ON THE PRACTICES OF AUTONOMOUS SYSTEMS DEVELOPMENT: SURVEY-BASED EMPIRICAL FINDINGS

A PREPRINT

Katerina Goseva-Popstojanova
Department of Computer Science
West Virginia University, Morgantown, WV 26506, USA
Katerina.Goseva@mail.wvu.edu

Denny Hood
Department of Computer Science
West Virginia University, Morgantown, WV 26506, USA
dlh0038@mix.wvu.edu

Johann Schumann
KBR/Wyle LLC, NASA Ames Research Center, Moffett Field, CA 94035, USA
johann.m.schumann@nasa.gov

Noble Nkwocha
NASA Independent Verification & Validation Facility, Fairmont, WV 26554, USA
noble.n.nkwocha@nasa.gov

ABSTRACT

Autonomous systems have gained an important role in many industry domains and are beginning to change everyday life. However, due to dynamically emerging applications and often proprietary constraints, there is a lack of information about the practice of developing autonomous systems. This paper presents the first part of the longitudinal study focused on establishing state-of-the-practice, identifying and quantifying the challenges and benefits, identifying the processes and standards used, and exploring verification and validation (V&V) practices used for the development of autonomous systems. The results presented in this paper are based on data about software systems that have autonomous functionality and may employ model-based software engineering (MBSwE) and reuse. These data were collected using an anonymous online survey that was administered in 2019 and were provided by experts with experience in the development of autonomous systems and / or the use of MBSwE. Our current work is focused on repeating the survey to collect more recent data and discover how the development of autonomous systems has evolved over time.

1 Introduction

The advancements of computing technologies (i.e., sensors, embedded processing, computer vision, hardware acceleration, machine learning) and communication technologies have enabled and motivated many organizations to work on development of autonomous systems. Domains such as space exploration, military, agriculture, and healthcare already have numerous use cases for autonomous systems. Many automotive companies, such as ArgoAI, Audi, Baidu, Cruise, Mercedes-Benz, Nissan, Tesla, Uber, and Waymo, have embraced and made enormous investments self-driving car technology (1). However, extremely high levels of complexity and safety criticality present fundamental challenges (2). Autonomous systems have complex interactions – they perceive the environment and execute tasks without detailed programming or under direct human control. Unlike automated systems, which execute a carefully engineered

sequence of actions that cannot be changed, autonomous systems are self-governing their course of action, that is, they are meant to understand and decide how to execute tasks based on goals, skills, and learning experience (3).

NASA defines autonomy as the ability of a system to achieve goals while operating independently of external control (4). An autonomous system can, and in the case of space domain often must, make decisions and take actions with little or no human involvement (5). Autonomous systems range from model-based diagnostics and prognostics to machine learning (ML) and artificial intelligence (AI) based systems, and may be both safety and mission critical, where failures can place human life and the mission in jeopardy. For example, in spite of the advances in the domain of autonomous vehicles, catastrophic accidents have happened, such as an autonomous car misinterpreting a white truck as a white cloud, and another one overlooking pedestrians on a road, which led to a fatal accident (3). More recently, Tesla has recalled 362,758 vehicles and warned that the driver assistance software, marketed as "Full Self-Driving Beta", may cause crashes (6).

Autonomous systems are composed of Autonomous Components (AUCs) that provide autonomous capabilities and support autonomous operations. Some examples of AUCs are vision-based navigation, system health management and prognostics, flight management systems that can operate without human intervention, and subsystems of deep space missions which perform complex operations without direct remote operator control. AUCs can implement different autonomous tasks related to (1) information acquisition, (2) information analysis, (3) decision and action selection, and (4) action implementation (7).

Autonomous systems impose novel and hard challenges for Verification and Validation (V&V). For example, based upon their literature review of verification and validation techniques for space autonomous systems (5), the authors concluded that verification and validation are still technological challenges and emphasize the need for providing assurances and guarantees towards reliable missions. Another literature review work focusing on testing autonomous vehicles (8) has identified four categories of significant challenges: complex and unpredictable environment, concerns with existing automotive testing methods, incompatible safety standards and certification, and machine learning induced challenges. To achieve dependable and trustworthy autonomous systems, intelligent V&V techniques that cover dynamic changes and learning are needed (3).

Software standards for safety-critical systems, like DO-178C (9), DO-331 (10), or ISO 26262 (11) for the automotive industry have been established at the time when this survey was held in 2019. These standards aim at complex, safety-critical systems that may have some automation capabilities, promote concpets like Reusable Software Components (RSCs) (9), and focus on model-based development and verification like the DO-331 supplement to DO-178C (10). Guidelines and standards for development and V&V of autonomous systems, e.g., (12; 13; 14), have been published several years after our survey has been administered.

Due to the emerging nature of autonomous systems and their proprietary nature, there is a lack of information about the state-of-the-practice, and the actual benefits and challenges observed in practice. This paper presents the first part of the longitudinal study focused on addressing this gap. The results are based on data collected from April 25, 2019 to June 20, 2019 by means of an anonymous survey. The survey focused on autonomous systems from different industry domains worldwide, and included aspects related to Model-based Software Engineering (MBSwE) and reuse. Out of 129 respondents to the survey, 110 used autonomous systems and/or MBSwE in their projects and answered questions beyond the first survey question. To assure that respondents' answers are not biased, the survey was administered anonymously. The survey had six specific sections addressing multiple research questions, which are listed in Table 1. The detailed analysis of the corresponding results led to the following contributions, pertinent to the time period in which the data were collected (i.e., 2019):

C1. established the **state-of-the-practice of developing autonomous systems** (i.e., answered the "Where", "What", "How", "Why", and "Who" questions) based on expert opinions collected using an anonymous survey,

C2. identified and quantified the **challenges and benefits of autonomy and reuse**,

C3. identified the **processes and standards** used to develop autonomous systems, and

C4. explored the **verification and validation used for the autonomy, models, and reuse**.

The work presented in this paper is comprehensive with respect to autonomous systems in general, from multiple domains, throughout the life cycle, and also incorporates the use of MBSwE and reuse. Most of the related studies focused on autonomous systems were based on literature review and systematization. Some of these studies were focused only on one domain (i.e., Unmanned Aerial Vehicles (UAV) (15) or Space (5) or Autonomous Vehicles (AV) (16; 1; 17; 8)), a particular aspect of the development, like verification and validation (18; 19; 20; 21; 16; 22; 8), or particular quality attribute (e.g., safety (16) or security (23; 17)). Different from most of the related works, our findings are based on the practical experiences of experts on autonomous systems, both from academia and industry. Only several prior works were based on experts practical experiences: our previous work which was focused only

Table 1: Survey Sections with corresponding research questions and contributions

Survey Section 1: Where? What? How? and Who?		
RQ1a	Which areas of industry are using autonomous systems and/or MBSwE?	C1
RQ1b	What is the level of safety criticality of the applications where autonomous systems and/or MBSwE are being used?	
RQ1c	What are the programming languages used during the development and deployment of the project?	
RQ1d	How was the code for autonomous functionality during development and deployment developed?	
RQ1e	Was special hardware and/or cloud used for code implementing autonomous functionality?	
RQ1f	What were the respondents' roles in the project?	
Survey Section 2: Details on Autonomy (for each AUC in the project)		
RQ2a	Was the AUC developed using MBSwE and which MBSwE tools were used?	C2
RQ2b	What is the level of autonomy of AUC?	
RQ2c	What algorithms and modeling paradigms were used to develop AUC?	
RQ2d	How were the requirements for the AUC specified?	
RQ2e	What were the challenges associated with the AUC?	
Survey Section 3: Details on Reuse of Software Artifacts		
RQ3a	Which artifacts were reused and to what extent?	C3
RQ3b	Were there any negative aspects of reuse?	
RQ3c	What were the difficulties due to reuse?	
RQ3d	What were the benefits of reuse?	
Survey Section 4: Processes and Standards		
RQ4a	Which life-cycle model was used?	C3
RQ4b	Which modeling standards and coding standards were used by the projects?	
RQ4c	Did the system go through a certification process and was the AUC part of the certified system?	
Survey Section 5: Verification & Validation		
RQ5a	Which quality attributes were verified and validated?	C4
RQ5b	How were the models verified & validated?	
RQ5c	How were the AUCs verified & validated during development?	
RQ5d	How was runtime behavior of AUCs monitored/assured?	
RQ5e	Were the reused artifacts verified & validated?	
Survey Section 6: Bugs		
RQ6a	Where there any bugs specific to autonomous functionality, model-based approach, and/or reuse?	C4

on MBSwE (24), another work based on a survey (25) published in 2006 which was focused only on space domain, and a more recent work that addressed only testing of autonomous systems and was based on focus discussions and interviews with a small sample of experts (22).

The remainder of the paper is organized as follows: Section 2 discusses the related works. Section 3 presents an overview of the survey, its design, and how the survey was administered. Detailed analysis of the results of the survey are discussed in Section 4. The threats to validity are described in Section 5. Finally, Section 6 provides a summary of the observations and recommendations.

2 Related Work

Due to its emerging nature, many papers on autonomy have been published. We start from the related works that were concerned with the level of autonomy and the types of tasks (i.e., information processing categories) (26; 7; 25; 27; 28; 15). Different levels of autonomy were first introduced by Sheridan et al. (26). Parasuraman et al. (7) introduced a framework for types and levels of autonomy that act as basis to determine the extent of autonomy for different tasks. The authors introduced four distinct information processing categories (i.e., tasks) of autonomy: information acquisition, information analysis, decision and action selection, and action implementation. The level of autonomy was defined on a scale from 1 to 10, where 1 is full human decision making and 10 is full computer

decision making; values in between represented combinations of different levels of human and computer interactions. Different levels of autonomy can be applied to each task. LaVallee et al. (25) proposed the following six levels of autonomy tailored to space domain: "manual", "automatic notification", "intelligent reasoning on ground with human control", "intelligent reasoning on ground with autonomous control", "intelligent reasoning onboard", and "autonomous thinking spacecraft". The Society of Automotive Engineers (SAE) has defined six levels of vehicle autonomy, with Level 0 (L0) being the lowest (i.e., no autonomy) and Level 5 (L5) being the highest (i.e., full autonomy in any driving environment) (27). Most of the modern vehicles have Level 1 autonomy, with at least one autonomous feature (e.g., braking, acceleration assistance, or cruise control). Vehicles with Level 2 autonomy use a combination of autonomous features to manage steering, acceleration and braking, but the driver must remain engaged and monitor the environment at all times. GM's Super Cruise and Tesla's Full Self-Driving are examples of Level 2 systems. Mercedes-Benz became the first automaker certified to sell vehicles with SAE Level 3 autonomous technology, which requires a human driver to be ready to take control over the vehicle if necessary. According to Mercedes-Benz, Level 4 autonomy may be possible by the end of the decade (29). Proud et al. (28) developed the NASA's SMART (Spacecraft Mission Assessment and Re-planning Tool) as a prototype to functionally decompose a flight management system with a suitable level of autonomy for the desired functionality. The authors introduced the Level of Autonomy Assessment Tool which was designed to find the optimum level of autonomy that minimizes the cost, and maximizes safety and efficiency. To measure the level of autonomy, Chen et al. (15) presented concepts related to autonomous systems, such as control level metrics. The architecture of autonomous UAV systems were divided into three levels: execution, coordination, and organization.

Based on published related works, the work presented in (1) summarized the state-of-the-art of computing systems for AV. That work considered seven performance metrics (i.e., accuracy, timeliness, power, cost, reliability, privacy, and security) and nine key technologies (i.e., sensors, data source, autonomous driving applications, computation hardware, storage, real-time operating systems, middleware systems, vehicular communication, and security and privacy). In addition, it identified twelve challenges to realizing autonomous driving: artificial intelligence for AVs, multisensors data synchronization, failure detection and diagnostics, how to deal with normal–abnormal, cyberattack protection, vehicle operating system, energy consumption, cost, how to benefit from smart infrastructure, dealing with human drivers, experimental platform, and physical worlds coupling.

Many related works were focused on different aspects of verification and validation of autonomous systems (18; 19; 20; 21; 22; 5; 8). Schumann et al. (18) presented a literature review of the use of neural networks in high assurance systems of various fields of study. They concluded that traditional verification and validation for safety-critical code is insufficient for neural network applications. Techniques for analysis and verification and validation of the System Health Management (SHM) were presented in (19). Specifically, a combination of n-factor combinatorial exploration and Monte Carlo techniques were used, which allowed for detection of potential weaknesses and unwanted parameter sensitivity in the health model. Schumann et al. (20) proposed a Bayesian method to diagnose and avert software faults in real-time using Software Health Management (SWHM). Using a two stage verification and validation process, both the model and code levels of a safety critical component were analyzed. The authors concluded that SWHM can provide an additional layer of safety during runtime, but was not suitable for replacement of verification and validation, and certification of the system. A paper by Nikora et al. (21) investigated the verification and validation techniques for systems with autonomous capabilities in the following areas: diagnostic model, diagnostic engine, and combination of model and engine. The authors noted that test procedures and expected results about a component's nominal functionality were more easily obtained than the off-nominal behavior. A recent review of verification and validation techniques for space autonomous systems discussed the model checking, theorem proving, runtime verification, software testing, and verification and validation of machine learning (5). The authors concluded that verification and validation were still challenges for space autonomous systems. Another recent literature review by Karunakaran et al. (8), which was focused on testing AV, identified four categories of significant challenges: complex and unpredictable world, concerns with existing automotive testing methods, incompatible safety standards and certification, and machine learning induced challenges. The work by Song et al. (22) was also focused on testing of autonomous systems, and in addition to literature review used focus group discussions consisting of 8 participants and semi-structured interviews with 5 participants. That work classified the challenges of testing autonomous systems to four categories: unpredictable environment; system and scenario complexity; data accessibility; and missing standards and guidelines. It also classified the available techniques, approaches, and practices for testing of autonomous systems.

Even though the empirical analysis of software bugs is a very active research area, only several recent papers were focused on studying software bugs in systems with autonomous functionality (30; 31; 32; 33; 34). Of these, two recent works were focused on AV bugs (30; 31). Garcia et al. (30) presented an empirical study of 499 bugs of the open source autonomous driving systems Apollo and Autowarein, and using manual analysis classified the root causes and symptoms of the bugs, as well as identified the autonomous components affected by these bugs. The study by Tang et al. (31) was based on the open-source driver assistant system OpenPilot, for which the authors collected 235 bugs

and classified them into five categories. Another two recent paper studied the bugs in UAS (32; 33), both based on two open source software suites PX4 (capable of controlling drones) and ArduPilot (capable of controlling unmanned vehicle systems such as drones, aircrafts, helicopters, ground rovers, boats, submarines, and antenna trackers). Taylor et al. (32) investigated the root causes, symptoms, reproducability of the bugs, and location at component level of 277 firmware bugs in the ArduPilot and PX4 code bases. Wang et al. (33) extracted 569 bugs of ArduPilot and PX4, and using a manual labeling process identified 168 UAS-specific bugs whose root causes were classified into 8 categories. In our work (34), we investigated software changes and bugs in the Autonomy Operating System (AOS) for UAS, which has 26 components with a total of 772 bugs. The results showed that autonomous components were significantly more prone to changes (measured in number of commits and code churn) and fault prone (measured in bugfixes per KLoC) than non-autonomous components. Furthermore, the distribution of the locations of bugs was skewed, both at component and file level (i.e., a small number of components / files contained the majority of bugs).

While safety considerations in complex hardware-software systems have been studied for many years, major work on safety for autonomous systems started after our survey, e.g., (35; 36; 37).

Some works were focused on security aspects of autonomous systems (23; 17). For example, the literature review paper by Jahan et al. (23) presented attack models that have been proposed over the years, proposed a taxonomy of attacks on autonomous systems, and identified the research gap that needs to be addressed. Another work, which was focused specifically on security of autonomous driving (17), elaborated the security issues along four dimensions: sensors, operating system, control system, and vehicle-to-everything (V2X) communication.

We next discuss the empirical works that used survey as an instrument to collect the relevant data (24; 25). These include our prior work which was based on conducting a survey focused only on the use of MBSwE and auto-generated code (AGC) in various industry domains worldwide (24). Specifically, based on the answers provided by 114 respondents to the survey, we explored the state-of-the-practice, the benefits and challenges, and software assurance of models and AGC. The only previous survey related to autonomous systems was conducted by Lavallee et al. (25). That work was focused only on space domain and presented a categorization of the level of autonomy and the complexity of the implementation, from a single component to an entire flight and ground systems. That work used six levels of autonomy and, based on 88 survey responses for 62 implementations, reported that lower levels of autonomy dominated (with 45% of implementations having Level 2 autonomy, and only 3% having Level 5 and 8% having Level 6 autonomy).

In this paper, we present the results of the anonymous online survey which was used to collect information about software systems that have autonomous functionality, may have employed model-based software engineering and some level of software reuse, for different domains, worldwide. The survey was conducted from April 25, 2019 to June 20, 2019. We aimed to fill the existing gaps in the knowledge related to autonomous systems development at that time by (C1) assessing the state-of-the-practice using autonomous systems, (C2) identifying and quantifying the benefits and challenges of autonomy and reuse, (C3) exploring the processes and standards used to develop autonomous systems, and (C4) investigating the verification and validation of the models, autonomy, and reuse. Unlike most of the related work papers which were based on literature review, the findings presented in this paper are based on the practical experience of experts on autonomous systems, both from academia and industry.

3 Survey Design and Execution

The survey was developed following the steps outlined in (38): (1) define the goals, (2) transform the goals into research questions, (3) design the questionnaire, (4) evaluate the questionnaire using pilot executions, (5) execute the survey, and (6) analyze and package results. Our Survey consisted of 122 multiple choice and free response questions divided into the following sections (see Table 1):

1. *"Where", 'What", "How", and "Who"* section addressed questions about areas of industry, level of safety criticality, programming languages used, how was the code developed and if the special hardware was used, as well as the respondents role in the project.
2. *Autonomy Details* section contained questions about the level of autonomy, modeling paradigms, requirement specification, and challenges developing autonomous systems.
3. *Reuse* section was concerned with the extent of reuse for AUC and non-AUC and the benefits and challenges related to reuse.
4. *Processes and Standards* section focused on lifecycle, modeling, and coding standards, and certification.
5. *Verification and Validation* section concentrated on questions about V&V of the models, autonomy, and reuse.
6. *Bugs* section dealt with bugs related to the models, autonomous functionality, and reuse of software artifacts.

The survey was divided into separate pages based on these sections. Depending on answers to specific questions, respondents were presented with some questions and skipped other questions. As can be seen in the flowchart of the survey shown in Figure 1, the first question was used to determine if the respondent has worked on autonomous systems, followed by the second question to determine if the respondent has used MBSwE. If the respondent neither worked on autonomous systems nor used MBSwE, they were led to the end of the survey. Otherwise, the respondent was asked the first set of questions about "Where", "What", "How" of their work. Then, the respondents were asked about the number of AUCs they worked on. Based on the response, the questions from the Autonomy Details section were repeated for each AUC. Next, the respondents were asked about the existence of reuse in their project. If the respondent chose yes, they were directed to the questions in the Reuse Details section. Otherwise, these questions were skipped. All respondents who were allowed to enter the survey went through the questions from the Processes and Standards, Verification and Validation, and Bugs sections.

The online questionnaire was created using Survey Monkey (39). A pilot study was used to evaluate the first version of the survey questionnaire. Our colleagues and contacts who had practical experience in developing autonomous systems participated in the pilot study and their comments and suggestions were used to revise and improve the survey questionnaire.

Invitations to complete the survey were distributed by email to people who work in related fields. The intended respondents were software engineering practitioners and researchers with experience in developing autonomous systems and / or using MBSwE in industry. 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. Additionally, over 300 authors of research papers related to autonomy were contacted via email invitations. The responses to the survey were collected from April 25, 2019 to June 20, 2019.

Of the 129 respondents who started the survey, 110 respondents have worked on autonomous systems and/or used MBSwE in their projects. Respondents were allowed to enter the survey, that is, answered more than just the first two survey questions. Note that it was not mandatory to answer each question.

4 Detailed Analysis of Survey Responses

This section presents the analysis of the survey responses provided by the 110 respondents who have used AUC and/or MBSwE in their projects. Since the survey did not require the respondents to answer all questions, for each figure and table, we provide the actual number of respondents and/or the percentages of the total number of respondents answering that specific question.

4.1 Survey Section 1: Where? What? How? and Who?

In the first section of the survey, we examined the state-of-the-practice of developing autonomous systems in order to provide answers to "Where", "What", "How", and "Who" questions.

4.1.1 RQ1a: Which areas of industry are using autonomous systems and/or MBSwE?

Autonomous systems are used in applications such as unmanned aerial and marine vehicles, self-driving cars, smart robots, and many other application areas. As shown in Figure 2, the space industry dominated by 40% of the respondents, followed by aviation at 16%, military at 13% and automobile at 9%. Domains included in "Other" were financial, government, transportation, Earth Ocean Sciences, and Department of Homeland Security. Compared to our previous study (24), which focused only on MBSwE, the number of respondents from the automotive industry in this survey was smaller (i.e., 33% versus 9%). This might be due to the the proprietary nature of autonomous systems in the automotive industry where respondents are not legally permitted to divulge information.

4.1.2 RQ1b: What is the level of safety criticality of the applications where autonomous systems and/or MBSwE are being used?

The level of safety criticality defines the impact of failures if the software or system fails. In the survey, we used the levels of safety criticality as defined in the DO-178C standard (9): catastrophic, hazardous, major, minor, and no effect. As shown in Figure 3, 45% of respondents indicated catastrophic or hazardous level of safety criticality, followed by 39% with major safety criticality, and 8% with minor criticality. Only 8% of respondents indicated an unknown level of safety criticality. Since a large portion of the respondents to our survey were from the space and aviation industry and from military, which typically deal with safety-critical applications, it is not surprising that many respondents

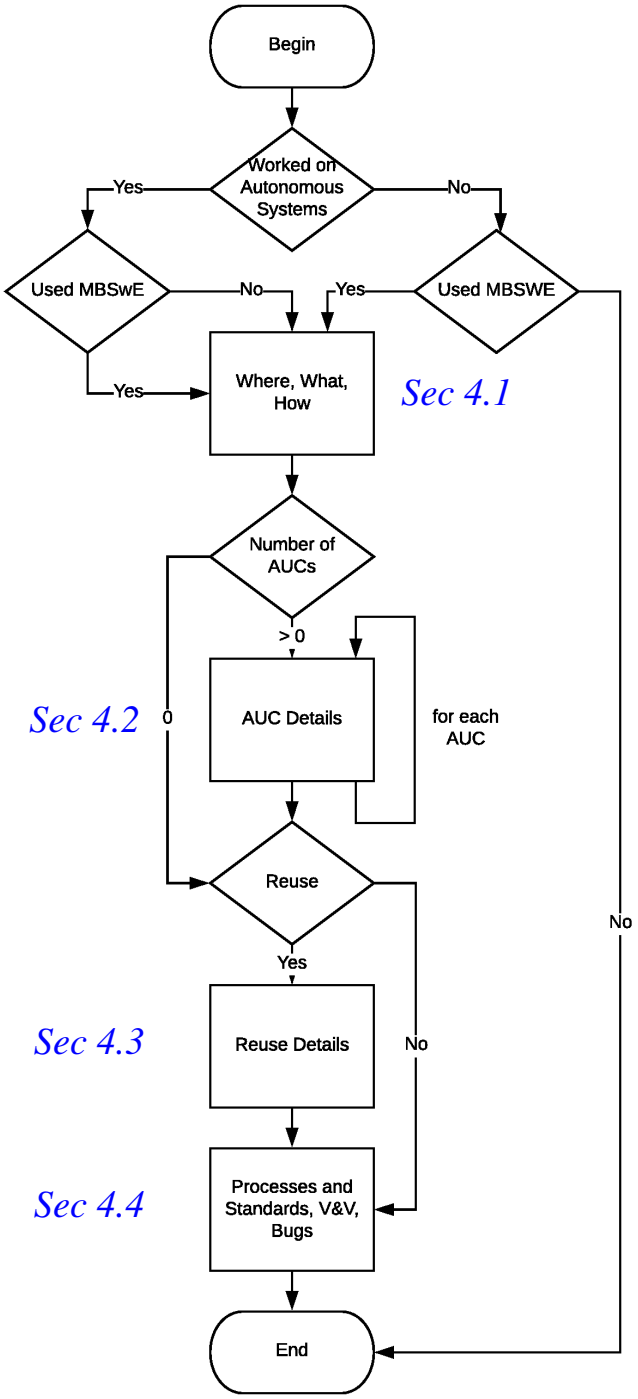


Figure 1: Flowchart of our survey

indicated a high level of safety criticality. These observations are consistent with the results from our previous survey (24), which used different values for the levels of criticality but had similar percentage of high criticality of 46%.

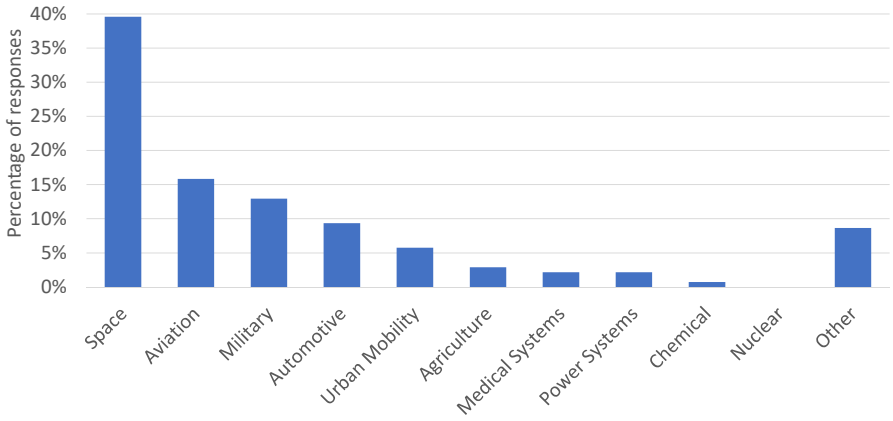


Figure 2: Areas of industry to which the respondents to our survey belong. (99 respondents, multiple answers possible)

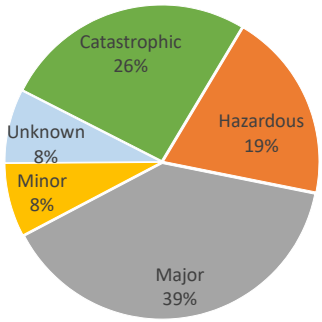


Figure 3: Safety criticality of applications (97 respondents)

4.1.3 RQ1c: What are the programming languages used during the development and deployment of the projects?

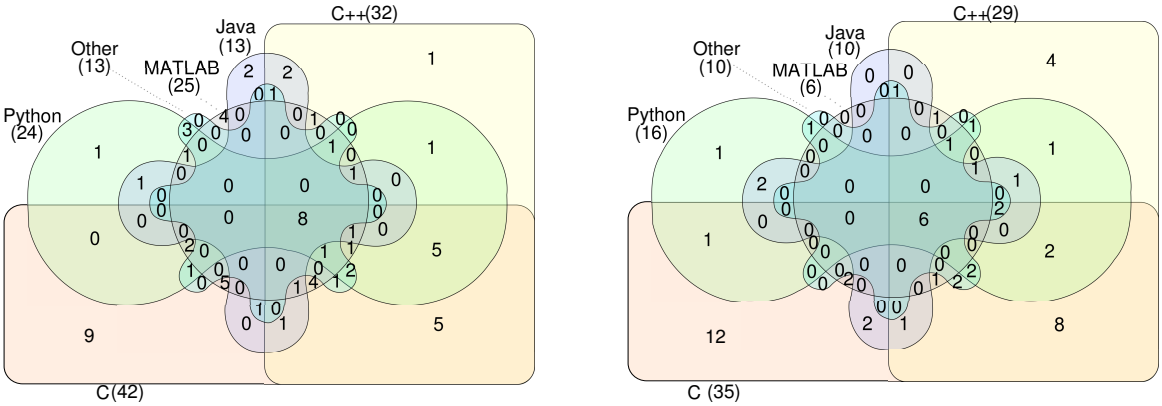
In our survey, we distinguished between programming languages and tools that were used during the development and those used in deployed code during operation. Programming languages used during development (e.g., Python or TensorFlow used to train a Deep Neural Network in the cloud) produce intermediate artifacts. Programming languages used in deployed code are executed during system operation. Figure 4a presents a Venn diagram of the programming languages used during development. Note that the respondents could choose multiple languages. In fact, 8 respondents indicated they used all of the programming languages shown in the figure and most respondents indicated use of multiple languages. C and C++ dominated the responses, with 42 and 32 responses respectively, followed by MATLAB and Python, with 25 and 24 responses, respectively.

Programming languages used during deployment are given in Figure 4b. Here as well the respondents could choose multiple languages. C and C++ dominated with 35 and 29 responses respectively, followed by Python with 16 responses and MATLAB with 6 responses. The category of "Other" programming languages in Figures 4a and 4b included C#, assembly, Lisp, JavaScript, and G2.

Table 2: Programming languages used during development and deployment (97 respondents, multiple selections possible)

	C	C++	Matlab	Python	Java	Other
Development	42	32	25	24	13	13
Deployment	35	29	6	16	10	10

To compare the use of programming languages during development and deployment, the total number of responses for each language shown in Figures 4a and 4b are also shown in Table 2. These values indicated that the languages used



(a) During development

(b) During deployment

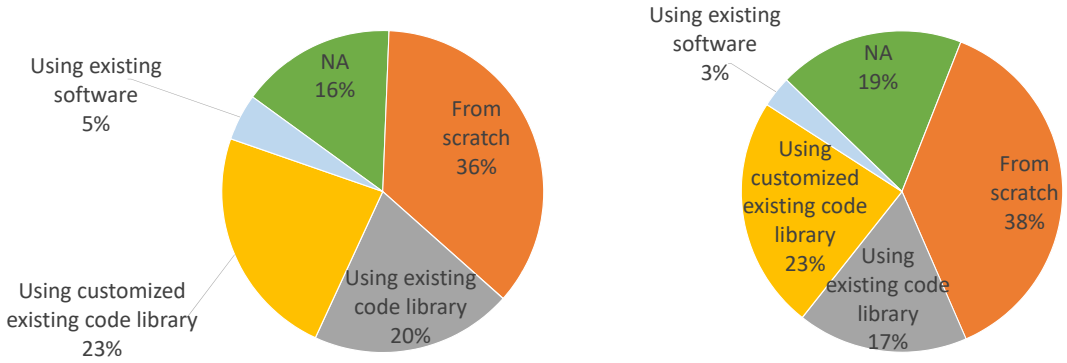
Figure 4: The use of different programming languages (97 respondents, multiple selections possible)

during development and during deployment were very similar. As expected, embedded AUCs are mainly implemented in C or C++, or a mixture thereof. It seems that some languages, which are not very widespread in safety-critical and embedded systems, like Python, are being used as implementation languages for AUCs. Note that there is stronger preference to using Python during development than during deployment. The much higher usage of MATLAB during development over deployment may be attributed to the use of Mathworks tools for model-based software development. Typically, models are developed using Simulink and MATLAB while the auto-generated code (generated by the Mathworks code generator) is run during deployment. Finally, no arcane or special-purpose languages have been used for development and implementation of AUCs. The most outstanding example here is Lisp.

4.1.4 RQ1d: How was the code for autonomous functionality during development and deployment developed?

Respondents were asked about how the autonomous code was developed and could only choose one answer from the provided answers in the survey. As can be seen in Figure 5a, over one third of the respondents indicated that the code during development was written from scratch, i.e., there was no reuse. 20% of respondents indicated use of (unmodified) existing code libraries and 23% of respondents indicated use of customized existing code libraries. Only 5% of respondents selected reuse of existing software.

Figure 5b depicts the origins of the autonomous code during deployment. (Here also the respondents could choose only one of the options.) The results are similar to those during development, with 38% of the deployed code developed from scratch, 17% using unmodified existing code library, 23% using customized existing code library, and only 3% reusing existent software.



(a) Origins of code during development

(b) Origins of code during deployment

Figure 5: How was the code for autonomous functionality developed (69 respondents)

4.1.5 RQ1e: Was special hardware and/or cloud used for developing and running code that has autonomous functionality?

We asked the respondents about the use of special hardware for development and running of the code with autonomous functionality. As shown in Table 3, only 16% of respondents used special hardware during development and 19% during operation. The special hardware that was specified by the respondents included touch screens, Nvidia Jetson, and custom designed space robot prototypes.

We also asked if any computations were executed in the cloud or on a remote server. Similarly as with the special hardware, the overwhelming majority (i.e., 70%) of respondents did not execute any software in the cloud or on a remote server. 9% of respondents used a cloud or remote server only during design, and additional 6% both during design and operation. (15% of respondents selected the NA option.)

The low usage of special hardware and cloud services was somewhat surprising. It may be due to the fact that, as described in RQ2c (Subsubsection 4.2.3), rule based algorithms still dominate as development approach of autonomy.

Table 3: Use of special hardware (69 respondents)

During:	development	operation
Special hardware	16%	19%
No special hardware	70%	66%
NA	14%	15%

4.1.6 RQ1f: What were the respondents roles in the project?

Figure 6 presents the respondents' roles in their projects. As expected, the two respondents' roles – Design and Research – dominated with 21 and 17 respondents, respectively. The roles of Model development, Programming, Software and System integration, Testing/QA/V&V, and Project Management were roughly evenly distributed with 12 to 13 respondents per category. Under the "Other" category the respondents mentioned the roles of Certification and Independent Verification and Validation (IV&V) analyst. Interestingly, no tool developers or vendors were among the respondents, which may be due to the way the survey was distributed.

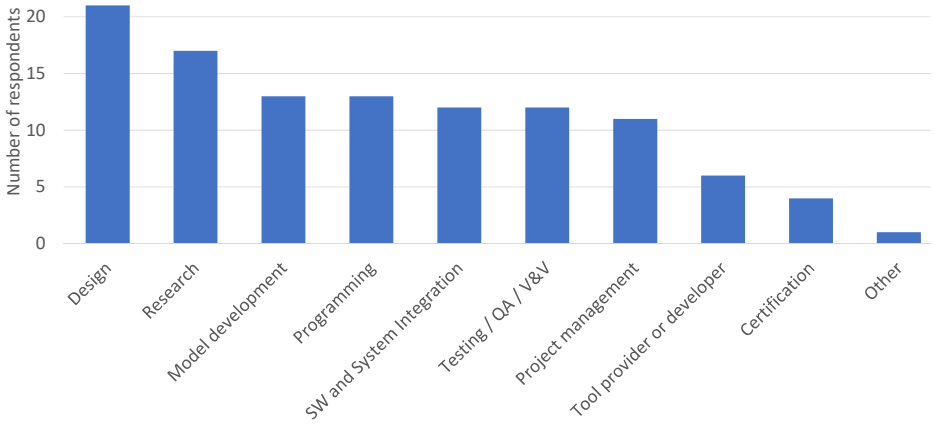


Figure 6: Respondents' roles in the project (34 respondents, multiple selections possible)

4.2 Survey Section 2: Details on Autonomy

Since an autonomous system can contain more than one AUC, we solicited details about autonomy for each AUC. For this purpose, the respondents were first asked to specify the number of AUCs in the system by choosing a number between 0 and 10. The respondents who selected 0 AUC were directed to the reuse section of the survey (see Figure 1). 38 respondents who chose one or more AUC were asked to answer a set of questions about autonomy details for each AUC. Of these 38 respondents, 28 (i.e., 74%) provided details for only one AUC. Six respondents provided details about two AUCs each. Three sets of two respondents each entered details about three, four and eight AUCs,

respectively. The analysis of the details on autonomy in this section is based on a total number of 58 AUCs entered by 38 respondents.

4.2.1 RQ2a: Was AUC developed using MBSwE and which MBSwE tools were used?

Only 38% of AUCs were developed using MBSwE. Specifically, 17% of AUCs were developed using MATLAB/Simulink, which explains the relatively high usage of MATLAB during development (see Figure 4a and Section 4.1.3). Another 13% of AUCs were developed using Rational Rhapsody. Several other less frequently used tools include Rational Rose and Magic Draw. The "Other" category included the following: Java, Prolog, Papyrus (40), and Generic Modeling Environment (GME).

4.2.2 RQ2b: What was the level of autonomy of AUCs?

An autonomous component and/or autonomous system may achieve different levels of autonomy. Multiple frameworks that define levels of autonomy have been proposed in the past (23). However, currently no universal framework for defining the levels of autonomy across different domains exists.

In this paper, we decided to use a simplified list of four levels of autonomy: (1) the autonomy is used only as a tool for assistance (i.e., "the human is primary and the computer is secondary", (2) "the computer operates with human interaction", (3) "the computer operates independently of the human" where the human has limited capabilities, and (4) "the computer has full autonomy".

In our survey, for simplicity and to accommodate different application domains, we asked the respondents to select from the four levels of autonomy mentioned above. Following (7), we included the following autonomy tasks in the survey:[1] Information acquisition (monitor), Information analysis (analyze), Decision and action selection (decide), and Action implementation (act).

Figure 7 presents the results for the level of autonomy for these four tasks, for 50 AUCs. Interestingly, for each task, a significant percentage of AUCs (i.e., between 50% and 67%) operated with the computer having a full autonomy. Of the four tasks, "Decision and action selection" had the highest dependence on human interaction. Only 5% to 12% of AUCs had the lowest level of autonomy (i.e., "the human primary and the computer secondary").

It appears that the level of autonomy has increased significantly when compared to the results reported in the related survey (25), which focused on autonomous systems from the space domain and concluded that the lower levels of autonomy dominated (with 45% of implementations having only automatic notification).

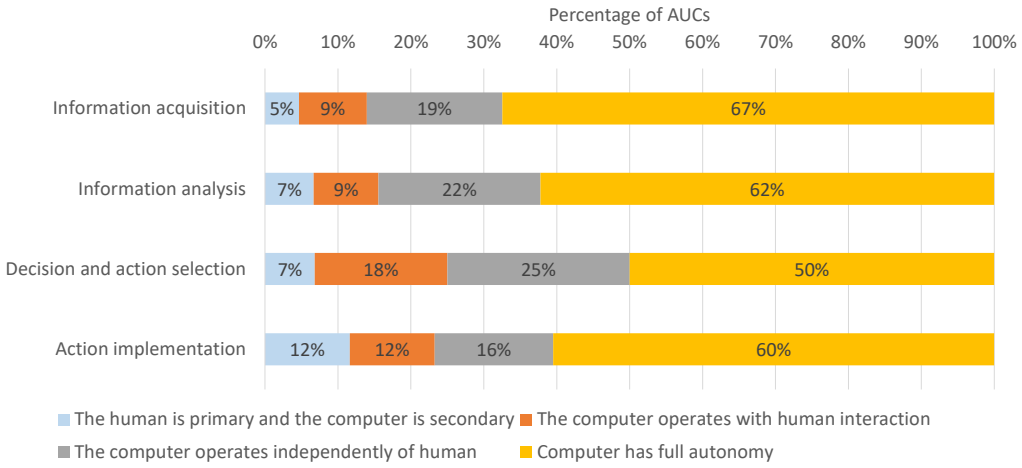


Figure 7: Level of autonomy for specific tasks (50 AUCs)

[1]These tasks are based on the Observe Orient Decide Act loop (OODA) cognitive model proposed by the military strategist Colonel John Boyd (41).

4.2.3 RQ2c: What algorithms and modeling paradigms were used to develop AUCs?

As can be seen in Figure 8, which shows the percentage of AUCs developed using different algorithms and modeling paradigms; "Rule-Based" algorithms and methods, which are based on program logic or using explicit rules, were used for development of 35% of AUCs. "Planning Systems/Languages" and "Statistical and filtering methods" were used for development of 22% and 18% of AUCs, respectively. Interestingly, at the time our survey was conducted (i.e., mid 2019) these traditional algorithm and methods dominated the development of AUCs. We were surprised that machine learning approaches, which at that time were much hyped by the media as the core for autonomy, were used much less frequently (i.e., only 12%). Specifically, only 9% of AUCs used offline machine learning and 3% used online machine learning. (The "Other" category included nonlinear programming techniques, model checking, and nonlinear optimization.)

The low usage of machine learning approaches explains the finding of RQ1d (Section 4.1.4) related to the high percentage of code developed from scratch. Namely, higher usage of machine learning may have led to use of off-the-shelf algorithms for data preparation and training (e.g., TensorFlow, Keras, etc.) and thus increased the percentage of reuse. Furthermore, the low usage of machine learning approaches may be the reason behind the low usage of special hardware and cloud services as found in RQ1e (Section 4.1.5).

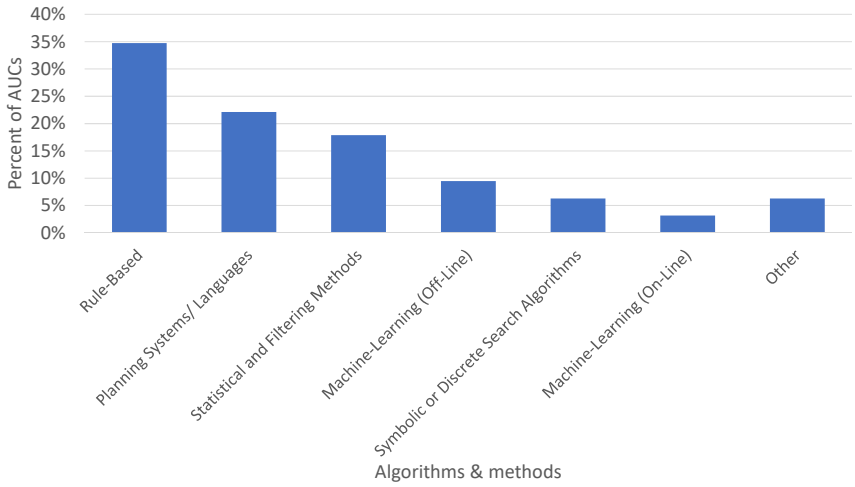


Figure 8: Algorithms and modeling paradigms used for development of AUCs (58 AUCs, multiple selections possible)

4.2.4 RQ2d: How were the requirements for the AUCs specified?

We also explored the ways requirements for AUCs were specified. The respondents could select more than one option for requirements specification of each AUC. As can be seen in Figure 9, only 10 AUCs were developed using more than one approach for requirement specification. For 31 AUCs, the requirements were specified same way as for non-AUCs. Natural Language was used more than twice as often as some form of formal specification.

The survey also included a question on the level of details of requirement specification for AUCs compared to non-AUCs. (For this question, the answer choices were mutually exclusive.) The majority of AUCs (i.e., 63%) were developed using requirements with the same level of details as the non-AUCs (Figure 10). However, for a quarter of AUCs the requirements specification was at higher level of details than for non-AUCs.

4.2.5 RQ2e: What were the challenges associated with the AUCs?

We also explored the challenges associated with AUCs, asking the respondents to use an ordinal scale for six different challenges given in Figure 11. (Since the numbers of responses were not the same for all six challenges, to allow for comparison, Figure 11 shows the percentages of the degrees of difficulty for each challenge.) If we consider the moderate and major difficulties together, "System complexity" was the most challenging (in 67% of the AUCs), followed by "High level of environment uncertainty" and "Achieving the desired level of autonomy", in 57% and 40% of the AUCs. While "Non-deterministic algorithms" led to some difficulties (in 23% of AUCs), "Lack of human's

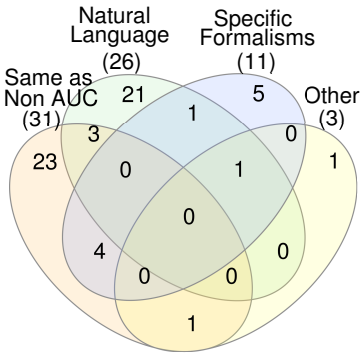


Figure 9: Specification of requirements for AUCs (51 AUC, multiple selections possible)

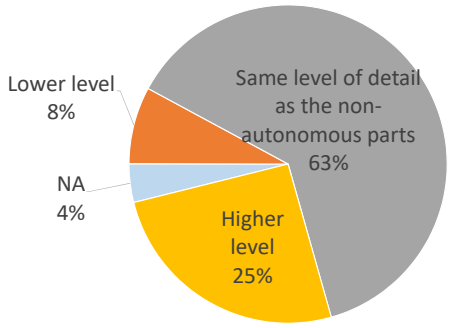


Figure 10: Level of details of requirements specification for AUCs compared to non-AUCs (51 AUC)

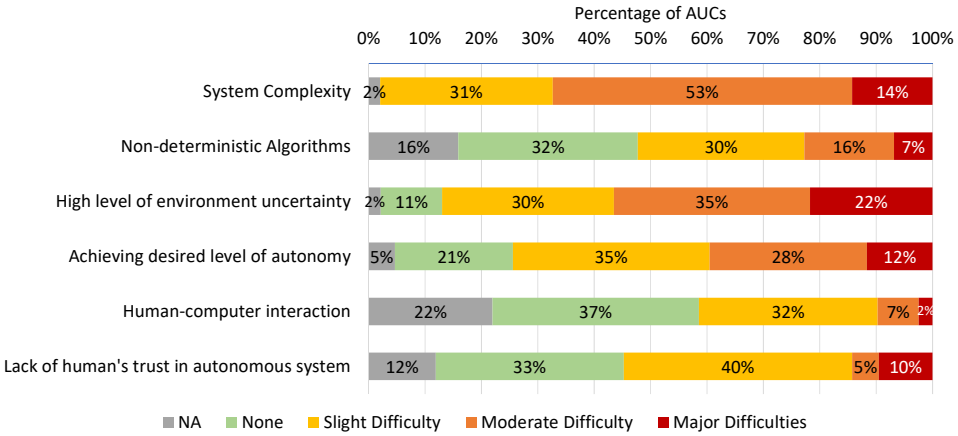


Figure 11: Degree of difficulty for challenges encountered during development, deployment and use of AUCs (50 AUCs)

trust in autonomous systems" and "Human-computer interaction" did not seem to be big issues (affecting only 15% and 9% of AUCs, respectively).

4.3 Survey Section 3: Details on Reuse of Software Artifacts

For the reuse of software artifacts, we explored reuse for both AUCs and non-AUCs. As can be seen from the survey flowchart in Figure 1, only respondents who answered "yes" to the question "Were software, data, or other artifacts reused in your project(s)?" were presented with the survey questions related to reuse (i.e., Survey Section 3).

4.3.1 RQ3a: What artifacts were reused and to what extent?

Figures 12 and 13 show the amount of reuse for different types of artifacts for AUCs and non-AUCs, respectively. For each type of software artifacts, respondents could select 30% or more reuse, less than 30% reuse, and Not Applicable (NA).

As can be seen in Figure 12, in case of AUCs, the code had the highest amount of reuse with 63% of the respondents' projects reusing 30% or more of the code, followed by 50% of respondents reusing more than 30% of "Model reuse: software models", and 39% of respondents having 30% or more "Data reuse: telemetry data and log files". When comparing the reuse of software artifacts in AUCs and non-AUCs (see Figures 12 and 13) similar trends were observed for all type of software artifacts except "Data reuse: offline training data" and "Data reuse: training machine learning model". As expected, these two artifacts have much less reuse in case of non-AUCs than in AUCs.

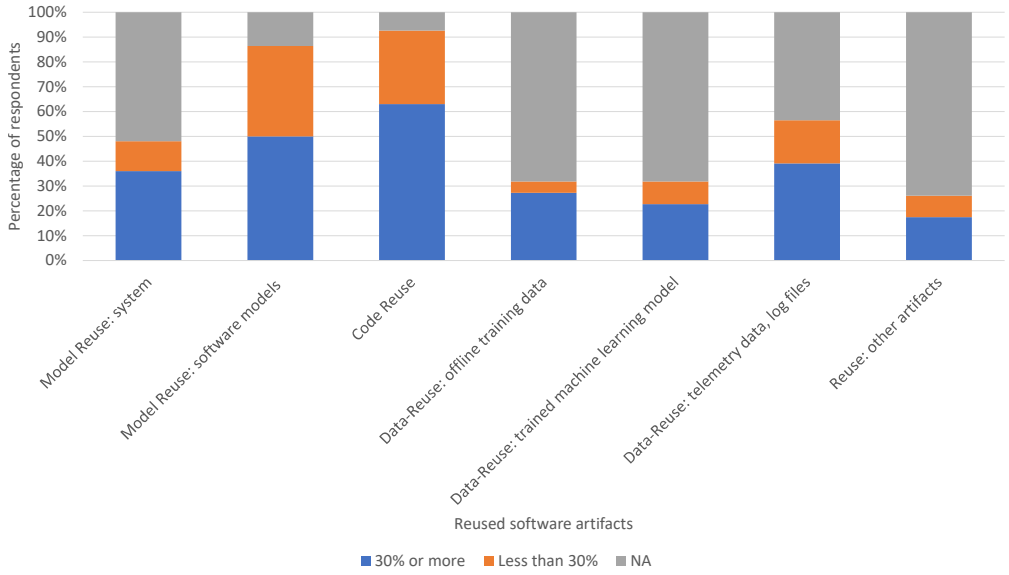


Figure 12: The extent of reuse of different software artifacts for AUCs (30 respondents)

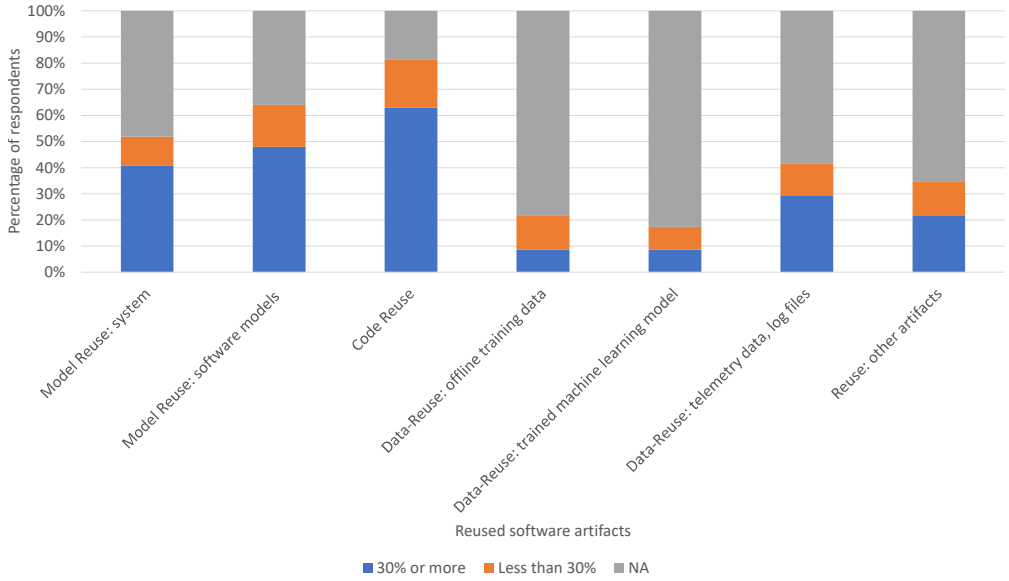


Figure 13: The extent of reuse of different software artifacts for non-AUC (30 respondents)

4.3.2 RQ3b: Are there any negative aspects of reuse?

For this research question we asked the respondents to rate (using an ordinal scale) four different negative aspects of reuse. Because it was not required to respond to each aspect, the numbers of responses for each aspect were slightly different. Therefore, Figure 14 shows the percentages of respondents. As seen in Figure 14, from 31% to 40% of respondents did not observe any negative aspects due to reuse. The largest negative aspect of reuse was due to "Hindering new ideas" with 14% of respondents selecting major and 11% selecting moderate negative aspect. This was followed by "Added complexity because of reuse" (with 4% major and 21% moderate) and "Additional cost due to reuse sustainability" (with 21% moderate negative aspects). Under "Other" negative aspects, respondents listed lack of documentation and difficulty adding patches into a formal design process.

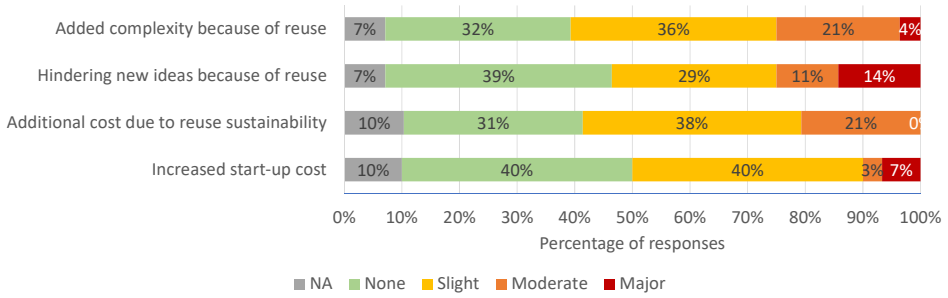


Figure 14: Negative aspects of reuse (30 respondents)

4.3.3 RQ3c: What were the difficulties due to reuse?

The amount of difficulty in different aspects of reuse are shown in Figure 15, using an ordinal scale. As seen in Figure 15 "Uncertain operational conditions / environment" led to most difficulties, with 11% of respondents indicating major and 29% indicating moderate difficulties. "Lack of planning for reuse in advance" and "Lack of information for reused software" each had 7% of respondents selecting major difficulties and 25% and 21% moderate difficulties, respectively.

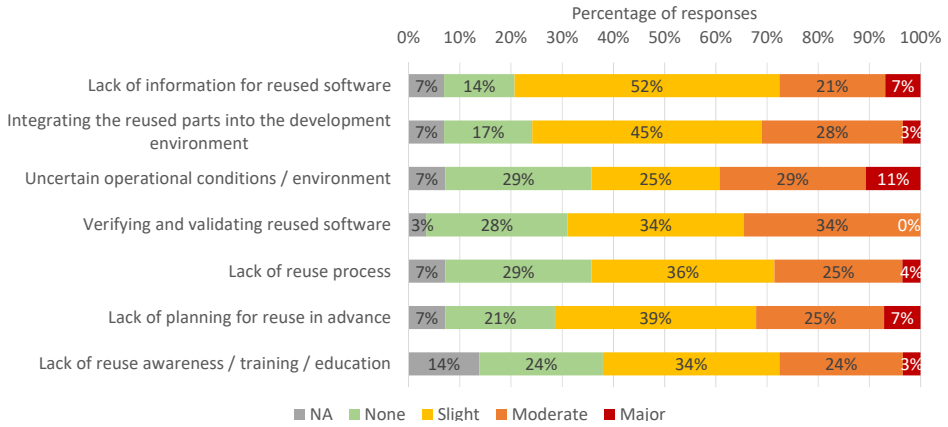


Figure 15: Difficulties due to reuse (29 respondents)

4.3.4 RQ3d: What were the benefits of reuse?

Figure 16 shows the benefits of reuse, in terms of productivity, quality, and cost. As can be seen, 63% of respondents reported increased productivity and 40% reported increased quality due to reuse. Interestingly, 23% of respondents reported decreased quality due to reuse. Surprisingly, only 37% of respondents reported decreased cost due to reuse. Note that the distribution was fairly uniform among "decreased", "about the same", and ''increased" cost due to reuse.

4.4 Survey Section 4: Processes and Standards

In this section we examine the processes and standards used when developing systems that have autonomous functionality and/or utilized model-based software development. As can be seen in the survey flowchart shown in Figure 1, all respondents of the survey were presented with the questions from Section 4 of the survey. However, since this section was towards the end of the survey, some respondents did not complete this section.

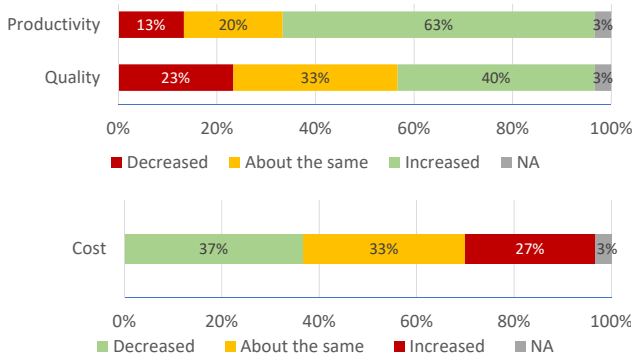


Figure 16: Benefits of reuse with respect to productivity, quality, and cost (30 respondents)

4.4.1 RQ4a: Which life-cycle model was used?

Figure 17 shows different types of life-cycle models used on projects. Interestingly, the Agile life-cycle model was used twice as often than the Waterfall model (i.e., 38% vs. 19%). Case-based development was used by 14% of respondents, followed by Rapid application development used by 11%, Spiral used by 8%, and Rational Unified Model used by 5% of respondents.

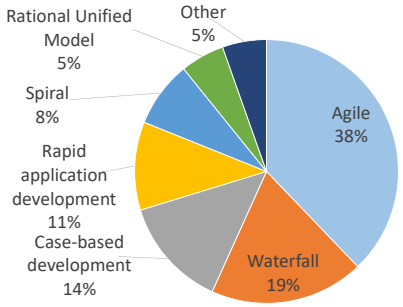


Figure 17: Life-cycle models used by projects (39 respondents)

4.4.2 RQ4b: Which modeling standards and coding standards were used by the projects?

As shown in Figure 18, almost half of the respondents (i.e., 48%) did not use any modeling standard. The most widely used standard was UML by 26% of respondents, followed by MathWorks MAAB (42) by 8% and SysML (43) by 5% of the respondents.

As can be seen in Figure 19, which depicts the coding standards used by the projects, about a quarter of the respondents did not follow any coding standard. Another quarter used standards not explicitly listed in the survey (i.e., "Other"), which included responses such as Orion, Thales, and internal coding standard. Large percentage of respondents used the NASA (19%) and JPL (18%) coding standards, followed by 13% who used MISRA coding standards.

4.4.3 RQ4c: Did the system go through a certification process and was the AUC part of the certified system?

As can be seen in Table 4, only 24% of systems went through certification and only 26% of AUCs were part of a certified system. The respondents who indicated their projects went through a certification process listed the following standards: NASA, Security Technical Implementation Guide (STIG), or an internal standard.

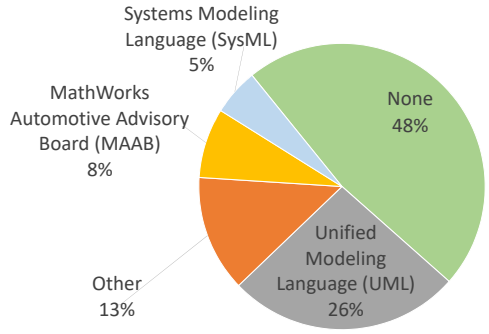


Figure 18: Modeling standards used by projects (40 respondents)

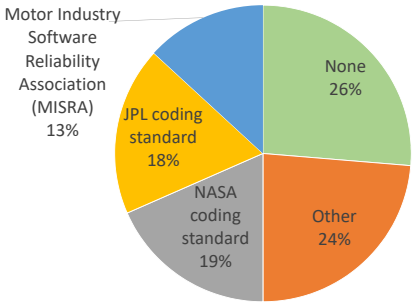


Figure 19: Coding standards used by projects (40 respondents)

Table 4: Information on certification process (41 respondents)

	Yes	No
System went through certification process	24%	76%
Autonomous components were part of a certified system	26%	74%

4.5 Survey Section 5: Verification and Validation

All respondents of the survey were presented with the questions from Section 5 of the survey (see the survey flowchart in Figure 1). However, since this section was towards the end of the survey, some respondents did not complete it.

4.5.1 RQ5a: Which quality attributes were verified and validated?

The survey had a question on which of the following quality attributes were verified and validated: correctness, performance, robustness, safety, and security. Respondents were allowed to select all that applied. The performance and correctness of the software were verified and validated most frequently, by 83% and 78% of the 36 respondents who answered this question, respectively. Smaller percentages of respondents verified and validated the robustness and safety (i.e., 61% and 56%, respectively). Surprisingly, only 11% of respondents verified and validated to security; these respondents verified and validated all the other quality attributes as well.

4.5.2 RQ5b: How were the models verified and validated?

Figure 20 shows the percentages of usage of different methods for verification and validation of the models. Testing was used most frequently (i.e., by 30% of respondents), followed by simulation used by 24%, manual model inspection/reviews used by 20%, and automated model analysis used by 10% of the respondents. Only 6% of respondents did not perform any verification and validation of the models.

As shown in Table 5 only 24% of respondents used tools to verify and validate the models. These tools included Design Verifier, Model Advisor, CoCoSim, and tools listed under "Other" which included UPPAAL (44), Jenkins (45), PRISM (46), and custom tools.

Table 5: Use of tools for verification and validation of the models (33 respondents)

Did you use tools for verification and validation of the models?	
Yes	24%
No	55%
NA	21%

Table 6: Verification and validation of autonomous functionality (32 respondents)

Did you specifically verify and validate autonomous functionality?	
Yes	56%
No	22%
NA	22%

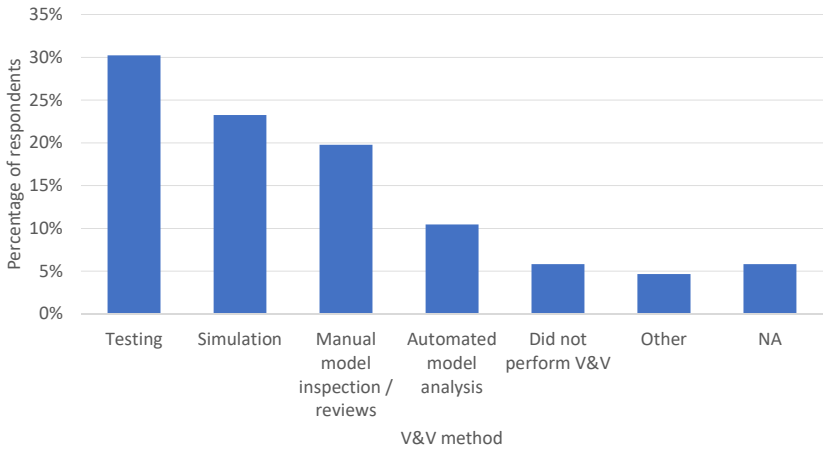


Figure 20: Methods used for verification and validation of the models (37 respondents, multiple selections possible)

4.5.3 RQ5c: How were the AUCs verified and validated during development?

As shown in Table 6, 56% of respondents specifically verified and validated the autonomous functionality. The use of different methods for verification and validation of AUCs during development are summarized in Figure 21. "Simulation-based testing" (20%), "Unit and integration testing" (19%), and "Manual model/code reviews or inspections" (18%) were used the most frequently. The least used verification and validation method was automated model analysis with only 3%. "Other" methods listed by some respondents included "Inspection of results by stakeholders" and "Comparing performance with human".

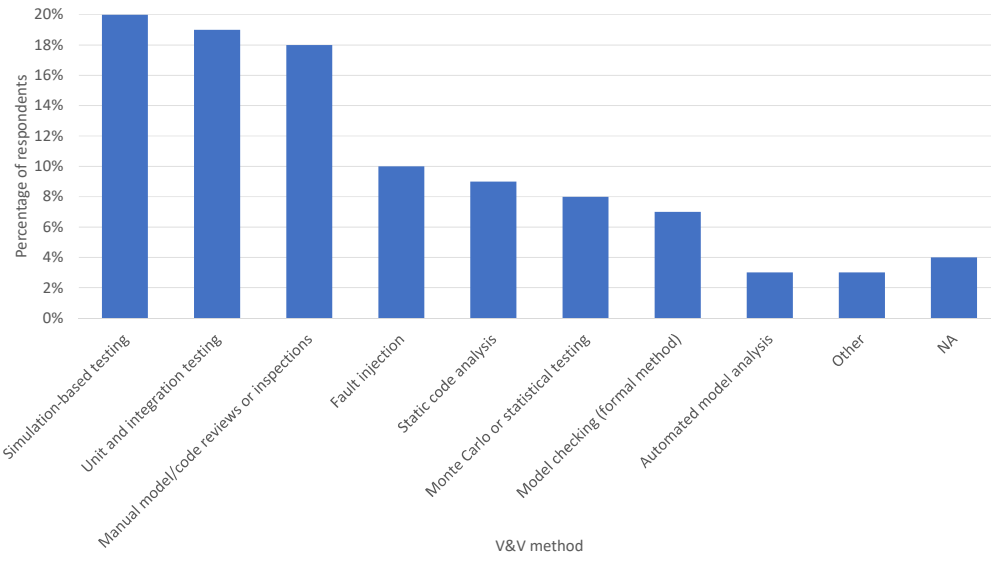


Figure 21: Methods used for verification and validation of AUCs during development (34 respondents, multiple selections possible)

4.5.4 RQ5d: How was runtime behavior of AUCs monitored/assured?

The bar chart in Figure 22 presents the results related to the methods used to monitor and assure the runtime behavior of AUCs. The most frequently used methods included "Monitoring of variable values and ranges" with 29%,

followed by "Monitoring of requirements" with 21%, and "Cross-checking that commanded actions meet intended post-conditions" with 16% of the respondents.

Respondents were also asked about methods used for handling the violations and errors of AUCs' behaviors. As shown in Figure 23, "Notification to human operator" was the most selected with 30%, followed by "Control handled over to human operator", "Automatic autonomous action", "Autonomous response and adaptation of mission", and "Like a system failure" with 20%, 16%, 14%, and 14% of the respondents, respectively. Only 3% of respondents indicated that "Control was taken away from the human operator". Under "Other", respondents listed "Failover to layered recovery mechanisms" and "Probability of correctness".

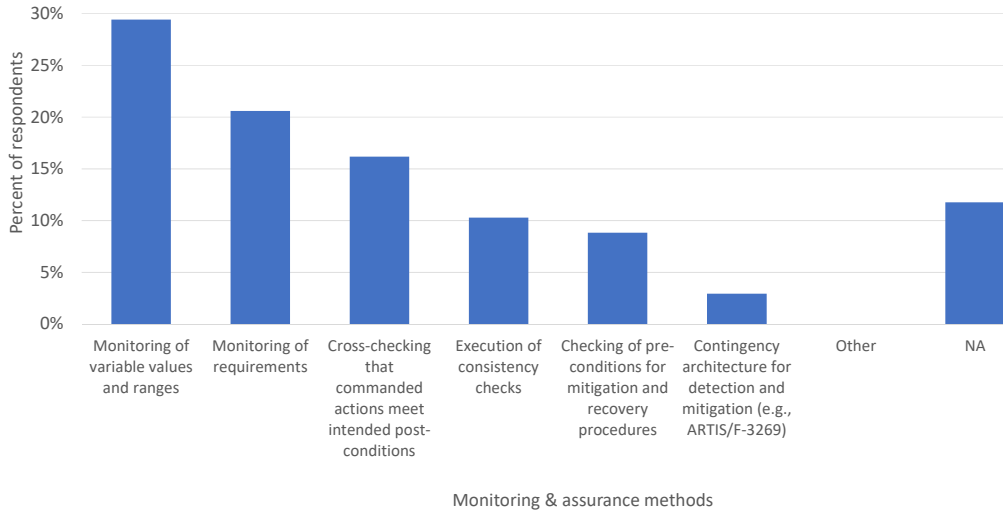


Figure 22: Methods used for monitoring/assurance of AUCs runtime behavior (34 respondents, multiple selections possible)

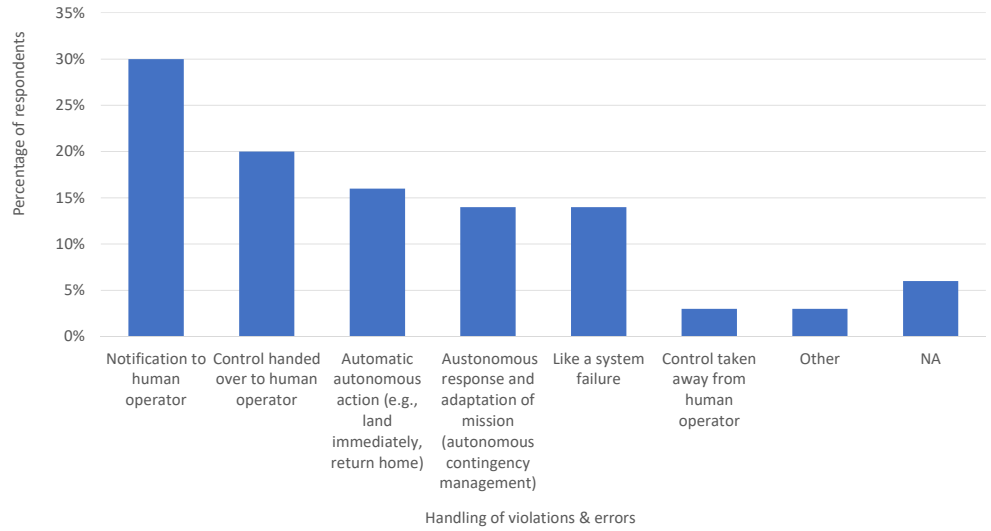


Figure 23: Methods used for handling the violations and errors of AUC (34 respondents, multiple selections possible)

4.5.5 RQ5e: Were the reused software artifacts verified and validated?

The respondents were also asked if they specifically verified and validated the reused artifacts. As can be seen in Table 7, only 19% of respondents verified and validated the reused artifacts while over half (56%) of respondents did not.

Table 7: Verification and validation of reused artifacts (33 respondents)

Did you specifically verify and validate the reused artifacts?	
Yes	19%
No	56%
NA	25%

Table 8: Bugs specific to autonomous functionality, model-based approach, and reuse (32 respondents)

Bugs due to	Yes	No	NA
Autonomous functionality	28%	47%	25%
Model-based approach	9%	44%	47%
Reuse	25%	53%	22%

4.6 Bugs

4.6.1 RQ6a: Where there any bugs specific to autonomous functionality, model-based approach, and/or reuse?

Our survey also explored the existence of bugs specific to autonomous functionality, model-based approach, and reuse of software artifacts. The responses are summarized in Table 8. 28% of respondents reported finding bugs specific to autonomous functionality. As expected, the ratio of specific bugs found vs. no-specific bugs found was high (approximately 1:2) for autonomous functionality. High complexity of the autonomous functionality code and novel/ unusual algorithms might be the likely reasons. Several respondents provided reasons for bugs related to autonomous functionality, which included overfitting/ underfitting and failures of commercial-off-the-shelf components.

Only 9% of respondents reported finding bugs specific to model-based approach. The ratio of specific bugs found vs. no-specific bugs found was relatively low (roughly 1:5), as expected.

Surprisingly, the percentage of reuse specific bugs was high (i.e., 25%), as well as the ratio of specific bugs found vs. no-specific bugs found (approximately 1:2). This means that a substantial number of reuse-specific bugs existed and also seemed to occur, in contrast to the typical assumption that reuse would substantially lower the number of bugs. Some respondents listed specific reasons for bugs attributed to reuse, which included version conflicts, incompatibilities, adaptation of existing software produced unintended consequences, and bugs that were not corrected in the previous release.

5 Threats to Validity

In this section, we describe threats to the validity of this study and the measures taken to mitigate them.

Construct validity addresses whether we are testing what we intended to test. One threat to construct validity is the use of inconsistent and imprecise terminology. To avoid this threat, in the survey and the paper, we provided the definitions of the terms being used. To ensure if we test what we intended to test, we carefully designed the survey questionnaire and used a pilot study to verify and validate it and to subsequently augment and improve it. The survey relied on voluntary participation, which introduces the risk of self-selection bias. We tried to address this threat to validity by reaching to experts from different domains, both from industry and academia, worldwide using different means of communication. Humans are know to have evaluation apprehension (i.e., a tendency to try to look better or fear of being evaluated). For example, participants may deliberately exaggerate or downplay their skills, affecting the validity of the survey results. Furthermore, the fear of negative consequences may result in a reluctance to report their true experiences accurately. Many of these social threats to construct validity were mitigated by keeping the survey anonymous, and allowing the respondents to skip any question they may have wanted to skip.

Internal validity concerns influences that can affect the variables and metrics without researchers' knowledge. The survey required participants to recall and self-report their experiences related to development of autonomous systems. Participants' recollections may be influenced by memory limitations or biases, potentially affecting the accuracy of their responses. Such biases affect individual participants, who individually responded to the survey. In our survey with more than 100 respondents, we belief that such biases cancel themselves out.

Conclusion validity concerns the ability to draw correct conclusions. One of the threats to conclusion validity is the sample size. We distributed the survey widely and collected responses from 110 experts who have used autonomous

systems and/or MBSwE in their projects. The answers related to the autonomous functionality were collected for 58 AUCs. In general, these are fairly large sample sizes that allow drawing valid conclusions. Some survey questions, however, have smaller sample sizes because answering each question was not mandatory. For clarity, the results are annotated with the number of participants who answered that specific question.

External validity concerns the generalizability of results. The results presented in this paper are based on opinions of experts on autonomous systems from different domains and world regions, both from industry and academia which ensures representative population and some generalizability of the results. Nevertheless, we cannot claim that the results based on one survey would be valid for all autonomous systems.

6 Recommendations and Conclusion

This paper presents the first part of a longitudinal study of software systems that have autonomous functionality and may employ MBSwE and reuse. The empirical results are based on data collected using an online survey which was conducted in 2019. The respondents to the survey were from different industry domains, worldwide. As main contributions of this paper, we (C1) assessed the state-of-the-practice of developing autonomous systems at the time of the survey, (C2) identified and quantified the benefits and challenges of autonomy and reuse, (C3) explored the processes and standards used to develop autonomous systems, and (C4) investigated the verification and validation of the autonomy, models, and reuse.

Our survey revealed that most project with autonomous functionality employed high levels of autonomy (see also Figure 7). Based on the findings presented in this paper, it can be concluded that the *safety-critical systems with a substantial degree of autonomy* have been successfully developed in the industry and that implementation of tasks with full autonomy was within the realm of the existing software technology. However, development of autonomous functionality was not without challenges. Among the respondents of our survey, the most challenging was to deal with the *system complexity*, followed by the high level of *environment uncertainty* and the difficulty achieving the *desired level of autonomy*. Surprisingly, at the time the survey was conducted (i.e., 2019), the development of autonomous functionality was still dominated by *traditional algorithms* (i.e., rule-based algorithm, planning systems/languages, and statistical and filtering methods), with only 12% of AUCs using machine learning approaches. More detailed summaries of our findings, grouped by research question, are given in Table 9. We are currently conducting the second part of our longitudinal study whose goals are to explore how the state-of-the-practice has evolved since 2019 and if the challenges and level of machine learning usage in autonomous systems remain the same or may have changed over the last six years.

Based upon our findings, the following recommendations can support and help to further enhance the development of autonomous systems:

- **High complexity of AUCs and environmental uncertainties require careful upfront study and planning**. This has a been major issue according to the study. Because of the often big and unclear expectations of system capabilities and performance, detailed requirements and operational envelopes should be defined early in the process.
- **Model-based Software Engineering can help toward successful development of an AUC**. As 37% of respondents used MBSwE, this is a strong indication of the usefulness of MBSwE. The decision to use MBSwE should be made early in the process and suitable tools set up and made available to the project team.
- **Projects that incorporate autonomy should be encouraged to use modeling and coding standards.** Our survey showed that modeling standards were used by roughly half of the projects whereas coding standards were applied in 74% of the projects. Enforcements of modeling and coding standards can contribute to project's success and may also help with reuse of software artifacts.
- **Certification of AUCs is still a tough issue.** Although many of the projects in our survey contained safety-relevant AUCs, only about a quarter of the AUCs went through certification. This may be due to unavailability of standards that are tailored toward AUC certification. Awareness of new and upcoming certification standards for autonomous systems and AUCs (e.g., IEEE P7009 for failsafe design (14)) and for learning-enabled systems (e.g., EASA Concept Paper (12; 13)). Aligning in-house development and QA processes toward such guidelines can help to streamline future projects that require certification.
- **Reuse of software artifacts for the development of autonomous systems can be helpful, but requires careful planning and considerations.** Over 60% of respondents to our survey reported increased productivity due to reuse. However, only 40% or less experienced positive impacts of reuse on quality and cost. Note that reuse always requires up-front investments to prepare artifacts for reuse.

- **Verification and validation should be performed on reused software artifacts as well** since surprisingly high percentage of respondents (i.e., 25%) experienced reuse specific bugs. Careful preparation of artifacts (e.g., documentation, attached certification, component-specific test cases) for later reuse is therefore important and can be seen as a good investment.
- **Software verification and validation should go beyond performance and correctness; it should also include robustness, safety, and security.** Our survey results showed that performance and correctness were verified most frequently (with 83% and 78%, respectively), followed by robustness and safety (with 61% and 56%, respectively). Verification and validation of security was rare, done by only 4% of the respondents.
- **The collection of empirical knowledge about software bugs specific to autonomy would be very valuable and has the potential to cost effectively increase the quality of AUCs.** Based on the survey responses, software contained bugs specific to autonomy more often than bugs specific to model-based approaches (i.e., 28% versus 9% of responses). For details see e.g., (34).

Our current work is focused on updating the survey to collect more recent data and discover how the development of autonomous systems has evolved over time. We will then compare and contrast those findings with the ones given in Table 9, and revisit the recommendations presented in this paper.

Acknowledgments

This work was funded by the NASA Software Assurance Research Program (SARP) in fiscal years 2019 and 2020. The authors would like to thank the colleagues from the NASA Ames Research Center and other colleagues who provided their feedback on the survey questionnaire. The authors would also like to thank all respondents to the survey for sharing their information, experience, and opinions.

References

[1] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2020.

[2] D. Harel, A. Marron, and J. Sifakis, "Creating a foundation for next-generation autonomous systems," *IEEE Design & Test*, vol. 39, no. 1, pp. 49–56, 2021.

[3] C. Ebert and M. Weyrich, "Validation of autonomous systems," *IEEE Software*, vol. 36, no. 5, pp. 15–23, 2019.

[4] L. Hall, "2015 NASA technology roadmaps," 2015.

[5] R. C. Cardoso, G. Kourtis, L. A. Dennis, C. Dixon, M. Farrell, M. Fisher, and M. Webster, "A review of verification and validation for space autonomous systems," *Current Robotics Reports*, vol. 2, no. 3, pp. 273–283, 2021.

[6] "Tesla recalls 362,758 vehicles, says full self-driving beta software may cause crashes," https://www.cnbc.com/2023/02/16/tesla-recalls-362758-vehicles-says-full-self-driving-beta-software-may-cause-crashes.html, February 2023, accessed: 2025-04-16.

[7] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, vol. 30, no. 3, pp. 286–297, 2000.

[8] D. Karunakaran, J. S. Berrio, S. Worrall, and E. Nebot, "Challenges of testing highly automated vehicles: A literature review," in *2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*. IEEE, 2022, pp. 1–8.

[9] RTCA, "DO-178C/ED-12C: Software considerations in airborne systems and equipment certification," 2012. [Online]. Available: http://www.rtca.org

[10] *DO-331, Model-Based Development and Verification Supplement to DO-178C and DO-278A*, RTCA Std., 2013.

[11] *Road vehicles — Functional safety — Part 6: Product development at the software level*, International Organization for Standardization Std. ISO 26 262-6, 2018.

[12] "EASA concept paper: First usable guidance for level 1 machine learning applications," European Aviation Safety Agency, Tech. Rep., 2021.

[13] "EASA concept paper: First usable guidance for level 1&2 machine learning applications," European Aviation Safety Agency, Tech. Rep., 2023.

Table 9: Summary of main findings for the research questions in Table 1

Findings 1: Where? What? How? and Who?	
RQ1a	Where: Space industry dominated with 40%, followed by aviation with 16%, military with 13% and automotive with 9% of respondents.
RQ1b	What: 45% of respondents indicated catastrophic or hazardous level of safety criticality, followed by 39% with major safety criticality of their systems.
RQ1c	What: C and C++ or a mixture thereof dominated as implementation languages, both during development and deployment, followed by use of Python. MATLAB was also used, but much more frequently during development than during deployment.
RQ1d	How: Both during development and deployment, over one third of the code was developed from scratch (i.e., 36% and 38%, respectively). Next most common were the use of customized existing code libraries (with 23% both during development and deployment) and using existing code libraries (with 20% and 17%, respectively for development and deployment).
RQ1e	How: Special hardware and cloud services were not used frequently. Thus, only 16% during development and 19% during operation used special hardware, and only 15% used cloud services.
RQ1f	Who: The two major group of respondents' roles included "Design and Research", followed by "Model development", "Programming", "Software and system integration", "Testing/QA/V&V", and "Project management".
Findings 2: Details on Autonomy	
RQ2a	38% of AUCs were developed using MBSwE.
RQ2b	For each task (i.e., Information acquisition, Information analysis, Decision and action selection, and Action implementation), a significant percentage of AUCs (i.e., from 50% to 67%) operated with the computer having a full autonomy.
RQ2c	The majority of AUCs used traditional algorithms and methods like rule-based algorithms (35%), planning systems/ languages (22%), and statistical and filtering methods (18%). Surprisingly, Machine learning approaches were only used for 12% of AUCs.
RQ2d	For 53% of AUCs the requirements were specified same way as for non-AUCs. Natural Language was used more than twice as often as formal specifications. The majority of AUCs (i.e., 63%) were developed using requirements with the same level of details as the non-AUCs.
RQ2e	When major and moderate challenges are considered together, "System complexity" was the most challenging (for 67% of the AUCs), followed by "High level of environment uncertainty" and "Achieving the desired level of autonomy" (for 57% and 40% of the AUCs, respectively).
Findings 3: Details on Reuse of Software Artifacts	
RQ3a	For both AUCs and non-AUCs software code was the most reused artifact, with over 60% of the responses indicating 30% or more of the code being reused.
RQ3b	When major and moderate negative aspects of reuse were considered together,"Hindering new ideas" and "Added complexity because of reuse" were most significant with 25%, followed by "Additional cost due to reuse sustainability" with 21%.
RQ3c	When major and moderate difficulties due to reuse were considered together, "Uncertain operational conditions / environment" led to most difficulties (40%), followed by "Verifying and validating reused software" (34%), "Lack of planning for reuse in advance" (32%) and "Integrating the reused parts into the development environment" (31%).
RQ3d	63% of respondents reported increase productivity, 40% reported increased quality, and only 37% reported decreased cost due to reuse.
Findings 4: Processes and Standards	
RQ4a	Twice as many respondents used Agile than Waterfall life-cycle process (i.e., 38% vs. 19%).
RQ4b	Almost half of the respondents (i.e., 48%) did not use any modeling standard and about a quarter of the respondents (i.e., 26%) did not follow any coding standard.
RQ4c	Only 24% of systems went through certification and in only 26% of cases AUCs were part of a certified system.

Findings 5: Verification & Validation	
RQ5a	Most of the respondents verified and validated the performance (83%) and the correctness (78%) of the software. Robustness and safety were verified and validated in 61% and 56% of the cases, respectively. Only 11% verified and validated security.
RQ5b	Testing was used most frequently for verification and validation of the models (i.e., by 30% of respondents), followed by simulation with 24%, manual model inspection/reviews with 20%, and automated model analysis with 10%. Only 6% of respondents did verify and validate the models.
RQ5c	During development, AUCs were most frequently verified and validated using simulation-based testing (20%), unit and integration testing (19%), and manual model/code reviews or inspections (18%). Automated model analysis was used only by 3% of the respondents.
RQ5d	Most frequently used methods for monitoring / assuring AUCs' runtime behavior included "Monitoring of variable values and ranges" (29%), "Monitoring of requirements" (21%) and "Cross-checking that commanded actions meet intended post-conditions" (16%).
RQ5e	Only 19% of respondents verified and validated the reused artifacts.
Findings 6: Bugs	
RQ6a	28% of respondents reported finding bugs specific to autonomous functionality. Only 9% of respondents reported finding bugs specific to model-based approach. Surprisingly, the percentage of reuse specific bugs was high (i.e., 25%).

[14] *IEEE Standard for Fail-Safe Design of Autonomous and Semi-Autonomous Systems*, IEEE Standard 7009, 2024, published by the IEEE.

[15] H. Chen, X. Wang, and Y. Li, "A survey of autonomous control for UAV," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 2, Nov 2009, pp. 267–271.

[16] Z. Tahir and R. Alexander, "Coverage based testing for V&V and safety assurance of self-driving autonomous vehicles: A systematic literature review," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2020, pp. 23–30.

[17] C. Gao, G. Wang, W. Shi, Z. Wang, and Y. Chen, "Autonomous driving security: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7572–7595, 2021.

[18] J. Schumann, P. Gupta, and Y. Liu, "Application of neural networks in high assurance systems: A survey," in *Applications of Neural Networks in High Assurance Systems*. Springer, 2010.

[19] J. Schumann, E. Reed, and O. Mengshoel, "Verification and validation of system health management models using parametric testing," in *AIAA Infotech@Aerospace*, 03 2011.

[20] J. Schumann, T. Mbaya, O. Mengshoel, K. Pipatsrisawat, A. Srivastava, A. Choi, and A. Darwiche, "Software health management with Bayesian networks," *Innovations in Systems and Software Engineering*, vol. 9, no. 4, pp. 271–292, 2013.

[21] A. Nikora, P. Srivastava, L. Fesq, S. Chung, and K. Kolcio, "Assurance of model-based fault diagnosis," in *2018 IEEE Aerospace Conference*, March 2018, pp. 1–14.

[22] Q. Song, E. Engström, and P. Runeson, "Concepts in testing of autonomous systems: Academic literature and industry practice," in *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 2021, pp. 74–81.

[23] F. Jahan, W. Sun, Q. Niyaz, and M. Alam, "Security modeling of autonomous systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–34, 2019.

[24] K. Goseva-Popstojanova, T. Kahsai, M. Knudson, T. Kyanko, N. Nkwocha, and J. Schumann, "Survey on Model-Based Software Engineering and Auto-Generated Code," Tech. Rep., 2016.

[25] D. LaVallee, J. Jacobsohn, C. Olsen, and J. Reilly, "Intelligent control for spacecraft autonomy - an industry survey," in *Space 2006*, 2006, p. 7384.

[26] T. B. Sheridan, *Telerobotics, Automation, and Human Supervisory Control*. Cambridge, MA, USA: MIT Press, 1992.

[27] SAE On-Road Automated Vehicle Standards Committee and others, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *SAE Standard J*, vol. 3016, no. 1, pp. 1–16, 2014.

[28] R. W. Proud, J. J. Hart, and R. B. Mrozinski, "Methods for determining the level of autonomy to design into a human spaceflight vehicle: A function specific approach," in *Proc. Performance Metrics for Intelligent Systems (PerMIS '03), NIST Special Publication 1014*, 2003.

[29] "Mercedes says level 4 autonomous driving will be a reality this decade," https://cars.usnews.com/cars-trucks/features/mercedes-level-4-autonomy-is-a-reality, March 2023, accessed: 2025-04-16.

[30] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and A. Qi, "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 385–396.

[31] S. Tang, Z. Zhang, J. Tang, L. Ma, and Y. Xue, "Issue categorization and analysis of an open-source driving assistant system," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2021, pp. 148–153.

[32] M. Taylor, J. Boubin, H. Chen, C. Stewart, and F. Qin, "A study on software bugs in unmanned aircraft systems," in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2021, pp. 1439–1448.

[33] D. Wang, S. Li, G. Xiao, Y. Liu, and Y. Sui, "An exploratory study of autopilot software bugs in unmanned aerial vehicles," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 20–31.

[34] K. Goseva-Popstojanova, D. Hood, J. Schumann, and N. Nkwocha, "The anatomy of software changes and bugs in autonomous operating system," in *47th IEEE Annual Computers, Software, and Applications Conference (COMPSAC)*, 2023, pp. 28–38.

[35] S. Burton, I. Habli, T. Lawton, J. McDermid, P. Morgan, and Z. Porter, "Mind the gaps: Assuring the safety of autonomous systems from an engineering, ethical, and legal perspective," *Artificial Intelligence*, vol. 279, p. 103201, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370219301109

[36] C. Menon and R. Alexander, "Ethics and the safety of autonomous systems," 02 2018.

[37] N. Webb, D. Smith, C. Ludwick, T. Victor, Q. Hommes, F. Favaro, G. Ivanov, and T. Daniel, "Waymo's safety methodologies and safety readiness determinations," 2020. [Online]. Available: https://arxiv.org/abs/2011.00054

[38] B. A. Kitchenham and S. L. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. Sjøberg, Eds. Springer London, 2008, pp. 63–92.

[39] "Surveymonkey," https://www.surveymonkey.com/, accessed: 2025-04-16.

[40] Eclipse.org, "Papyrus modeling environment." [Online]. Available: https://eclipse.org/papyrus/

[41] J. R. Boyd, "The essence of winning and losing," *Unpublished lecture notes*, vol. 12, no. 23, pp. 123–125, 1996.

[42] Mathworks, "The mathworks automotive advisory board." [Online]. Available: http://www.mathworks.com/solutions/automotive/standards/maab.html

[43] SysML.org, "SysML open source specification project." [Online]. Available: http://sysml.org

[44] "Uppaal." [Online]. Available: http://www.uppaal.org

[45] jenkins.io, "Jenkins." [Online]. Available: https://jenkins.io/

[46] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-time Systems," in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.