U.S. Department
of Transportation

**National Highway
Traffic Safety
Administration**

NHTSA

DOT HS 813 226

June 2022

Foundations of Automotive Software

DISCLAIMER

This publication is distributed by the U.S. Department of Transportation, National Highway Traffic Safety Administration, in the interest of information exchange. The opinions, findings, and conclusions expressed in this publication are those of the authors and not necessarily those of the Department of Transportation or the National Highway Traffic Safety Administration. The United States Government assumes no liability for its contents or use thereof. If trade or manufacturers' names or products are mentioned, it is because they are considered essential to the object of the publication and should not be construed as an endorsement. The United States Government does not endorse products or manufacturers.

NOTE: This report is published in the interest of advancing motor vehicle safety research. While the report may provide results from research or tests using specifically identified motor vehicle models, it is not intended to make conclusions about the safety performance or safety compliance of those motor vehicles, and no such conclusions should be drawn.

Suggested APA Format Citation:

Arthur, D., Becker, C., Epstein, A., Uhl, B., & Ranville, S. (2022, June). *Foundations of automotive software* (Report No. DOT HS 813 226). National Highway Traffic Safety Administration.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT HS 813 226		

4. Title and Subtitle	5. Report Date
Foundations of Automotive Software	June 2022
	6. Performing Organization Code

7. Authors	8. Performing Organization Report No.
David Arthur, Christopher Becker, Alex Epstein, Bill Uhl, and Scott Ranville	DOT-VNTSC-NHTSA-20-03

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
John A Volpe National Transportation Systems Center 55 Broadway Cambridge, MA 02142-1093	11. Contract or Grant No. 51HS2EA100/DTNH2217V00020

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Highway Traffic Safety Administration 1200 New Jersey Avenue SE Washington, DC 20590	Final Report
	14. Sponsoring Agency Code

15. Supplementary Notes
Bill Uhl and Scott Ranville of OrangeWire Systems LLC contributed to this report. Paul Rau was contracting officer representative for this project.

16. Abstract
This report chronicles the history as well as the current state-of-the-art practices of software development within the automotive sector. Key concepts, approaches, trends, and knowledge of automotive software development were collected to benchmark industry practices as well as to compare these to non-automotive industry sectors. The report provides a conceptual framework and taxonomy to articulate the relationships among relevant factors driving the evolution of automotive software development, which can be updated to incorporate new classifications and/or sub-classifications as the automotive software development landscape evolves. The automotive software industry's current practices are documented to enable high-level, future comparisons to similar domains—for example, to the aerospace industry where the primary overlap is in complex safety-critical systems and safety-critical software; and to the consumer electronics industry where the primary overlap is a product that is marketed, purchased, operated, and maintained by private, independent consumers. Actual practices across the automotive industry are diverse and the product of an evolutionary amalgamation of thousands of published consensus standards, internally developed processes, tools, practices, nomenclatures, architectures, and taxonomies. Vehicle software architectures and features have evolved over time, and this report explores the related factors in this evolution.

17. Key Words	18. Distribution Statement
ISO 26262, automotive open system architecture, AUTOSAR, automotive grade, Linux, model-based development, automatic code generation, automotive ECU, software complexity, ASPICE, V-Cycle, communications bus, software security, software safety, software dependability, critical systems, safety systems, software system requirements, software scheduling (OS technology), hardware (hardware technology), computer control (control technology and control systems), communication (communications technology, network architecture and topology), and software implementation (tools and programming languages and models of computation).	Document is available to the public from the DOT, BTS, National Transportation Library, Repository & Open Science Access Portal, rosap.ntl.bts.gov.

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	142	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Table of Contents

List of Abbreviations

Abbreviation	Term
ABS	antilock braking system
ACG	automatic code generation
ADL	architecture description language
ADAS	advanced driver assistance system
ADS	Automated Driving Systems
ADC	analog to digital converter
AEC	Automotive Electronics Council
AGL	automotive grade Linux
ALM	application lifecycle management
API	application programming interface
ASIL	Automotive Safety Integrity Level
ASPICE	automotive software process improvement and capability determination
AUTOSAR	automotive open system architecture
BEV	battery electric vehicle
CAFE	Corporate Average Fuel Economy
CAN	controller area network
COTS	commercial off-the-shelf
CV	connected vehicle
DAC	digital to analog converter
DAL	design assurance levels
DFP	data flow processor
DMCA	Digital Millennium Copyright Act
ECU	electronic control unit
E/E	electrical and electronic
EMS	engine management systems
FCEV	fuel cell electric vehicle
FMVSS	Federal Motor Vehicle Safety Standards
HIL	hardware-in-the-loop
HMI	human-machine interface
HPC	High-performance computing

IC	internal combustion
IO	input/output
ISO	International Organization for Standardization
IoT	internet of things
lidar	light detection and ranging
LIN	local interconnected network
LoC	lines of code
MaaS	mobility as a service
MBD	model-based development
MDSD	model-driven systems development
MIDI	multiple instruction multiple data
MISRA	Motor Industry Software Reliability Association
MIL	model-in-the-loop
MoC	model of computation
NCAP	New Car Assessment Program
NoC	network operations center
OBD	onboard diagnostics
OS	operating system
PWM	pulse width modulation
QM	quality management
RAM	random access memory
RNN	recurrent neural network
ROM	read-only memory
RTE	runtime environment
RTOS	real-time operating system
RCP	rapid control prototyping
SAE	SAE International (formerly Society of Automotive Engineers)
SIL	software–in-the-loop
SOA	service-oriented architecture
SoC	system on a chip
SOTIF	safety of the intended functionality
SDLC	software development lifecycle
SBD	simulation based development

SPEM	software process engineering meta-model
SPICE	software process improvement and capability determination
SR	synchronous reactive
STPA	systems theoretic process analysis
TSN	time sensitive network
UUT	unit under test
V&V	validation and verification
V2I	vehicle-to-infrastructure
V2V	vehicle-to-vehicle
V2X	vehicle to “x” where x is everything connected (e.g., vehicles, infrastructure, pedestrians, bicyclists)

1 Brief History of Automotive Software

The modern vehicle has been undergoing a transformation from the mechanical platform of the 20th century into a highly integrated system of "mechatronic"[1] subsystems. These subsystems are progressively defined by software-based (virtual[2]) controls that can emulate behavior of a physical interface or external system controller.[3] Vehicle manufacturers and suppliers are increasingly self-identifying as software companies as the industry has been shifting its focus from mechanical design and manufacturing to computer hardware and software development.[4]

In terms of software application, the motor vehicle is unique and complex. As an integrated system, it incorporates:

- Real-time control systems,
- Complex networks of communications busses and sensors,
- Application-specific embedded hardware and software,
- Safety systems,
- Maps and navigation software,
- Entertainment and communications systems,
- Connectivity with other consumer devices, and
- Emerging safety and driving automation systems that may leverage artificial intelligence (AI) techniques and machine learning technology (see Sections 1.4 and 3.3.3.1.3 for more information on AI).

The pace of innovation in vehicle design and electronic control of critical systems has accelerated in recent years. As software enhancements are added, such as driver automation, driver assistance, and security systems, automotive product designers must constantly assess their effects on the performance, safety, and reliability of the vehicle that features several other software-based technologies. Several of the sources cited within this document make arguments that automotive software and mechatronics systems are among the most complex systems engineered today that are challenging to analyze, verify and validate.[5] Due to the complex attributes of automotive software mentioned previously, vehicle production practices, software development lifecycles, systems engineering practices, and maintenance and sustainment practices are complex and specialized.

This report provides a foundation covering the basics of automotive software, including the historical evolution of vehicular software development including the build, test, and validation

---

[1] A mechatronic subsystem is the integration of a mechanical, electrical, computer, robotic, controls, and telecommunication subsystem. www.sciencedirect.com/topics/engineering/mechatronics

[2] Analog Devices, Inc. (2012, December 14). Virtual control interface [Web page]. https://wiki.analog.com/resources/tools-software/sigmastudio/usingsigmastudio/virtualcontrolinterface

[3] A controller is a hardware device or software program that manages or directs the flow of data between two entities. https://whatis.techtarget.com/definition/controller

[4] Ziegler, C., & Patel, N. (2016, April 7). *Meet the new Ford, a Silicon Valley software company*. The Verge. www.theverge.com/2016/4/7/11333288/ford-ceo-mark-fields-interview-electric-self-driving-car-software

[5] Molotnikov, Z., Schorp, K., Aravantinos, V., & Schaetz, B. (2016). *Future programming paradigms in the automotive industry*. Forschungsvereinigung Automobiltechnik e.V.

activities associated with the lifecycle, development, production, and maintenance of automotive control system software.

Automotive software complexity has surged during the past 15 years (Table 1). Early implementations of electronic controls were driven by the need to meet emissions regulations such as the Clean Air Act, and Corporate Average Fuel Economy. As computer processors became less expensive and more commonplace, manufacturers increasingly started to use computer systems to deliver competitive products to the market, resulting in advances in performance, value, comfort, entertainment, maintenance, diagnostics, and safety.

The drive for innovation has led to the replacement of mechanical control components with digital, electrically-actuated, computer-controlled systems, led by:

- Low-cost, miniaturized, and specialized electronic control unit and related technology for highly specialized, purpose-built, embedded computing;
- Advancements in actuation and sensing technology; and
- Advancements in communications networks.

Many factors have influenced (and will continue to influence) the use of software in automotive systems that include decreasing costs and increasing capabilities of electronic controllers, sensors, and actuators, market acceptance of innovative control applications and communication systems, published industry consensus standards, and regulations. As systems became more complex, new tools and processes have emerged to help manage the development of software.

Table 1. A timeline of representative E/E advancements and system complexity, correlated with emissions and safety regulations

Year	1950s and 1960s	1970s and 1980s	1990s	2000	2005	2010	2015
Lines of Code// vehicle[6] [7]	0	~100,000	~1,000,000		~15,000,000		~100,000,000
ECUs[8]**per vehicle**	0	1	~5		~15	~40	~100
Notable E/E Advancement Introductions[9] [10]	All Transistor Car Radio Alternator Electronic Ignition Antilock Braking Invented [11]	Engine Control Airbag Lambda (Air -Fuel) Sensor Antilock Braking Production Transmission Control	Cluster Body Electronics Wired Controller Area Network (CAN) Buses	Electronic Stability Control Advanced Restraints GPS Navigation	Adaptive Cruise Control Infotainment Electric Power Assisted Steering Telematics Displays Zero Emission Vehicles	Vehicle to Infrastructure Remote Diagnostics Adaptive Headlamps Steer by Wire Brake by Wire Fuel Cell Battery Management System NCAP (side impact and rollover sensors, backup cameras, collision avoidance [ESC, LDW, FCW])	Collision Avoidance Systems Crosswind Stabilization Automotive Night Vision Automatic Parking Wireless Buses

[6] Rough order of magnitude

[7] Antinyan, V. (2018). *Revealing the complexity of automotive software*. Volvo Car Corporation.DOI: 10.13140/RG.2.2.34697.29286.

[8] Ibid.

[9] E/E refers to Electrical and Electronics. The E/E Advancements section of the table provides a sampling of key electrical and electronics advancements for a representation of the evolution of automotive software electronics A complete list of electronics advancements would include hundreds of E/E features and capabilities.

[10] Davey, C. (2013, January 26-27). *Automotive software systems complexity: Challenges and opportunities*. INCOSE International MBSE Workshop, Jacksonville, FL.

[11] Invented in 1952 and ready for production in 1978: https://media.daimler.com/marsMediaSite/en/instance/ko/Mercedes-Benz-and-the-invention-of-the-anti-lock-braking-system-ABS-ready-for-production-in-1978.xhtml?oid=9913502

As described throughout this document, vehicle computing architectures are implemented using specialized embedded systems consisting of optimized processing for the intended functionality (to save cost), input/output, peripherals, real-time operating systems, and communication networks. They are designed to solve closed-loop controls with precision timing (see Section 2.4), and are defined by their associated model of computation (see Models of Computation, Section 2.4.4).

In the 1980s and 1990s, prior to the widespread availability of tools built around specific MoC, software engineers typically used general purpose programming tools for vehicle ECU programming (see Section 1.1). However, the evolution of MoC-specific developmental tools, such as model-based development (MBD), has had a transformational impact on the automotive industry. These improved developmental tools were instrumental in lowering technical and resource barriers to developing automotive control software.

The evolution of coding software is defined by three distinct phases, beginning with hand coding, then MBD, and culminating in the current practice of combining MBD with automatic code generation.

1.1 Hand-Coded Software

In the early years of computer programming, specialized tools designed to aid in development of software were unavailable. The first large mechatronics software projects (e.g., the software on the space shuttle) were "hand-coded" and static.[12] Unlike today's systems which can be re-flashed with new software by either physically plugging into a controller or receiving the update over-the-air, in static systems, it was impossible to alter the system once it was on the vehicle without replacing hardware.

Hand coding refers to a variety of editing practices used in software development where source code for a computer program was entered as lines of text using a (text) editor. When hand coding, a computer scientist, software engineer, or someone with a similar skillset, manually converted algorithms or mathematical models into textual computer language (such as C, C++, or Ada) for interpretation by the compiler.[13]

There were obvious inefficiencies that must be overcome when mathematical models and scientific notation must be translated into textual programming languages and syntax in order to be executed and solved on a computer. Hand coding tended to require more highly skilled programmers that stretched development budgets, was more prone to errors from easily missed syntax or coding mistakes, generated code that was more difficult to revise and maintain over time and was less conducive to collaborative environments with several programmers working in parallel.

As software engineering practices have evolved, engineers recognize that productivity can be greatly improved through the advancement of specialized tools built around domain specific language, notation, and MoC. One particular area where this occurred was in the field of mechatronics and control systems.

---

[12] Tomayko, J. E. (1988, March). *Computers in spaceflight: The NASA experience.* National Aeronautics and Space Administration. https://history.nasa.gov/computers/Ch4-5.html

[13] A compiler is a program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.www.lexico.com/en/definition/compiler

1.2 Model-Based Development

Model-based, graphical programming allows engineers to create programs from mathematical models using familiar notation, syntax, and formatting. Closed loop, feedback control systems are traditionally drawn in block diagrams by controls engineers (see Figure 3 in Section 2.4.8). Feedback control diagramming techniques lend themselves naturally to MBD methods.

In the 1980s the introduction of graphical languages such as MatrixX and Simulink allowed engineers to create computer programs by drawing control systems and transfer functions in block- and signal-based diagrams without converting them into text-based syntax, which is required when hand coding.[14]

The models are executable, meaning that as graphical elements are built into the model editor, the underlying software is configured as a program that may be run. The first graphical programming editors allowed models of closed-loop control systems to *execute*, mathematically, within the model editor.

Graphical programming methods introduced efficiencies in several areas over hand coding:

- Models reduce the need for cross domain specialization in computer programming languages. With MBD, software can be developed by engineers who are not specialized in computer science; the controls engineer does not need to have highly refined knowledge of programming syntax.
- Models are less prone to error than hand coding in textual programs because they are more easily inspected for correctness.
- Models are easily parameterized so that transfer function coefficients, values for physical properties, and other coefficients are stored in tables. Parameterization promotes reuse and refactoring[15] of software.

Models are intuitively understood by domain specialists and may be used for purposes other than software development, including documentation and analysis.

There is evidence that MBD yields 50 percent efficiency gains in coding software alone, and the benefits are leveraged across the entire SDLC.[16] MBD practices allow system architects to create libraries of parameterized, graphical software components (e.g., control strategies, actuators, and sensors) that may be quickly refactored and reused, yielding productivity gains each time the software is reused. MBD also allows ECU developers to separate and isolate software development practices from hardware development, thereby compressing and streamlining the respective development lifecycles while facilitating portability of software components across vehicle architectures.[17]

---

[14] Mohler, C. (n.d.). *A brief history of MATLAB*. MathWorks.www.mathworks.com/company/newsletters/articles/a-brief-history-of-matlab.html

[15] Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. www.sciencedirect.com/topics/computer-science/refactoring

[16] Broy, M. (2012). *What is the benefit of a model-based design of embedded software systems in the car industry?* IGI-Global. doi: 10.4018/978-1-61350-438-3.ch013.

[17] Ibid.

1.3 Automatic Code Generation

The introduction of automatic code generation in the 1990s took model-based development a step further. With ACG it became possible to use graphical system models (as explained in Section 1.2) to generate source code in order to run on hardware controllers outside of the graphical model editor (e.g., Simulink). ACG existed before 2000 but was not as memory efficient as hand coding and was limited to specialty applications.[18] Beginning in 1999 ACG from Simulink models was efficient enough to be a viable replacement of hand-written code. At that time, ECUs were also changing to support floating point operations[19] and ROM and RAM costs were coming down, which also contributed to a change in how automotive software was developed.

When combined with automatic code generation, model based software may be executed on a variety of embedded computing devices found in mechatronics applications (e.g., ECUs).[20]

ACG allows domain specialists, such as control system engineers, mechanical engineers, or system engineers, to create software from graphical representations of closed-loop feedback controls, state machines, sequence diagrams, and other abstractions of engineered systems. ACG tools automatically generate (textual) source code so that a software engineer can integrate the control code with low-level code on an ECU, including the RTOS, device drivers and IO, communications busses, and other hardware and software on the embedded system.[21] Prior to the invention of ACG, models developed in MBD editors were converted to code "by hand" for use on an ECU.

While MBD allowed control algorithm designers to test their control theory, ACG allowed MBD to overcome a significant hurdle that prevented graphical models from being used in production without first being translated into the programming language of the environment. Models created in MBD editors usually execute within the model editor on a personal computer. The operating environment is much different on an ECU however, where tasks are normally executed in discrete time steps managed by an RTOS, and so code from an MBD environment still needed some manual (hand coding) to operate on the ECUs.

The first implementations of ACG did not generate optimized code for use within the constrained resources on production (usually with less-expensive ECU hardware), but they did allow models to run on "unconstrained" rapid control prototyping hardware in order to test control strategies onboard developmental vehicles.[22] While RCP hardware was typically cost-prohibitive for a production vehicle, its benefit was especially realized in development and prototyping phases.

As ACG tools have evolved, code performance has improved and optimized so that today's ACG tools are capable of generating highly optimized code for production deployment on nearly any automotive ECU. Today, MBD/ACG processes provide seamless, automated, development

---

[18] ACG required more ROM/RAM than hand-coding, but was quicker to implement, resulting in a trade-off.

[19] Floating point numbers have decimal points. A floating point operation is the ability of a software system to perform mathematical calculations using floating point numbers.

[20] Liao, H. (2010, May 31-June 2). *A study of automatic code generation*. International Conference on Computational and Information Sciences, Amsterdam.

[21] Ibid.

[22] RCP systems are "unconstrained," high performance, MoC specific, prototype ECUs with high performance processors and large amounts of onboard memory and other resources, designed to run control code that has not been optimized for resource constrained production hardware.

capabilities from prototype to RCP to production. These capabilities have also led researchers to be able to train increasingly complex Automated Driving Systems that use machine learning with real-world observations. Sometimes, instead of a calibration engineer or domain specialist manually tuning the inputs and watching for a desired output as has been traditionally practiced, newer methods involve only providing output goals to control systems. Using a machine learning approach with training data, the software gathers its own response data while varying tuning parameters and essentially performs its own tuning and final calibration.

1.4 Emerging Artificial Intelligence Applications

As indicated in the bulleted list of software applications in the prior section, artificial intelligence (AI) techniques are among many applications increasingly being used within automotive software. It is beyond the scope of this report to comprehensively cover AI theory or detailed implementations, (see Section 3.3.3.1.3 for more information on AI beyond this section). To date AI is generally not considered foundational or a primary contributor to the evolution of automotive software and coding.

2 Key Themes in Automotive Software

Given the purpose of this study is to understand factors that drive lifecycle practices for control systems, it is important to understand how wide-ranging technical, industry, business, and safety requirements influence the processes and practices of the SDLC.

2.1 Critical Systems and Safety Systems

This section describes two related characteristics of automotive software—safety and criticality. Reliability is an important characteristic of *critical systems*. Reliability may describe different aspects of the system including dependability, safety, security, location awareness, or trustworthiness. As mechanical controls are replaced with mechatronic systems (see Table 1), and as new features are continuously added, there are an increasing number of critical software systems onboard vehicles.

Vehicle systems incorporate a range of critical systems properties and often incorporate several modes of criticality. The most notable and prevalent of these properties are listed below.

2.1.1 Dependability

A system is described as dependable when the system can be justifiably relied upon for correct and continuous service. Motor vehicle owners expect a high degree of dependability from mechatronic systems. It is important that a powertrain ECU provides expected levels of performance and economy when managing and controlling timing, ignition, and other functions. It is also important that the ECU meets performance expectations every time the vehicle is in operation.

Dependability often determines the success or failure of a vehicle in the marketplace. Consumers are influenced by numerous survey-based, test-based, and/or ratings-based publications, such as New Car Assessment Program (NCAP) ratings, *Consumer Reports*, and the *JD Powers Dependability Ratings,* which ranks vehicles based on the type and number of problems that owners have experienced with their 3-year-old vehicle in the preceding 12 months.

Incorporation of increased electrical and electronic content onboard motor vehicles has led to specific E/E requirements for reliability and robustness, as well as for related mechanical componentry. In the 1980s the Automotive Electronics Council was formed for the purpose of establishing common part-qualification and quality-system consensus standards. The AEC Component Technical Committee establishes consensus standards for reliable, high-quality electronic components. This committee has developed consensus standards, such as AEC-Q100, to ensure the advancement of reliable and robust connectors, mechanical components, enhanced plastics, semiconductors and complex integrated circuits (ICs) rated for high temperatures and harsh environments, etc.[23] [24] [25]

[23] Ardebili, H., & Pecht, M. G. (2000). *Encapsulation technologies for electronic applications*. 1st edition. Elsevier.
[24] Zarr, R.. (2018, April 11). *The future of high-reliability electronics*. Electronic Design. www.electronicdesign.com/technologies/analog/article/21806380/the-future-of-highreliability-electronics
[25] Automotive Electronics Council. (n.d.) [Untitled web page]. www.aecouncil.com/

For automotive software, dependability helps ensure that the driver interacts with a vehicle running software that operates correctly, robustly, and securely. According to Meyer,[26] correctness refers to the vehicle software's ability to perform according to its specification for uses within that specification. Robustness is the vehicle software's ability to prevent damage in cases of erroneous use outside of its specification. Finally, security is the vehicle software's ability to prevent damage in cases of hostile use[27] outside of its specification.

2.1.2 Safety

The word "safety" has specific meanings in the context of different consensus standards and regulations.

Within automotive E/E architectures, the following definitions are used in order to categorize safety systems and safety management activities.[28] [29] [30]

- **Passive safety** mechanisms minimize the severity of a crash and remain "passive" until needed. Examples of passive safety E/E systems include occupant and pedestrian protection systems (e.g., inflatable seat belts, air bags).
- **Active safety** mechanisms are designed to avoid crashes or to minimize crash severity and are intended to always be "active." Examples include predictive automatic emergency braking, antilock braking systems, traction control systems, and tire pressure monitoring systems.

Increasing levels of safety-critical and safety-engineered E/E content onboard vehicles has led to a host of safety consensus standards related to automotive hardware and software and E/E systems including the following.[31]

- ISO 26262
- AUTomotive Open System ARchitecture (AUTOSAR)[32] Guidelines for the use of the C++14 language in critical and safety-related systems.

---

[26] Meyer, B. (2006). "Dependable Software." In J. Kohlas, B. Meyer, & Schiper, A., eds., *Dependable Systems: Software, Computing, Networks*. Lecture Notes in Computer Science, Springer-Verlag. http://se.ethz.ch/~meyer/publications/lncs/dependability.pdf

[27] Hostile use implies an intentional use outside of the item's specified operating parameters, often with active circumvention of preventative mechanisms (as opposed to erroneous use, which is unintentional and does not involve active circumvention).

[28] Smith, D. J., & Simpson, K. G. L. *The Safety Critical Systems Handbook: A Straightforward Guide to Functional Safety: IEC 61508,* 4th edition. Butterworth-Heinemann.

[29] See the AUTOSAR document *Explanation of Application Interfaces of Occupant and Pedestrian Safety Systems Domain* for more information on the relevance of active and passive safety mechanisms related to E/E architectures.

[30] Kumar, A. (2017, July 18). *Active and passive automotive safety systems*. Electronic Specifier. https://automotive.electronicspecifier.com/safety/active-and-passive-automotive-safety-systems

[31] Van Eikema Hommes, Q. D. (2016, June). *Assessment of safety standards for automotive electronic control systems* (Report No. DOT HS 812 285). National Highway Traffic Safety Administration. www.nhtsa.gov/sites/nhtsa.gov/files/812285_electronicsreliabilityreport.pdf

[32] AUTOSAR provides a set of architectural consensus standards designed to allow interoperability between system hardware and software by describing and defining basic software modules, application interfaces, and a common development methodology based on standardized exchange formats.

- Motor Industry Software Reliability Association (MISRA) C (and C++) Guidelines for the Use of the C Language in Critical Systems.

One key safety concept for automotive E/E systems is functional safety. In the automotive industry, functional safety is typically addressed through the ISO 26262 standard (November 2011, revised December 2018).[33] ISO 26262 defines functional safety as ensuring "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems." As of 2012 all European OEMs and many of the rest of the world's OEMs were reportedly using ISO 26262 to some degree; full integration of ISO 26262 into internal processes is still on-going. One industry expert interviewed as part of this study indicated that their organization is estimated to be only 40 percent compliant with ISO 26262 within a few of the product lines as of 2019 U.S. OEMs initially resisted the standard but increasingly followed suit due to concerns about legal liability. The standard was developed by a working group that included most of the largest global OEMs.[34]

Safety engineered systems are concerned with avoidance of crashes or mishaps, or protecting occupants in the event of a crash, and are often defined in terms of external consequences.[35] As systems such as automatic emergency braking, active suspension, and electronic stability control increasingly implement active interventions to assist drivers to better control the behavior of the car, a wide range of interconnected sensors, actuators, and ECUs may affect the performance of the vehicle. In addition, there is the potential to cause harm should these systems fail to perform properly, including when they interact simultaneously with each other.[36]

Safety engineering involves implementing process steps throughout the development lifecycle in order to ensure that:

- hazards and risks are correctly identified,
- automotive safety integrity levels (ASILs) 37 or similar risk evaluation metrics, and safety goals are correctly specified,
- safety requirements flow down into the appropriate design processes (e.g., hardware, software), and

[33] *ISO 26262-2:2018 - Road vehicles -- Functional safety* is a derivative of IEC 61508, the generic functional safety standard for E/E systems.

[34] Czerny, B. J., D'Ambrosio, J., Debouk, R., & Stashko, K. (2010, August 30-September 3). *ISO 26262: Functional safety draft international standard for road vehicles: Background, status, and overview* [PowerPoint presentation]. 28th International System Safety Conference 2010, Minneapolis, MN. http://sesamo-project.eu/sites/default/files/downloads/publications/iso-26262-dis-tutorial-2010-final.pdf

[35] Rushby, J. (1994). Critical system properties: survey and taxonomy. *Reliability Engineering and System Safety, Vol. 43*, No. 2, pp. 189–219. www.csl.sri.com/users/rushby/papers/csl-93-1.pdf

[36] Ebert, C., Burton, S., Amsler, K., & Lederer, D. (2011). *Introducing automotive E/E safety engineering: Challenges and solutions.* Vector. https://assets.vector.com/cms/content/consulting/publications/Safety_White-Paper_Ebert.pdf

[37] Note that the concept of ASILs are specific to the ISO 26262 "Road vehicles – Functional Safety" standard.

- V&V is performed against the safety requirements to measure success or failure of the design and process.
    - A given safety requirement is verified if the associated test cases show a software product properly reflects a specified requirement, demonstrating the software was "built right" to satisfy a written requirement.
    - By contrast, a given safety requirement is validated if the software fulfills its intended use in the intended environment, i.e., the software engineers "built to the right requirement." Therefore, it is possible to have verified safety requirements that do not result in an intended safety benefit expected by end users—for instance, if the item does not operate in the intended environment.

Risk assessment and safety engineering processes are typically implemented for *any* system that has potential to impact vehicle safety. For example, an emissions control system may undergo a safety engineering process if it has potential to create an unsafe operating condition (e.g., fire) in a vehicle. The term "*safety system"* (e.g., active or passive safety system) is used to describe systems that are explicitly designed with the objective of improving vehicle safety.

2.1.3 Security

Vehicles today include a large number of interconnected ECUs, either physically through wired busses or wirelessly. With the trend of rising interconnectivity, in-vehicle networks are increasingly exchanging information with other vehicles, road infrastructure, and the internet. A growing need for networked E/E systems on motor vehicles has led to development and use of: [38]

- Mechanisms to manage the security of in-vehicle networks as part of the system design,
- Mechanisms to facilitate secure, efficient network traffic through authentication and authorization at runtime, and
- Mechanisms to enable security on legacy communication systems.

Security implications have led to a wide range of rapidly evolving technical advancements and consensus standards related to cryptography, software integrity, and anomaly detection, such as:

- ISO/IEC 27034-1, Information technology: Security techniques – Application Security.
- ISO/IEC 27036-1, 2, 3:2014, Information technology: Security techniques – Information security for supplier relationships.
- ISO/IEC 20243:2015, Information Technology: Open Trusted Technology Provider Standard (O-TTPS) – Mitigating maliciously tainted and counterfeit products.
- SAE International (SAE) AS6462A - AS5553A, Fraudulent/Counterfeit Electronic Parts: Avoidance, Detection, Mitigation, and Disposition Verification Criteria.

[38] Mundhenk, P. (2017). *Security for automotive electrical/electronic (E/E) architectures* [Dissertation, Technische Universität München]. Cuvillier Verlag. www.mundhenk.org/files/SecurityForAutomotiveEEArchitectures_PhilippMundhenk_Dissertation.pdf

- ISO 3011, Information technology – Security techniques: Vulnerability handling processes.
- ISO/SAE DIS 21434, Road Vehicles - Cybersecurity Engineering.

2.1.4 Real Time

Real-time systems are systems that rely on both (1) their input and output values *and* (2) on the timeliness with which those values are received and produced. In addition to other types of malfunctions, failures also occur when outputs are not produced before their deadline or produced with high variability.

Closed-loop feedback control of physical systems require real-time criticality. Ignition, for example, in an internal combustion engine, must occur within tight time tolerances with little variability ("jitter") to maintain the level of expected performance and avoid damaging the engine or generating unexpected emissions.

In automotive applications, real-time systems are characterized by:

- Real-Time Operating Systems, Schedulers, and Middleware;
- Discrete time domain control functions and algorithms and relevant MoC;
- Time-based state machines;[39]
- Event triggered or time triggered, deterministic, real-time communications busses, and
- Peripheral and IO architectures that support discrete and time domain-based models of computation.

Depending upon the specific application, other system properties may have real-time criticality, including:

- Trustworthiness,
- Location awareness,
- Maintainability, and
- Availability.

A *safety critical* system is a form of critical system that has direct influence over safety outcomes. It is important to understand that not all *critical* systems are *safety critical* systems. However, safety critical systems may be characterized by having both critical properties and safety properties.[40] The ability to incorporate high reliability, safety, security, and real-time systems in automotive E/E applications has heavily influenced the types of processes, tools, and technology used to develop automotive software.

A system's critical properties have strong influence over the MoC and lifecycle practices used during the development of the system, and they are significant drivers of overall system cost.

---

[39] AUTOSAR [AUTomotive *O*pen System ARchitecture] (2017.) *Specification of time service* (Document ID 624). www.autosar.org/fileadmin/user_upload/standards/classic/4-4-0/AUTOSAR_SWS_TimeService.pdf

[40] Rushby, 1994.

The impact of safety criticality on the cost of commercial aircraft software development serves as an example. In the study *DO-178C Costs Versus Benefits* Hilderman calculates that avionics software written to the highest level of criticality (DO-178C Level A) is 65 percent more expensive than software written to the lowest level of criticality.[41]

2.1.5 Fault Tolerance and Fault Recovery

An important concept related to critical systems is fault management. To manage risk, it is not only necessary to understand system criticality, but also how the system should behave in the event of faults, errors, or failures. The fault-management strategy is driven by the likelihood and severity of hazardous events that could result from system failures.

Fault-management strategies for automotive systems are typically developed based on hazard analysis according to ISO 26262.[42] Previously, the IEC 61508 standard for functional safety of E/E safety-related systems was employed, along with various internal processes developed over time.[43] For automotive E/E systems, fault-avoidance and human-intervention strategies have historically been chosen over more costly strategies involving redundant systems and fault isolation (see below and Table 4).[44]

This report does not provide a detailed analysis of failure recovery strategies. Rather, fault-management classification will be used as a means to compare safety processes across industries and domains, particularly as strategies relate to the SDLC. This report will define and use a high-level classification scheme to make macro comparisons of failure handling costs and sensitivity across industries and applications.

A fault-management taxonomy is not explicitly designated in AUTOSAR or ISO 26262—there are varying uses of descriptive terminology across the industry used to describe fault-management strategies ("fail gracefully," "fail silently," "fail over," and "safe-to-fail").

Safety systems incorporate a range of fault tolerance and recovery strategies, and often incorporate several methods. Hammet, Dubrova, and Koren and Krishna provide classification and definition for fault-tolerance techniques that are commonly used in today's E/E systems.[45] [46] [47] [48]

---

[41] Hilderman, V. (2014). DO-178C Costs Versus Benefits. Afuzion.

[42] Note that ISO 26262 is a derivative of IEC 61508. Hazard Analysis and Risk Assessment (HARA) for automobile software is described in Part 3 of ISO 26262.

[43] Czerny et al., 2010

[44] Wolf, J. (2015). *Is this What the future will look like? Implementing fault tolerant system architectures with AUTOSAR basic software*. Vector. https://assets.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Safety_ElektronikAutomotive_PressArticle_201511_EN.pdf

[45] Ibid.

[46] Dubrova, E. (2013). *Fault-tolerant design*. Springer.,"

[47] Hammett, R . (2016, August 8-12). *Developing electronic systems for safety critical applications*. 34th International System Safety Conference 2016, Orlando, FL.

[48] Koren, I., & Krishna, C. M. (2007). *Fault-tolerant systems*, 1st edition. Elsevier.

- **Fault Avoidance and Removal** - Achieved through quality control, mature and disciplined development processes, and conservative design.
- **Human Monitoring and Intervention** - System incorporation of self-tests, diagnostics, and reporting strategies including fault detection and isolation to allow human intervention in case of a failure.[49]
- **Safety Mechanisms** (e.g., "fail safe") - Incorporation of design features that cause the system to fail in a manner that allows for safety of the system to be maintained (e.g., circuit break, surge/over-speed protection, safety valve).[50]
- **Fault-Tolerant Systems** (e.g., "fail operational") - Use strategies such as containment and redundancy in order to tolerate faults and still deliver an acceptable level of service.

2.2 Complexity

Complexity, which will be defined throughout the forthcoming section, has become one of the biggest challenges for system engineers. There is growing recognition that there are cost-benefit limits in the implementation of systems of complex, interconnected, software defined E/E systems.[51] Redman describes this as the "*Affordability Limit,*" and predicts that, based on costs and expected selling prices for commercial airplanes, there is a point at which developmental costs may not be recaptured. For a commercial airplane, the report estimates this to occur at ~27M lines of code onboard with a software base cost of ~$7.8B.[52]

Wirthlin makes the case that different industries face different challenges when it comes to measuring and managing complexity (see Figure 1). Reinforcing this point, a recent automotive steer-by-wire system, whose function is to replace the mechanical steering linkage between the steering wheel and the driving wheels with wiring, contains ~14M LoC, approximately the same as the entire code base for the flight software on board the Boeing 787.[53] [54]

Methods for robustly measuring software complexity are an ongoing topic for debate within the software engineering community, whether within the automotive domain or elsewhere. LoC tend to be used as the default measurement of complexity as it is easy to quantify. However, many

---

[49] Note that "fail over" fault recovery is often used to describe the fault recovery case where a backup system takes control during failure of the primary system. Human monitoring and intervention might be considered to be an implementation of a fail over strategy.

[50] Note that ISO 26262-1:2018 defines *safety mechanism* as "*a technical solution implemented by E/E functions or elements (3.41), or by other technologies (3.105), to detect and mitigate or tolerate faults (3.54) or control or avoid failures (3.50) in order to maintain intended functionality (3.83) or achieve or maintain a safe state (3.131).*"

[51] Sheard, S. (2015, June 15). *Aircraft systems: Three principles for mitigating complexity* [SEI blog post]. Software Engineering Institute. https://insights.sei.cmu.edu/sei_blog/2015/06/aircraft-systems-three-principles-for-mitigating-complexity.html

[52] Redman, D. A., Ward, D. T., Chilenski, J., & Pollari, G. (2010). *Virtual integration for improved system design*. Software Engineering Institute.

[53] Nexteer Automotive. (2018, April 9.) *Nexteer expands strategic software investment & global team* [Web press release]. www.nexteer.com/release/nexteer-expands-strategic-software-investment-global-team/

[54] Norris, G., & Wagner, M. (2009). *Boeing 787 Dreamliner*. Zenith Press. The Boeing 787 Dreamliners' avionics and online support systems account for between 6 and 7 million LoC. The total flight software of the 787 amounts to ~14 million LoC.

studies[55] [56] [57] investigate whether LoC is truly a robust measurement of software complexity. In the 1970s and 1980s, when software was predominantly hand coded, LoC often served as an indicator of software complexity and effort. Today, many factors that influence overall complexity are not reflected by the LoC indicator. High product volume and the potential for tens of thousands of possible ECU permutations contribute to E/E lifecycle complexity. This complexity is compounded by short lifecycles and variability across dozens of E/E subsystems, which must be integrated into the vehicle (examples of powertrain and telematics ECUs are shown). Non-safety-critical systems such as infotainment, navigation, telematics, and comfort features, which interact among themselves and with other software subsystems, and which can have off-the-shelf rather than tailored code bases, are largely driving the increase in automotive software LoC.[58, 59] As seen in Table 2, the LoC of telematics can be six times higher than the LoC of safety-critical powertrain controllers. It is possible that with the introduction of ADS, portions of the automotive code base that are currently non-critical, e.g., navigation, may become more critical in the future.

At least two other metrics for complexity are worth noting, both of which are also somewhat easy to compute: Halstead Complexity, which is a measurement of data flows, and Cyclomatic Complexity, which is a measurement of control flows. While LoC is probably the most used and well-known complexity metric, Cyclomatic Complexity is also commonly used, and it is calculated by measuring the number of linearly independent paths through the source code.

Software can become complex for a variety of reasons. In addition to core functionality that software must provide, complexity is also often added for testing, verification, and validation purposes. This added LoC can be disabled in production, effectively reducing complexity before customer use.

---

[55] Tashtoush, Y., Al-Maolegi, M., & Arkok, B. (2014, June). The correlation among software complexity metrics with case study. *International Journal of Advanced Computer Research, 4*(2), 2277-7970. https://arxiv.org/ftp/arxiv/papers/1408/1408.4523.pdf

[56] Bloom, J. (2015, March 24). *Five reasons you MUST measure software complexity* [Web page]. Software Intelligence Pulse. www.castsoftware.com/blog/five-reasons-to-measure-software-complexity

[57] Bhatia, S., & Malhotra, J. (2014, August 1-2). A survey on impact of lines of code on software complexity. 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), Unnao, India. doi 10.1109/ICAETR.2014.7012875 https://ieeexplore.ieee.org/document/7012875

[58] Edelstein, S. (2015, May 14). *The Ford GT has more lines of code than a Boeing passenger jet* [Web press release]. Motor Authority. www.motorauthority.com/news/1098308_the-ford-gt-has-more-lines-of-code-than-a-boeing-passenger-jet#:~:text=The%20Ford%20GT%20Has%20More%20-Lines%20Of%20Code%20Than%20A%20Boeing%20Passenger%20Jet,-Stephen%20Edelstein%20May

[59] Saracco, R. (2016, January 13). Guess what requires 150 million lines of code ... [Blog post]. IEEE Future Directions. https://cmte.ieee.org/futuredirections/2016/01/13/guess-what-requires-150-million-lines-of-code/

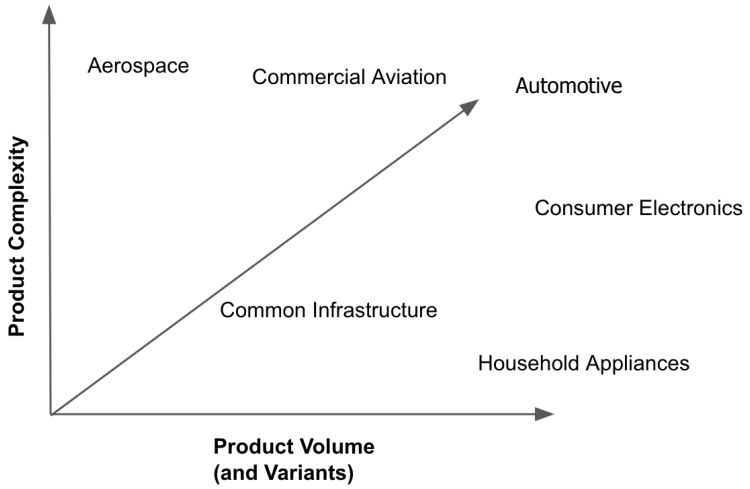

Figure 1. Comparison of several industries and products in terms of complexity and product volume

The term complexity is subject to varying definitions. In terms of product variants and volume the automotive industry has become uniquely complex.[60] In terms of complexity, the automotive industry is rapidly differentiating itself from other industries, particularly those that develop large systems with relatively long life cycles and small production volumes (see Figure 1). The automotive industry in contrast can be characterized by high-production volumes, high variance within production models (e.g., options, configurations), and high development life-cycle variability within products.

Examples of automotive software lifecycle variability are shown in Table 2. Within the vehicle lifecycle (an average light vehicle age of 11.8 years[61]) each subsystem has its own lifecycle and associated complexities.

On one hand, increasing use of software benefits the automotive industry. With high production volumes, software allows addition of features and functionality using existing hardware without adding significantly to manufacturing costs. On the other hand, high volume and high variance within models also means that there are greater permutations of configurable software that must be developed, tested, and properly activated in production vehicles.

[60] Wirthlin, R. (2018, March 29). *Embedded software in products: The convergence of ALM with systems engineering* [Powerpoint]. Exploring Application Lifecycle Management and Its Role in PLM, 2018 Spring Meeting, PLM Center of Excellence, Purdue University [Powerpoint]. https://polytechnic.purdue.edu/sites/default/files/files/Embedded%20software%20in%20products%20-%20the%20convergence%20of%20ALM%20with%20Systems%20Engineering.pdf

[61] Bureau of Transportation Statistics. (n.d.) Average Age of Automobiles and Trucks in Operation in the United States [Web page, embedded Excel dataset]. www.bts.gov/content/average-age-automobiles-and-trucks-operation-united-states

Table 2. Examples of automotive software lifecycle and complexity variation

Lifecycle Property and the Associated Complexity Considerations		
Lifecycle Property	**Complexity Metric**	**Complexity Measurement**
Production Volume[62]	Units Produced (e.g. Ford F-150/250/350/450, 2018)	~ 75,000/Month
Variants and Options[63]	Variants (e.g. Ford CD Class)	*~10,000 Vehicle-Series-Variants (based on ECU component permutations per vehicle line)*
	Automotive (relative) Mechanical Domain Complexity	10^3 Dependencies
	Automotive (relative) Onboard and Off Board Software Domain Complexity	10^6 Dependencies
Automotive Powertrain[64] **Controller Lifecycle**	Memory	256 Kb
	LoC	50,000
	Time to Market	~24 Months
	Development Effort	40 person-years
	Validation Time	5 Months
Automotive Telematics Unit Lifecycle[65]	Memory	8Mb
	LoC	300,000
	Time to Market	~12 Months
	Development Effort	200 person-years
	Validation Time	2 Months

[62] Ford Authority. (n.d.) Ford F-Series sales numbers [Web page]. http:/fordauthority.com/fmc/ford-motor-company-sales-numbers/ford-sales-numbers/ford-f-series-sales-numbers/

[63] Davey, 2013.

[64] Ferrari, A. (n.d.). *An overview of (electronic) system level design: beyond hardware-software software co-design* [PowerPoint]. Parades GEIE [Gruppo Europeo di Interesse Economico] Roma. www.sti.uni-urb.it/events/sfm06hv/slides/Ferrari.pdf

[65] Ibid.

Sheard, of the Software Engineering Institute, notes that, "While complexity is often blamed for problems, the term is usually not defined. There is widely acknowledged lack of agreement as to how to define system complexity."[66]

A wide range of factors drive complexity across the automotive SDLC.[67] Despite lack of agreement on, or standardized approaches for, how to best define and measure complexity, Antinyan (Volvo), Davey (Ford), and Wirthlin (GM) make similar observations related to the most common sources of automotive software complexity.

Software complexity in the automotive domain can be classified into the following groups:

Process Complexity - Interconnections between requirements, code, tests, interface definitions, hardware architecture, software variants, software versions, and development tools is the source of prodigious complexity that builds up in automotive software over the software evolution.[68]

Code Complexity - Structure of the computer code can have different magnitudes of influence on complexity, and different implementations of the same task can have substantially different code complexities.

Architectural and Variant Complexity - Different architectural perspectives yield different requirements sets that must be reconciled:

- Electrical architectures deal with ECUs and interconnections,
- Software architectures deal with software components and their interconnections, and
- Variant architectures deal with different software configurations that may run on the same ECU.

Requirements Complexity - Complexity of requirements for automotive functional and performance characteristics and also for related E/E SDLC processes is driven by a wide range of technical requirements and market factors.

2.3 Business Factors

IEC 61508 (see Section 2.1.5 Fault Tolerance and Fault Recovery) addresses safety systems that are predominantly owned, operated, and maintained by commercial or government entities. Nuclear power plants, rail systems, medical devices, and commercial airplanes are typical examples of safety systems within the power, healthcare and transportation domains.

2.3.1 Ownership Models

Ownership, maintenance, and operation of commercial and government safety systems are often accompanied by:

- Design for maintainability, design for durability;
- High utilization (hours in operation per day);
- Long product and sustainment life cycles;

[66] Sheard, 2015.
[67] Antinyan, 2018
[68] Ibid.

- Fewer "options" for product personalization, brand identity, comfort, entertainment;
- Maintenance by professional, certified technicians; and
- Required, regulated, and audited maintenance intervals.

Some motor vehicles, such as buses and commercial trucks, have ownership models more closely aligned with commercial and industrial systems. However, in general, consumer ownership and operation of vehicles are much different, characterized by independent ownership,[69] low utilization, widely varying models of private use, high levels of product customization, and widely varying levels of individual vehicle care and maintenance.

The large variety of product offerings and configurations, along with direct-to-consumer marketing, are more akin to smartphones and other consumer electronics. For example,

- 77 percent of vehicle owners drive themselves as their primary means of transportation to work - total car utilization is typically 1 to 2 hours per day (see Table 3 and Table 4); personal vehicles are parked for much of their lifespan.
- Vehicle advertising in 2011 contributed to nearly 25 percent of all advertising revenue, emphasizing the direct-to-consumer marketing approach.[70]

This section describes how consumer-driven demands, particularly where they differ from safety systems found in other industries or domains, influences SDLC requirements.

2.3.2 Business Factors Today

Automotive manufacturers are incorporating more technology in new vehicles. The data in Table 3 shows that software and electronics are a large and growing portion of a new vehicle purchase, and analysts expect this portion to grow from 10 percent of the vehicle purchase in 2019 to as much as 50 percent by 2030.[71] [72] Much of the projected growth in automotive software stems from multi-billion dollar private sector investment in ADS technology and supporting software.

[69] Note that fleet sales made up approximately 17 percent of the new-vehicle market in 2019. Bond, V. Jr. (2019, October 7). *Fleet gains keep sales pace above 17 million* [restricted web press release]. Automotive News. www.autonews.com/sales/fleet-gains-keep-sales-pace-above-17-million

[70] Marketing Schools. (2020, December 3). *Marketing cars: How successful car manufacturers market their vehicles to you ...* [Web page]. www.marketing-schools.org/consumer-psychology/marketing-cars.html

[71] Grand View Research. (2019, July). *Automotive electronic control unit market size, share, & trends analysis report by application, by propulsion type, by capacity, by vehicle type, by region, and segment forecasts, 2019 - 2025* (Report ID: 978-1-68038-367-6). www.grandviewresearch.com/industry-analysis/automotive-ecu-market

[72] Burkacky, O., Deichmann, J., Doll, G., & Knochenhauer, C. (2018, February 14). *Rethinking car software and electronics architecture.* McKinsey & Company. www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture

While individually-owned passenger vehicles are expected to continue to hold their position as the largest automotive software submarket, emerging submarkets could create demand for different models of vehicle ownership.[73] For instance, some forecasts project that the market share for mobility as a service (MaaS) (e.g., car clubs and shared-use services) will grow by more than 30 percent per year through 2030.[74] A shift from individual ownership to ride-sharing and car-sharing services may have an impact on the ownership model discussed in the previous section. For instance, vehicles may begin to experience higher utilization rates (hours-in-service). Fleet-operated vehicles may also benefit from more regular maintenance service from certified professionals. There may also be a decrease in vehicle models and configurations as fleet operators opt for more standardized equipment to simplify maintenance. However, the effect of fleet ownership has not yet manifested as reduced complexity for software development.

This rapid change in ownership model is further accelerated when the vehicle is viewed *as a software platform,* creating a unique, growing market for consumer software that is *integrated on the vehicle with critical software*. Software developers can expect that automotive software practices will continuously evolve, driven by rapidly changing market factors, including:

- Non-traditional suppliers of new automotive software platforms (e.g., NVIDIA, Tesla, Google/Waymo, Apple);
- Heavy investment in new mobility models by large start-up ventures (e.g., Uber, Lyft);
- Demand for increasing automation;
- New and emerging vehicle architectures (e.g., battery electric vehicle (BEV), fuel cell electric vehicle (FCEV), hybrid);
- New and increasing data exchanges between vehicles and other data sources (e.g. weather);
- Traditional vehicle manufacturers (e.g., Volkswagen, GM) expanding product offerings and investing in software, AI, new mobility models; and
- Increasing market segmentation and diversification of automotive software domains based on:
- Demographics and geographies (e.g., “Gen X”, “millennial”, rural, urban);
- Technology (e.g., BEV, hybrid vehicle (HV), V2X, ADS, fuel cell);
- Transportation model (e.g., consumer, MaaS);

---

[73] Kumar, A., Manda, and Atreya, A. (2020, May). *Automotive software market by application (ADAS & safety, connected services, autonomous driving, HMI, V2X, infotainment), software layer (OS, middleware, application), EV application (charging, battery, V2G), vehicle and region - Global forecast to 2025.* MarketsandMarkets. www.marketsandmarkets.com/Market-Reports/automotive-software-market-200707066.html?gclid=EAIaIQobChMIwebBuP7o4AIVZB6tBh0ksAxHEAAYASAAEgJlN_D_BwE

[74] Godbole, A. (2020, August). *Mobility as a service market by service (ride hailing, car sharing, micro mobility, bus sharing, train), solution, application, transportation, vehicle type, operating system, business model and region - Global forecast to 2030*. MarketsandMarkets. www.marketsandmarkets.com/Market-Reports/mobility-as-a-service-market-78519888.html

- Propulsion energy supply and availability (e.g., electricity, gasoline, hydrogen, diesel, natural gas, ethanol mixes, biodiesel, propane); and
- Automotive technology subdomain (e.g., ADS, ADAS).

Vehicle manufacturers are under growing pressure to innovate while containing costs. The industry is increasingly investing in and relying upon software for vehicle innovation, while also changing and adapting to new models of vehicle maintenance, delivery, and ownership. As a result, the auto industry will continue to see pressure to vigorously improve the critical behavior (reliability, dependability, security, safety, maintainability) of automotive E/E systems. [75] [76] [77]

Table 3. Overview of Some Current Business and Market Statistics in the Automotive Sector

Ownership[78]		
	Total vehicles registered in USA	~240M Vehicles (2015)
	Total vehicles in fleets in USA	5.7M Vehicles (2015)
	Number of vehicles per Household (USA)	1.95 (2015)
Contribution to Vehicle Content[79] [80] [81] [82]		
	Software as a Percent of Overall Vehicle Cost (Average)	10% (2018)
	Electronics parts as a Percent of Overall Vehicle Cost	30% (2018)
	Forecasted Software as a Percent of Overall Vehicle Cost	30% - 50% (2025)
	Global Automotive ECU Market	$33.5B (2016)
	Forecasted Global Automotive ECU Market	$60.5B (2025)

[75] Glance, D. (2017, September 10). *As your car becomes more like an iPhone, get ready to update its software regularly* [Web page]. The Conversation US, Inc. https://theconversation.com/as-your-car-becomes-more-like-an-iphone-get-ready-to-update-its-software-regularly-83780

[76] Gelles, D., Tabuchi, & H. Dolan, M. (2015, September 26). "Complex car software becomes the weak spot under the hood." *New York Times*.

[77] Hars, A. (2017, September 23). *Self-driving vehicles: The "platform" business model* [Web page]. Driverless car market watch. www.driverless-future.com/?p=1091

[78] Bronzini, M., Fletcher, W., Rick, C., Camp, J., Firestine, T., Schmitt, R., Beningo, S., Ford, C., Liu, M., Menegus, D., Nazareth, R., Nguyen, L., Reschovsky, C., Riddle, J., Smallen, D., Smith-Pickel, S., & Tang, C. (2017). *Transportation Statistics Annual Report 2017.* Bureau of Transportation Statistics. https://cms7.bts.dot.gov/sites/bts.dot.gov/files/docs/browse-statistical-products-and-data/transportation-statistics-annual-reports/215041/tsar-2017-rev-2-5-18-full-layout.pdf

[79] Burkacky, et al., 2018.

[80] Statista. (2021). *Automotive electronics cost as a percentage of total car cost worldwide from 1970 to 2030* [Web page]. www.statista.com/statistics/277931/automotive-electronics-cost-as-a-share-of-total-car-cost-worldwide/

[81] Grand View Research, 2019.

[82] Ibid.

Product Lifecycle[83] [84] [85] [86]		
	Average Vehicle Age (USA)	11.5 years
	Average Vehicle Life Expectancy (USA)	11-17 years
	Average Vehicle Product Development Lifecycle	6.7 years
	Number of Light Vehicle Retail Sales in the US	~17M units (2018)
Usage and Mobility Model [87]		
	Principal Means of Transportation to Work - Drives Self	77% (2015)
	Principal Means of Transportation to Work - Carpool	9% (2015)
	Principal Means of Transportation to Work - Public Transportation, Cab, Bicycle, Motorcycle, Other	14.% (2015)

Emerging and Changing Business Environment

Automotive software incorporates the challenges of being complex and safety critical, while also meeting consumer-driven demands for a variety of features and options. This trend is coupled with an industry knowledge base that is still transitioning from mechanical to software-based system development, following architectural consensus standards and software process consensus standards that are themselves still maturing (e.g., ISO 26262 and AUTOSAR).

[83] Bureau of Transportation Statistics (2015). *Average age of automobiles and trucks in operation in the United States, 2015.*

[84] Consumer Reports. (2018, November 6). Make your car last 200,000 miles: How to go the distance and save tens of thousands of dollars [Web page]. www.consumerreports.org/car-repair-maintenance/make-your-car-last-200-000-miles/

[85] Center for Automotive Research. (2017, September 20). Automotive Product Development Cycles and the Need for Balance with the Regulatory Environment [Web page]. www.cargroup.org/automotive-product-development-cycles-and-the-need-for-balance-with-the-regulatory-environment/

[86] Lassa, T. (2018, January 4). U.S. auto sales totaled 17.25-million in 2017: Trucks and SUVs now account for two-thirds of the market, Toyota estimates [Web page]. Motortrend. www.automobilemag.com/news/u-s-auto-sales-totaled-17-25-million-calendar-2017/

[87] Bronzini et al., 2017.

The research for this study has identified several key business trends that are most likely to impact SDLC practices:

- Growing demand for new software and technology features on traditional vehicle platforms (including conversion of legacy mechanical systems into mechatronic systems).
- New mobility models.
- Demand for automation and ADAS.
- Connectivity.

Table 4 shows how these trends are likely to impact different requirements that drive developmental practices. [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100] [101] [102]

[88] Wolski, C. (2016). *How to leverage fleet vehicle purchases* [Web page]. Fleet Financials. www.fleetfinancials.com/156847/how-to-leverage-fleet-vehicle-purchases

[89] Shin, D., Park, K., & Park, M. (2018). *Effects of vehicular communication on risk assessment in automated driving vehicles.* Applied Science, 8, 2632. doi.org/10.3390/app8122632

[90] Saberi, A. K., Barbier, E., Benders, F., & van den Brand, M. (2018, April 24). On functional safety methods: A system of systems approach. 12th Annual IEEE International Systems Conference (SysCon), Vancouver, Canada. doi: 10.1109/SYSCON.2018.8369598

[91] Ivanov, I., Maple, C., Watson, T., & Lee, S. (2016, March 28-29). Cyber security standards and issues in V2X communications for Internet of Vehicles. Living in the Internet of Things: Cybersecurity of the IoT – 2018, London.

[92] Bernon-Enjalbert, V., Blazy-Winning, M., Gubian, R., Lopez, D., Meunier, J.-P., & O'Donnell, M. (n.d.) *Safety-integrated hardware solutions to support ASIL-D applications.* Freescale. https://www.nxp.com/docs/en/white-paper/FUNCSAFTASILDWP.pdf

[93] Muehlhauser, L. (2013, August 25). *Transparency in safety-critical systems* [Web page]. Machine Intelligence Research Institute. https://intelligence.org/2013/08/25/transparency-in-safety-critical-systems/

[94] Dawson, N. (2017). *Designing effective policies for safety-critical AI* [Web page blog]. Bits and Atoms. https://bitsandatoms.co/effective-policies-for-safety-critical-ai/

[95] Hewitt, C. (2012). *What is computation? Actor model versus Turing's model in a computable universe: Understanding computation & exploring nature as computation.* World Scientific Publishing Company.

[96] Koopman, P., & Wagner, M., (2016). Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety, 4*(1). doi.org/10.4271/2016-01-0128

[97] Silver, A. (2016, October 7). *Why AI makes it hard to prove that self-driving cars are safe: Engineers weigh in on the pitfalls of machine learning and autonomous driving* [Web page]. IEEE Spectrum.

[98] Aitrends. (2019, February 15). Current generation of self-driving cars AI needs a safety certification process [Web page]. Cambridge Innovation Institute. www.aitrends.com/ai-world-2018/current-generation-of-self-driving-cars-ai-needs-a-safety-certification-process/

[99] U.S. Department of Homeland Security. (2018). *AI: Using standards to mitigate risks*. www.dhs.gov/sites/default/files/publications/2018_AEP_Artificial_Intelligence.pdf

[100] Muehlhauser, 2013.

[101] Dawson, 2017.

[102] Bernon-Enjalbert, et al., n.d.

Table 4. High level analysis of how global business requirements impact development practices for critical systems

Business Trends Likely to Impact Lifecycle Practices	Potential Impact on the Automotive Software Lifecycle
Increasing Number of Mechatronic and Software Features	• Increasing complexity (architectural, code, requirements, process, product variant) driven by more interconnected software features • Increasing numbers of critical and safety critical systems • Increasing numbers of mixed criticality systems (e.g., interactions between telematics, entertainment, and ADAS systems) • Continuing need to integrate legacy systems with new architectures • Increasing criticality related to security, maintainability, reliability, etc. • Increasing reliance on existing reference architectures (e.g., AUTOSAR, automotive grade Linux [AGL]) • Increasing need for effective fault management, process maturity, quality in order to drive down the number of E/E related defects
Mobility as a Service (MaaS) and Shared Use	• Decreasing product variant complexity - lower number of options and models due to bulk/fleet sales • Changing criticality related to security, maintainability, reliability, location awareness, and availability due to increased utilization/duty cycle, commercial use, fleet management system vulnerabilities • Increasing need for fleet maintenance processes including health monitoring.
Automated Driving Systems[103]	• Increased architectural, code, process, and requirements complexity due to growing number of E/E systems, new/changing MoC, changing and increasing criticality of E/E systems • Changing MoC drive demand for new hardware architectures based on GPU, high bandwidth busses, high bandwidth sensors, highly integrated/arbitrated subsystems, etc. • Changing failure management strategies with increasing numbers of critical E/E systems, (e.g., arbitration and redundancy are used more) • Changing criticality for systems related to trustworthiness, location awareness, AI, ethics • New and changing MoC including stochastic and machine learning, probabilistic computing, arbitration • Changing SDLC methodologies including AI-driven software development • Need for consensus standards maturity related to AI, learning algorithms, probabilistic computing, AI coding practices, and AI algorithms • Evolution of existing reference architectures (e.g., AUTOSAR) to accommodate new MoC and architectures; introduction of new reference architectures
Software Interconnectivity with Systems Outside the Vehicle	• Increasing risk management complexity due to more interconnections to other software systems • Changing risk models; likelihood and severity of failures are increased as vehicle software integrates with other/larger systems. • Changing criticality for systems related to security, geolocation, quality of service

[103] This is a reference to a wide range of system capabilities covered by implementation of software under the levels of driving automation (3-5) as defined in SAE J3016, June 2018.

Business Trends Likely to Impact Lifecycle Practices	Potential Impact on the Automotive Software Lifecycle
	• Changing failure management strategies with increasing numbers of interconnected critical systems • Need for consensus standards maturity related to security, communications protocols, interoperability, specific and appropriate for the intended use cases. • Evolution of existing reference architectures (AUTOSAR, etc.) and the introduction of new reference architectures

2.4 Current State-of-the-Art

Today, highly specialized processes are used for hardware and software development for automotive E/E systems, driven by functional and performance requirements and global considerations. The uniqueness of global, functional, and performance requirements across the automotive industry has led to SDLC processes and tooling that are specialized and unique to the industry.

The research indicates that best practices for today's SDLC processes are designed to support the following objectives:[104]

- Minimize software development time,
- Maximize model-based software,
- Use ACG in support of necessary MoC and languages,
- Use reference architectures to maximize reuse and interoperability, and to minimize complexity,
- Maximize reuse of hand-written software components, and
- Minimize hardware platform change requests.

2.4.1 Challenges for Software Requirements

In order to understand current and future impacts on software development, production, and maintenance practices for vehicular software, it is important to understand the demands on software processes that are being used.

The study of requirements analysis and flow down is a complex field of study in and of itself.[105] The SDLC process is designed by systems engineers with the *goal* of ensuring that as many of the design requirements are captured and completed as possible.[106] The ability to achieve this goal is influenced by many factors outlined in this report, and engineers are continuously faced with the challenges of requirements completeness, testability, traceability (of test results to

[104] Ferrari, n.d.

[105] Lee, S., & Duo, S. (2013, November 19-21). *Safety analysis of software requirements: model and process*. 3rd International Symposium on Aircraft Airworthiness, Toulouse , France.

[106] Jackson, S. (1997). Systems Engineering for Commercial Aircraft: a domain specific adaptation. Ashgate.

requirements and derived requirements to top-level requirements), "requirements creep" when extra requirements that inadvertently expand scope are added to the core set, and a myriad of other difficulties that make requirements management a highly complex endeavor. In 2014 Braun noted that "*the growing complexity (of software intensive embedded systems in the automotive industry) drives current requirements engineering practices to the limits.*"[107]

For manufacturers of vehicles, requirements management across the entire product development lifecycle and subdomains, including the software development lifecycle, is a great undertaking. Correct and complete specification of requirements is particularly important for software. Software safety is not driven by random failures. Rather, studies have shown that most errors in software result from flawed or incomplete requirements,[108] and system engineers often use three categories for defining requirements.[109]

- **Functional Requirements** - the functional needs of the vehicle, without any performance specifications.
- **Performance Requirements** - the measure of the extent to which the system performs a function.
- **Constraints** - non-performance requirements that cannot be traced to a function (also called *global requirements*).

Critical and safety systems impose further requirements and constraints on the SDLC. As part of the item definition in Part 3 of ISO 26262, the functional and non-functional (e.g., performance) requirements of a system must be known in order to determine whether an observed behavior is malfunctioning:

> *"Absence of unreasonable risk due to hazards caused by malfunctioning behavior of Electrical/Electronic systems*." (ISO 26262)[110]

2.4.2 Challenge of Resolving Lifecycle Practices against Requirements

The practice of developing automotive performance and functional requirements is complex and subject to rapidly changing and contradictory demands that are often difficult to interpret. Usable requirements can be extremely challenging to develop, particularly when trade-offs often emerge among levels of costs, performance, safety, security, and other factors.

2.4.3 Key Software Development Approaches

Automotive software onboard the vehicle is predominantly described as *embedded*. Embedded systems are dedicated computers that are built into the system; they are usually based on a microcontroller programmed and controlled by an RTOS or task scheduler, with a dedicated

---

[107] Braun, P., Broy, M., Houdek, F., Kirchmayr, M., Müller, M., Penzenstadler, B., Pohl, K., & Weyer, T. (2010). Guiding requirements engineering for software-intensive embedded systems in the automotive industry. *Computer Science - Research and Development, 29*, 21-43.

[108] Leveson, N. (2012). *Engineering a safer world.* MIT Press.

[109] Jackson, S. (1997). *Systems engineering for commercial aircraft: A domain-specific adaptation*. Ashgate.

[110] Note: clause 5.4.1 of Part 3 of ISO 26262 explicitly states that: "The requirements of the item shall be made available [as a prerequisite of the ISO 26262 functional safety process] Note: If the functional and non-functional requirements are not already available, their generation can be triggered by the requirements of this clause."

function within a larger mechatronic or electronic system, often with real-time computing resource constraints, such as limited ROM, EEPROM, etc.

2.4.4 Models of Computation

MoCs describe the types of mathematical functions that a computer architecture can process. Many consumer devices are "general purpose" computing devices, such as personal computers, which are designed to handle a broad range of MoCs. In vehicles, computer architectures are dedicated embedded systems, designed around a specific MoC.

In automotive embedded software, concurrent tasks (processes) and threads are scheduled by the RTOS. Tasks interact with each other and the environment, with behavior and interaction mechanisms defined by the MoC.[111] For automotive embedded systems, MoCs center on the ability to run tasks that rely on the following.

- Concurrency
- Communication and synchronization
- Time
- Hierarchy

Models of computation for networked real-time control systems found in automotive software applications are specialized. An examination of the MoC provides insight into how to classify automotive software systems and compare them with other industries and domains.

Automotive MoC include:

- **Communicating Finite State Machines** - Used frequently for modeling communication protocols;[112]
- **Dataflow Process Networks** - Used for visual dataflow programming and model based design;[113]
- **Discrete Event** - Used for modeling and simulation of communication networks, hardware architectures, systems of systems, and in the design of and software synthesis for sensor networks, distributed real-time software, and hardware software systems;[114]
- **Codesign Finite State Machines** - Used for MBD of embedded software systems;[115] and
- **Synchronous Reactive** - Used to model embedded control systems where safety must be preserved.[116]

---

[111] Lee, E. A. (2011). Concurrent models of computation [PowerPoint Slides for course of same name]. UC Berkeley.
[112] Lee, 2011.
[113] Lee, E. A., & Parks, T. M. (1995). Dataflow process networks. *Proceedings of the IEEE 83*(5):773-801. doi: 10.1109/5.381846,"
[114] Lee, 2011.
[115] Mahapatra, R. N. (2002). Co-design finite state machines [PowerPoint]. Texas A&M University.,"
[116] Edwards, S. A., Lee, E. A., Tripakis, S., & Whitaker, P. (2014). Synchronous-reactive models. In Ptolemaeus, C., ed., *System design, modeling, and simulation using Ptolemy II*. Ptolemyy.org, Berkeley, CA.,"

2.4.5 Architectural Standards

The models of computation used in automotive software development rely on highly integrated and structured lifecycle practices for safety, architecture, and design. The automotive industry has developed several process consensus standards, including:

- **Automotive** software process improvement and capability determination **(ASPICE)** - A standard for implementation of enterprise software capability that is specific to the automotive domain. The ASPICE consensus standard is a process assessment and capability model, and is older and more generic than ISO 26262, which is focused on safety.
- **AUTOSAR** software process engineering meta-model (**SPEM)** an open and standardized software architecture for automotive ECUs. SPEM is a meta-model for defining processes and their components. ("Software Process Engineering Meta-model Specification," Object Management Group, 2001)
- **Part 6 of ISO 26262** - "Road vehicles – Functional safety," an international standard for functional safety of E/E systems in production automobiles defined by the International Organization for Standardization in 2011, specifically deals with software.

A common thread among these consensus standards is the automotive industry's view of the SDLC. The V-Cycle is referenced as the prevailing SDLC methodology in each of the above consensus standards.

Dvorak's *NASA Study on Flight Software Complexity* identifies establishment of reference architectures as critically important to address risks associated with the growth in size and complexity of flight software.[117]

Vehicle manufacturers and suppliers have identified similar needs due to the complexity and heterogeneity of wide-ranging, often proprietary, computing architectures; communications protocols; and sensing and actuation technologies. The automotive ecosystem consists of hundreds of suppliers across thousands of model variants and vehicle implementations.[118] [119]

The challenge of maintaining real-time, dependable, and safe system architectures has led the automotive industry to promote architectural consensus standards that allow interoperability across platforms and across product development lifecycles that often extend across supply chain boundaries. However, standardization can be a slow, inconsistent process, and it can be slowed by conflicting interests and competitive factors.

Standards and Licensing

As mechanical systems are replaced by mechatronics, vehicle content is progressively being defined by software-based control systems. Unlike mechanical and electrical components, software is often licensed, and has the potential to be upgraded, monitored, and changed by the software producer. Software content is beginning to be offered under end-user licensing that more closely resembles the plan for a cell phone or personal computer than a traditional vehicle

[117] Dvorak, D. L. (2009, April 6-9). NASA study on flight software complexity. AIAA Infotech Aerospace Conference, Seattle WA.
[118] Davey, 2013.
[119] Wirthlin, 2016.

purchase, lease, or maintenance agreement.[120] [121] Some models of automotive software licensing are explicitly claiming acceptance of end-user licensing by the consumer.

Despite the competitive factors driving proprietary content, there are benefits to and activities that target reducing complexity and facilitating open consensus standards for interoperability, for example, to integrate supplier workflow, and to standardize component and subsystem interfaces (e.g., AUTOSAR, Section 2.4.5.1).

Two of the most noteworthy standardization efforts with impacts on automotive software are AUTOSAR and Automotive Grade Linux.

2.4.5.1 AUTOSAR

Perhaps the most widely used and accepted architectural standard for automotive E/E architectures, AUTOSAR is a partnership of automotive manufacturers and suppliers working together to develop and establish an open industry standard for automotive E/E architectures.[122] [123] Today, AUTOSAR core and premium partners constitute a large, cross section of vehicle manufacturers, component, and tools suppliers.

The Verband der Automobilindustrie survey referenced in *Future Programming Paradigms in the Automotive Industry* found that AUTOSAR is used in 82 percent of automotive software projects and OSEK in 55 percent of automotive software projects.[124] [125]

AUTOSAR provides a set of architectural consensus standards designed to allow interoperability between system hardware and software by describing and defining basic software modules, application interfaces, and a common development methodology based on standardized exchange formats. It is designed to allow increased use of commercial-off-the-shelf hardware, software, and tools, and increased use of model-based development practices by automotive manufacturers and suppliers.

An in-depth study of AUTOSAR may provide more insight into how the automotive industry is tackling the challenges of portability and composability of software components from a multitude of suppliers with varying competitive interests, across automotive subdomains with differing functional, safety, critical, and performance requirements.

---

[120] Bloomberg, J. (2017, April 30). *John Deere's digital transformation runs afoul of right-to-repair movement* [Web page]. Forbes. www.forbes.com/sites/jasonbloomberg/2017/04/30/john-deeres-digital-transformation-runs-afoul-of-right-to-repair-movement/#3eaf04975ab9

[121] Glance, 2017.

[122] AUTOSAR. (2008). *Technical overview V2.2.1 R3.0 Rev 0001* (Document ID 067).

[123] Hurley, B. (2011, March 1). *Global car platforms: Automotive design with the world in mind* [Web page]. Techbriefs.

[124] Molotnikov, A., Schorp, K., Aravantinos, V., & Schätz, B. (2016). *Future programming paradigms in the automotive industry*; German Association of the Automotive Industry.www.vda.de/dam/vda/publications/2016/FAT/FAT-Schriftenreihe_287.pdf

[125] OSEK (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen [Open Systems and their Interfaces for the Electronics in Motor Vehicles])is a standards body that has produced specifications for the embedded operating system ISO 17356-3; AUTOSAR "Specification of Operating System" uses OSEK OS:ISO 17356-3 as the basis for the AUTOSAR OS.

AUTOSAR's "classic platform" sets consensus standards for embedded real-time ECUs and operating systems and for hardware and software components (including real-time elements) in vehicle architectures.

The motivations[126] and perceived benefits and drawbacks[127] behind the development of key AUTOSAR features and consensus standards are shown in Table 5 and Table 6.

Table 5. Reported motivations of automotive manufacturers and suppliers for using AUTOSAR solutions

AUTOSAR Solution	Motivation
Standardization of specification exchange formats	● Improvement in specification (format and content). ● Opportunity for a seamless tool chain.
Basic Software Core	● Enhancement of software quality. ● Concentration on functions with competitive value.
Microcontroller Abstraction	● Microcontroller can be exchanged without the need for adaptation of higher software layers.
Runtime Environment (RTE)	● Encapsulation of functions creates independence of communication technology. ● Communication easier through standardized mechanisms. ● Partitioning and relocation of functions possible.
Standardization of Interfaces	● Reduction/avoidance of interface proliferation within and across vehicle manufacturers and suppliers. ● Eased implementation of hardware independent software functionality by using generic interface catalogues. ● Simplifies the model-based development and allows the use of standardized AUTOSAR code generation tools. ● Reusability of modules across several vehicle manufacturers. ● Exchangeability of components from different suppliers.

[126] AUTOSAR, 2008.

[127] Martínez-Fernández, S., Ayala, C. P., Franch, X., & Nakagawa, E. Y. (2015, May 4). *A survey on the benefits and drawbacks of AUTOSAR.* doi: 10.1145/2752489.2752493

Table 6. Survey response summary of perceived benefits and drawbacks of using AUTOSAR by engineers and managers within automotive tool suppliers, vehicle manufacturers, and component suppliers

AUTOSAR Benefits	AUTOSAR Drawbacks and Risks
Standardization Reuse Interoperability Improved Communication Reduced Development Costs Knowledge Repository Reduced Time to Market Reduced Maintenance Costs Best Practices Enhanced Quality Increased Productivity Risk Reduction Mission and Strategy Reputation	Complexity Initial Investment Learning Curve Term Confusion Abstractness Dependency Inefficient Instantiation Bad Documentation

2.4.5.2 Automotive Grade Linux

Open Source Software

The future of proprietary versus open source software onboard vehicles has become a prominent topic in the automotive industry.

Vehicle manufacturers, such as Tesla, have developed large amounts of software under Linux open source licensing, and are required to publish much of their internally developed software under the terms of the open licensing agreement. [128]

While competition for share of the vehicle software platform drives many producers to create proprietary intellectual property (IP), opposing requirements to lower cost and standardize software architectures has led suppliers to use open source components. This in turn affects how suppliers' business models and views of proprietary rights are evolving, affecting the value of the automotive software platform in terms of IP and *data*.

Beyond the need for standardization, one of the key motivations given by AGL collaborators for contributing to the open-source code base concerns the belief that the value of the software is not related to IP, but rather the ability to keep and control data that could otherwise be captured by third-party applications and smartphone interfaces.[129] Collaboration on the open-source platform allows vehicle manufacturers to leverage collective resources in order to compete against emerging In-Vehicle-Infotainment and Telematics offerings by "non-traditional" suppliers such as Apple (CarPlay) and Google (Android Auto).

---

[128] Vaughan-Nichols, S. (2018, May 30). *Tesla starts to release its cars' open-source Linux software code.* ZDNet. http://www.zdnet.com/article/tesla-starts-to-release-its-cars-open-source-linux-software-code/

[129] Tajitsu, N. (2017, May 31). *Toyota uses open-source software in new approach to in-car tech* [Restricted web page]. Automotive News. www.autonews.com/article/20170531/OEM04/170539963/toyota-uses-open-source-software-in-new-approach-to-in-car-tech

The evolution of open-source software onboard vehicles has the potential to significantly impact software lifecycle practices. As described elsewhere in this document, ISO 26262 processes and software product liability are two leading contributors to the overall cost of critical software development, and both are significantly impacted by the use of open-source safety software.

AGL Software Defined Connected Car Architecture

There is evidence that an increasing number of vehicle manufacturers and suppliers are looking at open-source software for a wide range of automotive applications.[130] [131] [132]

AGL members comprise 120+ component makers, software developers, and vehicle manufacturers, including Jaguar Land Rover, Mazda, Suzuki, Honda, Nissan, Ford, Toyota, and Daimler.

Linux is attractive due to the industry's desire to drive interoperability and standardization around common backbone components (e.g., transport layer, security framework, network interfaces, audio control interfaces), where there is little added value or possibility for brand differentiation by "reinventing the wheel."

As described by Vaughan-Nichols, as vehicle manufacturers develop more and more open-source content, including "Autopilot" and other critical features, it is likely that open-source platforms will quickly gain inertia as they have in cloud, industrial, and desktop computing applications.

While AGL is still a relatively new platform architecture, the AGL roadmap plans to address all software on the vehicle:

> *"Although initially focused on infotainment, AGL is the only organization planning to address all software in the vehicle: infotainment, instrument cluster, heads-up-display, telematics/connected car, advanced driver assistance systems, functional safety and autonomous driving."*[133]

2.4.5.3 Communications Bus Standards

The evolution of automotive E/E architectures has led to increasingly complex distributed systems with demanding requirements for determinism,[134] reduced cycle times, and increased bandwidth. Modern vehicles may integrate up to 150 ECUs on several Communications Bus Networks. Busses are specified based on requirements for safety, performance/bandwidth, and ECU application within the subdomain (see Table 7).[135]

---

[130] Vaughan-Nichols, 2018.

[131] Holmes, F. (2018, October 8). *Auto industry's thirst for software is quenched by open source*. Automotive World. www.automotiveworld.com/articles/auto-industrys-thirst-for-software-is-quenched-by-open-source/

[132] Cauchy, D. (2018, September 5). *How open source is transforming the automotive industry* [Web page]. The Linux Foundation.

[133] The Linux Foundation. (2016). *About automotive grade Linux* [Web page]. www.automotivelinux.org/about

[134] Determinism describes the predictability and repeatability of a software component when generating output from a set of inputs.

[135] Keskin, U. (2009). In-vehicle communication networks: A literature survey. *Computer Science Reports, Vol. 0910*. Technische Universiteit Eindhoven.

Table 7. Automotive communications bus classifications and correlated applications and performance requirements

	Powertrain	Chassis (Active Safety)[136]	Body	Telematics	Passive Safety[137]
Program Size	2MB	4.5MB	2.5MB	100MB	1.5MB
Number of ECU	3-5	6-10	14-30	4-12	11-12
Bandwidth	500 Kb/s	500 Kb/s	100 Kb/s	200 Mb/s	10 Mb/s
Cycle Time	10ms - 10s	10ms - 10s	50ms - 2s	20ms - 5s	50ms
Safety Requirements	high	high	low	low	very high
Bus Type	Class C	Class C	Class A	Class D	Class D
			Class B		

SAE has classified automotive bus[138] applications into classes A, B, C, and D, in increasing order of criticality for real-time and dependability constraints.[139]

- **Class A** denotes low-speed networks with data rates <10 kb/s, mostly in the body domain.
- **Class B** networks operate at data rates between 10 and 125 kb/s, are used for general information exchange and body domain networks.
- **Class C** networks operate from 125 to 1,000 kb/s and are used in powertrain and chassis domains.
- **Class D** networks require high-speed communication data rates up to or higher than 1 Mb/s and are mainly used for telematics and x-by-wire applications.

2.4.6 Common Processes and Practices

2.4.6.1 The V – Cycle

The V-Cycle, shown in Figure 2, is the dominant software development lifecycle methodology for safety and control systems in the automotive industry. It is a requirements-driven methodology. As described above, it is the foundation of many consensus standards that are used by the automotive industry for safety, enterprise process development, and functional and

[136] E/E Active Safety systems are responsible for avoiding crashes, and are always "active." Examples include electronic stability control, automated emergency braking, and lane keeping assistance.

[137] E/E Passive Safety Systems remain passive until needed and act to lessen or prevent harm in the event of a crash, – for example, air bags.

[138] A bus is a special form of a communication network in which all of the various devices in the network are connected to a single cable or line.

[139] SAE International. (2006, September 12). Class A application/definition, J2057/1_200609 [Web page]. www.sae.org/standards/content/j2057/1_200609/

performance design (including ASPICE, ISO 26262, and AUTOSAR SPEM).[140] The V-Cycle methodology has become highly specialized in automotive applications, where it tightly integrates hardware (ECU) and software development practices through component, unit, system and vehicle development and test.

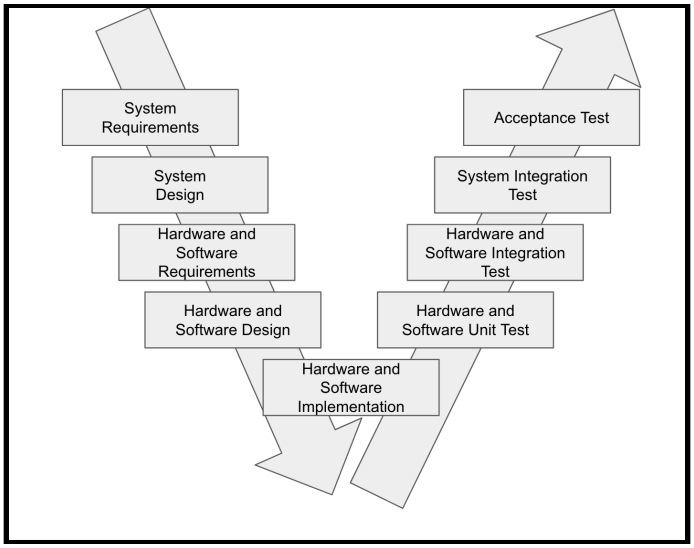

Figure 2. A simplified representation of the V-Cycle SDLC

The history of the systems engineering V-Cycle dates back to the early 1990s when it was developed as an extension to the waterfall SDLC model, with steps for software validation and verification added to the waterfall, forming the "right side of the V."[141] Requirements are decomposed and flow down the left side of the V-Cycle through the development process. Verification activities are typically integrated through the right side of the V-Cycle, including unit, integration, and acceptance tests. Validation is typically considered to occur at the upper-right part of the V-Cycle.

In the automotive V-Cycle, hardware and software development practices are tightly coupled in accordance with the specialized nature of computing platforms used onboard vehicles.[142] Over the years, integrated V-Cycles have evolved in complexity as automotive electronics have advanced. Today these individual processes integrate into a highly specialized SDLC that is uniquely tailored to the automotive industry.

2.4.6.2 Simulation-Based Development and "In-the-Loop" Verification and Validation

The V-Cycle is often described in terms of developmental activities on the left and right side of the V. MBD and requirements decomposition occur on the "left side of the V," and design

[140] Munassar, N. M. A., & Govardhan, A. (2010, September). A comparison between five models of software engineering. *IJCSI International Journal of Computer Science Issues, 7*(5). www.ijcsi.org/papers/7-5-94-101.pdf

[141] ReQtest AB. (2016, April 1). *V-Model vs scrum, who wins?* [Web page]. https://reqtest.com/agile-blog/v-model-versus-scrum-who-wins/

[142] Hanselmann, H. (1993). *Hardware-in-the loop simulation as a standard approach for development, customization, and production test of ECU's* (SAE Technical Paper 931953). Society of Automotive Engineers. https://doi.org/10.4271/931953.

verification against the requirements to check if the product was "built right" occur on the "right side of the V." Validation is typically understood to occur on the upper-right side of the V to check if the developer "built the right thing," which would mean that requirements reflected what a customer actually wanted or needed. In-the-loop testing is a common validation and verification strategy used in the automotive industry. "In-the-loop" testing methods are used to test software functions against simulations of the required stimulus and loads, and are broken into Model, Hardware, and Software in the loop (MIL, HIL, SIL) tests depending on whether the device in the loop is a model, a piece of hardware (ECU, sensor, actuator), or code (e.g., "hand coded" or code generated by an ACG process).[143]

MIL/HIL/SIL are called "simulation" because the behavior of the loads, sensors, and stimulus are based on models of the dynamic systems. Figure 3 showed that there are dependencies between blocks in the feedback control system (Closed Loop Control, Plant, Actuators). The behavior of blocks in the feedback loop are dependent upon the respective inputs (W,U,Y,X,R) and dynamic behavior of the subsystem under control and of various actuators and sensors.

In order to perform unit, integration, and system-level software tests, the block under development, or unit under test, requires representative W, R, and U IO stimulus and loads in order to properly exercise software functions. Simulations representing the external system behavior (e.g., internal combustion engine, the environment, chassis dynamics, actuator dynamics) drive MIL/HIL/SIL interfaces (W, R, and U -in hardware and software).[144] The success or failure of MIL/HIL/SIL strategies is highly dependent on availability and quality (or "fidelity") of models used in the simulation. Tests that cannot be performed in simulators are often required to be performed on more expensive dynamometer test stands or in road tests. Several vehicle manufacturers have introduced the term "digital twin" to refer to virtual models and environments for developing, validating, and verifying their hardware and software designs.[145] Aircraft and vehicle manufacturers and suppliers are increasingly able to place models of hardware and software that comprise their products into complete virtual worlds where virtual tests can be conducted.

[143] This differentiates between "model" and "code" where the code is typically the output of an ACG process or handwritten and optimized for a specific target implementation, MoC, etc. MIL and SIL processes are designed to incorporate "models" or highly optimized "code" depending on the requirements of the test.

[144] King, P. J., & Copp, D. G. (2006, February 1). Hardware in the loop for automotive vehicle control systems development and testing. *Measurement and Control, 39*(1). http://doi.org/10.1177/002029400603900103

[145] O'Heron, P. J. & Chown, W. (2017, September 18-21). *Aerospace product engineering & verification:: The digital twin* [PowerPoint]. Global Product Data Interoperability Summit 2017, Los Angeles, CA. https://gpdisonline.com/wp-content/uploads/2017/11/Siemens-OHERON_Chown-DigitalTwin-MBSE-Open_9_14_2017.pdf

2.4.7 Tools and Implementations

Automotive software development tool suppliers offer compliance packages for various architectural, design, and developmental consensus standards such as MISRA, AUTOSAR, ISO 26262, and SPEM.

There are several integrated toolchains/methods on the market that map to specific models of computation and are designed to support the automotive V-Cycle. The following are examples of tools used in the automotive industry.

- **IBM Rational** is a visual construction and simulation platform incorporating simulation-based testing, requirements engineering tools, and model-driven systems development (MDSD).
- **EAST-ADL** is an architecture-descriptive language (ADL) for automotive embedded systems developed in several European research projects.
- **Mentor AUTOSAR** is a family of AUTOSAR enabled products based on Mentor Graphics Vehicle Systems Architect - a systems design tool for AUTOSAR-based systems.
- **The MathWorks MATLAB/Simulink/Stateflow** is a graphical programming environment for modeling, simulating, and analyzing multi-domain dynamical systems. Used in Model-Based Design.
- **dSPACE ControlDesk, SystemDesk, TargetLink** is a toolchain that implements the V-Cycle in hardware and software through an MDB process.

2.4.8 Application of MBD to Automotive ECU Software Development

For automotive mechatronic systems, MBD is based on underlying control theory, block diagrams, and transfer functions that describe the relationships between components.

Figure 3 through Figure 5 show a system-level view of a generic feedback control block diagram. The blocks represent different transfer functions and mathematical models of physics and dynamics for each element of the system.[146] While a closed-loop feedback control system can be achieved without software, either mechanically or only with electrical circuits and hardware, the information examines a system that does include software. Explanatory notes for Figure 3 through Figure 5 are as follows:

- The closed-loop control block represents the ECU. Automotive ECUs often incorporate "look up" tables to improve control loop speed by elimination of complex math and transfer functions running on the ECU processor.
- Actuators represent different drives, motors, and machine components needed to exert control over the subsystem (control systems engineering sometimes refer to this as the plant, the combination of process and actuator).
- The subsystem represents the physical system under control (e.g., internal combustion engine and transmission; braking system; or steering system).

[146] Phillips, C. L., & Parr, J. M. (2010). *Feedback control systems*, 5th edition. Prentice Hall..

- Sensors represent different instruments that provide dynamic data to the closed loop control in order to calculate command values for the actuators (e.g., engine speed sensor).
- The signals (W, U, Y, X, R) represent the flow of information between blocks.
- The system is called a feedback control system because the output of the system, R, is fed back into the controller, allowing the control algorithm to correct for error (R - W), in order to command the actuators with signal U.

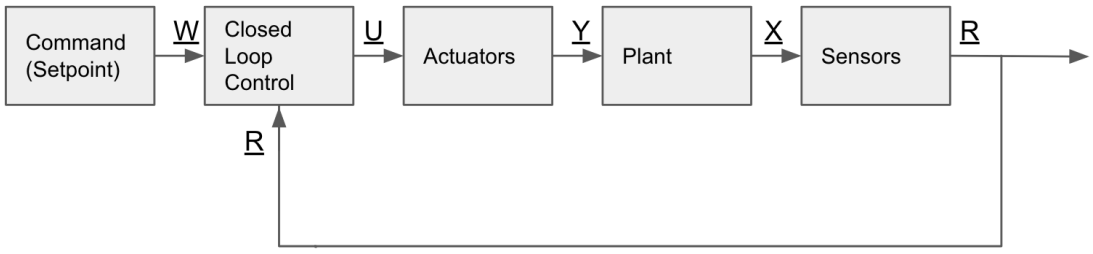

Figure 3. Closed loop feedback control diagrams are used by control system engineers to design automotive control systems in mechatronics applications

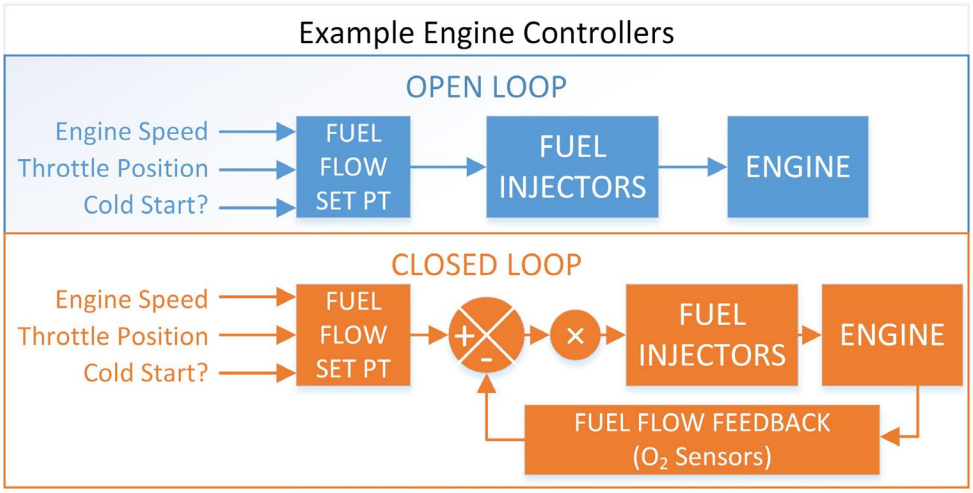

Figure 4. Simplified "Open Loop" (without feedback control) and "Closed Loop" (with feedback control) diagrams for fuel control onboard an IC engine

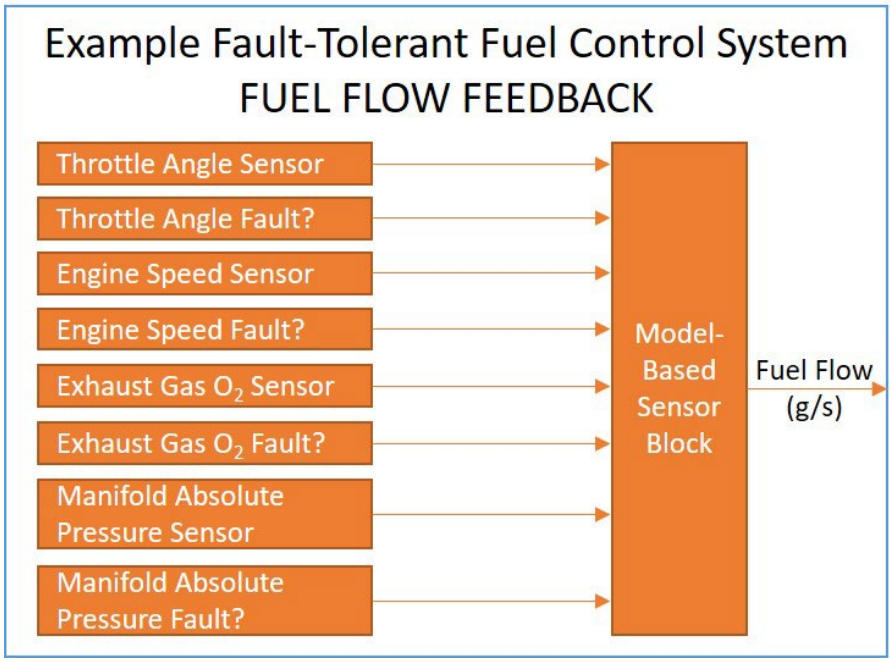

Figure 5. Simplified "Fuel Flow Feedback" diagram that considers several inputs, including fault conditions, to calculate a single fuel flow sensor signal for an IC engine. A fault-tolerant design would allow for a fuel flow signal to remain usable despite one or more sensor faults

Several aspects of the above block diagram notation make it suitable for use in model-based software design.

- The closed-loop control, subsystem, underlying models, and transfer functions can be represented using similar symbolic notation. This allows MBD process to be used for the development of control algorithms for MBD/ACG and also for development of subsystem, actuator, and sensor simulation models for use with "in-the-loop" validation and verification (V&V) methods.
- Blocks and signals correspond to underlying software functions, arguments, and return values.[147]
- The same graphical model may be used for different MoC during automatic code generation (e.g., continuous, discrete).

Control system engineers often create models of nested blocks within blocks to define the behavior of each feedback control element. For example, an engine control model may contain thousands of blocks. Since models represent the dynamics of the control system, actuators and sensors, it is possible to test a control theory against a mathematical representation (simulation) of the actuators and sensors in software without introducing any physical hardware into the system.

(See Section 2.4.6.2 "Simulation Based Development and "In-the-Loop" Validation and Verification.")

[147] Mbihi, J. (2018). *Analog automation and digital feedback control techniques*. Wiley.

The introduction of MBD opened the door to affordable simulation based testing of automotive ECUs, reducing a need for costly field and test track trials. Moreover, simulation based development (SBD) allows for automatic software testing against model-based requirements, diminishing the need for tedious, error prone verification against written requirements. MBD and SBD describe practices that are very similar, to a point that the terms are sometimes used interchangeably. When the MBD approach to modeling a subsystem is used for development of ECU software, including actuators, sensors, and even communications busses and outside influences (the environment), the technique is called simulation based development (SBD).[148]

Simulation technology has accelerated E/E development and compressed design cycles by reducing the need for in-vehicle testing. Simulations have also enabled component reuse through development of libraries of parameterized and able-to-be-calibrated subsystem, actuator, and sensor models.

2.5 Comparison to Approaches in Other Industries

To better understand challenges faced by the automotive industry related to software build, test, and maintenance, particularly with regard to critical systems, it is beneficial to benchmark practices against other comparable industries. The next chapter introduces a comparative framework that shows one possible way to compare the automotive industry with other industries.

To use the framework, it is important to understand different lifecycle practices and why they are used. This is apparent in Table 8, which demonstrates that commercial aircraft and automotive software share many similarities with regard to technical (functional and performance) requirements. Looking further into the table and comparing business, risk, and certification models and the overall safety approach, the data show differing requirements that lead to wide-ranging and different requirements in SDLC practices. It is apparent from the table that automotive software complexity, as measured by LoC, has surpassed that of aviation software. Several factors help to explain this trend:

- Consumer-facing systems in cars, such as infotainment, navigation and comfort features, which are software controlled and interact with yet other software driven subsystems, are primarily driving the increase in automotive software LoC. OEMs offer every conceivable permutation of features to the market, resulting in a massive code base even though some features are never sold to consumers.
- Automotive OEMs are also using more off-the-shelf components rather than tailored software developed from scratch, leading to more lines of code than is strictly necessary when the components are combined.[149, 150]

[148] Chrisofakis, E., Junghanns, A., Kehrer, C., & Rink, A. (2011, March 20-22). *Simulation-based development of automotive control software with Modelica*. 8th International Modelica Conference, Technical University, Dresden, Germany.
[149] Edelstein, 2015.
[150] Ibid.

- In contrast, standards require that aviation code is inspected to be free of dead code, i.e., code that is not reachable or is never used. Hence, the aircraft industry is motivated to reduce software complexity, which has resulted in a *decrease* in the LoC in that industry over the last ten years.[151] For example, Boeing reportedly made significant cuts in the amount of code used in the 787 Dreamliner compared to previous airliners.[152]

To use the framework, it is a must to identify and differentiate global requirements placed upon the SDLC practices of the proxy industries versus the practices used in automotive industries.

As the vehicle has become increasingly specialized and complex, its associated SDLC processes and methods have become unique, making it more difficult to compare consensus standards and practices between the automotive domain and other comparable domains. Some challenges in comparing the automotive domain with other domains include:

- Certain domains such as consumer electronics and medical devices are highly diversified and heterogeneous and must be segmented before comparisons may be made. For example, within the range of applications classified as "consumer electronics," the SDLC practices and associated global, functional and performance requirements for televisions and smartphones differ with respect to complexity, models of computation, criticality, etc.
- The SDLC practices in mature industries are hierarchical, consisting of layers of processes and sub-processes. In mature industries where lifecycle practices share some similarities (e.g., automotive and aerospace) the framework may require comparisons deep within the SDLC process hierarchy in order to understand the differences between the SDLC processes.
- In certain other domains, levels of accelerated growth, uniqueness, and complexity are comparable to the automotive industry, leading to a need for deep industry analysis in order to understand and establish functional, performance, and global requirements and comparative framework practices. If, for example, the comparative framework is used to compare MoC between vehicle and smartphone software practices, the framework shows that critical software in smartphones relies on emerging MoCs. These include models of trust and models of location that are not prevalent in automotive software.[153]

Table 8 provides an example of a comparative analysis between automotive onboard software and civil and commercial aircraft onboard software using the global requirements described above and constructed using a portion of the framework.

[151] Saracco, 2016.
[152] Edelstein, 2015.
[153] National Research Council. (2001). *Embedded, everywhere: A research agenda for networked systems of embedded computers*. The National Academies Press. https://doi.org/10.17226/10193

Table 8. A comparison of global requirements and constraints for automotive onboard software versus commercial aircraft onboard software, based on a literature review

	Automotive Onboard	Aerospace Onboard (Civil and Commercial Aircraft)
Technical Model[154]		
MoC[155]	Real Time, Distributed, Finite State Machines, Dataflow Process Networks, Discrete Event, Synchronous Reactive (SR)	Real Time, Distributed, Finite State Machines, Dataflow Process Networks, Discrete Event, SR
Communications, Busses	Class A, B, C, D (e.g. CAN, LIN, FlexRay, MOST)[156]	Class A,B,C,D (e.g. ARINC 429, ARINC 664, AFDX, CAN)
Reference Architectures	AUTOSAR/OSEK	ARINC 653
Published Safety Standard & Model[157] [158]	Functional Safety	System Safety Engineering
Types of Critical Systems	Real Time Systems	Real Time Systems
	Dependable Systems	Dependable Systems
	Functional Safety Engineered Systems	System Safety Engineered Systems
	Secure Systems	Secure Systems
Fault Tolerance[159]	Fault Avoidance, Human-Machine Interface, Fail Safe	Fault Avoidance, HMI, Fail Operational
Safety Characteristics		
Fatalities/Billion Passenger Miles[160]	7.8 (22,697 fatalities/year) Passenger Vehicles[161]	0.038 (21 fatalities/year) Air Carrier 4.11 (42 fatalities/year) Commuter and Air Taxi
	~$22 Billion/year (recall costs)	~ $1.5 Billion/year (warranty claims)

[154] Ramsey, J. W. (2005, June 1*). Boeing 787: Integration's next step*. Aviation Today. www.aviationtoday.com/2005/06/01/boeing-787-integrations-next-step/

[155] Lee, 2011.

[156] Malik, H., Avatefipour, O., Hafeez, A., & Raj,. P. (2017, April 4-6). *Comparative study of CAN-bus and FlexRay protocols for in-vehicle communication.* SAE World Congress Experience WCX 17, Detroit, MI. doi: 10.4271/2017-01-0017 www.researchgate.net/publication/315781234_Comparative_Study_of_CAN-Bus_and_FlexRay_Protocols_for_In-Vehicle_Communication

[157] International Organization for Standardization. (2018, December). *Road vehicles-Functional safety, ISO 26262-2:2018*.

[158] Society of Automotive Engineers. (1996, December 1). *Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*, (SAE No. ARP4761).

[159] Dubrova, 2013.

[160] Waycaster, G. C., Matsumura, T., Bilotkach, V., Haftka, R. T., & Kim, N. H. (2017, January 17). Review of regulatory emphasis on transportation safety in the United States, 2002–2009: Public versus private modes; *Risk Analysis, 38*(5). doi: 10.1111/risa.12693

[161] National Center for Statistics and Analysis. (2020, October). *Passenger vehicles: 2018 data* (Traffic Safety Facts. Report No. DOT HS 812 962). National Highway Traffic Safety Administration. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812962

	Automotive Onboard	Aerospace Onboard (Civil and Commercial Aircraft)
SDLC Process Standards		
Software Safety Guidelines	ISO 26262	DO-178C
Hardware Safety Guidelines	ISO 26262	DO-254
Hardware Software Certification/Compliance Certification Authority	Supplier/vehicle manufacturer, Self-Certification	FAA, Type Certification
Business Model		
Design Philosophy	Performance, Comfort, Safety, Consumer Features	Safety, Reliability, Performance, Maintainability
Product Life[162] [163] [164]	150,000 miles/10 Years	40,000 Pressurization Cycles (Take-Off and Landing Cycles)//25 Years
		150,000 Flight Hours
Utilization[165]	Individual Use: 1-2 hours/day	Shared/Fleet Use: 9-10 hours/day
Cost Per Passenger-Mile[166]	~$1.00 (Passenger Vehicle)	~$0.10 (Air Carrier)
Operator	Owner Operator Driver State Driver's License	Commercial Operator Pilot Commercial Pilot License
Business Model - Maintenance		
Responsibility	Owner/Operator/Individual	Government/Fleet
Performed By	Variable/vehicle manufacturer/ Independent	Certified Maintenance Technicians
Software Complexity	High - Limited in complexity by vehicle manufacturer risk model, driven by volume, variants	High - Limited in complexity by cost of certification, driven by demand for new capabilities
	~15 - 100 MLoC[167]	~5 - 15 MLoC[168] [169] [170]

[162] Weiss, M. A., Heywood, J. B., Drake, E. M., Schafer, A., & AuYeung, F. F.. (2000, October). *On the road in 2020: A life-cycle analysis of new automobile technologies* (Energy Laboratory Report No. MIT EL 00-003). Energy Laboratory, Massachusetts Institute of Technology. https://web.mit.edu/sloan-auto-lab/research/beforeh2/files/weiss_otr2020.pdf

[163] Airline Data Project. (n.d.). *Aircraft and related* [Web page]. Massachusetts Institute of Technology. http://web.mit.edu/airlinedata/www/Aircraft&Related.html

[164] Berla Corporation. (n.d.). [Untitled web page and portal]. https://berla.co/average-us-vehicle-lifespan/

[165] Morris, D. Z. (2016, March 16). *Today's cars are parked 95% of the time* [Web page]. Fortune. http:/fortune.com/2016/03/13/cars-parked-95-percent-of-time/

[166] Condon, P. M., & Dow, K. (2009, November). A cost comparison of transportation modes. *Foundational Research Bulletin, 7*.

[167] Edelstein, 2015.

[168] Redman et al., 2010.

[169] Nexteer Automotive, 2018.

[170] Norris & Wagner, 2009.

2.6 Future Challenges

Changing technology and market demands can be at odds with suppliers' ability to manage cost, complexity, and safety. Functional and performance requirements are derived from these changing constraints and drive the SDLC requirements. Market factors can often change more rapidly than vehicle manufacturers are able to modify developmental practices. For instance, the automotive industry is still adjusting to changes in current automotive developmental practices stemming from ISO 26262, but it must now also look forward to new software challenges introduced by ADS.

It will be important to consider how these changes will impact critical systems and the automotive SDLC. For instance, some questions raised by these changes include:

- How will changes in requirements impact the functional and performance requirements required of software?
- What are the impacts on functional and performance requirements of critical systems (reliability, security, safety and real-time)?
- What are the impacts on complexity?

The automotive industry is changing, particularly with regard to technology and software onboard vehicles. Existing, highly evolved V&V practices for automotive control systems that incorporate MBD, ACG, MIL/HIL/SIL, and dynamometer testing are designed to validate deterministic control systems against defined requirements by producing repeatable test results.

Moving forward, the following are examples of what may continue to occur (see Table 4).

- Industry struggles with an "affordability limit" and "wall of complexity."[171]
- Growing and progressively varying software content and increased pressure to improve software reliability.
- Emerging technology driving new models of computation, system criticality, and architectures.
- Vehicles as subsystems within larger critical systems and infrastructure, yielding new risk models.
- Changing models for vehicle ownership, shared use, commercial management and maintenance.
- Increasing use of new technology and an increasing number of software systems that are safety critical.
- Incorporation of probabilistic and non-deterministic subsystems into safety-critical systems (e.g., vision subsystems for driving automation systems).
- New consensus standards as emerging technologies are introduced.
- Changing risk models and changing fault management strategies.

[171] This refers to the "affordability limit" cited by Redman and "complexity limit" (wall of complexity) cited by Davey, where the authors identify diminishing returns for onboard software value versus developmental costs with limits defined by what the market will pay for the finished product.

- Increasing need to ensure that legacy automotive and infrastructure systems are compatible with emerging systems and technology.
- An evolution in lifecycle practices and increasing use of methodologies other than, or in addition to, the V-Cycle.

3 Automotive Software Evolution Framework

3.1 Introduction

Due to automotive electrical and electronic systems' rapid evolution over the past three decades, the automotive industry is facing a host of software-related challenges that were absent during the industry's first century of evolution.[172] As a result of rapid software proliferation in vehicles, a number of the authors and documents cited throughout this work note that software processes and methods are approaching the capability limits of what these processes and methods are able to handle in terms of managing complexity and criticality, emerging architectures, automation, affordability, and adaptability.[173]

Challenges faced by the industry are compounded by the fact that the vehicle could become connected to infrastructure, smart highways, etc. In the future, it is likely that software onboard a vehicle may need to be viewed from the larger Internet of Things and connected infrastructure.

For these reasons, capturing the essence of the evolution of the build, test, and validation activities associated with the development, production, and maintenance of automotive software is complex and challenging. The scope of onboard vehicle software, including software that is connected with off-board systems (e.g., OTA updates, V2X), is large and rapidly changing. This study established a framework to help understand and categorize the challenges facing automotive software development. One fundamental goal of the study is to use the framework to identify unique characteristics of automotive software used in modern motor vehicles.

The framework described herein is derived from a matrix of "best-fit" evidence compiled by the authors. The research questions assigned to the study formed the basis for research and synthesis. An initial step was examining these questions through the lens of a number of change factors over time, i.e., drivers for change arising from either the external or internal environment of the industry. The goal of the framework is to develop a structured approach that can be used to compare and contrast automotive software development practices over time, as well as to other transportation sectors and other industry domains.

One way of interpreting the framework is to consider it an aggregation of many consensus standards, practices, processes, and tooling into a representation that captures the state of practice across the industry for development, production, and maintenance of automotive software.

The actual practices used across the industry are diverse and are the product of an evolutionary amalgamation of thousands of published consensus standards, internally developed processes, tools, practices, nomenclatures, architectures, and taxonomies practiced across hundreds of original equipment manufacturers and suppliers.

[172] The introduction of the gasoline-powered automobile is often credited to Karl Benz in 1885. Steam-powered automobiles were introduced as early as 1769. See Eckermann, E. (2001, August 1). *World history of the automobile*. Society of Automotive Engineers.

[173] For further information see Antinyan, 2018; Davey, 2013.

3.2 First-Pass Checklist

The framework is a checklist of major development process steps and technologies used in the creation of automotive software. The first-pass checklist came from analyzing two commonly used consensus standards, ASPICE and ISO 26262. The ASPICE consensus standard is a process assessment and capability model, and is older and more generic than ISO 26262, which is focused on safety.

The ISO 26262 standard was first published in 2011 and first revised in December 2018. ISO 26262 adds functional safety to automotive product development, including software. Functional safety as a general concept has been employed in the automotive industry for many decades, as safety has been a high priority for automotive companies. However, the more formal definition and standardization of functional safety is relatively new to the industry. At the systems level, ISO 26262 is sometimes implemented within software companies to define systems that are functionally safe or not. However, as of early 2019, companies were still in the process of integrating formal functional safety activities into their software development processes.

It has become clear that a first pass checklist is not ideal for being able to differentiate between companies and industries. For example, the consensus standards do not care how the software is implemented. The consensus standards are more solution-neutral, and they care only whether requirements are met and whether the development is done with appropriate quality control in place. For example, the requirement may be for the vehicle to respond within a specified time with a defined performance characteristic. The standard is not concerned if this is accomplished with a simple scheduler, a minimal RTOS, or a desktop-like RTOS. However, from a historical, current practice, and future projections perspective, the RTOS is a key differentiator between systems within a given vehicle and also between different industries.

Table 9 shows the overarching themes of the framework.

While the “Traditional Automotive - Current” era continues from 2002 to today, the automotive industry is undergoing the next major shift, to the “Projected Automotive (i.e., Automated)” era. This next era is centered on ADSs. While ADSs are being tested on roads today, they are prototypes and experimental rather than mass production vehicles. Because ADSs are undergoing rapid change and development/testing, consensus standards around software practices either do not exist or are only in discussion stages, and not all the framework fields can be filled in at this time.

The framework is refined and fully developed with additional data in the following sections:

Table 9. Evolution of Automotive Software Development Process

Software Development Process Step	Historical	Current	Projected (e.g., automated)
Documentation management plan	Ad Hoc	Ad Hoc	Ad Hoc
Continuous Improvement	Functionality: Yes Process: Ad hoc	Functionality: Yes Process: Recommended by ASPICE/ISO 26262, but implementation more ad hoc	Functionality: Yes Process: Too soon to know
Change Management	Minimal	Becoming more standard (supporting tools)	Common
Configuration Management	Some	Common (supporting tools)	Common
Trace all artifacts from initial requirements through final test results	No	On the rise with tool support, but still not universally implemented	Comprehensive trace of all artifacts
Distributed Development (i.e. Suppliers)	No → Common	Common	Common

3.3 Taxonomy

The framework incorporates thematic, inductive synthesis in order to address guiding research questions.[174] [175] [176] The source material and process for developing the material is shown in Figure 6 and the decomposition process of this source material; is shown in Figure 7. Iterations of the process yield "frames" that integrate new evidence in response to research questions, as shown in Figure 6.

[174] The numbers in parentheses refer to the current and future framework versions. The current version of the framework is Version 1.0 ("Frame 1").

[175] ISO 26262:2018, Road vehicles – Functional safety, was released during the research phase of this project. References used throughout this report refer to ISO 26262:2011 unless explicitly specified.

[176] ISO/SAE CD 21434, Road Vehicles -- Cybersecurity engineering, was unreleased during the research phase of this project. References to ISO 21434 refer to pre-release material and are subject to change. Concepts from ISO/SAE CD 21434 could be incorporated into future iterations of the framework.

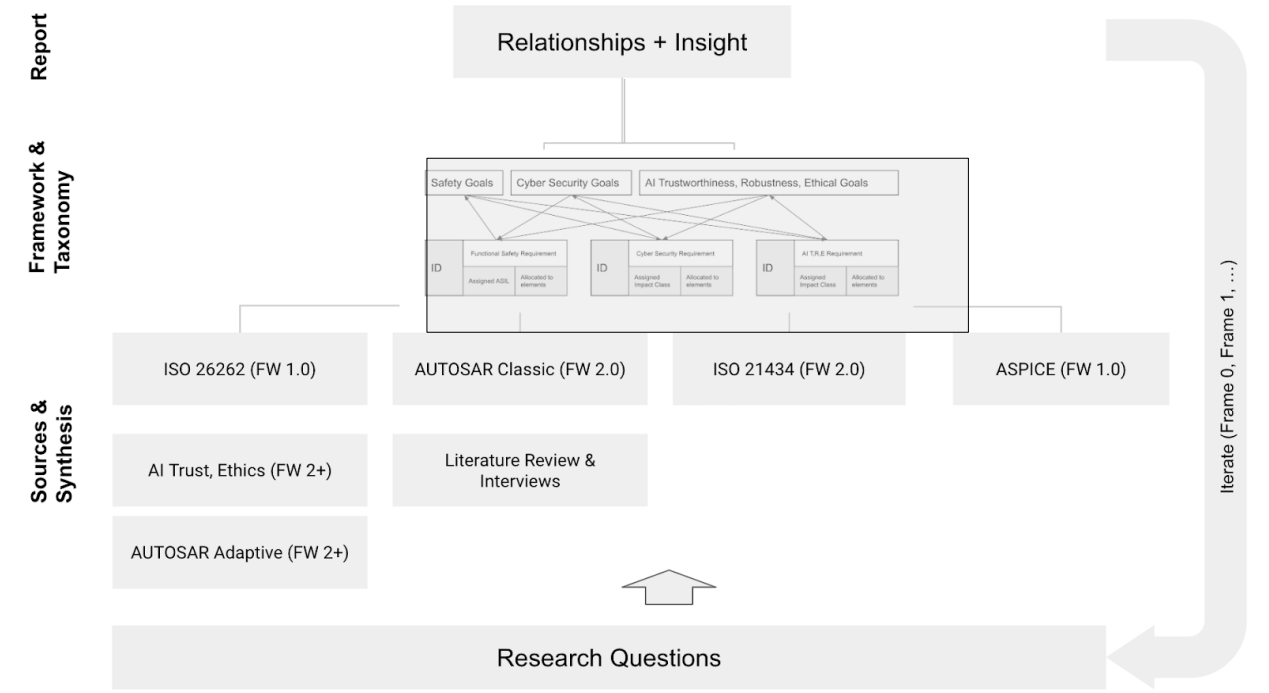

Figure 6. Summary of framework sources and roadmap by framework (FW) version. Figure 9 provides a closer view of the Framework & Taxonomy section.

In order to synthesize the common threads of change factors that have affected evolution of control systems, the research team investigated industry, market, government, and technical requirements and influences on automotive software development lifecycle practices.

The resulting framework taxonomy from Figure 6, shown in Figure 7, consists of four levels[177] (with an example slice from the research on the right):

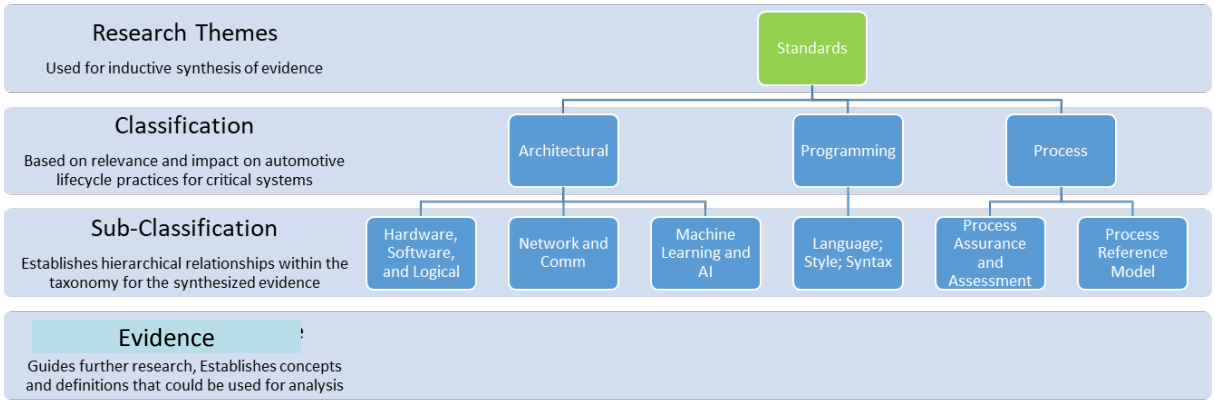

Figure 7. Framework taxonomy levels

[177] One of the fundamental goals of the study is to understand what is unique about modern motor vehicle software. For this reason, the study focuses primarily on *critical* software and the uniqueness of practices associated with automotive critical software development. The framework taxonomy identifies and establishes various modes of criticality and the types of control systems used in automobiles which may be used to identify and characterize the specialized software processes used for the development of related software.

The framework was established by using a conceptual model.[178] [179] "Frame 0" of the comparative framework was established using *a priori* consensus standards. The framework then consists of two components: (1) a categorization of change factors and (2) key elements of the SDLC derived from the *a priori* standard. This approach produces a comprehensive framework that can be used to compare across both change factors and the specific resulting SDLC practices.

ISO 26262 and ASPICE were selected as the *a priori* consensus standards since they closely and directly impact the criticality, process steps, process assurance, and enforcement of lifecycle practices for today's automotive-critical systems. These consensus standards were chosen not only for their relevance to the research questions, but also because they are reliable sources of industry definitions, methods, and relationships that may be used as the basis for taxonomy. Process models, ISO 26262 and ASPICE, were considered most relevant to the automotive industry.

ISO 26262 is a widely accepted standard for functional safety in the automotive industry and provides guidelines for the development of safety critical software.[180] ASPICE provides a process reference model for automotive software development and related management functions. It also provides a process assessment methodology that allows automotive software suppliers to determine the capabilities and maturity level of their software development processes.[181] By incorporating these two consensus standards in the initial framework taxonomy, a baseline is set for classifying the reference processes, concepts, and terminology that establish the foundation for critical software and related automotive development practices.[182]

Framework Scope, Use Cases, and Objectives

The objective of this section is to develop a framework that can be applied to compare automotive software development practices over time and relative to both other transportation sectors and other industries. The SDLC practices used in the development of commercial airplanes, vehicles, nuclear power plants, and consumer electronics share many things in common, including similar (and in many cases, identical) process steps, computing architectures, models of computation, and programming languages. A comparative framework must provide a broad taxonomic basis for a user to establish common definitions between comparative targets. It must also provide a mechanism to reduce the broad taxonomy into smaller, manageable groupings for comparative analysis.

---

[178] A priori within the context of the best-fit framework synthesis research method is defined as knowledge that is known prior to, and independent of the research. The framework begins with a "frame 0 seed frame" based on a priori standards that are known by the research team to be widely used across the automotive industry

[179] The study team was provided a set of high-level "a priori" research questions by a diverse team of automotive engineers, researchers, and transportation professionals. The questions were designed to seed the frame-work with topics and research objectives in order to establish frame 0 and thematic research topics

[180] ISO-26262: 2011, Road vehicles – Functional safety, is an international industry standard for functional safety of electrical and/or electronic systems in production automobiles developed by the International Organization for Standardization

[181] ASPICE: VDA QMC Working Group 13 Automotive SIG. (2015) is a widely recognized reference model for the disciplined evaluation of an organizational unit's processes against a process assessment model.

[182] Frame 0, driven by the a priori research questions and study guidelines, is focused on software practices related to control systems and critical software.

To accomplish these objectives, the framework was developed under the following guidelines:

- It is intended to provide a *comparative* taxonomy for automotive SDLC processes and practices, rather than a *comprehensive* taxonomy.
    - It was agreed at the outset that the research would look across a broad range of themes rather than perform a deep dive on select topics.
- The study is qualitative.
- The best-fit framework is adaptive and driven by research questions and guided by the research team.
- Successive iterations ("frames") may be required in order to provide clarification, detail, and to map relationships in the taxonomy. Subsequent frames will yield more classification layers in the taxonomy. It is possible that the taxonomy will change as automotive technology evolves, and thus the framework may require maintenance or revisions.
- The industry consensus standards that are referenced as framework sources throughout this document are in and of themselves taxonomies. The comparative framework is not intended to reclassify or provide definitions for information that is already published, but rather it is meant to extract the defining characteristics that allow for a comparative analysis.
- The framework must provide a chronological taxonomy in order to reflect the continuously changing nature of supporting consensus standards, methods, and definitions. Software lifecycle practices across the automotive industry may change considerably during a vehicle product development lifecycle (e.g., four to seven years).

Taxonomy and Comparative Framework

The concepts of "taxonomy" and "framework" are closely related. In order to differentiate between the two, the following definitions have been used for the study:

- The "taxonomies" are the classification schemes that articulates the relationships among factors that influence automotive SDLC practices.
- The "framework" is the underlying structure that allows the taxonomy to be used for comparative analysis.

The framework taxonomy is broken into two separate parts (Figure 8):

- The "taxonomy of process change factors" synthesizes common threads that impact SDLC practices for automotive software.
- The "taxonomy of software development lifecycle practices" synthesizes SDLC practices (e.g., architecture, design, validation, and verification) associated with the development, production, and maintenance of automotive software.

In Figure 8, the taxonomy reduction process allows the large taxonomy to be reduced into a smaller taxonomy comprised of elements of interest for comparative analysis. A taxonomic reduction may be useful for reasons such as (1) interest in conducting a comparative analysis along one research theme (e.g., the consensus standards research theme), or (2) removing subclassifications that are irrelevant to the target industry.

The organization of the rest of this document maps to the four levels of the framework hierarchy, organized consistent with the framework organization shown in Figure 8.

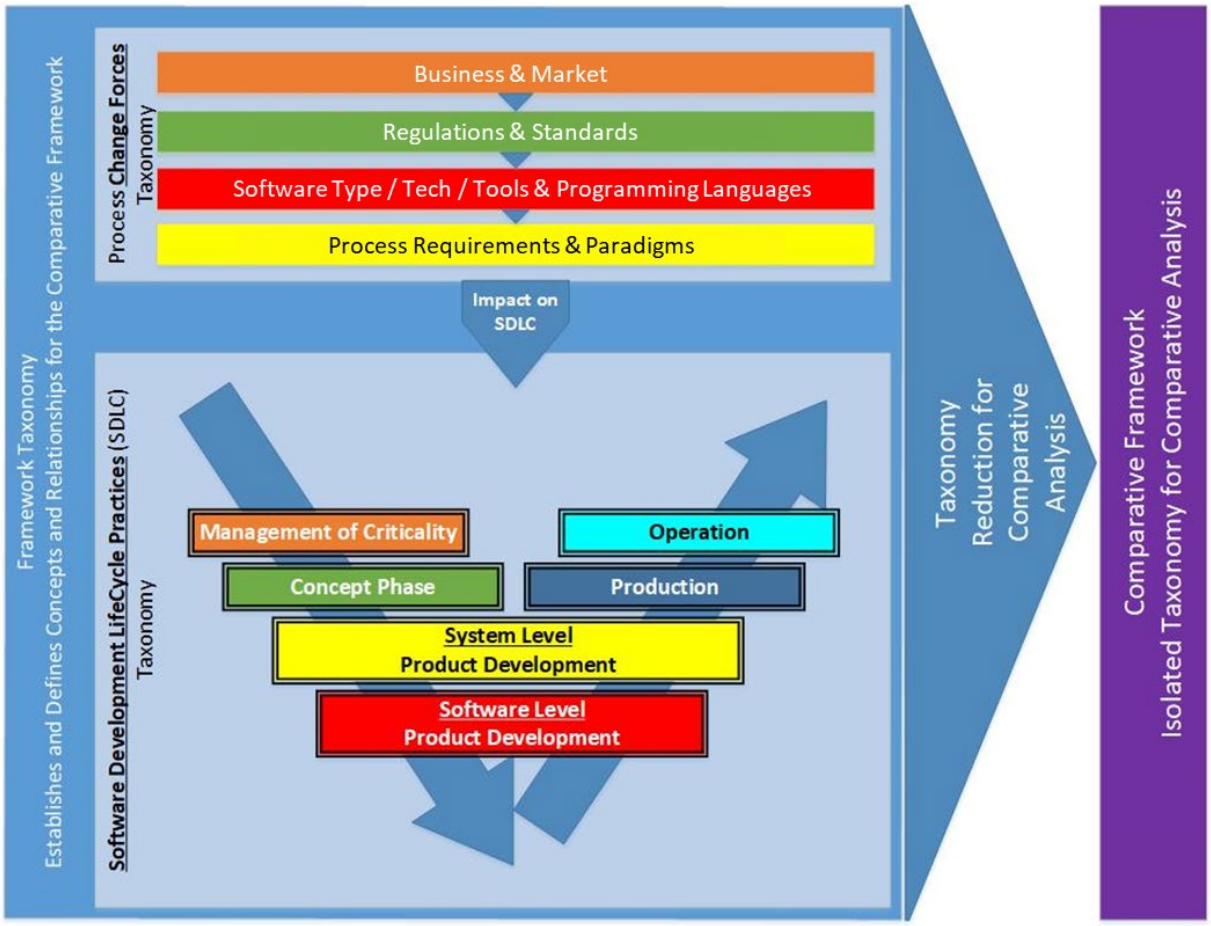

Figure 8. A block representation of the framework taxonomy, comparative framework and the relationship with process change factors. This block diagram also serves as a document map for subsequent sections, tables, and figures.

3.3.1 Taxonomy of Process Change Factors

The research team previously identified unique lifecycle constraints (e.g., non-functional requirements) that are imparted on automotive software development processes. For example, since demand for vehicles is consumer-driven in nature, this market factor results in a lifecycle constraint that promotes ever-increasing product complexity. Not only has the number of vehicle variants supported by vehicle manufacturers increased throughout the last decade, but the number and complexity of electronic features demanded by consumers has also dramatically increased during this time. These electronic features must function properly and coexist with each other on every single unique variant that the vehicle manufacturers produce.

Table 10 and Table 11 provide a chronological view of the framework taxonomy, capturing influential process change factors identified by the study. The research team has divided the timeline into three *eras,* described in this application of the framework as "Generations" (e.g.,

Gen 1, Gen 2, and Gen 3). Each era has defining characteristics based on the elements of the framework taxonomy, as described in Table 10.

Throughout this document, tables capturing the evolutionary aspects of the framework taxonomy will be provided to give a chronological perspective that could be used during a comparative analysis. The tables describe defining characteristics for each generation of the framework.

Table 10. Key high-level characteristics that define each framework generation

Summarized Framework Taxonomy by Generation	
Gen 1 (circa 1950-2002)	Emergence of isolated (e.g., not networked) automotive electrical and electronic systems; emerging E/E-related developmental tools and consensus standards; emergence of E/E-related SDLC processes; emergence of critical software practices for real-time embedded systems and digital control theory.
Gen 2 (circa 2002-2017)	Widespread use and standardization of networked and distributed E/E architectures (e.g., electronic control units); widespread use and standardization of developmental tools and SDLC processes and practices for E/E systems; widespread use and standardization of critical software practices for real-time embedded systems; emerging vehicle and powertrain architectures (e.g., hybrid and electric vehicles) affecting E/E implementation and critical software modalities.
Gen 3 (circa 2017-Present)	Legacy support for distributed and networked real-time embedded systems; emergence of integrated and service-oriented software architectures; integration of features resulting in fewer overall ECUs within the vehicle; emergence of connected vehicle pilots (e.g., V2X) and automation architectures; emergence of developmental tools and software practices for ADS and connected vehicles; emerging critical modalities related to automation, cybersecurity.[183]

[183] The terms "automation" and "autonomy" are both widely used by engineers to describe supervisory control systems (also described as *authority* control systems), and both are included in the taxonomy. SAE J3016defines "automation," and the framework will incorporate the definitions provided by SAE 3016 for levels of driving automation.

Table 11. Evolution of process change factors and lifecycle practices influencing automotive critical E/E system development (fill colors correspond to SDLC taxonomy from Figure 8)

Evolution of Process Change Factors on Automotive Industry by Framework Generation									
Chronological References and Benchmarks Related to E/E Evolution	**Year**	**1950s & 1960s**	**1970s & 1980s**	**1990s**		**2000s**		**2015**	**2017 to present**
	Framework Era	**Gen 1**				**Gen 2**			**Gen 3**
	Vehicle Lines of Code [184]	0	100,000	1,000,000		15,000,000		100,000,000	TBD
	Vehicle ECUs[185]	1		5		15 (2005)	40 (2010)	100	75
	Commoditization of HW: Cost of Computer Memory ($/MB)[186]	$400M	$700K	$30 (early 1990s)	$1 (2000)	$0.12 (2005)	$0.02 (2010)	$0.01	TBD
Framework Subclassification		**Example Characteristics//Unclassified Categories (not exhaustive)**							
Macro Factors		Changing mobility model; changing demographics and geographies; improving propulsion technology; concerns with energy supply and availability; changing maintenance and sustainment; increasing product feedback; evolving consumer requirements; changing supply chain; increasing number of mechatronic and software features; increasing use of automation and connectivity; concerns over environmental impact; concentration on innovation/first to market with intellectual property; increasing commoditization of E/E hardware and components; increasing manufacturability of E/E systems; increasing capability for in field software updates and influence of an end user licensing agreement							
Design Philosophy		Consumer product; cost optimized around performance, comfort; safety and consumer features							TBD
Utilization		Predominantly individual use (< 8% of registered vehicles in the US are fleet), 1-2 hours/day							TBD
Properties of Relevant Standards		Continuously evolving; self-certification by producers (US) or type-certification (Europe)							
Software Complexity		Process complexity; code complexity; architectural complexity; variant complexity; requirements complexity; no accepted consensus standards for measuring complexity.							

[184] Antinyan, 2018.
[185] Davey, 2013.
[186] The cost of memory metric is used as a representation of the overall cost and commoditization of E/E hardware that contributes to the proliferation of automotive E/E systems. For more information and metrics (e.g., microprocessor cost per transistor, microprocessor clock speed, miniaturization of mechanical components).

Year	1950s & 1960s	1970s & 1980s	1990s	2000	2005	2010	2015	2017 to present
Framework Era	**Gen 1**				**Gen 2**			**Gen 3**
Framework Subclassification	**Example Characteristics//Unclassified Categories (not exhaustive)**							
Hardware Standards		Proprietary			Proprietary; AUTomotive Open System Architecture Classic Platform			Proprietary; AUTOSAR Classic; Adaptive AUTOSAR
Software and Logical Standards		Proprietary			AUTOSAR Classic Platform			Automotive Grade Linux; AUTOSAR Classic Platform; Adaptive AUTOSAR
Network (Bus) and Protocol Standards			Controller Area Network Specification: 1991; Local Interconnect Network Specification: ~2003; Media Oriented Systems Transport Specification: ~2008; Time-Triggered Ethernet Specification: ~2008; FlexRay/ISO 17458-1:2013; Time-Synchronous Networking/IEEE 802.1AS: 2018					
Network Class Standards					Class A, B, C, D[187]			
Machine Learning and AI Standards								Emerging computational approaches and architectures; emerging trustworthiness; emerging use cases and applications; emerging foundational consensus standards; emerging system consensus standards
Language, Style, Syntax Standards			Motor Industry Software Reliability Association (C, C++, ACG); AUTOSAR 068;[188] Mathworks Automotive Advisory Board[189]				MISRA (C, C++, ACG); AUTOSAR 068; MAAB; Emerging AUTOSAR C++14;[190] Emerging High Integrity C/C++;[191] Emerging Computer Emergency Response Team;[192] Secure Coding Standards[193]	

[187] For more information, see Hall, E. (2000, February). Internet Core Protocols: The Definitive Guide: Help for Network Administrators [Web page]. O'Reilly Online Catalog. https://web.archive.org/web/20110401192204/ http://oreilly.com/catalog/coreprot/chapter/appb.html

[188] AUTOSAR. (2008). *AUTOSAR methodology. V2.2.1 R3.0 Rev 0001* (Document ID 068). www.autosar.org/fileadmin/user_upload/standards/classic/3-2/AUTOSAR_Methodology.pdf

[189] MathWorks. (n.d.). *MathWorks Advisory Board (MAB) guidelines*.www.mathworks.com/solutions/mab-guidelines.html

[190] AUTOSAR. (2017, March 31). *Guidelines for the use of the C++14 language in critical and safety-related systems* (Document Identification No 839). www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_RS_CPP14Guidelines.pdf

[191] Perforce Software, Inc. (2021). *High integrity C++ is a coding standard developed by experts at PRQA (Now part of Perforce)*. www.perforce.com/resources/qac/high-integrity-cpp-coding-standard

[192] ScienceDirect. (2021). *Computer emergency response team* [Web page]. www.sciencedirect.com/topics/computer-science/computer-emergency-response-team

[193] Schiela, R. (2020, Nov. 18). *SEI CERT coding standards* [Web page]. Carnegie Mellon University Software Engineering Institute. https://wiki.sei.cmu.edu/confluence/display/seccode

Year	1950s & 1960s	1970s & 1980s	1990s	2000	2005	2010	2015	2017 to present
Framework Era		**Gen 1**			**Gen 2**		**Gen 3**	
Framework Subclassification	**Example Characteristics//Unclassified Categories (not exhaustive)**							
Process Assurance and Assessment Standards						ASPICE		TBD
Process Reference Model	Ad-hoc; emerging generic V-Cycle					ASPICE V-Cycle; ISO 26262 V-Cycle; ISO 21434 V-Cycle; emerging agile		TBD
Environmental Regulations and Standards	Motor Vehicle Air Pollution Act (1965); Air Quality Act (1967); Clean Air Act (1963 and 1970); Clean Air Amendments (1990)			California Emissions Standards (Various); Clean Fuels Alternatives; Natural Low Emissions Vehicles; Tier 2 Tailpipe Emissions (2004 - 2009)				One National Program on Federal Preemption of State Fuel Economy Standards (2019)[194]
Programming Languages	Assembly; emerging C				Assembly; C; emerging C++			C; C++; Accelerated Massive Parallelism; SYCL; Open CL; compute unified device architecture; very high definition language
Modes of Criticality		Reliability; emerging real time; emerging safety			Reliability; real time; safety; emerging cybersecurity; emerging mixed criticality			Reliability; real time; safety; mixed criticality onboard computing; integrated cybersecurity; mixed criticality ad-hoc networks; location awareness;
Hardware Architecture		Dedicated (predominantly fixed); emerging standardized			Standardized ECU elements (memory, processing elements, input output, dedicated peripheral and communications bus interfaces)			Adaptive and multi-purpose; high-performance computing; sensor fusion; several and heterogeneous processing units (e.g., microprocessor units, graphics processing unit, data flow processor, field programmable gate array); adaptive peripheral interface ;Ethernet bus backbone
Software Architecture		Dedicated proprietary			Independent subdomain architectures (e.g., body electronics, powertrain, chassis, occupant and pedestrian safety, multimedia, telematics, and human-machine interface); functional/dedicated applications; fixed applications; virtual interfaces (e.g., application layer; Runtime Environment; service layer (e.g., OS, Mode, Diagnostic, Firmware, Memory, communications); ECU abstraction layer, microcontroller abstraction layer)			Emerging Service Oriented Architecture; integrated domain architectures (e.g., integrated ADAS); functional clusters; adaptive applications

[194] Environmental Protection Agency. (n.d.). R *Regulations for greenhouse gas emissions from passenger cars and trucks* [Web page and portal]. /www.epa.gov/regulations-emissions-vehicles-and-engines/regulations-greenhouse-gas-emissions-passenger-cars-and

Year	1950s & 1960s	1970s & 1980s	1990s	2000	2005	2010	2015	2017 to present
Framework Era		**Gen 1**			**Gen 2**		**Gen 3**	
Framework Subclassification	**Example Characteristics//Unclassified Categories (not exhaustive)**							
Fault Management Strategy	Human monitoring and intervention; safety mechanisms; fault avoidance and removal						Human monitoring and intervention; safety mechanisms; fault avoidance and removal; fail safe; fail operational (including redundancy, mitigation)	
Modes of Software Safety Criticality				Emerging functional safety (Automotive Software Safety Integrity Level); "Bottom-Up" approach based on Element out of Context (EooC); bottom-up risk assessment (e.g., failure mode and effects analysis from Society of Automotive Engineers Recommended Practice 1739)			Functional safety; emerging system safety model ("top-down") and top-down safety analysis (e.g., System Theoretic Process Analysis (STPA)); emerging Safety of the Intended Functionality	
OS Technology		Proprietary			Open Systems and their Interfaces for the Electronics in Motor Vehicles; proprietary			OSEK; proprietary; emerging thread safe and Portable Operating System Interface for Unix compliant including Linux[195]
Hardware Technology		Emerging PU, 8 and 16 bit single core PU; proprietary and custom IO; ratiometric (output directly proportional to an input) and differential sensors, electromechanical actuators			Emerging 32 bit PU, reduced instruction set computing, complex instruction set (CISC) and multiple instruction multiple data processor instruction sets on single core PU (advanced RISC machine, microprocessor without interlocked pipelined stages, power PC architectures); ASIC; inter-ECU communications/dedicated busses; standardized IO and peripheral interfaces (DIO, analog to digital, digital to analog converter, PWM, pulse width demodulation, capture compare unit, watchdog timer, timer); differential and ratiometric sensors; emerging multi-core PU; emerging solid state and smart sensors and actuators.			Legacy support; emerging HPC and System on Chip; emerging heterogeneous microcontroller unit, GPU, field-programmable gate array (FPGA) processing architectures; emerging Network on Chip; Ethernet backbone; inertial sensors; global positioning system; wideband sensors (e.g., camera(s), radar, light detection and ranging, ultrasonic); solid state sensors and actuators; smart sensors and actuators; emerging systems on ECU (e.g., integrated Advanced Driver Assistance System controller)

[195] Zahir, A., & P. Palmieri, P. (1998, November 13). *OSEK/VDX-operating systems for automotive applications*. IEE Seminar on OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), London. doi: 10.1049/ic:19981075. https://ieeexplore.ieee.org/document/744164

Year	1950s & 1960s	1970s & 1980s	1990s	2000	2005	2010	2015	2017 to present
Framework Era		**Gen 1**			**Gen 2**		**Gen 3**	
Framework Subclassification	**Example Characteristics//Unclassified Categories (not exhaustive)**							
Control Strategies and Control Systems		Open loop; closed loop digital (e.g., Proportional/Integral/Derivative control); dedicated (foundational) controls (e.g., closed loop motor control)			Closed loop digital; functional control systems (e.g., Electronic Fuel Injection, engine management system, antilock braking system, Traction Control System, Adaptive Transmission Control, Electronic Climate Control, ACC, Power Assisted Steering); emerging ADAS			Closed loop digital; guidance; trajectory; functional control systems; supervisory and authority control systems (e.g., ADS; ADAS; integrated control systems
Communications Strategy, Network Architecture and Topology		Sensors directly connected to ECU; emerging communications bus and ECU - ECU		In-vehicle communications bus and ECU- ECU (e.g., CAN); emerging V2V; emerging smart and networked sensors and actuators; emerging wireless and RF communications (e.g., 3/4,DSRC)				ECU-ECU; networked smart sensors and actuators; emerging V2V and V2X; in-vehicle backbone bus (e.g., TSN[196]); emerging NoC[197]; wireless and RF communications (e.g., 3/4/5G, DSRC)
Model of Computation				Real time; distributed; finite state machines; dataflow process networks; discrete event; synchronous reactive				Concurrent and parallel; multi thread; pragma based[198]; accelerator; service-oriented architecture (SOA); real time; distributed; finite state machines; dataflow process networks; discrete event; SR; AI
Development Tools				Modeling ISO 26262 concepts; requirements management; architecture (definition); modeling tools; modeling style guide enforcement; model metrics; model debugging; data dictionary; model diff and model merge; automatic test vector generation; test execution/management; model coverage measurement; model viewers; model documentation; automatic code generation; ACG - low level drivers; static code analyzers; HIL; calibration; requirements traceability; compilers; real-time operating system; version control; issue tracking; product languages; process tools; reviews				TBD

[196] Institute of Electrical and Electronics Engineers, Inc. (2017). *Time-Sensitive Networking Task Group* [Web page]. www.ieee802.org/1/pages/tsn.html

[197] Arteris IP. (2020, November 4). *Application driven network-on chip architecture exploration & refinement for a complex SoC: How to identify bottlenecks and converge towards the NoC implementation* [Web page]. Semiconductor Engineering. https://semiengineering.com/application-driven-network-on-chip-architecture-exploration-refinement-for-a-complex-soc/

[198] Pragma (from "pragmatic") is a language construct that specifies how a compiler (or other translator) should process its input.

3.3.2 Business and Market Factors

Table 12 presents the framework taxonomy classifications for business and market factors that act as process change factors for the automotive SDLC.

Table 12. Taxonomy of business and market factors with impacts on SDLC constraints and requirements.

Business and Market Factors with Influences on the Automotive SDLC				
Research Theme	**Classification**	**Subclassification**	**Change Factors**	**Framework Sources**
Business and Market Factors	Business and Market Factors	Macro Factors	Mobility model; demographics and geographies; propulsion technology; energy supply and availability; maintenance and sustainment; product feedback; consumer requirements; supply chain; increasing number of mechatronic and software features; increasing use of automation and connectivity (V2X); environmental impact; innovation/first to market with IP including from new entrants; commoditization of E/E hardware and components; manufacturability of E/E systems	[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]
		Design Philosophy	Consumer product; cost optimized around performance; comfort; safety and consumer features	
		Utilization	Predominantly individual use (< 8% of registered vehicles in the US are fleet), 1-2 hours/day	[12] [13] [14]
	Complexity	Software Complexity	Process complexity; code complexity; architectural complexity; variant complexity; requirements complexity; no accepted consensus standards for measuring complexity	[15] [16] [17]

The following sections summarize classifications and sub-classifications within the business and market factors theme.

3.3.2.1.1 Business and Market Factors Taxonomy

Section 2.3 provided an overview of business and market factors that shape the automotive industry and heavily influence software practices. The following section captures these influences in terms of their impact on critical lifecycle practices.

3.3.2.1.1.1 Macro Factors

Macro factors include non-functional requirements such as business and economic factors, demographics, legal, social, supply chain, and natural forces that influence the SDLC. Additionally, competitive decisions come into play, such as when the development pace for ADSs increased significantly following Google's automated vehicle debut in 2014.[199] This had a

[199] Glon, R., & Edelstein, S. (2020, July 31). *The history of self-driving cars.* Digital Trends. www.digitaltrends.com/cars/history-of-self-driving-cars-milestones/

significant impact on automotive software development as traditional vehicle manufacturers shifted focus to developing ADS technologies (i.e., starting the shift from Gen 2 to Gen 3).

3.3.2.1.1.2 Design Philosophy

Design philosophy classifies fundamental requirements related to style, performance, safety, consumer demand, economics, manufacturing processes, and end use. Among producers of critical software, the automotive industry is, perhaps uniquely, characterized by heavy influences on requirements related to consumer driven utilization, stylistic, and performance characteristics.

3.3.2.1.1.3 Utilization

Utilization describes the end-use model and duty cycle of the vehicle during daily operation. The utilization model influences requirements and costs related to durability and longevity. It thereby influences functional performance requirements related to dependability and availability, design margins, and overall life of the vehicle.

Emerging "last mile" delivery services (e.g., FedEx Freight Direct) and mobility as a service providers (e.g., Uber, Lyft) have provided evidence of changing business models that may influence utilization.

3.3.2.1.2 Complexity Taxonomy

Complexity describes the "state of complication" of the software, processes, architectures, and other aspects of automotive E/E systems. There is widely acknowledged lack of agreement as to how to define and measure complexity related to the growing software content on modern vehicles. Still, many of the research sources cite "complexity" as a key driver behind SDLC process costs.

This research identified process, code, and architectural complexity as three of the leading sources of overall complexity for critical E/E software.

Table 13. Impacts on SDLC practices due to business and market constraints and non-functional requirements

Influences due to Business and Market Constraints on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Business and Market Factors	• Evolving and increasing use of E/E control systems onboard motor vehicles drives constraints for software criticality (e.g., safety, reliability, security, dependability) • Expanding product variability introduces added software complexity due to variant and configuration management (e.g., models and model variants, software options and configurations) • Consumer use cases drive private, independent and consumer managed maintenance and operation of motor vehicles impacting requirements for usability, maintainability, reliability, and robustness of E/E systems • Consumer driven mobility models and limited utilization impact requirements for reliability, robustness and dependability for automotive E/E components, and result in E/E systems that are designed for ~10 years/150,000 miles of service • Energy supply and availability influence vehicle E/E architectures and propulsion (e.g., electric drive versus internal combustion engine) • Increasing demand for ADAS drives complexity, changing risk models, changing MoC, changing failure management strategies, and changing criticality for onboard software		• Changing mobility models (e.g., MaaS) drive requirements for increasing utilization and duty cycles, commercial maintenance and repair, and changing critical software constraints • Changing control strategies and paradigms (e.g., ADS, ADAS) drive changing critical software constraints (e.g., location awareness, trustworthiness) • Changing communications strategies and connectivity (e.g., telematics) drive increasing complexity and exposure to cybersecurity threats
Impact on SDLC	• Emerging processes, consensus standards, and MoC for critical software development • Process complexity and variant management drive costs and limits to affordability of onboard software • Changing failure management strategies with increasing numbers of interconnected critical systems • Need for consensus standards maturity related to security, communications protocols, interoperability • Evolution of reference architectures (AUTOSAR, etc.) • Evolution of distributed critical software development and critical element out of context for software development across automotive supply chain boundaries		• Emerging safety paradigms (e.g., safety of the intended functionality (SOTIF)) and increasing need for top-down/system safety approach, (e.g., STPA) • Changing criticality for systems related to security, location awareness, quality of service • Lifecycle impacts on municipalities, other entities, possessing limited experience managing large safety critical software systems • New and changing MoC, including stochastic and machine learning, probabilistic computing, indeterminate concurrency, arbitration. • Increasing use of statistical testing, road testing, "game play testing" • Changing SDLC methodologies including software development leveraging ML techniques • Need for consensus standards maturity related to learning algorithms, probabilistic computing, ML Coding Practices, ML algorithms, trust • Evolution of reference architectures to accommodate new MoC and architectures

3.3.3 Consensus Standards Research Theme

In highly regulated industries, such as aviation, software processes and architectural consensus standards are often driven by regulatory agencies to be well defined, harmonized, and consistently applied across many applications.[200] Automotive manufacturers and suppliers use voluntary industry oversight and market discipline to inform consensus standards. Automotive consensus standards often originate from proprietary documentation and integrate a wide range of internally developed protocols, external consensus standards, and ad-hoc process definitions. Various standards bodies and associations (e.g., AUTOSAR, SAE) have been organized by the industry to maintain public consensus standards.

During the research, several conversations with vehicle manufacturers and other industry experts[201] indicated that the automotive industry's position of standardization through self-certification (both to regulations and consensus standards) and voluntary market discipline in the United States provides a level of resistance to widespread acceptance of universal consensus standards. The conversations revealed a full spectrum of approaches to standardization related to industry reference models. Some companies have evolved their processes internally with no explicit effort to look at external consensus standards, others pick and choose portions from several consensus standards, and some strive to fully implement published consensus standards.

An example of the gap between consensus standards and practice can be seen in a response an expert provided during a research conversation, where the respondent described the organization's view that ISO 26262 is widely accepted as the leading consensus standard for functional safety in automotive E/E systems. The respondent's organization has publicly established the goal of becoming completely ISO 26262 compliant across all product lines yet is estimated to be only 40 percent compliant within a few of the product lines as of 2019. That is, while vehicle manufacturers may recognize ISO 26262 as the state-of-the-art for functional safety, a gap exists where internal processes still might not be updated to comply with ISO 26262.

[200] For example, the FAA released an advisory circular describing implementation of the DO-178 consensus standard, but notes that adherence to DO-178 is one way, but not the only way, to satisfy applicable airworthiness regulations.

[201] To elicit more complete and candid responses, the contractor interviewed industry experts anonymously.

Table 14. Taxonomy of key regulations and consensus standards that influence SDLC constraints and requirements

Regulations and Standards with Influences on the Automotive SDLC				
Research Theme	**Classification**	**Sub classification**	**Change Factors**	**Framework Sources**
Regulation	Regulatory Jurisdiction	N/A	International, Federal, regional	[109] [110]
	Regulatory Standards	Environmental	Motor Vehicle Air Pollution Act (1965); Air Quality Act (1967); Federal Motor Vehicle Safety Standards (1967 to present); Clean Air Act (1970); Clean Air Amendments (1990); California Emissions Standards (Various); Clean Fuels Alternatives; Natural Low Emissions Vehicles; Tier 2 Tailpipe Emissions (2004 - 2009)	[50]
Standards	Architectural	Hardware Software and Logical	AUTOSAR Classic Platform; Adaptive AUTOSAR; Proprietary; AGL; Proprietary	[18] [19] [20] [21] [22]
		Network and Communications	CAN Specification: 1991; LIN Specification: ~2003; MOST Specification: ~2008; TTEthernet Specification: ~2008; FlexRay/ISO 17458-1:2013; TSN/IEEE 802.1AS: 2018 (deterministic Ethernet); Class A, B, C, D	[23] [24] [25] [26] [27] [28] [29] [30]
		Machine Learning and AI	Emerging computational approaches and architectures; emerging trustworthiness; emerging use cases and applications; emerging foundational consensus standards; emerging system consensus standards (e.g., ADSs)	[31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42]
	Programming	Language; Style; Syntax	MISRA (C, C++, ACG); AUTOSAR 068; MAAB; AUTOSAR C++14; HIC/HIC++; CERT	Appendix A Table 8 [43]
	Process	Process Assurance and Assessment	ASPICE process reference model; measurement framework; Process Assessment Model; performance indicators	[44]
		Process Reference Model	ISO 26262; ISO/SAE 21434; ASPICE; SAE J3061; ISO PAS 21448	[45] [46] [47] [48] [49]

Table 15. Impacts on SDLC practices due to regulations and consensus standards

Influences due to Regulations and Standards on the Automotive SDLC[202]			
	Generation 1	**Generation 2**	**Generation 3**
Regulations and Standards	• Continuously evolving • Governed by voluntary market discipline; private litigation using public rules/consensus standards; performance based regulation (e.g., FMVSS beginning in 1967)		• Emerging consensus standards driven by automation, connectivity, and evolving MoC • Changing reference architectures and consensus standards • Specialized consensus standards and processes related to application specific HW/SW (e.g., system on a chip (SoC))
Impact on SDLC	• Market discipline and self-certification drives the implementation of widely varying processes and methods across the industry • Emerging critical consensus standards provide process reference models for critical software • Widely varying processes across supply drive increasing interdependence between SDLC processes and architectural consensus standards		• Emerging architectures and programming languages drive changing and new reference models, programming syntax and style guides

[202] Because the regulations and standards in the taxonomy provide industry definitions and taxonomies that define products, software, programming languages, architectures, and other aspects of E/E systems and software, the influences shown in Table 15 are given for the regulations and standards theme and not the individual standards provided at the classification and sub-classification levels of the framework.

3.3.3.1 Architectural Standards Taxonomy

Automotive E/E systems are produced by distributed and vertically integrated suppliers that progressively integrate components and subsystems through the supply hierarchy. SDLC processes are often distributed across supply chain boundaries, a fact that affects requirements management, validation and verification, process integration, release management, and nearly every step of the SDLC. The distributed nature of SDLC processes makes it difficult to produce an exact description of how the industry produces software; thousands of methods are practiced by hundreds of vehicle manufacturers, Tier I suppliers, and other suppliers.

In order to achieve product and process compatibility, the industry produces a wide variety of architectural consensus standards that define hardware, logical and software interfaces between operating systems, middleware, interconnected electronic control units (ECUs), peripherals (e.g., sensors and actuators), and software applications that allow diverse, distributed networks of suppliers to produce seamlessly integrated products (Figure 14).

As the "software-defined vehicle" takes form, E/E architectural definitions will increasingly define and reflect the overall composition of the vehicle. Hardware and software architectures, developmental processes, and consensus standards are interrelated. The corresponding framework taxonomy may be required to capture complex relationships between classifiers due to interdependencies between wide ranging factors, as described above.

Table 14 provides a taxonomic overview of regulations and consensus standards that influence automotive critical software practices. Further information is available in Appendix A, Table 8.

3.3.3.1.1 Hardware, Software, and Logical Consensus Standards

Hardware, software, and logical consensus standards provide structural descriptions of system architectures and give developers the necessary abstraction layers, interface definitions, and logical structures to allow integration of E/E systems from distributed suppliers and developers.

Table 14 provides a classification for architectural *consensus standards* used in the automotive industry.

Section 3.3.4.1.2 provides the taxonomy for architectural types used in the framework.

3.3.3.1.2 Network and Communications Consensus Standards

Network and communications consensus standards define protocols, use cases, and functional specifications for busses and other communications mechanisms (e.g., NoC) that are used to interconnect ECUs, actuators, sensors, and other devices on the vehicle.

Table 14 provides a classification for network and communication consensus standards used in the automotive industry.

Section 3.3.4.3.1 provides a taxonomy of hardware technology that is used in automotive networks and communications systems, and Section 3.3.4.5.1 describes a framework taxonomy for communications strategies, network architecture, and topology.

3.3.3.1.3 Machine Learning and AI Standards

Table 14 provides a classification for AI and machine learning consensus standards based on current consensus standards and from numerous working groups for upcoming consensus

standards.[203] Appendix A, Table 8 provides a summary of emerging consensus standards related to lifecycle processes, tools, and methods for nomenclature, trustworthiness, computational approaches, and MoC in AI frameworks. As these approaches mature, the framework can be updated to capture emerging machine learning and AI consensus standards.

The diversity of new automation-related technology and software methods, lack of existing consensus standards, and variety of closely held, proprietary approaches to machine learning and AI all presented challenges to developing a framework. This research revealed a wide range of predictions and speculation related to how consensus standards and practices will evolve with respect to the safety, trustworthiness, and morality of rapidly evolving AI software.

As "Gen 3" emerges, the landscape of tools, processes, consensus standards, and technology is rapidly changing, particularly with respect to machine learning and AI, automated safety critical systems, and cybersecurity. For example, in early 2020, UL 4600 became the first published consensus standard specifically for documenting the safety evaluation of ADS.[204] The framework is adaptive and designed to be revised in order to reflect changes and trends across the automotive industry. Table 29 provides a short-term roadmap to capture consensus standards that have emerged during this study.

As consensus standards and practices emerge, additional frames will be required.

3.3.3.2 Programming Standards Taxonomy

The SDLC's ability to meet requirements across the functional hierarchy is strongly influenced by the coding practices, style, coding constructs, syntax, and conventions used during construction of source code and models. For example, Part 6 of ISO 26262 describes different modelling and coding guidelines needed to comply with the standard. Software programming and modeling language consensus standards define best practices and use it to ensure that software can meet requirements for performance, functionality, and criticality.

3.3.3.2.1 Language, Style, and Syntax Standards

Research conversations with experts revealed that the automotive industry uses a number of well-established and mature consensus standards for language, style, and syntax. For "Gen 2" development, MISRA-C is used for C code and MAAB and/or MISRA SL/SF, for model-based development. Interviewees described widespread acceptance for both consensus standards and indicated that company-specific extensions to the consensus standards are common, particularly in large software organizations.

Rapidly evolving use of other programming languages in "Gen 3" applications indicate that a number of new language-related consensus standards are on the horizon. Changes to the existing consensus standards may also come as emerging languages and programming frameworks are tailored to accommodate new HPC, GPU, and FPGA-based E/E platforms that incorporate programs written in Open CL, CUDA, VHDL, and other languages.

---

[203] For this document "AI" and "Machine Learning" are used to generally describe a wide range of algorithms, technologies, and automation techniques that are used in Gen 2 and emerging in Gen 3 in support of automation. Examples of machine learning techniques include supervised and unsupervised learning techniques, classification and regression learning algorithms, route planning, sensor fusion and image classification.

[204] Underwriters Laboratories Inc. (2021). *Presenting the standard for safety for the evaluation of autonomous vehicles and other products* [Web page]. https://ul.org/UL4600

Table 14 provides a classification for programming language, style, and syntax consensus standards used in the automotive industry.

3.3.3.3 Process Consensus Standards Taxonomy

Interdependencies between architectural consensus standards, models of computation, standardized commercial tooling (e.g., MBD, ACG), and various consensus standards for programming syntax and style drive the need for standardized SDLC process steps.

Standardized process models ("*reference model*") have a profound impact on the way E/E systems are conceived, designed, constructed, tested, and delivered to customers. Reference models enable successful operations across geographic, cultural, and supply chain boundaries.[205]

3.3.3.3.1 Process Assurance and Assessment

Process assurance and assessment is related to an organization's ability to ensure that process steps are implemented correctly and are being followed according to the intent and requirements of the process. *Software process assessment* examines whether implemented software processes are effective and efficient in accomplishing the goals set forth in process requirements. A software *capability* or software *maturity* assessment is performed against a process reference model in order to determine the relative capability of an organization's SDLC when measured against the idealized process reference model.

Table 14 provides a classification for process assurance and assessment consensus standards that are used in the automotive industry.

3.3.3.3.2 Process Reference Model

Process reference models provide abstractions of software lifecycle processes, including process capabilities (e.g., process documentation), process steps (e.g., software validation and test), and supporting process infrastructure (e.g., quality, culture, qualified personnel). Software producing organizations may measure real, implemented enterprise SDLC processes against these models.

ASPICE and ISO 26262 are examples of consensus standards that provide process reference models for the measurement of process maturity. For instance, the automotive industry developed the ASPICE process reference model and process assessment methodology as a generalized model for E/E systems development. Process reference models for critical modalities (e.g., ISO 21434 and ISO 26262) establish idealized practices for the respective modalities (e.g., Hazard Analysis and Risk Assessment for safety critical systems, Threat Assessment and Remediation Analysis for cybersecurity).

Table 14 provides a classification for process reference models that is used in the automotive industry.

3.3.3.4 Environmental Regulations and Standards Taxonomy

Environmental regulations are highly influential over the evolution of E/E technology on motor vehicles. Innovation related to electronically controlled fuel delivery (e.g., EFI), ignition and ignition timing, and CO_2 sensing (among others) has been at the forefront of the vehicle industry's efforts to reduce emissions and improve fuel efficiency. Emerging architectures for

[205] Prikladnicki, R., Audy, J. L. N., & Evaristo, J. R. (2007, March). *A reference model for global software development.* Working Conference on Virtual Enterprises, Guimarães, Portugal. doi: 10.1007/1-4020-8139-1_39.

high efficiency and reduced-emissions vehicles (e.g., H-EV, EV, and FCEV) are made possible by digital control systems and vehicle electrification.

Onboard diagnostics consensus standards provide an example of how environmental regulations may impact automotive architectures and lifecycle practices. Many of today's OBD-related automotive consensus standards, including establishing a common set of diagnostic trouble codes (DTCs), originate in environmental regulations.[206] Automotive software systems must be capable of detecting and logging the faults that trigger these DTCs.

Various regulations for fuel economy, emissions, and diagnostics provide general constraints that influence SDLC practices, which are shown in Table 15.

[206] For example, see Environmental Protection Agency. (n.d.). *Vehicle emissions on-board diagnostics (OBD)* [Web page and portal]. www.epa.gov/state-and-local-transportation/vehicle-emissions-board-diagnostics-obd

3.3.4 Software Type, Technology, Tools and Programming Languages Research Theme

Table 16 summarizes the taxonomy classifications for a wide range of technology, tools, programming languages, and other types with strong influence over lifecycle practices for software on motor vehicles.

Table 16. Software typology and classification of technology, tools, and programming languages with impacts on SDLC constraints and requirements

Software Type, Technology, Tools, and Programming Languages with Influences on the Automotive SDLC				
Research Theme	**Classification**	**Subclassification**	**Change Factors**	**Framework References**
Software Type, Technology, Tools & Programming Languages	Type	Modes of Criticality (Table 17)	Reliability; real time; safety (e.g., functional safety, system safety, SOTIF); cybersecurity; mixed criticality onboard computing; emerging mixed criticality within ad-hoc (off board) V2X networks; location awareness criticality; trustworthiness ; emerging morality criticality	[51] [52] [53] [54] [46] [47] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60]
		Architecture (Table 18)	Logical, software, and hardware architectures; standardized; dedicated (predominantly fixed); standardized ECU elements (e.g., memory, processing elements, IO/peripherals); distributed/decentralized controller networks; dedicated and standardized peripheral and communications bus interfaces; emerging adaptive and multi-purpose peripheral interfaces; emerging HPC architectures; emerging sensor fusion; multiple and heterogeneous PUs (including multicore microprocessor unit (MPU), graphics processing unit (GPU), data flow processor (DFP), FPGA); emerging adaptive peripheral interfaces; emerging Ethernet bus backbone; emerging integrated domain controllers (e.g., ADAS); independent subdomain architectures (e.g., body electronics, powertrain, chassis, occupant and pedestrian safety, multimedia, telematics, and HMI); functional/dedicated applications; fixed applications; virtual interfaces (e.g., application layer, RTE, service layer (e.g., OS, Mode, Diagnostic, Firmware, Memory, COM); ECU abstraction layer; microcontroller abstraction layer); emerging SOAs; emerging functional clusters and adaptive applications (integrated subdomains, integrated foundational/functional applications)	
		Fault Management (Table 19 and Table 20)	Human monitoring and intervention; fault avoidance and removal; safety mechanisms; fail safe; fail operational (including redundancy, mitigation)	[62] [63] [64] [65] [66] [67] [68] [69] [70]

Research Theme	Classification	Sub classification	Change Factors	Framework References
Software Type, Technology, Tools & Programming Languages	Software Scheduling	OS Technology (Table 21)	Simple scheduler, OSEK; proprietary; emerging POSIX compliant including Linux	[18] [19] [22] [71] [72][73] [74]
	Hardware	Hardware Technology (Table 22)	8, 16, and 32 bit single core PU; proprietary and custom IO; ratiometric and differential sensors; electromechanical actuators; RISC, standard instruction set computing and MIMD processor instruction sets on single core PU (ARM, MIPS, PPC architectures); inter-ECU communications/dedicated busses); standardized IO and peripheral interfaces (DIO, ADC, DAC, PWM, PWD, CCU, WDT, timer); emerging multi-core PU; emerging solid state and smart sensors and actuators; HPC/SoC: MCU, GPU, FPGA; NoC; Ethernet backbone; inertial sensors; GPS; wideband sensors (e.g., camera(s), radar, LIDAR, ultrasonic); solid state sensors and actuators; smart sensors and actuators; emerging systems on ECU (e.g., integrated ADAS controller)	[51] [59] [60] [75]
	Computer Control	Control Technology and Control Systems (Table 23)	Open loop; closed loop (digital); guidance and trajectory; functional and foundational; supervisory and authority (e.g., automation and autonomy)	[76] [51] [52] [77] [78] [79] [80] [81] [113]
	Communication	Communications Technology, Network Architecture and Topology (Table 24)	Sensors directly connected to ECU; ECU-ECU; V2X; in-vehicle shared inter-ECU bus (e.g., TSN); smart and networked sensors and actuators, NoC	[23] [24] [25] [26] [27] [28] [29] [30] [83]

Research Theme	Classification	Sub classification	Change Factors	Framework References
Software Type, Technology, Tools & Programming Languages	Software Implementation	Tools and Programming Languages (Table 33)	Modeling ISO 26262 concepts; requirements management, architecture (definition); modeling tools ; modeling style guide enforcement; model metrics; model debugging; data dictionary; model diff and model merge; automatic test vector generation; test execution/management; model coverage measurement; model viewers; model documentation; ACG; ACG - low level drivers; static code analyzers; MIL/Hardware in the Loop/SIL; calibration; requirements traceability; compilers; RTOS; version control; issue tracking; product languages; process tools; reviews	[84] [85] [86]
		Model of Computation (MoC) (Table 25)	Real-time; distributed; finite state machines; dataflow process networks; discrete event; SR; concurrent and parallel; multi thread; pragma based; accelerator; SOA; AI	[22] [19] [60] [53]

3.3.4.1 Software Type Taxonomy

The following sections provide a general typology for influences on automotive software lifecycle practices.

3.3.4.1.1 Critical

Section 2.1 of this study provided an overview of critical modalities that affect the automotive SDLC, e.g., dependability, safety, security, real time, trustworthiness, location awareness, maintainability, and availability.

The "level of criticality" for a system is measured in terms of critical assurance levels, which are based on mode of criticality. The critical assurance level (e.g., ASIL for functional safety) is determined from the risk assessment process that has been designated for the specific modality of criticality (e.g., Hazard Analysis and Risk Assessment (HARA) for functional safety, Threat Assessment and Remediation Analysis (TARA) for cybersecurity). Reference models for automotive software criticality are emerging and relatively new. For example, ISO 26262, Road vehicles - Functional safety, was first released in 2011, and ISO 21434, Road Vehicles – Cybersecurity engineering, was issued in February 2020.[207]

A detailed analysis of critical assurance levels and consensus standards associated with criticality is outside of the scope of this paper. For this phase, the objective is to establish a high-level typology and taxonomy for relevant process reference models related to the SDLC in order to use them for comparison with other industries and domains.

Impacts of Criticality on SDLC Practices
The ASIL assignment process happens when an ASIL is calculated for hazardous events and is assigned to system components. Hardware or software components may realize several functions with different ASIL ratings. If this occurs, the hardware or software component inherits the most critical ASIL, as described in ISO 26262. Appendix A, Table 10 provides an example of ASIL ratings for automotive functional safety across various E/E systems.

The ASIL assignment process provides a method for developers to integrate systems from elements with different levels of criticality. The benefit is that the number of developmental process steps may be reduced across the integrated system through reduced process overhead for elements with lower ASIL ratings. Benefits are realized, however, at the cost of increased complexity in order to manage the ASIL assignment process.

ASIL classification has wide-ranging impacts on automotive E/E systems, production, and the complexity and number of SDLC process steps. The assignment of an ASIL to a software element within an automotive E/E system significantly impacts the level of effort and number of process steps required for the software to be developed across the lifecycle (Appendix A, Table 11). As an example of this, one of the study experts revealed that the respondent's organization estimates approximately 25 percent in increased developmental costs for each ASIL level that must be achieved for a software component. industry experts indicated that software development practices that are designed to satisfy the highest levels of criticality (e.g., ASIL D)

[207] See ISO 26262: 2011, Road Vehicles- Functional safety,; ISO 21434: 2019. Road Vehicles – Cybersecurity Engineering for detailed information on the respective critical assurance levels and hazard, threat, and risk treatment.

can cost up to 100 percent more than the lowest levels (e.g., quality management or ASIL A). Analysis published by software development organizations indicate similar estimates of 50-80% cost increases from the lowest to the highest levels of criticality, as well as one estimate of 10-fold increased effort.[208] Cost multipliers are expected to be higher for systems as the requirements for mixed criticality become more complex (e.g., increasing number of critical modalities for a system).

Various consensus standards, including ISO 26262 and ISO 21434, provide processes for defining levels of criticality, not only for the software, but also for the entire electromechanical system.

Emerging automotive E/E systems are susceptible to several, concurrent critical modalities (e.g., safety and security), which introduce demands on the SDLC for process harmonization and integration. In order to provide a classification of the critical modalities encountered during automotive software development it is necessary for the framework to address the increasing occurrence of mixed criticality within automotive E/E systems. *Mixed criticality* occurs when components with different levels of criticality coexist in the same system or in interacting systems.[209] Automotive E/E systems may be implemented with mixed criticality within a single modality, such as safety critical subsystems constructed from items with different ASILs. Mixed criticality may also occur when E/E systems are constructed from *items* with criticality across several critical modalities, such as systems constructed from items with different ASIL and cybersecurity assurance levels. Requirements for mixed criticality can conflict with each other. For instance, security measures to satisfy cybersecurity assurance levels hypothetically may slow down the system responsiveness to the extent that the functional safety requirements are no longer met.

According to D'Ambrosio and Debouk, challenges related to mixed criticality are compounded when criticality is considered vertically through the system hierarchy. The functional safety approach used in the automotive industry is designed to allow for the "bottom-up" integration of critical subsystems from a multitude of suppliers with the goal of achieving system-level (e.g., vehicle) safety assurance. System-level safety requirements developed by vehicle manufacturers are often developed using "top-down" hazard, threat, and risk assessments. As a result of the "bottom-up" nature of production across the automotive supply ecosystem, vehicle manufacturers are increasingly faced with the challenge of integrating "top-down" and "bottom-up" approaches to criticality in order to integrate safety systems vertically within the safety hierarchy.[210]

---

[208] Table 1, ASIL cost heuristics, in Gheraibia, Y., Kabir, S., Djafri, K. & Krimou, H. (2018). An overview of the approaches for automotive safety integrity levels allocation. *Journal of Failure Analysis and Prevention, 18*, doi: 10.1007/s11668-018-0466-9 https://link.springer.com/article/10.1007/s11668-018-0466-9/tables/1; Tom-M.[sic] (2019). *What does it cost to implement functional safety?* [Web page]. ADI EngineerZone. https://ez.analog.com/b/engineerzone-spotlight/posts/what-does-it-cost-to-implement-functional-safetyand Hilderman, 2014.

[209] Crespo, A., Alonso, A., Marcos, M., de la Puente, J. A., & Balbastre, P. (2014, August 24-29). *Mixed criticality in control systems* 19th World Congress, International Federation of Automatic Control, Cape Town, South Africa. Also in IFAC Proceedings, 47(3)..

[210] D'Ambrosio, J., & Debouk, R. (2013). *ASIL decomposition: The good, the bad, and the ugly* (SAE Technical Paper 2013-01-0195). SAE International. https://doi.org/10.4271/2013-01-0195

Table 17. Modes of Criticality with impacts on SDLC

Criticality and Influences on the Automotive SDLC			
	Generation 1	Generation 2	Generation 3
Modes of Criticality	• Emerging requirement for real time performance, safety and reliability in emerging E/E systems	• Widespread use of E/E systems in automotive critical applications • Development of consensus standards for reliability and functional safety and the emergence of design assurance levels (e.g., ASIL, design assurance levels (DAL)) • Emergence of multi-function/functional control systems (e.g., engine management systems (EMS)) • Emergence of mixed criticality systems • Emergence of V2X, networked vehicles, and requirements for cybersecurity engineering • Increasing presence of single mode mixed criticality (e.g., different levels of criticality (e.g., ASIL) within a system for a single mode of criticality (e.g., safety)) driven by increasing number of critical functions on distributed ECUs	• Increasing cybersecurity criticality due to OTA and V2X threat surface/exposure • Increasing presence and complexity of multi-mode mixed criticality systems (e.g., systems with concurrent modes of criticality) • Emerging modes of criticality (e.g., SOTIF, AI) and associated Models of Computation (MoC) due to increasing use of automation (e.g., ADS, ADAS, AI) and interconnectivity (e.g., V2X, OTA) across E/E systems
Impact on SDLC	• Development of foundational practices for critical software engineering (e.g., consensus standards for language, style, syntax, proprietary process reference models)	• Emergence of critical reference models and consensus standards for criticality and critical software development (e.g., ISO 26262, ISO 21434) • Increasing use of standardized MBD, MIL/HIL/SIL practices across increasingly industry standardized V-Cycle and Agile • Development of process assessment models, development of standardized reference (HW/SW) architectures for critical E/E systems	• Cybersecurity threat and remediation require changing end-user license agreement (EULA) models and increasing use of continuous updates for software in the field, including OTA updates and Software as a Service • Emerging and unknown lifecycle practices for systems with new and emerging MoC and modes of criticality related to ADS/ADAS, V2X, OTA • Increased reliance on virtualized development and test (e.g., “game testing”) and road/field testing for increasingly indeterminate critical systems • Increasingly difficult to execute deterministic V&V coverage due to the complexity of test requirements related to mixed criticality systems

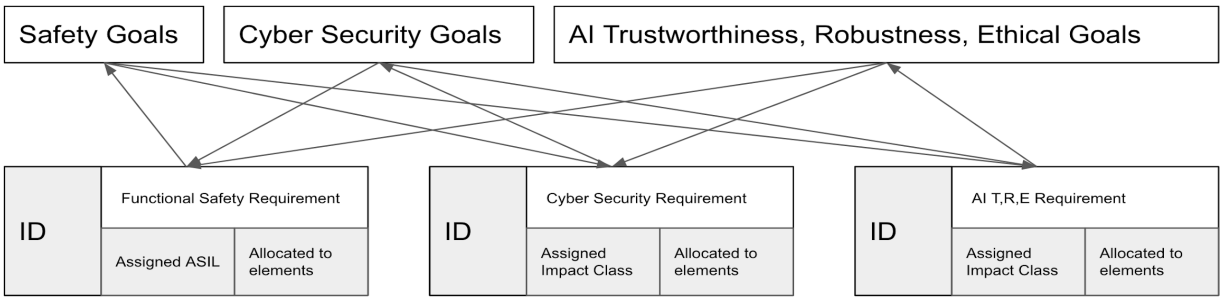

Figure 9. Mixed criticality across several critical modalities [211]

Figure 9 provides a graphical representation of the impacts of mixed criticality on E/E systems. The complexities introduced by mixed criticality are compounded by the complexities of automotive E/E supply chains. The automotive industry is heavily influenced by the need to construct vehicles from interoperable subsystems from diverse networks of suppliers.[212] The industry addresses this requirement, as well as the associated challenge of constructing safety systems from supplier-produced safety subsystems, by developing critical systems in vehicles from the "bottom-up." In this approach, the industry seeks to establish a target system-level ASIL from component elements that have some notion of (mixed criticality) ASIL already associated with them.[213]

D'Ambrosio and Debouk describe how this is achieved through the concept of "element out of context":

> *A Safety Element out of Context (SEooC) is a safety-related element which is not developed for a specific system in the context of a particular vehicle. Assumptions are made at the component level and requirements are developed that can meet a given safety integrity level. This can be seen as a bottom up application of an ASIL decomposition concept*.[214]

They further note that:

> *Design teams need to explicitly consider the (expected) top down requirements decomposition even when the system is being designed bottom up, where design elements have preassigned ASIL*[215]

It is likely that top-down requirements decomposition will become increasingly important and relevant as the industry moves toward SOAs. This trend will drive system-level integration of *ad hoc* integrated safety systems potentially introduced by V2X and V2V interoperability, where

[211] D'Ambrosio& Debouk, 2013.
[212] Ibid.
[213] Ibid.
[214] Ibid. See Appendix A, Table 10 for representative automotive E/E subsystems and their respective ASIL classifications.
[215] Ibid.

top-down system-level criticality will have growing influence over the safety requirements of subordinate E/E subsystems.

3.3.4.1.2 Architecture

Bach et al. developed *A Taxonomy and Systematic Approach for Automotive System Architectures,* which provides a basis for the framework taxonomy's classification of architectural types with impacts on automotive SDLC practices.[216] Mirroring the study, Table 16 captures change factors for logical, software, and hardware architectures.

Differentiation between distributed and integrated system architectures (e.g., centered on distributed domain controllers versus integrated domain controllers) may be used to further sub-classify the architectural taxonomy in future versions of the framework.

Emerging automation functions rely on object recognition, image processing, and other specialized services that require general purpose graphics processing units, field-programmable gate array (FPGA), and other advanced hardware technology. Moving into Gen 3, a shift toward service-oriented architectures built around high-performance integrated domain controllers is reversing the Gen 2 expansion of functions across distributed functional networks.

As an example of how architectural and logical consensus standards impact SDLC practices, one expert described integration tests as modules that are increasingly developed for "plug and play use." For modules developed by suppliers, integration testing by the vehicle manufacturer may end up being minimal, resulting in integration issues being found late in the development process. The expert believed that some companies are outsourcing larger and larger subsystems to suppliers to minimize the amount of integration testing that needs to be performed by the vehicle manufacturer.

During separate conversations, study sources revealed that these practices could lead to unintended results, including increasing levels of defects and problem reports during the latest stages of testing, and during in-vehicle trials.

Table 18 provides a high-level summary of impacts on the SDLC due to architectural evolution.

[216] Bach, J., Otten, S., & Sax, E. (2017, April 22-24). *A taxonomy and systematic approach for automotive system architectures: From functional chains to functional networks*. Proceedings of the 3rd International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2017), Porto, Portugal.

Table 18. HW, SW, and logical architectures and effects on the SDLC

HW Architectures, SW Architectures, and Logical Architectures with Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Architecture	• Emerging use of E/E systems implementing proprietary, non-standard, and dedicated ECU elements (e.g., peripherals, IO) • Emerging communications bus consensus standards and distributed ECU architectures • Proprietary, dedicated/nonstandard programming interfaces and interface definitions	• Emerging standardized peripheral interfaces (e.g., analog to digital converter (ADC), digital to analog converter (DAC), pulse-width modulation (PWM)), device driver interfaces and architectural hardware (HW) consensus standards (e.g., AUTOSAR Classic Platform) • Communications bus consensus standards and classifications (e.g., CAN, e.g., A,B,C,D) for standardized, distributed domain architectures • Processor selection driven by requirements for closed loop control • Emergence of standardized, distributed domain architectures (e.g., chassis, powertrain) (e.g., AUTOSAR Classic Platform) • Standardized programming and logical interfaces (e.g., AUTOSAR RTE) • Scalable through standardized interfaces, scalable by adding distributed functional ECUs • Emergence and increasing use of calibration for software modules [107]	• Emergence of integrated domain controller • Emerging secure and fail safe hardware • Emerging adaptive interfaces • Processor selection driven by requirements for high volumes of data (e.g., 64 bit microprocessing unit (MPU), general purpose graphics processing unit (GPGPU)) • Emerging specialized, dedicated and application specific hardware architectures (e.g., SoC) • Emergence of Ethernet backbone between integrated domain controllers for time sensitive data • Open, SOA software environment for "plug and play" services and potential V2X networks • Scalable through reuse, services/functional clusters • Increased complexity for real time systems due to multi-threaded programming and heterogeneous controller architectures • Emergence of increasingly indeterminate systems due to SOA and ad-hoc/multi-function software architectures
Impact on SDLC	• Foundational and functional programming • Dedicated, purpose built and hand programmed software • Emergence of logical software architectures • Not scalable	• Hardware abstraction and middleware allowing distributed development, reuse, standardization and interoperability across suppliers • Standardized models representing the logical architecture (e.g., Simulink models for increasing use of standard architectural template libraries) and model based architectures (e.g., Unified Modeling Language (UML), EAST-Architecture Description Language) • Emergence and increasing use of RCP for ECU development due to the availability of commercial RCP systems based on standardized HW and SW architectures [107] • Emergence of programming style guides and consensus standards (e.g., MISRA ACG) • Emergence and increasing use of standardized process reference models (e.g., ASPICE) • Emerging use of object oriented programming and C++ • Scalable through reuse and standard block libraries (MBD)	• Emergence of SOA, multi-function programming • Emergence of agile programming techniques for E/E systems • Emergence of virtualized testing methods for guidance and trajectory control • Emergence of specialized programming languages for data and image processing on specialized processors (e.g., SYCL, CUDA programming languages and GPGPU processing architectures) • Emergence of scene generation HIL testing and real time broadband sensor simulation for HIL testing [108] • Increasing use of object oriented programming • Increasingly complex and indeterminate software and test conditions, driven by potential V2V networks, wide ranging ADS driving scenarios, and SOA • Increasing use of road tests and less reliance on MIL/HIL/SIL due to test complexity • Emerging practices for developing redundant and multi-threaded critical software processes

Evolution of Hardware, Software, and Logical E/E System Architectures With Impacts on the SDLC

Figure 10, Figure 11, and Figure 12 provide graphical representations of the architectural changes captured in Table 18.

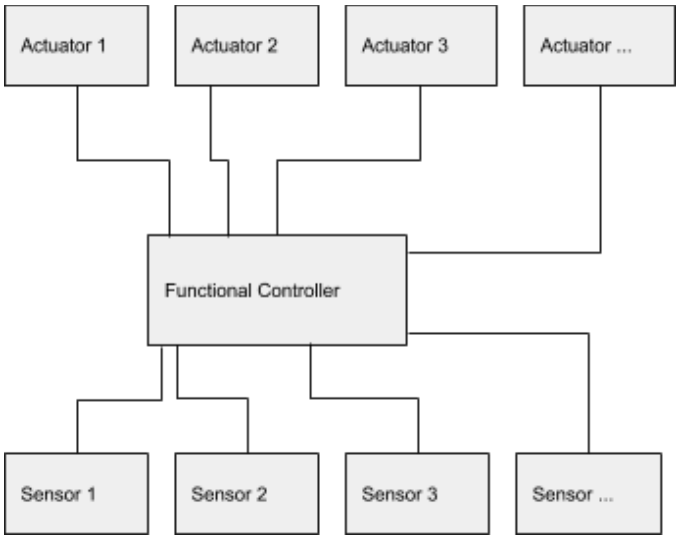

Figure 10. Framework "Gen 1" is characterized by emerging automotive ECUs that incorporate isolated, digital, foundational and functional controls (e.g., throttle body control, ABS).

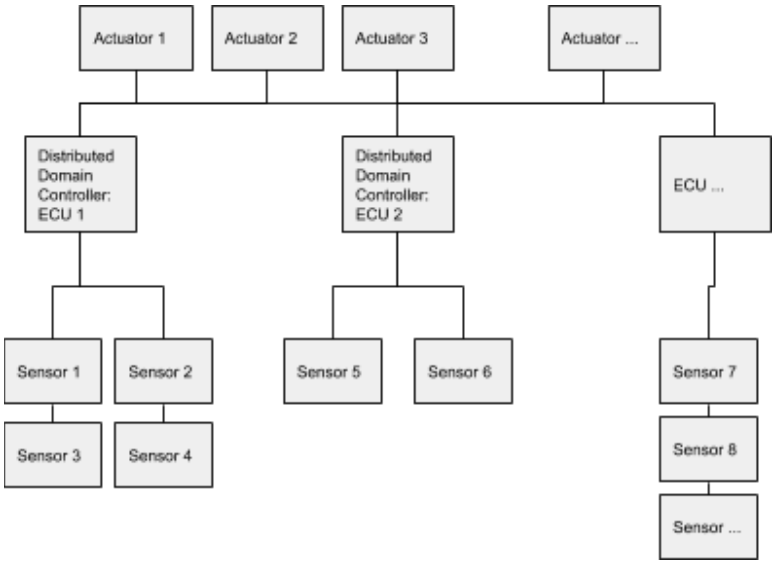

Figure 11. Framework "Gen 2" ECU hardware is increasingly distributed and networked (e.g., ECU-ECU via CAN).

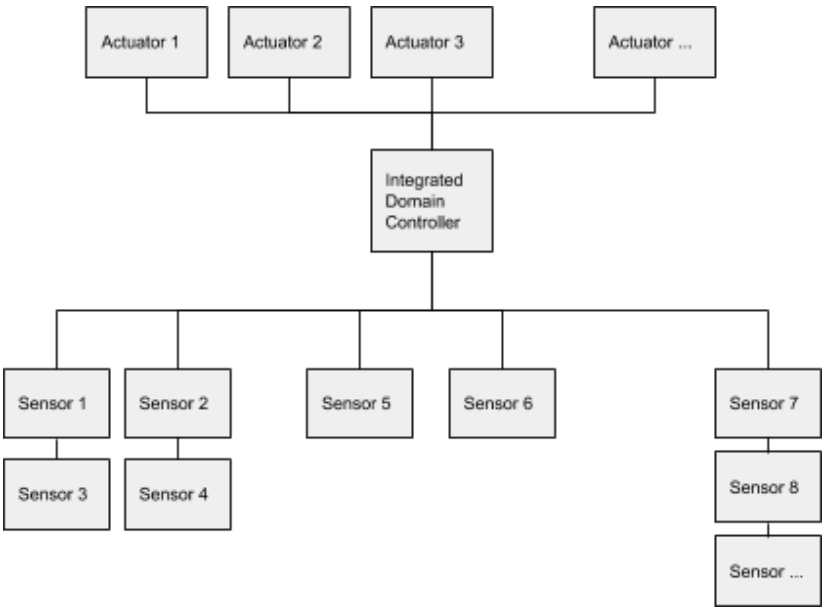

Figure 12. The emergence of "Gen 3" domain controllers continues the trend of function aggregation but reverses the trend of increasing numbers of distributed ECUs onboard the vehicle.

3.3.4.1.3 Fault Management

Section 2.1.5 for this study provided a taxonomy and overview of fault tolerance and fault-recovery strategies with impacts on automotive SDLC practices (i.e., fault avoidance and removal, human monitoring and intervention, safety mechanisms, and fault tolerant systems).

An understanding of the failure modes that are addressed by fault management strategies provides insight into how fault management practices impact the SDLC.

ISO 26262 defines two categories of failures for E/E systems (Table 19). Table 20 provides examples of impacts on the SDLC due to various fault management strategies and implementations.

Table 19. Fault management - example failure modes for E/E systems

Fault Management - Example Failure Modes for E/E Systems[217] [218]		
Category	**Sub Category**	**Example**
Fault Management: Systematic Failures	Process Related	Incorrect specification
	Software Related	Programming error
	Hardware Related	Insufficient EMC immunity due to changing environmental conditions
Fault Management: Random Hardware Failures	Hardware Related	Degradation, wear, oxidation of components

[217] Tyagi, R. (2018). Functional safety architectural challenges for autonomous drive. Infineon.
[218] For reference on failure modes see ISO 26262-6, Annex D. Also see ISO 26262-1 (1.14), ISO 26262-9, ISO 26262-10.

Table 20. Effects of fault management strategies on software lifecycle practices

Fault Management Strategy	Representative Influences on SDLC Constraints and Requirements[219]
Fault avoidance and removal	Added process steps to eliminate or reduce systematic failures that can be eliminated during the development process (e.g., incorrect specification).
Human monitoring and intervention	Affects the number and complexity of process steps due to the level of criticality. For example, elements such as driver warnings, control transitions, and fail safe mechanisms all contribute to increased SDLC complexity to achieve the requisite level of criticality. Affects the test and verification methodology that is used in order to verify the intervention strategy.
Safety mechanism	Affects the number and complexity of process steps due to the level of criticality. Affects the test and verification strategy to test the safety mechanism.
Fault tolerant system	Affects software reusability due to the use of specialized architectures (e.g., redundancy). Affects the ASIL level (e.g., due to impact on operator controllability). Affects the test and validation strategy due to the challenges of testing fail operational systems. Affects the overall software complexity.

3.3.4.2 Software Scheduling Taxonomy

To implement software with the required MoC, criticality, and architecture, software functions must be sequenced and scheduled to execute within the boundaries of functional and performance requirements for timing and determinism, jitter, and latency.

Operating systems provide the foundation for standardized architectures and often implement standardized service layers with industry-accepted APIs for memory access, timing and synchronization, peripherals, device drivers, and communications.

In simpler applications, particularly where libraries of device drivers are not needed (e.g., Gen 1), "software schedulers" are used.

3.3.4.2.1 Operating Systems

Table 16 provides the framework classifications for software scheduling and OS technology.

Standardized operating systems (e.g., OSEK, AGL) implement architectural interfaces through virtualization and abstraction layers between various service layers and the ECU. This allows E/E suppliers to develop and innovate with rapidly changing designs, while using consistent "plug-and-play" interfaces and APIs to integrate emerging technology within standard architectures.

[219] ISO 26262 classifies faults as single point, residual, detected multi-point, and latent-multi point; for further information on fault management see ISO 26262-5:2011, Annex B, Figure B.2.

The ECU and system logical architectures, HW/SW architectures, and OS implementations are tightly coupled and highly-integrated, and they share dependencies across the spectrum of system requirements.

In support of standardized operating systems and abstraction layers, manufacturers use calibration methods to implement parameterized software within automotive ECUs and to allow ECU software to be targeted and tuned for a wide variety of applications. This allows delivery of adaptable software from Tier I suppliers to a multitude of vehicle manufacturers, and it requires minimal modification of the supplied software by the vehicle manufacturer to deploy the supplier's software.

Table 21 summarizes effects of operating system characteristics on lifecycle practices.

Table 21. Impacts of operating system characteristics on SDLC practices

Operating System Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
OS Technology	• Undefined OS, simple scheduler (one type of software scheduler), purpose built, proprietary RTOSs • Emerging commercial operating systems	• Standardized RTOSs (e.g., OSEK) • Standardized OS compatibility with standardized HW/SW architectures (e.g., OSEK and AUTOSAR CP) • Abstraction layers (e.g., Virtual Function Bus (VFB), ECU and Microcontroller abstraction) allowing multipurpose/general purpose IO and programming interfaces • Standardized specifications for device drivers, communications bus interfaces (See AUTOSAR Classic)	• Service oriented, multi-function programming in thread safe OS (e.g., POSIX) • Increasing use of function clusters and sensor fusion allow for SOA
Impact on SDLC	• Lack of consensus standards for interoperability • Dedicated programming interfaces and interface definitions • Application specific device drivers • Manual C & Assembly language coding	• Standardized programming, device driver, and logical interfaces allowing reuse and interoperability (e.g., AUTOSAR CP, OSEK Operating System) • Integrates with model based design workflow	• Increasing use of POSIX standard programming interfaces and multi-thread programming techniques for parallel programming in heterogeneous processing architectures. • Open software environment for “plug-and-play” services. • Increasing use of multi-threaded and redundant MoC and OS.

3.3.4.3 Hardware Taxonomy

Automotive *embedded systems* are built so that the underlying hardware and software architectures are tightly integrated, allowing the system to implement the required critical modalities and MoC and to meet functional and performance requirements. Hardware technology for automotive E/E systems has evolved to become increasingly specialized and is designed for specific purposes and applications within the motor vehicle.

3.3.4.3.1 Hardware Technology

The recent architectural evolution of automotive E/E systems is driven by rapid innovation, commercialization, and commoditization across a wide range of hardware and electronic components, processors, FPGA, GPU, memory, peripherals, smart sensors, actuators, wideband sensors (e.g., LIDAR, RF, radar, ultrasonic), and communications network technology.

Documented taxonomic sources for automotive E/E hardware are available in published literature and were used for this study. The representative advancements shown in the framework (Table 22) provide context for architectural changes and related effects on the SDLC and lifecycle framework.

Table 22. Effects of hardware architecture and implementation on SDLC practices

Hardware Technology Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Hardware Technology	• Sensors and actuators interface with ECU through semi-custom and dedicated peripheral interfaces including analog, discrete//timing (e.g., for ratiometric and differential sensors, hall effect sensors, switches, encoders) • Dedicated communications busses (ECU - ECU) • 8 and 16 bit single core microprocessors (e.g., Motorola 6802, 68HC11)	• Standardized peripheral, bus, and IO interfaces between sensors and actuators and the ECU • Emerging smart sensors and actuators (e.g., with onboard processing, FPGA, bus interface with ECU) • Increasing use of digital, integrated circuit-based sensors (e.g., CMOS, MEMS) • Distributed, decentralized control systems organized by functional groups including domain and bus classification • Functional groupings by bus category across bus endpoints (e.g., A, B, C, D) and functional requirements (e.g., sensors and actuators, ECU-ECU, broadband (e.g., camera, LIDAR)) • Functional groupings by vehicle subdomain (e.g., chassis, powertrain) • Emerging multicore and heterogeneous processors (e.g., microprocessor unit and FPGA) • Emergence of memory mapped IO for efficient Real Time IO processing, increased memory address space and low cost memory for larger application size	• Adaptive interfaces • Progression from distributed, networked ECU architecture to centralized architecture with integrated domain controllers integrated with clusters of functional controllers • Low level IO (feedback controls) through functional controllers (Gen 2), High level IO interfaces (Authority Guidance, and Trajectory) through SoA • Increasing use of Sensor Fusion and SoA • Backbone integration of domain architectures (e.g., via TSN) between integrated domain controllers and functional busses (e.g., low level and closed loop control systems (CAN), broadband sensors (MOST)) • Reduced onboard ECU count and increased ECU complexity • Emerging GPGPU, many core processors, System on Chip architectures. • Emerging specialized automotive processing architectures for deep learning and sensor processing (e.g., data flow processor (DFP))
Impact on SDLC	• Hand coded, custom software, custom device drivers, extensive use of assembly language	• Increasing use of the C programming language and MBD/ACG • Standardized device drivers, standardized function libraries • Increasing use of MIL/HIL/SIL	• New MoC related to service-based architectures • New lifecycle paradigms including increased use of virtualization and "game testing" • Increasing use of emerging programming languages (e.g., SYCL, CUDA) • Increasing use of agile programming techniques, increased use of redundancy and fail safe development techniques • Increasing use of multithreaded programming techniques

Sensors, Actuators and Peripherals

E/E systems in framework "Gen 1" and "Gen 2" incorporate a variety of ratiometric and differential sensors for the measurement of various vehicle parameters (e.g., hall-effect sensors for crank and cam position, thermocouple-based temperature sensors, air fuel sensors, and knock sensors220). The evolution and commoditization of solid-state sensing technology, processing and communications busses, and software/OS architectures has led to advancement of a host of new sensing technologies across the full spectrum of automotive E/E applications.[221]

Automotive actuation technology has evolved in parallel with sensing technology and covers a range of applications including active steering, power steering, electromechanical brakes, clutch and shift actuators, suspension, damping and stabilization actuators, heating, ventilation, air conditioning, starter-generators, and emerging x-by-wire (e.g., steer-by-wire, brake-by-wire).[222]

The evolution of E/E systems from "Gen 1" of the framework to "Gen 2" is partially characterized by standardization of ECU peripheral interfaces (Table 9). During evolution from framework Gen 1 to Gen 2, mechanical devices were replaced by open and closed loop electromechanical devices, followed by "smart devices" with onboard controllers and integrated communications busses. Moving into Gen 3, vehicle electromechanical architectures are anticipated to continue to replace mechanical linkages and to incorporate emerging safety critical "by wire" systems. In addition to the Gen 1 and Gen 2 sensor types, Gen 3 architectures are increasingly implementing general-purpose sensors (e.g., RADAR, LIDAR). The implication for the SDLC is that these general-purpose sensors require more complex software, such as "sensor fusion" over service-oriented architectures and function clusters. To help address the increasing complexity associated with sensor fusion, which will substantially impact the SDLC, there is ongoing standardization activity. For example, the semantic interfaces of sensor systems is being standardized in the ISO 23150, "Data communication between sensors and data fusion unit for automated driving functions." This consensus standard defines which sensor data or signals are mandatory or optional and how are they defined, e.g., in terms of coordinate systems and units. For each sensor type (radar, LIDAR or camera), ISO 23150 will specify many optional sensor data or signals in addition to the required signals.[223] To reduce development costs, especially in terms of functional safety, the set of options must be fixed at design-time. The consensus standard is used to specify logical sensor interfaces within AUTOSAR, including the AUTOSAR Adaptive Platform Standard, which will provide a specification to handle the optional sensor data and signals during design-time.[224]

---

[220] Knock sensor detect vibrations that come from an irregularity in combustion and send a signal to the engine control computer, which then adjusts timing to compensate.

[221] For more information on the evolution and application of automotive sensing technologies see First Sensor Inc.. (2021, February 03). *Sensor technologies for automotive systems.* AZoSensors. www.azosensors.com/article.aspx?ArticleID=1241 and Fleming, W. (2001, December). *Overview of automotive sensors*; IEEE Sensors Journal, 1(4).

[222] Iles-Klumpner, D., Serban, I., & Risticevic, M. (2006, September 6-8). *Automotive electrical actuation technologies*. 2006 IEEE Vehicle Power and Propulsion Conference, Windsor, UK. https://doi.org/10.1109/VPPC.2006.364364

[223] van Driesten, C., & Schaller, T. (2019). *Overall approach to standardize AD sensor interfaces: Simulation and real vehicle*. Springer Fachmedien Wiesbaden. www.springerprofessional.de/en/overall-approach-to-standardize-ad-sensor-interfaces-simulation-/16401376

[224] AUTOSAR. (2019, November 28). *Explanation of sensor interfaces* (Document ID 913). www.autosar.org/fileadmin/user_upload/standards/adaptive/19-11/AUTOSAR_EXP_SensorInterfaces.pdf

Communications Busses

The rapid evolution of automotive communications busses is a key trend captured in the "Gen 2" taxonomy and reflects the increasingly decentralized, distributed, and functionally organized automotive E/E paradigm shift between "Gen 1" and "Gen 2." Throughout the 1990s and 2000s, communications busses were developed for specific applications and function groupings within the vehicle (e.g., Class A, B, C, D [Table 13]).

The evolution of "Gen 3" architectures is reversing this, with architectural trends across the industry moving toward increasingly centralized domain controllers, responsible for function clusters implemented through SOA. Looking forward, through insight gained from the emergence of "Gen 3" vehicles and from the Adaptive AUTOSAR framework, the communications bus strategies deployed in "Gen 3" appear to be moving toward centralized architectures based on networked domain controllers integrated using "backbone" bus structures such as Ethernet-based TSN technology. Functional clusters (e.g., closed-loop control systems/sensors/actuators) will continue to be integrated using lower-level busses (Class A, B), while broadband and sensor fusion applications will be hosted on Class D networks.

Processors

Microprocessor capabilities have progressed throughout the history of the vehicle ECU.

During framework "Gen 1" and the beginning of "Gen 2", processing capabilities were added across the E/E architecture through addition of more decentralized, distributed, and networked ECUs with increasingly standardized ECU and processing architectures. Microprocessing power was applied based on a relatively limited number of processing options, and driven by cost/volume sensitivity (e.g., 16- and 32-bit controllers for powertrain applications and 8- and 16-bit controllers for chassis applications).

Through the course of "Gen 2," manufacturers began to produce specialized automotive microcontrollers with progressively more power in terms of clock speed, instruction size, instruction set and architecture (e.g., RISC, MIPS, PPC, and ARM architectures). The cost of microcontrollers has decreased through the course of Gen 2, allowing greater use in vehicle systems. Moving into Gen 3, specialized processors (e.g., GPGPU) used in integrated domain controllers may reverse this trend.

As high-performance computing capabilities have become affordable over the past decade, an increasing number of heterogeneous processing and SoC)options have become available, contributing to the evolution of increasingly capable ADAS and ADS systems. Moreover, the progression of hardware architectures into "Gen 3" has yielded specialized automotive microcontroller architectures targeted at critical applications, with lockstep capability for functional safety, "smart watchdog" capabilities, and dedicated peripherals for hardware-based cybersecurity.[225] External computing power (cloud computing) has also been used in order to handle the

[225] Fault-tolerant "lockstep" systems are designed to run parallel operations at the same time in order to ensure that the operating state of the controller does not change until all required operations are complete.

increasing complexity and dataflow of non-safety-critical automotive software, such as infotainment content, high-bandwidth map services, OTA functional upgrades, remote diagnosis, emergency-call processing, and connectivity with external infrastructure.[226] For example, Amazon Web Services touts: "You can use [the Connected Vehicle Solution] to address a variety of use cases such as voice interaction, navigation and other location-based services, remote vehicle diagnostics and health monitoring, predictive analytics media streaming services, vehicle safety and security services, head unit applications, and mobile applications."[227]

Increased use of multicore MPUs, FPGA, GPGPU/GPU, and Dataflow processors has driven the consolidation of Gen 3 ECU functions and the evolution of *integrated domain controllers*—and the evolution from decentralized to centralized vehicle system architectures.[228]

3.3.4.4 Computer Control Taxonomy

The evolution of electromechanical control systems, and the progressive differences between "Gen 1, 2, and 3" E/E implementations can be characterized in terms of control theory. The following sections provide classifications for control systems and control strategies used in automotive E/E systems.

3.3.4.4.1 Control Strategies and Control Systems

In the framework, "Gen 1" is characterized by the emergence and implementation of closed-loop electromechanical control systems and by progressive replacement of mechanical components with electromechanical systems.

As technologies related to sensors, actuators, processors and ECU architectures have evolved, so have control strategies. Advancements in sensing and actuation technology, AI/Machine Learning, and advanced broadband sensors have led to increasingly sophisticated hierarchical control frameworks incorporating supervisory, trajectory, and real-time closed-loop control layers**.** "Gen 2" and "Gen 3" control systems are increasingly characterized by high-level *supervisory* functions performed by computers in place of human operators. This may include both simple functions such as speed control and parking assist, and more advanced functions such as route planning (e.g., guidance navigation and control) and collision avoidance (e.g., trajectory control). Supervisory control systems and associated fault management strategies are tightly coupled and impact the ASIL rating for the system, complexity of the system, and cost. Table 23 summarizes effects on lifecycle practices due to influences related to control systems and control strategies.

---

[226] Ebert, C., & Favaro, J. (2017, May-June). *Automotive software*. IEEE Software. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7927926 ; Milani, F. & Beidl, C. (2018, December 5-7). *Cloud-based vehicle functions: Motivation, use-cases and classification*. 2018 IEEE Vehicular Networking Conference, Taipei, Taiwan. doi: 10.1109/VNC.2018.8628342. https://ieeexplore.ieee.org/document/8628342

[227] Amazon Web Services. (2021). *Connected vehicle solution* [Web page and portal]. https://aws.amazon.com/automotive/solutions/?nc=sn&loc=3

[228] For more information on integrated domain controllers and centralized vehicle control systems architectures see Reinhardt, D., & Kucera, M. (2013, February 19-21). *Domain controlled architecture - A new approach for large scale software integrated automotive systems.* Proceedings of the 3rd International Conference on Pervasive Embedded Computing and Communication Systems, Barcelona, Spain. doi: 10.5220/0004340702210226.

Table 23. Impacts of control strategies and control systems on SDLC practices

Control Strategy Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Control Strategy	• Emerging open loop, closed loop digital controls • Calibration of control systems allowing deployment on many model variants • Evolution of digital control theory leading to increasing number of distributed digital systems onboard the vehicle. • Evolution of MoC including real time, distributed, finite state machine, dataflow process network, discrete event, SR. • Emerging and evolving standardization and integration of closed loop digital control strategies, MoC (e.g., finite state machines), and associated tools (e.g., Simulink), and standardized process reference models (e.g., V-cycle) • Evolution of sensors and actuators for closed loop control applications (e.g., crank/cam sensors, lambda sensors, electronic fuel injector and ignition actuators)		• Evolving use of supervisory and authority (e.g., guidance and trajectory) control strategies • Emerging use of AI and machine learning for supervisory control, with emerging MoC (e.g., adaptive control, model predictive control, neural network) implemented using concurrent and parallel, multi thread, pragma based, accelerator, SOA, and AI • Increasingly integrated functions (e.g., function clusters, SOA), and sensor fusion • Emerging modes of criticality related to AI/machine learning (e.g., trustworthiness, morality).
Impact on SDLC	• Evolution of model based programming and related V-Cycle process around MoC for control systems development • Reference models for the development of closed loop control systems • Evolution of MBD/ACG and MIL/HIL/SIL practices for distributed control systems.		• Increasingly difficult to perform deterministic test coverage due to the large number of widely varying use cases and scenarios • Emerging MoC related to parallel and redundant control systems and SOA • Increasing levels of software complexity due to increasing ASIL and fault management • Emerging techniques for virtual V&V (e.g., game testing) • Requirements for new consensus standards and defined MoC related to AI and Machine Learning.

3.3.4.5 Communication Taxonomy

This classification captures the overall strategy that is used to implement digital message and signal passing mechanisms between control elements (e.g., controlled subsystem, controller) and E/E subsystems.

3.3.4.5.1 Communications Strategy, Network Architecture, and Topology

Automotive E/E systems consist of integrated and networked subsystems including ECUs, actuators, and sensors. The evolution of E/E communication strategies corresponds with the evolution of logical and system architectures (Figure 10, Figure 11, and Figure 12).

Domain functions across Body Electronics, Powertrain, Chassis, Occupant and Pedestrian Safety, Multimedia, and Telematics subdomains are constrained by performance characteristics required of communications on each subdomain network. Network class specifications allow networked E/E architectures to be partitioned to support performance and functional requirements, including bandwidth, cycle time, determinism, and fault latency (Table 13).

A number of changes in "Gen 3" architectures have the potential to impact network and communications consensus standards. Emerging ADAS and ADS applications are driving demand for high bandwidth deterministic communication bus technology (e.g. TSN) and the consolidation of distributed software functions within integrated domain controllers. In contrast to traditional bus-based approaches, NoC strategies are emerging to address shortcomings inherent in traditional bus architectures as developers integrate functions on integrated domain controllers.

The research revealed that E/E engineers are anticipating that the number of onboard busses and endpoints for class A and B busses will be reduced and simplified, since "Gen 3" vehicles use fewer ECUs interconnected by deterministic Ethernet (e.g., TSN) "backbones."[229] Based on conversations for this research, the consensus of E/E engineers seemed to be that any reduction in architectural complexity that is achieved due to reductions in the number of onboard ECUs will be more than offset by increased complexity of system architectures for the remaining ECUs—for instance, incorporating SOA and function clusters, NoC, heterogeneous computing (e.g., GPU, FPGA, and ECU), and sensor fusion.

Sensor proliferation and the emergence of networked "smart actuators," is expected to increase the number of bus endpoints and demands on class B and C busses. Several of the experts indicated that they expect the automotive industry to continue to support LIN, CAN, and FlexRay consensus standards for many years.

[229] See Table 13 for classification of Type A, B, C, D busses and associated applications by subdomain.

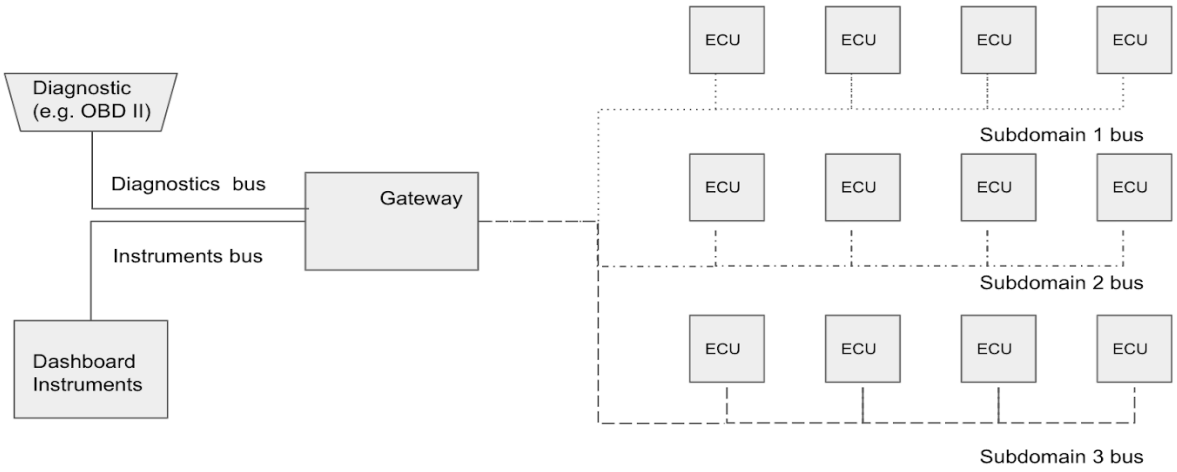

Figure 13. Representation of a generic Gen 2 network strategy showing several connected ECUs.

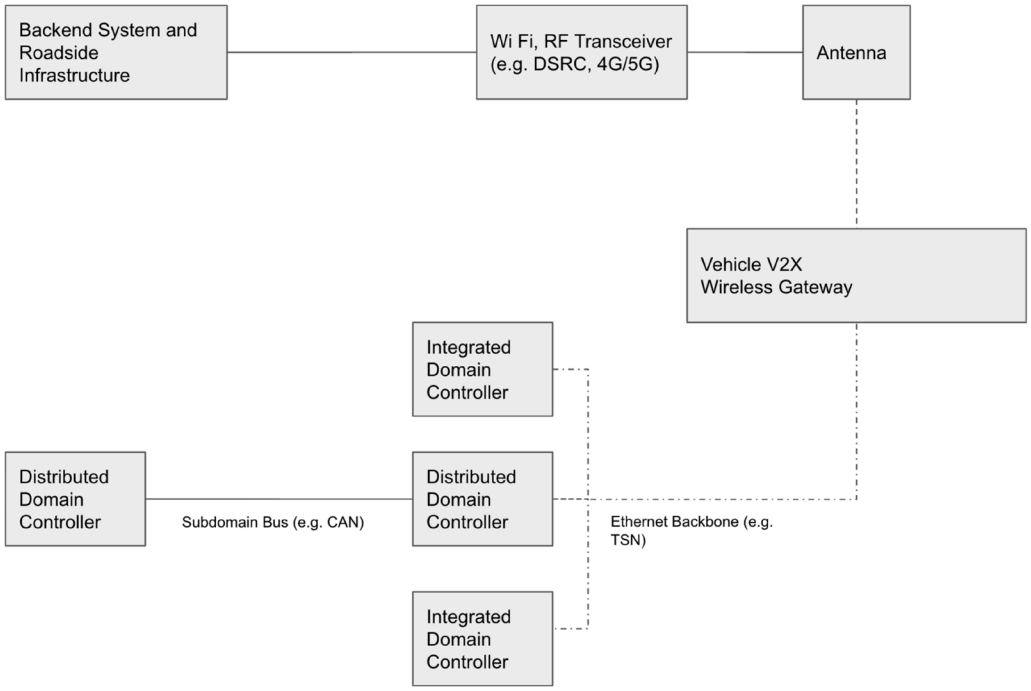

Figure 14. Representation of a generic Gen 3 network architecture showing reduced network connections through use of distributed domain controllers.

Table 24. Impacts of communications strategy and network architecture on SDLC practices

Communications Strategy, Network Architecture and Topology with Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Communications Strategy, Network Architecture, and Topology	• Emerging independent and isolated ECUs (e.g., limited or no ECU-ECU integration) • Sensors directly connected to ECU • Emerging automotive communications bus consensus standards	• Decentralized, distributed control systems architectures • In-vehicle communications bus and ECU-ECU (e.g., CAN) networks • Emerging V2X networks • Emerging smart and networked sensors and actuators	• Increasingly centralized control systems architectures • ECU-ECU, networked smart sensors and actuators connectivity between domain controllers • In-vehicle backbone bus (TSN) and emerging NoC between integrated domain controllers • Integration of on and off board computing (e.g., vehicle to vehicle, V2X) via wireless/RF (e.g., 4G, 5G, satellite)
Impact on SDLC	• Direct and proprietary IO interfaces using hand coded device drivers (assembly language) • Limited reuse • Emerging requirements for system level architecture resulting from ECU-ECU networks	• System level software design, network design • Rest bus testing/HIL integration testing • Emergence of model based architectures and systems engineering • Increasing levels of single mode mixed criticality on distributed ECUs	• Increasing support for infrastructure required for in-field operation and sustaining engineering for Software Defined Car with OTA updates, including integration with cloud, V2X • Increasing levels of security criticality, demand for system-level ("top-down") approach for several critical modalities • Increasing use of inter-process communication and multi thread programming techniques within integrated domain controllers • Increasing levels of mixed criticality across several modalities due to integration of functions into a single domain controller, SOA, integration of on/off board functions

3.3.4.6 Software Implementation Taxonomy

This classification captures the programming languages, tooling, and programming and computing theory that allow the software to be implemented in the E/E system.

3.3.4.6.1 Tools and Programming Languages

There is a close relationship between software tools, programming languages, and associated SDLC practices.

Tools

Automotive software workflows are built around a combination of commercial off the shelf and custom "homegrown" tools.[230] Table 33 provides a representative list of some of the more common tools used for automotive software development. ISO 26262 introduces the concept of qualifying tools for use in the functional safety process to provide confidence in software tools and ensure their suitability to support ISO 26262 activities.

Integration of COTS tools into an organizational workflow may require process changes, and conversely, custom tools are often developed in order to facilitate process requirements. Often, tool suppliers work closely with vehicle manufacturers in order to develop lifecycle tooling that satisfies the automotive suppliers' SDLC process requirements.

Programming Languages

Software development during Gen 1 and Gen 2 of the framework was primarily performed using assembly language and C.[231] Emerging models of computation and computing and processing architectures (e.g., GPU, FPGA) have led to the emergence of a range of new programming languages that are being used or are likely to be used as integrated domain controllers emerge with increasing requirements for data and image processing and performance.

3.3.4.6.2 Models of Computation

MoC describe the types of mathematical functions that a computer architecture is capable of processing.

Some of the most familiar computing devices, such as personal computers, are designed to handle a broad range of MoC. In contrast, the computer architectures used in vehicles are dedicated embedded systems, designed around specific MoC. Earlier in this report, we described the MoC that are predominantly used in automotive software.

Table 25 provides a summary of MoC by framework generation, with impacts on the automotive SDLC.

[230] During the research phase of this project, one expert described a recent shift toward COTS functional safety tools in order to avoid issues related to self-qualification of tools.

[231] Model-based development may be defined as a third "programming language;" however, the models are used to generate C source code.

Table 25. Impacts of Model of Computation on SDLC practices

Influences due to Models of Computation on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Model of Computation	• Evolution of real time, distributed, finite state machines, dataflow process networks, discrete event, SR MoC		• Emerging MoC including multi thread, pragma based, accelerator, SOA, AI
Impact on SDLC	• Emerging real time, discrete event, and SR MoC drive process reference model, RTOS and tool selection (e.g., MBD/ACG with RTOS target and step-wise solver)		• Multi thread and SOA MoC drives requirements for thread safe OS (e.g., POSIX), MBD tools with POSIX (or similar) support, lockstep programming. • Pragma based and accelerator MoC drive SDLC requirements for tool/language support (e.g., object oriented, CUDA)

3.3.5 Process Requirements Research Theme

Table 26 summarizes process requirements that act as non-functional requirements for the automotive SDLC.

The table incorporates framework elements shown in Figure 8. As the framework evolves, this section of the taxonomy can be expanded to include additional classifications and subclassifications.

Section 3.3.3.3.2 presented taxonomy of various consensus standards that are used as software process reference models across the automotive industry. The following sections classify these consensus standards with respect to these processes.

3.3.5.1 Process Reference Model Taxonomy

Section 2.4.6.1 described the evolution of the V-Cycle used in automotive software development processes.

ASPICE, ISO 26262, and emerging ISO 21434 are all based on the V-Cycle as the underlying process reference model and share many common process groups and process steps. This can be seen in the taxonomy of SDLC practices, which captures the first two, nearly identical levels of the ASPICE and ISO 26262 consensus standards at the classification and subclassification levels. This will hold true for ISO 21434 based on preliminary drafts of the standard. Deeper levels of the taxonomy reflect increasing variability between reference processes, indicating that the number of branches in the taxonomy will greatly increase as classification layers are added.

ASPICE may be viewed as the most general of the consensus standards because it is designed to describe key attributes of a mature SDLC, and not specific process steps. Reference models for critical lifecycle practices and for implementations of software development methods (e.g., critical reference models) provide more specific and detailed process steps.

3.3.5.2 Critical Reference Model Taxonomy

The framework differentiates between “critical” and “process” reference models because as critical reference models emerge for cybersecurity (ISO 21434), trustworthiness, AI and machine learning, and other modes of criticality, it is likely that the taxonomy will branch here, and a classification system will be required in order to identify different characteristics of the various reference models.

The critical reference taxonomy is likely to become the framework focus during the next generation of vehicle development and corresponding framework frames. This is because the evolution of criticality and associated SDLC process steps are key differentiators between framework generations. Understanding the differences between critical process paradigms is crucial for understanding the evolution and uniqueness of automotive software development practices.

There is a relative harmonization between the few critical lifecycle processes used today in the automotive industry. Process harmonization, however, is a growing concern as lifecycle practices become more complex and as MoC, critical modalities, and automotive computing architectures incorporate more and more dissimilar process paradigms.

Requirements for evolving MoC and critical modalities drive different process requirements. For example, ISO 26262 requires hazard and risk analysis in the concept phase, and ISO 21434 (preliminary) requires threat and remediation assessment. As might be expected, the supporting process steps for a threat surface analysis on a networked E/E system are much different than a hazard analysis on a closed-loop, embedded control system. For this reason, as diverging process methods are captured within deeper levels of the taxonomy of SDLC practices, the taxonomy is expected to become more complex and more differentiated between classification elements.

Table 26. Taxonomy of process requirements with impacts on SDLC constraints and requirements

Processes with Influences on the Automotive SDLC				
Research Theme	**Classification**	**Subclassification**	**Change Factors**	**Framework References**
Process Requirements	Process reference model	(not classified)	V-Cycle SDLC (e.g., ASPICE), critical V-Cycle SDLC (e.g., ISO 26262 functional safety), emerging Agile, "Bottom-Up" approach based on Element out of Context (EooC), Bottom-up critical analysis (e.g., FMEA), Emerging system criticality model ("top-down") and top-down critical analysis (e.g., STPA), emerging SOTIF	[44] [45] [46] [49]
	Critical reference model	(not classified)		

Table 27. Impacts of software processes on SDLC practices

Process Influences on the Automotive SDLC			
	Generation 1	**Generation 2**	**Generation 3**
Process reference model and Critical Reference model	• Ad-hoc, emerging generic and proprietary V-Cycle	• Standardized process reference model and V-Cycle including ASPICE, ISO 26262, ISO 21434 • Emerging Agile • Emerging SOTIF	• Domain controllers developed using ASPICE V-Cycle, ISO 26262 V-Cycle, ISO PAS 21434 • V-Cycle processes for domain controllers using MBD/ACG techniques • Agile programming practices on integrated domain controllers for SOA and parallelization, used with emerging MoC including pragma, accelerator, and parallel programming
Impact on SDLC	• Ad-hoc software development and emerging process paradigms	• ASIL provisions for "bottom-up" approach based on Element out of Context (EooC) and distributed development, [232] • Emerging system safety model ("top-down") and top-down critical analysis and assessment (e.g., STPA)[233]	• Emerging integration of V-Cycle and Agile process models • Integration of emerging MoC and process reference models and tools • Increasing need for process harmonization between emerging and evolving critical process models

[232] For more information on requirements decomposition, and top-down versus bottom-up considerations in mixed criticality systems, see D'Ambrosio & Debouk, 2013.

[233] See "Risk Management" and the Taxonomy of Software Development Lifecycle Practices (Table 6). Risk analysis and assessment are part of the risk management sub category in the risk management process group.

3.3.6 Software Development Lifecycle Practices Taxonomy

The following section provides a taxonomy of SDLC practices that are driven by process change factors.

Table 28 summarizes the Framework Taxonomy of Software Lifecycle Practices for Framework v1.0 (see also, Figure 8). The SDLC process and sub process categories are derived from ISO 26262 and ASPICE. The "characteristics" column in Table 28 provides unclassified concepts and definitions that may be classified in future revisions of the framework.[234] Further reading on these concepts may be found in the literature cited in the references column.

The taxonomy of SDLC practices presented in this section supplements the taxonomy of process change factors described in Section 3.3.1 . Together, the change factors and SDLC practices taxonomies comprise the complete framework taxonomy. Once the complete framework taxonomy is established, it may be reduced into comparative elements, allowing it to be compared and contrasted with other transportation sectors and industries.

[234] One of the key trends identified in the study is the increasing level of harmonization behind automotive software standards ISO 26262, ISO 21434, and ISO PAS 21448 are designed with similar structure around a similar process reference model based on the V-cycle. For these reasons, ISO 26262 and ASPICE were chosen as the *a priori* models and form the foundation of frame 1 of the framework.

Table 28. Software development lifecycle practices for automotive software taxonomy

Software Development Lifecycle Practices Taxonomy - Version 1.0			
SDLC Process Category	**SDLC Process Sub Category**	**Characteristics**	**References**
Management of Criticality	Item Definition	Indirect SDLC activity	[45] [46] [48]
Supporting process group	Quality Assurance, Verification, Joint Review, Documentation, Configuration Management, Problem Resolution Management, Change Request Management		
Management process group	Project Management, Risk Management, Measurement	Indirect SDLC activity	
Process improvement group	Process Improvement	Indirect SDLC activity	
Reuse process group	Reuse Program Management	Model-based component libraries, parameterized models, model repositories, data dictionary, modular logical/functional architecture, supplier interoperability/SEooC, software calibration, open source software	[48] [87] [88] [89]
Concept phase	Item Definition	Risk analysis, Impact assessment, Risk assessment (or critical analysis and assessment (e.g., TARA, HARA)), risk treatment, item definition, initiation of critical lifecycle, concept for criticality	[45] [46] [48] [98] [99] [100] [101]
Risk management	Risk Management		

SDLC Process Category	SDLC Process Sub Category	Characteristics	References
System engineering process group	Requirements Elicitation	Requirements engineering (needs analysis, requirements analysis and requirements specifications), requirements management and traceability and tools (e.g., DOORS), model based requirements (e.g., Simulink/Stateflow, UML), Written Requirements (e.g., word document, spreadsheet)	[45] [46] [48] [90] [91] [90] [91]
	System Requirements Analysis		
	System Architectural Design	Model-based software architecture, UML/Simulink/Stateflow models and system design models (software and hardware components (SWC), (HWC), networks, Interface Control Definitions (ICD)), Diagnostic Architecture, integrated consensus standards based reference architecture (e.g., AUTOSAR Basic Software, VFB, RTE), Architectural Description Language (e.g., EAST ADL), integration with PLM Tools and behavior modeling tools (BMT), AUTOSAR XML description files (ARXML)	[45] [46] [48] [92] [93] [94]
	System Integration and Integration Test	MIL/HIL/SIL Simulation and Test, Dyno Test, Road/Field in-vehicle testing, Virtual drive testing (e.g., Mechanical Simulation CarSIM, IPG CarMaker), test automation (e.g., dSPACE AutomationDesk, National Instruments TestStand), traceability and test integration with requirements tools (e.g., AutomationDesk integration with DOORS), test reporting (e.g., TestStand, AutomationDesk)	[45] [46] [48] [95] [96] [97]
	System Qualification Test		
Software engineering process group	Software Requirements Analysis	Requirements engineering (needs analysis, requirements analysis and requirements specifications), requirements management and traceability and tools (e.g., DOORS), model-based requirements (e.g., Simulink/Stateflow, UML), written requirements (e.g., Word document, spreadsheet)	[45] [46] [48] [90] [91] [90] [91]
	Software Architectural Design	MIL/SIL, UML, MBSE/Simulink/Stateflow, RCP, MBD/ACG	[45] [46] 48] [92] [93] [94]
	Software Detailed Design and Unit Construction		
	Software Unit Verification	MIL/HIL/SIL Simulation and Test, HIL unit and integration testing, virtual drive testing, rest bus testing, test automation, traceability and test integration with requirements tools, test reporting	
	Software Integration and Integration Test		

SDLC Process Category	SDLC Process Sub Category	Characteristics	References
Software engineering process group	Software Qualification Test	Develop software qualification test strategy Develop specification for software qualification test Select test cases Test integrated software Establish bidirectional traceability Ensure consistency Summarize and communicate results	[45] [46] [48] [95] [96] [97]
Product development at the system level	System design	Initiation of product development at the system level, Specification of critical requirements (e.g., requirements engineering (needs analysis, requirements analysis and requirements specifications)), Requirements management and traceability and tools (e.g., DOORS, model-based requirements (e.g., Simulink/Stateflow, UML), Written requirements (e.g., word document, spreadsheet))	[45] [46] [48] [103] [91] [92] [93] [94]
		Model based software architecture (e.g., UML/Simulink/Stateflow models and system design models (software and hardware components (SWC), (HWC)), Network architecture, Interface Control Definitions (ICD), Diagnostic architecture, Integrated consensus standards based reference architecture (e.g., AUTOSAR BSW, VFB, RTE), Architectural Description Language (e.g., EAST ADL), Integration with PLM Tools and behavior modeling tools (BMT), AUTOSAR XML description files (ARXML)	
	Item integration and testing	MIL/HIL/SIL Simulation and Test, Dyno Test, Road/Field in-vehicle testing, virtual drive testing (e.g., Mechanical Simulation CarSIM, IPG CarMaker)), test automation (e.g., dSPACE AutomationDesk, National Instruments TestStand), traceability and test integration with requirements tools (e.g., AutomationDesk integration with DOORS), Test reporting (e.g., TestStand, AutomationDesk)	[45] [46] [48] [95] [96] [97]
	Validation of critical requirements		
	Critical assessment		
	Release for production		

SDLC Process Category	SDLC Process Sub Category	Characteristics	References
Product development at the software level	Initiation of product development at the software level	Specification of software requirements (e.g., requirements engineering (needs analysis, requirements analysis and requirements specifications), requirements management and traceability and tools (e.g., DOORS), model based requirements (e.g., Simulink/Stateflow, UML), written requirements (e.g., word document, spreadsheet)	[45] [46] [48] [90] [91]
	Software architectural design	MIL/SIL, MBSE (e.g., Simulink/Stateflow/System Composer,), UML (e.g., SysML), RCP (e.g., Simulink/TargetLink), ACG (e.g., TargetLink, SCADE)	[45] [46] [48] [92] [93] [94]
	Software unit design and implementation	MIL/SIL, MBSE/Simulink/Stateflow, RCP, ACG (e.g., TargetLink, SCADE)	[45] [46] [48] [95] [96] [97]
	Software unit testing	MIL/HIL/SIL simulation and test, HIL unit and integration testing, virtual drive testing, rest bus testing, test automation, traceability and test integration with requirements tools, test reporting	
	Software integration and testing	MIL/HIL/SIL Simulation and Test, Dyno Test, Road/Field in-vehicle testing, virtual drive testing (e.g., Mechanical Simulation CarSIM, IPG CarMaker), Test automation (e.g., dSPACE AutomationDesk, National Instruments TestStand), Traceability and test integration with requirements tools (e.g., AutomationDesk integration with DOORS), test reporting (e.g., TestStand, AutomationDesk)	
	Verification of critical requirements	MIL/HIL/SIL simulation and test, HIL unit and integration testing, virtual drive testing, rest bus testing, test automation, traceability and test integration with requirements tools, test reporting, critical assessment, release for production	
Production	N/A	On and off board diagnostics (e.g., OBD, OBD-II), OTA updates, in field service updates and installation	[101] [102] [103] [104]
Operation	Operation, service, and repair		

3.3.7 Taxonomy Reduction and Comparative Framework

3.3.7.1 Applications

The framework taxonomy provides a foundation for defining terminology, concepts, and relationships between influencing "change factors," non-functional requirements, constraints, and related SDLC process steps, and the lifecycle practices used for automotive critical software. The framework taxonomy can be used to identify fundamentally different approaches to the SDLC, focusing on process steps, constraints, and non-functional requirements. It is possible to describe and compare these approaches using a subset of the framework taxonomy. For instance, this framework could be used to:

- Better understand the evolution of automotive software development, as shown by comparing across Gen 1, Gen 2, and Gen 3 systems.
- Compare and benchmark automotive SDLC best practices against those in other industries.
- Develop harmonized processes through comparative process analysis within the automotive industry.
- Conduct process, cost, and complexity analysis.

Appendix B proposes and illustrates one such methodology for reducing the framework taxonomy and developing a comparative framework. Examples are used from the evolution of automotive software development across Gen 1, Gen 2, and Gen 3.

4 Conclusion

During this research activity, the team performed thematic framework synthesis and inductive analysis activities resulting various technology development eras. The pictorial display on Figure 15 synthesizes the key concepts and governing factors of interest with respect to generations of technology. Content within this report, including the various framework tables that can be used to prompt comparisons in specific automotive software areas, are based on the following activities:

- Developed an *a priori* reference framework ("Frame 0") based on ASPICE and ISO 26262.
- Established Frame 1.0 research questions and themes by incorporating Volpe/NHTSA-provided *a priori* questions and by generating sub-questions, developed and led by team subject matter experts for each of the research themes. The framework developed from Frame 1.0 can provide an initial starting point and can be revisited in the future as the software development process in the automotive industry continues to evolve.
- Conversed with industry experts, including automotive functional safety engineers, systems and software engineers, engineering managers, consensus standards experts, software architects, and process engineers.
- Performed a literature review of publications, conference presentations, industry reports, consensus standards publications, journals, etc. (Publications that were used as part of the literature review are captured in the Mendeley database at Mendeley.com.)
- Recorded evidence related to research questions in the inductive synthesis matrix.
- Created a traceability matrix allowing thematic evidence to trace to sources in the inductive synthesis matrix.
- Performed a framework analysis and generated the "Frame 1.0" revision (Figure 15), incorporating evidence that was captured during research.

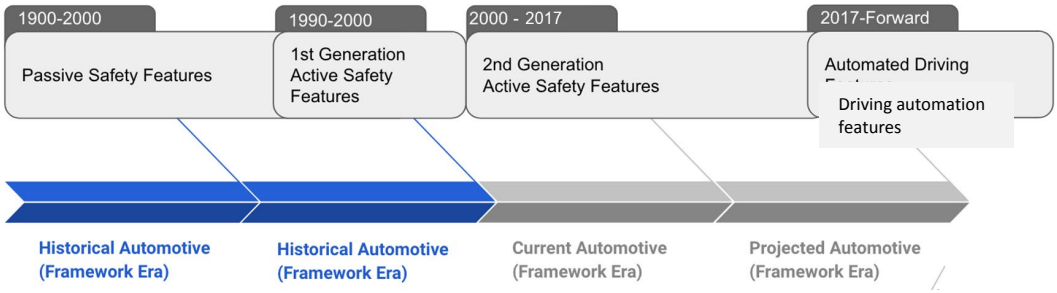

Figure 15. Frame 1.0 of the comparative framework

4.1 Research Summary

This document, provides a foundation for fundamentals of automotive software over various eras of technology introduction. This document reflects the rapidly emerging and accelerating change factors related to market demand, consensus standards, technology, and processes for the automotive industry. The document presents a taxonomy of SDLC practices that resulted from these change factors. The framework taxonomy is proposed as a potential tool to facilitate comparison of automotive SDLC practices both over time as well with respect to other industries.

Examples of how the comparative framework may be used include:

- Basis for better understanding the evolution of automotive software development and what technologies were or are being used,
- Tool to compare and benchmark automotive SDLC best practices against best practices applied in other industries, and
- Tool to be used in developing harmonized processes through analysis across various factors.

4.2 Research Findings

Uncertainty surrounding the implications of growing software complexity is compounded by the expanding spectrum of novel E/E architectures, non-traditional sensing and computational platforms, and increasing levels of automotive software automation and connectivity. As a result, the impacts on criticality and safety have yet to be fully understood as the number of critical software modalities on vehicles expands to bring more factors into consideration such as machine learning, cybersecurity, trust, and morality.

Responses during discussions with industry experts for this project indicate that software development practices that are designed to satisfy the highest levels of criticality (e.g., ASIL D) can cost up to 100 percent more than the lowest levels (e.g., quality management or ASIL A). Analysis published by software development organizations indicate similar estimates of 50-80% cost increases from the lowest to the highest levels of criticality, as well as one estimate of 10-fold increased effort.[235] It is expected that mixed criticality software systems will amplify costs through added layers of complexity (e.g., satisfying both safety and security criticality requirements). Moreover, there is evidence that newer automotive business models driven by MaaS and other commercialized automation models (e.g., last-mile delivery) may drive requirements for criticality and complexity due to the differing models for utilization, reliability, and availability for commercial fleet users versus private consumers.

For the reasons described above, future SDLC practices across the wide spectrum of automated systems and SOA, emerging agile software frameworks, emerging process reference models, legacy systems, and changing programming paradigms are expected to be increasingly diverse and complex.

The growing integration of increasingly complex software in motor vehicles implies that cost structures across the transportation industry will become increasingly dependent on the complexity of underlying software, which is in turn highly sensitive to levels of software

[235] Gheraibia et al., 2018 , Tom-M.[sic], 2019 ; Hilderman, 2014.

criticality and to mixed modes of criticality. As a result, pressure to understand and optimize software practices that drive safe, reliable, and secure critical software will increase.

The need to perform comparative analytics on software processes will only increase, with the objective of allowing automotive software developers and producers to study and design harmonized, cost-effective lifecycle practices, to ultimately manage growth of the vehicle industry's software infrastructure. The framework contemplated in this study may be useful as a foundation for software producers, automotive engineers, and transportation scientists to perform comparative assessments and analysis on highly diverse and complex automotive software processes.

Framework Sources

[1] Ziegler, C., & Patel, N. (2016, April 7). *Meet the new Ford, a Silicon Valley software company*. The Verge. www.theverge.com/2016/4/7/11333288/ford-ceo-mark-fields-interview-electric-self-driving-car-software

[2] Grosse-Ophoff, A., Hausler, S., Heineke, K., & Möl, T. (2017, April 18). *How shared mobility will change the automotive industry.* McKinsey and Company. www.mckinsey.com/industries/automotive-and-assembly/our-insights/how-shared-mobility-will-change-the-automotive-industry

[3] Hurley, B. (2011, March 1). *Global car platforms: Automotive design with the world in mind.* Techbriefs. www.techbriefs.com/component/content/article/tb/pub/features/articles/9410

[4] Tajitsu, N. (2017, May 31). *Toyota uses open-source software in new approach to in-car tech*. Reuters. www.reuters.com/article/us-toyota-tech/toyota-uses-open-source-software-in-new-approach-to-in-car-tech-idUSKBN18R1CW

[5] Appel, T. (2016, April 25). *How many different ways are there to build a Ford F-150? Would you believe 2 billion?* The Daily Drive. http://blog.consumerguide.com/how-many-different-ways-are-there-to-build-an-f-150-would-you-believe-2-billion/

[6] Burkacky, O., Deichmann, J., Doll, G., & Knochenhauer, C. (2018, February 14). *Rethinking car software and electronics architecture.* McKinsey & Company. www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture

[7] J. D. Power. (2018, February 14). Most owners still in love with their three-year-old vehicles, J.D. Power finds [Press release]. www.jdpower.com/business/press-releases/jd-power-2018-us-vehicle-dependability-study

[8] Steinkamp, N. (2017). 2017 Automotive Warranty and Recall Report. Stout Advisory.

[9] MarketsandMarkets. (2020, May). Automotive Software Market by Application (Infotainment, Powertrain, ADAS & Safety), Vehicle Type (Passenger Vehicle, Commercial Vehicle), EV Type (BEV, HEV, PHEV), and Region (Asia Pacific, Europe, North America, and RoW) - Global Forecast to 2025. [Wep page] www.marketsandmarkets.com/Market-Reports/automotive-software-market-200707066.html?gclid=EAIaIQobChMIweb-BuP7o4AIVZB6tBh0ksAxHEAAYASAAEgJlN_D_BwE

[10] Clarence-Smith, T. How software will dominate the automotive industry. Toptal [Web page] www.toptal.com/insights/innovation/how-software-will-dominate-the-automotive-industry

[11] Glance, D. (2017, September 3). As your car becomes more like an iPhone, get ready to update its software regularly. *Futurism.* https://futurism.com/as-your-car-becomes-more-like-an-iphone-get-ready-to-update-its-software-regularly

[12] Bureau of Transportation Statistics. National Transportation Statistics 2015 [Web page]. www.bts.gov/product/national-transportation-statistics

[13] Statista. (2018, July). Number of light vehicles per household in the United States from 2006 to 2016. www.statista.com/statistics/551403/number-of-vehicles-per-household-in-the-united-states/

[14] Gross, A. (2015, April 16). New study reveals when, where and how much motorists drive. AAA. https://newsroom.aaa.com/2015/04/new-study-reveals-much-motorists-drive/

[15] Antinyan, V. (2018). *Revealing the complexity of automotive software*. Volvo Car Corporation.

[16] Davey, C. (2013, January 26-27). *Automotive software systems complexity: Challenges and opportunities*. INCOSE International MBSE Workshop, Jacksonville, FL.

[17] Wirthlin, R. (2018, March 29). Embedded Software In Products: The Convergence of ALM with Systems Engineering [Powerpoint]. Exploring Application Lifecycle Management and Its Role in PLM, 2018 Spring Meeting, PLM Center of Excellence, Purdue University [Powerpoint]. https://polytechnic.purdue.edu/sites/default/files/files/Embedded%20software%20in%20products%20-%20the%20convergence%20of%20ALM%20with%20Systems%20Engineering.pdf

[18] AUTOSAR Classic Platform [Web page]. www.autosar.org/standards/classic-platform/

[19] Adaptive AUTOSAR Platform [Web page]. www.autosar.org/standards/adaptive-platform/

[20] Cordes, J., & Zetlmeisl, M. (2012). AUTOSAR gets on the road - More and more (SAE Technical Paper 2012-01-0014). SE International. doi: 10.4271/2012-01-0014.

[21] Martínez-Fernández, S., Ayala, C. P., Franch, X., & Nakagawa, E. Y. (2015). A survey on the benefits and drawbacks of AUTOSAR. doi: 10.1145/2752489.2752493

[22] AGL Specification [Web page]; www.automotivelinux.org/software/agl-specification

[23] Tuohy, S., Glavin, M., Hughes, C., Jones, E., Trivedi, M., & Kilmartin, L. (2015, April). Intra-vehicle networks: A review. *IEEE Transactions on Intelligent Transportation Systems*, Volume 16, Issue 2. https://ieeexplore.ieee.org/document/6819448

[24] Robert Bosch GmbH. (1991). CAN Specification. http://esd.cs.ucr.edu/webres/can20.pdf

[25] LIN Consortium. (2003). LIN Specification Package, Revision 2.0.

[26] MOST Cooperation. (2008). MOST Specification, Revision 2.3.

[27] TTTech Computertechnik AG. (2008, November). TTEthernet Specification.

[28] Keskin, U. (2009, January). *In-vehicle communication networks: A literature survey*. Eindhoven University of Technology.

[29] IEEE 802.1 Working Group. Time-Sensitive Networking (TSN) Task Group [Web page]. https://1.ieee802.org/tsn/

[30] [Deleted, duplication]

[31] J2057/1_200609. (n.d.). SAE Class A application/definition. Society of Automotive Engineers.

[32] ISO/IEC JTC 1/SC 42/SG 1. (n.d.). Computational approaches and characteristics of artificial intelligence systems.

[33] ISO/IEC WD 22989. (n.d.). Artificial intelligence -- Concepts and terminology.

[34] ISO/IEC WD 23053. (n.d.). Framework for artificial intelligence (AI) systems using machine learning (ML).

[35] ISO/IEC NP TR 24027. (n.d.). Information technology -- Artificial intelligence (AI) -- Bias in AI systems and AI aided decision making.

[36] ISO/IEC NP TR 24028. (n.d.). Information technology -- Artificial intelligence (AI -- Overview of trustworthiness in artificial intelligence.

[37] ISO/IEC NP TR 24029-1. (n.d.). Artificial Intelligence (AI) -- Assessment of the robustness of neural networks.

[38] ISO/IEC NP TR 24030. (n.d.). Information technology -- Artificial intelligence (AI) -- Use cases.

[39] ISO/IEC NP 38507. (n.d.). Information technology -- Governance of IT -- Governance implications of the use of artificial intelligence by organizations.

[40] [Deleted, duplication]

[41] IEEE P7008. (n.d.). Standard for ethically driven nudging for robotic, intelligent and autonomous systems.

[42] IEEE P7010. (n.d.). Wellbeing metrics standard for ethical artificial intelligence and autonomous systems.

[43] [Deleted, duplication]

[44] ASPICE: VDA QMC Working Group 13 Automotive SIG. (2015). Automotive SPICE, 132.

[45] ISO/SAE CD 21434. (n.d.). Road Vehicles -- Cybersecurity engineering.

[46] [Deleted, duplication]

[47] SAE J3061A. (n.d.). Cybersecurity Guidebook for Cyber-Physical Vehicle Systems.

[48] [Deleted, duplication].

[49] ISO 21448:2019. (n.d.). Road Vehicles — Safety of the intended functionality.

[50] National Highway Traffic Safety Administration. (n.d.) A drive through time. https://one.nhtsa.gov/nhtsa/timeline/index.html

[51] AUTOSAR. (2017). Explanations of Adaptive Platform Design AUTOSAR, DID 706, (17-03).

[52] Glas, B., Gebauer, C., Hänger, J., Heyl, A., Klarmann, J., Kriso, S., Vembar, P., & Wörz, P. (2015). Automotive safety and security integration challenges. Lecture Notes in Informatics, Proceedings - Series of the Gesellschaft Fur Informatik.

[53] Rushby, J. (1994). Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety, Vol. 43*, No. 2. www.csl.sri.com/users/rushby/papers/csl-93-1.pdf

[54] Zarr, R.. (2018, April 11). *The future of high-reliability electronics*. Electronic Design. www.electronicdesign.com/technologies/analog/article/21806380/the-future-of-highreliability-electronics

[55] Ardebili, H., & Pecht, M. G. (2000). *Encapsulation technologies for electronic applications*. 1st edition. Elsevier.

[56] Mundhenk, P. (2017). *Security for automotive electrical/electronic (E/E) architectures* [Dissertation, Technische Universität München]. Cuvillier Verlag. https://www.mundhenk.org/files/SecurityForAutomotiveEEArchitectures_Philipp-Mundhenk_Dissertation.pdf

[57] Van Eikema Hommes, Q. D. (2016, June). *Assessment of safety standards for automotive electronic control systems* (Report No. DOT HS 812 285). National Highway Traffic Safety Administration. www.nhtsa.gov/sites/nhtsa.gov/files/812285_electronicsreliabilityreport.pdf

[58] Ebert, C., Amsler, K., Lederer, D., & Burton, S. (2011). *Introducing automotive E/E safety engineering: Challenges and solutions.* Vector.

[59] AUTOSAR. (2017). AUTOSAR ECU Template (AUTOSAR), DID 060, (4.3.1).

[60] AUTOSAR, 2008.

[61] Pelliccione, P. (n.d.). *Software architecture for automotive*.

[62] Wanner, D., Trigell, A., Drugge, L., Jerrelind, J. (2012). *Survey on fault-tolerant vehicle design*. World Electric Vehicle Journal.

[63] Wolf, J. (2015). *Is this what the future will look like? Implementing fault tolerant system architectures with AUTOSAR basic software*. Vector. https://assets.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Safety_ElektronikAutomotive_PressArticle_201511_EN.pdf

[64] Czerny, B. J., D 'ambrosio. J. G., Murray, B. T., Sundaram, P. (2005, April). Effective application of software safety techniques for automotive embedded control systems. doi: 10.4271/2005-01-0785.

[65] [Deleted, duplication]

[66] Molotnikov, Z., Schorp, K., Aravantinos, V., & Schaetz, B. (2016). *Future programming paradigms in the automotive industry*. Forschungsvereinigung Automobiltechnik e.V.

[67] Tatourian, A. (2018). *Highly-dependable automotive software* [PowerPoint]. Intel Automotive.

[68] Dubrova, E. (2013). *Fault-tolerant design*. Springer.

[69] Hammett, R . (2016, August 8-12). *Developing electronic systems for safety critical applications*. 34th International System Safety Conference 2016, Orlando, FL.

[70] Koren, I., & Krishna, C. M. (2007). *Fault-tolerant systems*, 1st edition. Elsevier.

[71] Berntsson, P. S., Strandén, L., & Warg, F. (2017). *Evaluation of open source operating systems for safety-critical applications*; Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence, Lecture Notes in Bioinformatics. doi: 10.1007/978-3-319-65948-0_8

[72] Shelly, P. (2013). Operating systems for cars [PowerPoint]. Mentor Graphics. www.roadmapforth.org/program/presentations/PatShelly.pdf

[73] Lardinois, F. (2018). *With its new in-car operating system, BMW slowly breaks with tradition*. TechCrunch. https://techcrunch.com/2018/06/15/with-its-new-in-car-operating-system-bmw-slowly-breaks-from-tradition/

[74] Aroca, R., & Caurin, G. (2009). *A real time operating systems (RTOS) comparison*. www.semanticscholar.org/paper/A-Real-Time-Operating-Systems-(-RTOS-)-Comparison-Aroca-Caurin/3dc51976f8fb9408f5c991d457fa27f7b5adb737

[75] Lee, J.-C., Han, T. & Kim, S.-H. (2009, August 27-29). *Implementation of ECU configuration framework based on AUTOSAR methodology*. Conference: Proceedings of the 2009 International Conference on Hybrid Information Technology, Daejeon, Korea.

[76] Mbihi, J. (2018). *Analog automation and digital feedback control techniques*. Wiley

[77] SAE J3016 Levels of Driving Automation.

[78] U.S. Department of Transportation. (2018, October). *Preparing for the Future of Transportation: Automated Vehicles 3.0*. www.transportation.gov/sites/dot.gov/files/docs/policy-initiatives/automated-vehicles/320711/preparing-future-transportation-automated-vehicle-30.pdf

[79] General Motors Inc. (2018). *2018 Self-Driving Safety Report*. www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf

[80] Waymo. (2018). On the road to fully self-driving. www.auto-mat.ch/wAssets/docs/171019_waymo-safety-report-2017-10.pdf

[81] Fraade-Blanar, L., Blumenthal, M. S., Anderson, J. M., & Kalra, N. (2018). Measuring automated vehicle safety: Forging a framework. RAND Corporation. www.rand.org/pubs/research_reports/RR2662.html

[82] AUTOSAR. (n.d.). Specification of Communication; AUTOSAR CP Release 4.3.1.

[83] Arteris. (2005). *A comparison of network-on-chip and busses*. Design and Reuse. www.design-reuse.com/articles/10496/a-comparison-of-network-on-chip-and-busses.html

[84] Bock, F., Homm, D., Siegl, S., & German, R. (2016). *A taxonomy for tools, processes and languages in automotive software engineering*. Computer Science & Information Technology. doi: 10.5121/csit.2016.60121

[85] Molotnikov, Z., Schorp, K., Aravantinos, V., & Schaetz, B. (2016). *Future programming paradigms in the automotive industry*. Forschungsvereinigung Automobiltechnik e.V. https://trid.trb.org/view.aspx?id=1412727

[86] Voget, S. (2010, March 8). *AUTOSAR and the automotive tool chain*. Proceedings of the Conference on Design, Automation and Test in Europe, Dresden, Germany.

[87] Hardung, B., Koelzow, T. & Krüger, A. (2004, September 27-29). *Reuse of software in distributed embedded automotive systems.* 4th ACM International Conference On Embedded Software, Pisa, Italy.

[88] Khalil, M. (2018, October 25). Design patterns to the rescue: Guided model-based reuse for automotive solutions. 28th Conference on Pattern Languages of Programs .

[89] Lewis, B. (2017, March 28). *Automotive adopts Linux open source and software reuse principles for IVI and autonomous drive*. Embedded Computing Design. www.embedded-computing.com/embedded-computing-design/automotive-adopts-linux-open-source-and-software-reuse-principles-for-ivi-and-autonomous-drive

[90] Westman, J. (2013).; *A reference example on the specification of safety requirements using ISO 26262*. Royal Institute of Technology (KTH)/Scania.

[91] Matsubara, M. & Aoyama, M. (2017). *An analysis method of safety requirements for automotive software systems*. 24th Asia-Pacific Software Engineering Conference, Nanjing, China. doi: 10.1109/APSEC.2017.47

[92] Halbach, S., Sharer, P., Pagerit, S., Rousseau, A., & Folkerts, C. (2010). *Model architecture, methods, and interfaces for efficient math-based design and simulation of automotive control systems* (SAE Technical Paper 2010-01-0241). SAE International.

[93] Giese H., Karsai G., Lee E., Rumpe B., & Schätz B. (2007). The EAST-ADL architecture description language for automotive embedded software; Model-based engineering of embedded real-time systems. *Lecture Notes in Computer Science, vol. 6100*. Springer.

[94] AUTOSAR. (2006). *Layered software architecture*. AUTOSAR CP Revision 4.3.1.

[95] Fathy, H. K., Filipi, Z. S., Hagena, J., & Stein, J. L. (2006). *Review of hardware-in-the-loop simulation and its prospects in the automotive area*. Proceedings, Defense and Security Symposium, Orlando FL. doi: 10.1117/12.667794

[96] King, P. J. Whitley, C., & Copp,D. G. (2006, February). Hardware in the loop for automotive vehicle control systems development and testing. *Measurement+Control, 39*.

[97] Conrad, M., Fey, I., & Sadeghipour, S. (2005). Systematic model-based testing of embedded automotive software. *Electronic Notes in Theoretical Computer Science, 111*. doi: 10.1016/j.entcs.2004.12.005

[98] Ma, Z., & Schmittner, C. (2016). Threat modeling for automotive security analysis. *Advanced Science and Technology Letters, 139*. doi: 10.14257/astl.2016.139.68

[99] Macher G., Höller A., Sporer H., Armengaud E., Kreiner C; Koornneef F., & van Gulijk C. *A combined safety-hazards and security-threat analysis method for automotive systems; Computer safety, reliability, and security*. SAFECOMP 2014. Lecture Notes in Computer Science, vol. 9338. Springer.

[100] Ward, D., & Ibarra, I. (2013, October 16-17). *Practical experiences in applying the concept phase of ISO 26262*.8th IET International System Safety Conference Incorporating the Cyber Security Conference 2013, Cardiff, UK[102] New Electronics. (2018, November 23). *Software coding standards in automotive is* [sic] *becoming vital*. www.newelectronics.co.uk/electronics-technology/software-coding-standards-in-automotive-is-becoming-vital/195498/

[102] Sax, E., Reussner, R., Guissouma, H., & Klare, H. (2017). *A survey on the state and future of automotive software release and configuration management*. Karlsruhe Reports in Informatics.

[103] Jonston, B. (2016). *Harman updates ECUs OTA with NXP Gateways*. Auto Connected Car News.

[104] Majeed, A. (2016). *OTA software updates now serving ECUs for engine, brakes and steering*. Embedded Computing.

[105] Taub, E. (2016). Your car's new software is ready, Update now? *The New York Times*.

[106] Lang, K., Kropinski, M., & Foster, T. (2010). *Virtual powertrain calibration at GM becomes a reality.* SAE Technical Paper 2010-01-2323. doi.org/10.4271/2010-01-2323

[107] AUTOSAR. (2006). *Specification of I/O hardware abstraction*. AUTOSAR CP Release 4.3.0.

[108] Yan, Q.-Z., Williams, J. M., & Li, J. (2002). Chassis control system development using simulation: software in the loop, rapid prototyping, and hardware in the loop. *SAE Transactions, 111* www.jstor.org/stable/44719352

[109] Knight, W. (n.d.). *Self-driving cars can learn a lot by playing Grand Theft Auto.* MIT Technology Review.

[110] Schleifer, A. (2005). *Understanding regulation*. Harvard University. doi: 10.1080/09528822.2014.970769

[111] University of Pennsylvania Law School. (2016). *Penn Program on Regulation*.

[112] Wonham W. M. (2015). Supervisory control of discrete-event systems. In J. Baillieul, T. Samad (eds), *Encyclopedia of Systems and Control*. Springer.

Appendix A

The following section provides supplementary tables and supporting information related to the framework taxonomy.

Table 29. A representative list of industry consensus standards with potential impacts on automotive E/E software development practices

Representative List of Standards with Impacts on Automotive E/E SDLC Processes					
Category	**Sub Category**	**Standard**		**Description**	**Status**
Environmental	Diagnostics	Road vehicles Communication between vehicle and external equipment for emissions-related diagnostics	ISO15031-4:2014	A set of standard diagnostic services to be provided by vehicles (OBD services)	Active Published 2014
Cybersecurity	Encryption	Information technology – Security techniques – Lightweight Part 1: General;	ISO/IEC 29192-1: 2012	Specifies two block ciphers suitable for lightweight cryptography	Active Reviewed 2017
	Hardware Assurance	Requirements for Hardware-Protected Security for Ground Vehicle Applications	SAE J3101	A common set of requirements for security to be implemented in hardware for ground vehicles to facilitate security enhanced applications	Under Development
		Counterfeit Electronic Parts; Avoidance Protocol, Distributors.	SAE AS6081-2012	Standardizes practices to identify reliable sources to procure parts, assess and mitigate risk of distributing fraudulent/counterfeit parts, control suspect or confirmed fraudulent/counterfeit parts, and report suspect and confirmed fraudulent/counterfeit parts	Active Published 2012
	Software Assurance	Information technology -- Security techniques – Application Security	ISO/IEC 27034-1	Provides guidance to assist organizations in integrating security into the processes used for managing their applications	Active Reviewed 2017
	Supply Chain Risk Management	Information Technology – Open Trusted Technology Provider Standard (O-TTPS) – Mitigating maliciously tainted and counterfeit products;	ISO/IEC 20243:2015	A set of guidelines, requirements, and recommendations that address specific threats to the integrity of hardware and software COTS ICT products throughout the product life cycle. This release of the Standard addresses threats related to maliciously tainted and counterfeit products	Active Published 2018
		Information technology – Security techniques – Information security for supplier relationships	ISO/IEC 27036-3	Provides an overview of the guidance intended to assist organizations in securing their information and information systems within the context of supplier relationships	Active Published 2014
		Fraudulent/Counterfeit Electronic Parts; Avoidance, Detection, Mitigation, and Disposition Verification Criteria;	SAE AS6462A - AS5553A,	Establish compliance, and grant certification to AS5553, Aerospace Standard; Counterfeit Electronic Parts; Avoidance, Detection, Mitigation, and Disposition	Active Revised 2019

Category	Sub Category	Standard		Description	Status
Cybersecurity	Identity and Access Management	Information technology – Security techniques – Vulnerability handling processes;	ISO/IEC 15026-2:2011	Specifies minimum requirements for the structure and contents of an assurance case to improve the consistency and comparability of assurance cases	Active Published 2016
	System Security Engineering	Cybersecurity Guidebook for Cyber-Physical Vehicle Systems	SAE J3061	Provides guidance on vehicle cybersecurity	Active 2016
		Road Vehicles – Cybersecurity engineering	ISO/SAE DIS 21434	Automotive cybersecurity engineering for use of embedded controllers, long lifecycle of vehicles, safety implications	Under Development
Architecture	General E/E	AUTOSAR Classic	AUTOSAR	AUTOSAR Classic Platform architecture distinguishes on the highest abstraction level between three software layers which run on a microcontroller: application, RTE and BSW	Active 2003
		Adaptive AUTOSAR	AUTOSAR	AUTOSAR Adaptive Platform implements the AUTOSAR Runtime for Adaptive Applications)	Active 2017
	System/Vehicle	Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles	SAE J3016	Describes motor vehicle driving automation systems that perform part or all of the dynamic driving task by providing a taxonomy with detailed definitions for six levels of driving automation	Active Revised 2018
Safety		Road Vehicles - Functional safety	ISO 26262	ISO 26262 addresses possible hazards caused by malfunctioning behavior of E/E safety-related systems, including interaction of these systems. It does not address hazards related to electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, release of energy and similar hazards, unless directly caused by malfunctioning behavior of E/E safety-related systems.	Active Revised 2018
		Standard for Safety for the Evaluation of Autonomous Products	UL 4600	Covers safety principles and processes for evaluation of autonomous products, specifically their ability to perform the intended function without human intervention based on their current state and sensing of the operating environment. The standard also covers the reliability of hardware and software necessary for machine learning, sensing of the operating environment, and other safety aspects of autonomy.	Published

Category	Sub Category	Standard		Description	Status
Safety		Standard for Fail-Safe Design of Autonomous and Semi-Autonomous Systems	IEEE P7009	Establishes a practical, technical baseline of specific methodologies and tools for the development, implementation, and use of effective fail-safe mechanisms in autonomous and semi-autonomous systems. The standard includes (but is not limited to): clear procedures for measuring, testing, and certifying a system's ability to fail safely on a scale from weak to strong, and instructions for improvement in the case of unsatisfactory performance. The standard serves as the basis for developers, as well as users and regulators, to design fail-safe mechanisms in a robust, transparent, and accountable manner.	Working Group
		Road vehicles -- SOTIF	ISO/Publicly Available Specification 21448:2019	Provides guidance on the applicable design, verification and validation measures needed to achieve the SOTIF. This document does not apply to faults covered by the ISO 26262 series or to hazards directly caused by the system technology (e.g., eye damage from a laser sensor).	Active 2019
Machine Learning/AI		Computational approaches and characteristics of AI systems	ISO/IEC JTC 1/SC 42/SG 1	Standardization in the area of AI	Technical Committee
		AI -- Concepts and terminology	ISO/IEC WD 22989	Standardization in the area of AI	Under Development
		Framework for AI Systems Using Machine Learning (ML)	ISO/IEC WD 23053	Standardization in the area of AI	Under Development
		Information technology -- AI -- Bias in AI systems and AI aided decision making	ISO/IEC NP TR 24027	Standardization in the area of AI	Under Development
		Information technology -- AI (Overview of trustworthiness in AI)	ISO/IEC NP TR 24028	Standardization in the area of AI	Under Development
		AI -- Assessment of the robustness of neural networks	ISO/IEC NP TR 24029-1	Standardization in the area of AI	Under Development

Category	Sub Category	Standard		Description	Status
Machine Learning/AI		Information technology -- AI -- Use cases	ISO/IEC NP TR 24030	Standardization in the area of AI	Under Development
		Information technology -- Governance of IT -- Governance implications of the use of AI by organizations	ISO/IEC NP 38507	Standardization in the area of AI	Under Development
		Framework for AI Systems Using Machine Learning (ML)	ISO/IEC AWI 23053	Standardization in the area of AI	Under Development
		Standard for Ethically Driven Nudging for Robotic, Intelligent and Autonomous Systems	IEEE P7008	Standardization in the area of AI	Under Development
		Wellbeing Metrics Standard for Ethical AI and Autonomous Systems	IEEE P7010	Standardization in the area of AI	Under Development
Coding Style//Standard	MBD/ACG	Generic modeling design and style guidelines	MISRA AC GMG	Provides a set of rules, in a similar fashion to the MISRA C rules, which encourage good modeling practices and avoid poorly-defined features of the modeling language. In light of automotive industry trends, some rules will be aimed at the use of automatic code generators in safety-related systems.	Active 2009
		Modeling design and style guidelines for the application of Simulink and Stateflow	MISRA AC SLSF		
		Modeling style guidelines for the application of TargetLink in the context of ACG	MISRA AC TL		
		Guidelines for the application of MISRA-C:2004 in the context of ACG	MISRA AC AGC		
	Programming	Guidelines for the use of the C language in critical systems	MISRA C	Provides a "restricted subset of a standardized structured language" as required in the 1994 MISRA Guidelines for automotive systems being developed to meet the requirements of Safety Integrity Level (SIL) 2 and above.	Active Revised 2012

Category	Sub Category	Standard		Description	Status
Coding Style//Standard		Guidelines for the use of the C++ language in critical systems	MISRA C++		Active 2008
		MISRA Safety Analysis Guidelines	MISRA SA	Provide an extension to the original MISRA Development Guidelines for Vehicle Based Software, in that they give extended detailed advice on the sections on Integrity and Safety Analysis	
		Guidelines for the use of the C++14 language in critical and safety-related systems	AUTOSAR Adaptive Document 839	Specifies coding guidelines for the usage of the C++14 language as defined by ISO/IEC 14882:2014 [3], in the safety-related and critical systems	Active 2017
		Programming Language Secure Coding Standard	CERT C	Provides rules for secure coding in the C programming language	Active Revised 2016
Process Model//Maturity		Automotive SPICE Process Assessment//Reference Model	ASPICE	For use when performing conformant assessments of the process capability on the development of embedded automotive systems	Active Revised 2015

Table 30. Representative classification of ECU elements and subsystems [59]

AUTOSAR Classic - Taxonomy of ECU Elements and Subsystems[236]

ECU Memory Types	ECU Memory Implementations	ECU Processing Units by Name	ECU Processing Units by Architecture	ECU Processing Units by Implementation/ Technology	ECU Peripherals	ECU Electronics
Volatile: stores data and program code only during the operation of the ECU.	**ROM (Read Only Memory):** The program and constant data are fixed onto the chip during the manufacturing process. This data cannot be modified.	**ARM:** A series of low-cost, 32-bit RISC microprocessor cores for embedded control. It was the first commercial RISC.	**RISC:** A processor whose design is based on the rapid execution of a sequence of simple instructions rather than on the provision of a large variety of complex instructions.	**Microprocessor (µP):** A microprocessor is a PU without any peripherals and a significant amount of memory.	**Digital IO**	**Oscillator**
Non-volatile: stores data and program code during the operational and non-operational mode of the ECU.	**PROM (Programmable Read Only Memory):** data can only be written once. Programming is not part of the operational mode. Used for program code and constant data.	**MIPS:** A project at Stanford University intended to simplify processor design by eliminating hardware interlocks between the five pipeline stages.	**CISC:** A processor where each instruction can perform several low-level operations such as memory access, arithmetic operations or address calculations.	**ASIC:** A chip that implements dedicated functionality in hardware, such as a transmission protocol or a lambda-IC.	**ADC**	**Clock**
Shared: Memory that is used from more than one PU concurrently. The memory resource is available to all PUs connected to that memory.	**EPROM (Erasable Programmable Read Only Memory):** Data can be erased completely by UV light and then written one time until next erasure. Erasure and programming is not part of the operational mode. Used for program code and constant data.	**PowerPC:** The PowerPC standard specifies a common instruction set architecture (ISA), allowing anyone to design and fabricate PowerPC processors, which will run the same code.	**Vector PU; SIMD PU:** Type of a PU, which can process different operands and instructions at the same clock cycle.	**FPGA:** A reconfigurable chip, which implements a digital function in hardware.	**DAC**	**Communications transceiver**
Multi ported: Memory that can be accessed by more subscribers at one time.	**Flash:** Electrically Erasable Memory.		**MIMD PU:** machines have a number of processors that work asynchronously and independently.	**Digital Signal Processor:** A digital signal processor is a PU that is specialized for limited functions to process signals.	**Pulse Width Modulator**	**Power Driver**

[236] Definitions in this table are taken from the "AUTOSAR Specification of ECU Resource Template," AUTOSAR Classic Platform 4.3.1; the table is provided in order to show common ECU elements and subsystems found in Gen 2 ECU architectures.

ECU Memory Types	ECU Memory Implementations	ECU Processing Units by Name	ECU Processing Units by Architecture	ECU Processing Units by Implementation/ Technology	ECU Peripherals	ECU Electronics
Data retention: Data retention time is the time between programming a sample of non-volatile memory and the observation of a prescribed failure rate when verifying the programmed pattern.	**EEPROM (Electrical Erasable Memory):** Data is stored as the presence of electrical charges via tunneling effects in floating gates. The erasure and programming use similar physical effects. Erasure and programming takes the same amount of time approximately. Each EEPROM cell represents only a single bit. Depending on the internal architecture EEPROM can be erased and programmed in bit, 4-bit, byte or word size.				**PWD**	**Power Supply**
Architectural Quality: Special hardware implementations can help to improve the overall quality of an ECU. Error Correction Code and Parity are usual technologies. The choices of the storage media and the according quality and reliability requirements have to match.	**RAM (Random access memory):** Data is stored in electrical form either in the switching state of a Flip-flop or in the charge of a capacitor. RAM is used for temporary program code and variables data. This is a volatile memory.				**(CCU)**	
Dynamic Memory Allocation: In embedded real time systems dynamic memory allocation is not recommended as no reliable and predictable system behavior can be achieved and guaranteed.	**SRAM (Static Random Access Memory):** Data is stored in the switching state of a Flip-flop. Data can be accessed at any time and very fast. The data is valid as long as the power is supplied. This is a volatile memory.				**Watchdog Timer (WDT)**	

ECU Memory Types	ECU Memory Implementations	ECU Processing Units by Name	ECU Processing Units by Architecture	ECU Processing Units by Implementation/ Technology	ECU Peripherals	ECU Electronics
Mass-Storage Devices: Data stored on CD-ROM, DVD, as well as memory cards	**DRAM (Dynamic Random Access Memory):** Data is stored in a capacitor. DRAM can be implemented cheaply and in a high density. Due to the leakage of the capacitor the DRAM needs a refresh cycle in a defined time frame. This is a volatile memory.				**Timer**	
	Cache: Usually implemented as fast SRAM. Caches are used to increase performance of memory implementations.					
	Shadowed NV-RAM: A special form of non-volatile memory which uses extra mechanisms to increase the performance.					

Table 31. Representative classification of automotive E/E systems by safety critical assurance level (ASIL)

Classification of Representative E/E Systems by ASIL Rating[237] [238]						
Severity	**Probability of Exposure**	**Controllability by Driver***				
Light or Moderate Injury	**Very Low/Low**	All Classes of Controllability				
	Medium		Difficult to Control			
	High		Normally Controllable	Difficult to Control		
Severe Injury/ Survival Probable	**Very Low/Low**	All Classes of Controllability				
	Medium		Normally Controllable	Difficult to Control		
	High				Difficult to Control	
Life Threatening Injury	**Very Low/Low**	Simply Controllable	Normally Controllable	Difficult to Control		
	Medium		Simply Controllable	Normally Controllable	Difficult to Control	
	High			Simply Controllable	Normally Controllable	Difficult to Control
		Representative ASIL Ranking				
		QM	**A**	**B**	**C**	**D**
Automotive E/E Subsystem		GPS/Navigation System				
		Movie/Game Systems				
		Connectivity (USB, etc.)				
		Accent Lighting				
			Rear Lights			
			Headlights			
			Body Control Units			
			Instrument Clusters			
			HVAC			
			Body Gateway			
				Rear Camera		
				Active Suspension		
					ACC	
					EPS	

[237] MIPS. (n.d.) *Functional Safety: Functional Safety, ISO 26262 and MIPS* [Web page]. www.mips.com/markets/automotive/functional-safety/

[238] Vincelli, R., & Yasumasu, T. (2012, October 22-25). *Mastering functional safety and ISO 26262* [PowerPoint]. Renesas Electronics Corporation. DevCon 2012 Conference, Anaheim, CA.

		QM	A	B	C	D
				Transmission Control		
				Engine Control		
				Throttle Control		
				Ignition		
						Airbag
						ABS

Controllability (C class) represents the level of the ability to avoid harm and is one of the parameters that determine the ASIL in the ISO 26262 functional safety standard, which applies to the electrical and/or electronic systems.
* Simply Controllable (>99% of Drivers are able to control)
* Normally Controllable (>90% of Drivers are able to control)

Table 32. Examples of the additional SDLC process steps that are required to achieve ASIL D (safety) criticality over ASIL A criticality[239]

Process Considerations Required for ASIL D, but Optional in ASIL A
Defensive programming techniques (for example: divide by 0 protection)
Plausibility checks
Use of established design principles
Use of unambiguous graphical representation
Use of style guides for code/model
SW Architecture: High cohesion within a component
SW Architecture: Restricted coupling between components
SW Architecture verification via simulation of dynamic parts of the design
SW Architecture verification via prototype generation
Restricted use of interrupts
External monitoring facility (for example with hardware, such as a watchdog timer)
Control flow monitoring/analysis
Data flow analysis
Diverse software design/Independent parallel redundancy (for example two software development groups that do not talk to each other develop software for the same set of requirements. An "arbitrator" software will need to decide which one to use)
Graceful degradation
No dynamic objects or variables, or else online testing during their creation
No multiple use of variable names
Avoid global variables or else justify their usage
Limited use of pointers
No implicit type conversions
No hidden data flow or control flow
No recursions
Semi-formal verification of the unit design
Static code analysis
Fault injection testing
Resource usage tests (for example for memory and execution time)
Back-to-back comparison test between model and code (if using models)
Unit testing via equivalence classes
Unit testing via analysis of boundary values
Definition of done for Integration testing is Function and Call coverage
HIL testing of safety requirements

[239] International Organization for Standardization, 2018

Table 33. Classification and representative examples of software tools used in the development of automotive E/E systems

Classification/Examples of Representative Tools Used Across the SDLC	
Process Function	**Representative Tools used in SDLC**
Modeling ISO 26262 Concepts	ANSYS' medini analyze, MetaCase
Requirements Management	Microsoft's Word/Excel/PowerPoint, IBM's DOORS, Polarion, Jama, VSEM, SimuQuest UniPhi, Sparx Systems Enterprise Architect, PTC's Integrity, In House
Architecture (Definition)	UML/SysML, Word, Excel, PowerPoint, Matlab/Simulink, ASCET, UniPhi, ANSYS' medini analyze, ANSYS' SCADE Architect, IBM's Rhapsody, SPARK UML Architect
Modeling Tools (some are simulatable)	Altair's Embed, ANSYS' SCADE, ETAS' ASCET, Mathworks Simulink/Stateflow
Modeling Style Guide Enforcement	Mathworks Model Advisor, MES' Model Examiner
Model Metrics	Mathworks Model Metrics, MES' MXRAY
Model Debugging	Mathworks Simulink/Stateflow, Reactive System's Reactis
Data Dictionary	dSPACE TargetLink, Mathworks Simulink, SimuQuest UniPhi
Model Diff and Model Merge	ANSYS' medini unite, DiffPlug, EnSoft SimDiff/SimMerge, Mathworks Report Generator
Automatic Test Vector Generation	BTC, Mathworks Simulink Design Verifier, Piketec TASSIMO, Reactive System's Reactis
Test Execution/Management	BTC, Mathworks Simulink Test, MES' Test Manager, Piketec TPT
Model Coverage Measurement	Mathworks Simulink Coverage (formerly V&V), Reactive Systems' Reactis
Model Viewers	DiffPlug, Reactive System's Reactis/Model Inspector
Model Documentation	Mathworks Simulink Report Generator
Automatic Code Generation	Altair's Embed, ANSYS' SCADE, dSPACE TargetLink, ETAS' ASCET, Mathworks Embedded Coder
Automatic Code Generation - Low Level Drivers	Altair's Embed, Ecotrons EcoCoder, New Eagle's Rapture, SimuQuest QuantiPhi, Woodward MotoHawk
Static Code Analyzers	AbsInt, lint, LDRA, Polyspace, (many others)
HIL	dSPACE, National Instruments, SpeedGoat, (numerous others)
Calibration	ETAS' Inca, Vector tools
Requirements Traceability	ANSYS' medini analyze, Claytex Reqtify, Jama, Mathworks Requirements Management, MES' Test Manager
Version Control	SVN, git, Collabnet TeamForge
Issue Tracking	Jira
Product Languages	Assembler, C, C++, Java for informatics
Process Tools	MS Visual Studio, Eclipse, gcc, autoconf, binutils, clang, cmake, yocto, bitbake, powershell, batch scripts, MS-Project, Jenkins, Python, Code Composer Studio
Reviews	SmartBear Collaborator

Table 34. Classification of representative communications busses used in automotive E/E systems

Automotive Communications Busses by Class and Subdomain [23] [28] [31] [82]					
	Powertrain	**Chassis (Active Safety)**	**Body**	**Telematics**	**Passive Safety**
Program Size	2MB	4.5MB	2.5MB	100MB	1.5MB
Number of ECU	3-5	6-10	14-30	4-12	11-12
Bandwidth	500 Kb/s	500 Kb/s	100 Kb/s	200 Mb/s	10 Mb/s
Cycle Time	10ms - 10s	10ms - 10s	50ms - 2s	20ms - 5s	50ms
Safety Requirements	high	high	low	low	very high
Bus Type (typical)	Class C	Class C	Class A	Class D	Class D
			Class B		
Busses Used (typical)	CAN-C	CAN-C	LIN	MOST	MOST
	FlexRay	FlexRay	CAN-B		
			J1850		

Appendix B

Example

The following example demonstrates the framework reduction process in order to provide a comparative framework. It may be done differently for different scenarios depending on the comparison of interest. For example, different taxonomy elements may be used in one reduction versus another.

Step 1: Framework Reduction by Subclassification

The first step is to start with the framework taxonomy (e.g., Taxonomy of Software Development Lifecycle Practices, Taxonomy of Process Change Factors) classifications and subclassifications.

Subclassifications that do not differentiate the framework taxonomy for comparative purposes should be left out of the comparative framework. For example, "Supporting Processes," "Continuous Improvement - Functionality" is a potential subclassification. However, if all generations of the comparative targets (e.g., the reference framework and the target framework for comparison) use identical methods to try to continuously improve the functionality over time, the subclassification does not differentiate the taxonomy and should be removed from the comparison.

Table 30 through Table 33 provide a working example:

Table 35. The Framework Taxonomy reduced to comparative elements (a single element is shown)

Framework Reduction Example	
Classification	**Subclassification**
Software Implementation	Programming Language

Step 1a: Framework Reduction by Characteristics of the Subclassification

For each subclassification, a list of "Possible Answers" or "characteristics" can be created. The possible answers are ways that the subclassification can be implemented and compared.

Table 36. The Framework Taxonomy reduced into comparative elements with 4 comparative characteristics

Framework Reduction Example		
Classification	**Subclassification**	**Possible Answers (Characteristics)**
Software Implementation	Programming Language	Assembler
		C
		C++
		Other

Step 1b: Framework Reduction by Framework Generation

To perform comparative analysis, it is important that the framework taxonomy provide a chronological reference. This allows the framework to be used, for example, to compare the varying lifecycle practices found in "Gen 1" versus "Gen 3" in the automotive industry with changing practices of the comparative target.

For each "Possible Answer," the probability that this answer (without knowledge of any other subclassification answers) leads to classifying the overall software development process as a given framework "generation" was estimated (Table 32).[240]

Table 37. Assignment of probabilities to isolated characteristics allows the comparative target to be compared chronologically against reference framework.

Framework Isolation Example					
Classification	**Subclassification**	**Possible Answers**	**Gen 1 Probability**	**Gen 2 Probability**	**Gen 3 Probability**
Software Implementation	Language	Assembler	80	10	10
		C	10	80	10
		C++	0	10	90
		Other	0	0	100

Step 2: Comparative Analysis

With the probabilities for each subclassification defined, the comparative framework can be applied to specific scenarios (e.g., as a comparative framework). For any "Possible Answers" that are not used, do not count these "points" in the total number of possible points. For each generation, the number of points is the same as the probability for this generation.

The final result is a probability that the scenario fits each generation and the associated framework taxonomy for that generation.

Example Scenario:
In order to see how the comparative framework could be used to evaluate and assess lifecycle practices in other industries against the automotive industry, the following example is presented.

The scenario assumes that an aerospace software researcher is interested in understanding how the automotive industry tackles the challenges faced by integrated modular avionics architectures within hierarchical development processes that are distributed across a supply chain.

[240] The probabilities used are engineering best guesses based on the qualitative analysis used for this study and are for illustrative purposes.

The researcher has identified that the automotive industry uses harmonized consensus standards that incorporate process steps for:

- Distributed Development,
- SEooC, and
- (Emerging) SOTIF

The researcher has determined that interrelated, relevant consensus standards between the automotive and aerospace industries may be mapped as shown in Figure 15 and Figure 16.[241]

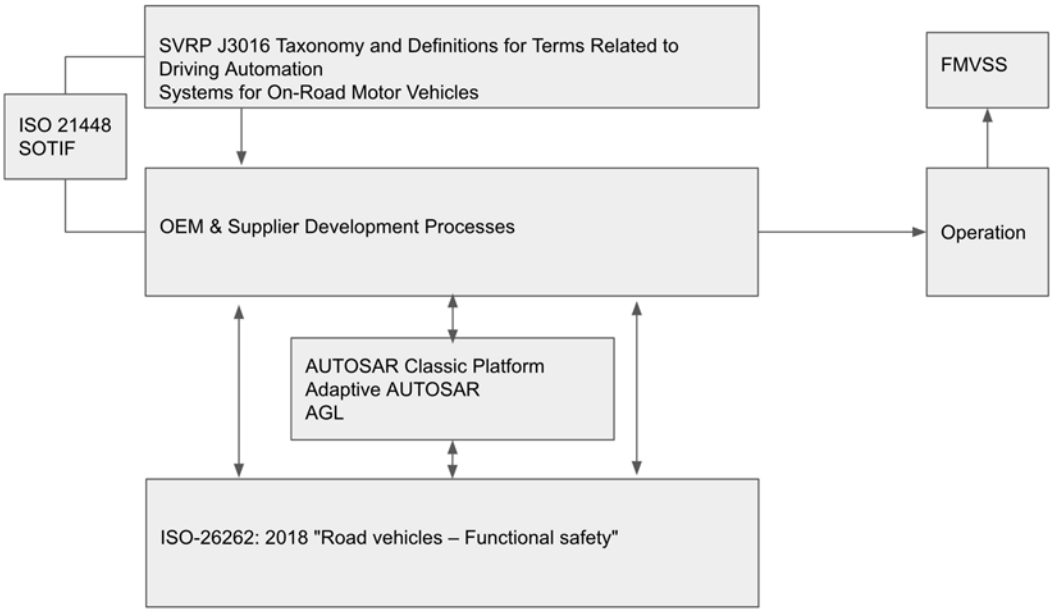

Figure 16. Example taxonomy of regulations and consensus standards used in automotive critical software development.

[241] Leveson, N., Wilkinson, C., Fleming, C., Thomas, J., & Tracy, I. (2014, October). *A comparison of STPA and the ARP 4761 safety assessment process*. MIT PSAS Techynical Report. http://sunnyday.mit.edu/papers/ARP4761-Comparison-Report-final-1.pdf.

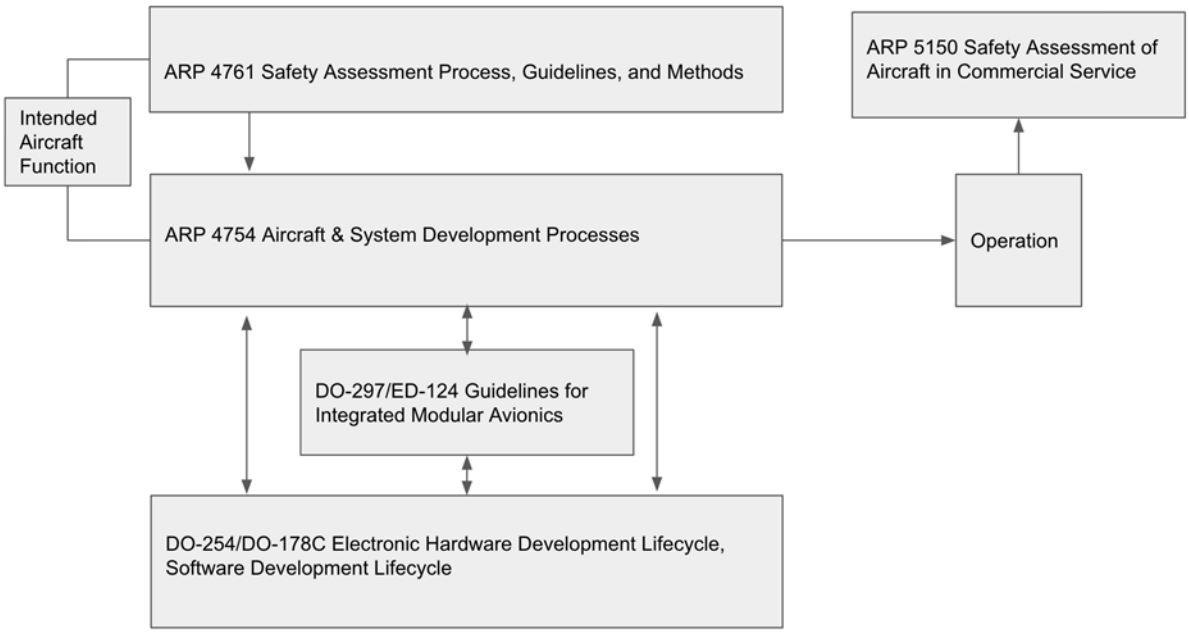

Figure 17. Example taxonomy of regulations and consensus standards used in commercial airplane critical software development

Using a framework reduction process as described above, the researcher has identified the elements shown in Table 33 to be used for comparative analysis between concepts used in the two industries.

Table 38. Automotive comparative analysis using the isolated framework

Framework Comparative Analysis[242]					
Classification	**Subclassification**	**Possible Answers**	**Gen 1 Probability**	**Gen 2 Probability**	**Gen 3 Probability**
Supporting Processes	Change Management	Yes	0	50	50
	Configuration Management	Yes	0	50	50
	Distributed Development	Yes	10	80	20
Concept Phase	Item Definition	Yes	0	80	20
	SEooC	Yes		50	50
Product Development at the Software Level	Development Process	V	10	85	5
	Code/Model Guidelines	MAAB/MISRA/ Other; Simulink	0	95	5
	Who does Software coding	Embedded Software Engineer	75	25	0
		Controls Engineer	0	90	10
	Software Design	Executable graphic (e.g., Simulink)	0	95	5
Software Implementation	Convert Software Design to Code	Autocode	0	50	50
	Language	C	10	80	10
	Language: Hardware Considerations	Floating Point	0	50	50
		Limited ROM/RAM/CPU	75	20	5
	RTOS	Embedded RTOS	0	95	5
	Dynamic/Self Adapting Code	No	45	45	10
Software Testing	Software Unit Testing	Yes	0	80	20
	Unit Testing Definition of Done	Structural Coverage	0	90	10
	Software Integration Testing (without hardware)	Yes	0	90	10
Post Release	In-Field Updates	Reflash ECU	0	95	5

Using the probabilities as defined (Table 33), a comparative analysis can be performed, and each process category can be assessed in order to determine which framework generation the process

[242] One of the goals of the example is to show the challenges with characterizing software practices in the automotive industry due to the wide range of constantly changing processes and tools that are used. The probabilities assigned were designated by the research team as "best guesses" and are used for illustrative purposes only. In order to perform a reliable analysis further research should be performed in order to capture probabilities that more accurately reflect industry practices.

category falls in. The results can be aggregated (e.g., weighted average) to classify the entire target process. In this way, the context of the taxonomic elements is established, allowing elements to be studied relative to other elements that are used in the same context.

Once the generational likelihood is established for the target process scenario, process steps and characteristics of the comparative target may be compared with the reduced framework, allowing a researcher to analyze the target process against other aspects of the reference taxonomy by generation.

Note: In this example, unused “Possible Answers” are not shown.

The a priori framework in this report only addresses the first two layers of detail. However, for the comparative framework, some interesting differentiators occurred at “lower levels” of detail that the a priori framework did not cover, such as what programming language is used. Some of these lower levels of detail were included in the comparative framework to allow for better differentiation of fundamentally different approaches.

DOT HS 813 226
June 2022

U.S. Department
of Transportation

**National Highway
Traffic Safety
Administration**

15352-061622-v3