

Report from a workshop held at the Scandinavian Conference on System and Software Safety (SCSSS) 2022.

Managing Continuous Assurance of Complex Dependable Systems

Fredrik Warg[§] and Anders Thorsén
RISE Research Institutes of Sweden

Anders Cassel, Omar Jaradat and Negin Nejad
Qamcom Research and Technology AB, Sweden

Stig Ursing
Semcon Sweden AB

DeJiu Chen
KTH Royal Institute of Technology, Sweden

Abstract—The SALIENCE4CAV project has done work on enabling continuous assurance, which aims to ensure safety is maintained throughout the entire lifecycle of a product, system, or service. One key technique is the use of safety contracts and modular assurance cases for systematically managing safety responsibilities and requirements across different stakeholders. This report summarizes outcomes from a workshop where discussions were held around this work. The participants were predominantly working in domains with high dependability requirements, such as automotive. Knowledge, tools, and organizational issues are seen as some key obstacles, but interest is high, and the community realizes the need for enabling continuous assurance.

Index Terms—dependable systems, safety, cybersecurity, contract-based design, safety contracts, continuous assurance

I. INTRODUCTION

Systems where safety and cybersecurity assurance are vital are increasing in complexity amid a growing demand for significantly faster update cycles than before. Various factors drive this shift. For example, the successful market introduction of an automated driving system (ADS) likely requires an ability for post-release updates to improve performance and grow the number of use cases over time. In addition, changes in operating conditions may require system updates, e.g., re-training of machine learning models in the perception subsystem for new types of objects, or updates to the driving policy based on new incident data. Hence, integrating field monitoring into the development loop also becomes necessary. Connected dependable systems also face a constant need for rapid updates due to the evolving cybersecurity threat landscape, requiring adaptations to new threats and security fixes. Some driving reasons and necessary timescales may be:

- Adapt to changing customer and business needs. → Year–Months
- Enabler for gradual feature and performance growth. → Months–Weeks
- Adaption to changes in operational conditions. → Weeks–Days
- Security fixes / protection against new threats. → Days–Hours

Version 1.3, 2023-11-06. This work is licensed under CC BY 4.0. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>. The work was supported by the Strategic Vehicle Research and Innovation (FFI) programme in Sweden, via the project SALIENCE4CAV (ref. 2020-02946).

[§]Corresponding author: fredrik.warg@ri.se.

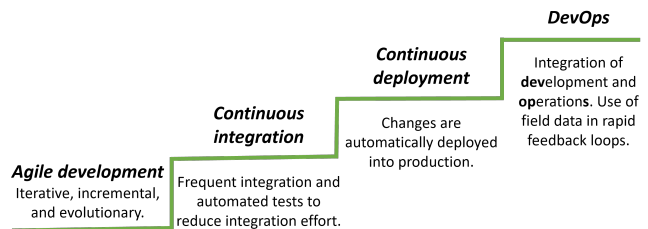


Fig. 1: The “Stairway to heaven” towards continuous updates.

In many domains, iterative practices have seen increasing adoption in the last two decades; e.g., as described in the “stairway to heaven” model [1] (see Fig. 1) where organizations move from traditional to agile development, through continuous integration (CI) and continuous development (CD), to finally using feedback and data collection to continually improve their products. For dependability-critical product development, however, the added challenge of managing assurance of e.g. safety and cybersecurity—often including compliance to standards (e.g., [2], [3] for road vehicles)—for each release has slowed the uptake of such methods, even if attempts have been made to define suitable methodologies, e.g., [4]–[6].

The SALIENCE4CAV project¹ has been working on continuous assurance using safety contracts and modular assurance cases [7], [8]. A workshop with industry practitioners was held in conjunction with the Scandinavian Conference on System and Software Safety (SCSSS) 2022. We discussed the state of practice, as well as presented and discussed the use of safety contracts as part of a way of working (WoW) with continuous assurance. This report summarizes live poll questions and discussions from this workshop. Some key observations were that a majority of the practitioners were unfamiliar with the concept of safety contracts, despite over a decade of work in the research community on this topic. There is an interest in, and perceived need for, trying to apply such practices, however organizational and tool issues are seen as obstacles.

¹<https://salienc4cav.se>

II. PARTICIPANT PROFILES AND STATE OF PRACTICE

Before diving into specific topics, we asked the participants some background questions. Note that the results were collected using a live polling tool², and hence the number of answers per question may vary since some participants may have failed to, or elected not to, answer specific questions.

The first set of questions (see Fig. 2 for poll questions and results) were related to the background of the participants, i.e., the type of products they work with, and their role in the company. Almost all participants work with safety-critical products in some sense, and a majority are in the automotive domain. Most participants have safety-related roles.

The second set of questions (see Fig. 3) concerned the use of agile development, CI, CD, and DevOps respectively in their current workplace (either the company where they are currently employed, or where they have their current assignment if they do consulting). The results show that agile development and continuous integration are common, but not universal, practices. Far fewer answer that they practice continuous deployment and DevOps, but a few already do apply these practices in safety-critical domains. The participants who answered "not applicable" mainly relate to persons not working in development organizations, such as researchers.

In the third set of questions (see Fig. 4), participants were asked whether they thought that CI/CD increased or decreased the effort required for safety assurance. The majority indicated that it increased the effort, while a notable portion mentioned a decrease. Additionally, participants were asked whether they believed that system safety could be enhanced through a CI/CD WoW. In this case, the majority expressed confidence that it could indeed lead to safety improvement.

III. CHALLENGES WITH CONTINUOUS DEPLOYMENT

The second part of the workshop focused on challenges associated with CI/CD in the context of dependability-critical development. These challenges were introduced in the presentation in Appendix A.

After the presentation, a discussion took place in a world café format. This involved participants forming groups to address a series of predetermined questions set by the organizers. Towards the end of the session, the essential take-aways from each group’s discussions were shared with the entire workshop. Groups had the flexibility to choose which questions they discussed, as time constraints often made it challenging to cover all topics. Lastly, participants were invited to individually respond to a set of live poll questions.

A. World café on "Challenges with CI/CD"

Questions asked in this part of the workshop were:

- How to manage and reconcile the impacts of product-line variability, necessary changes, and unexpected side effects?
- What parts of the development life cycle need to be considered in a CI/CD pipeline to support safety assurance?

²We used QuestionPro LivePolls. See <https://www.questionpro.com/>

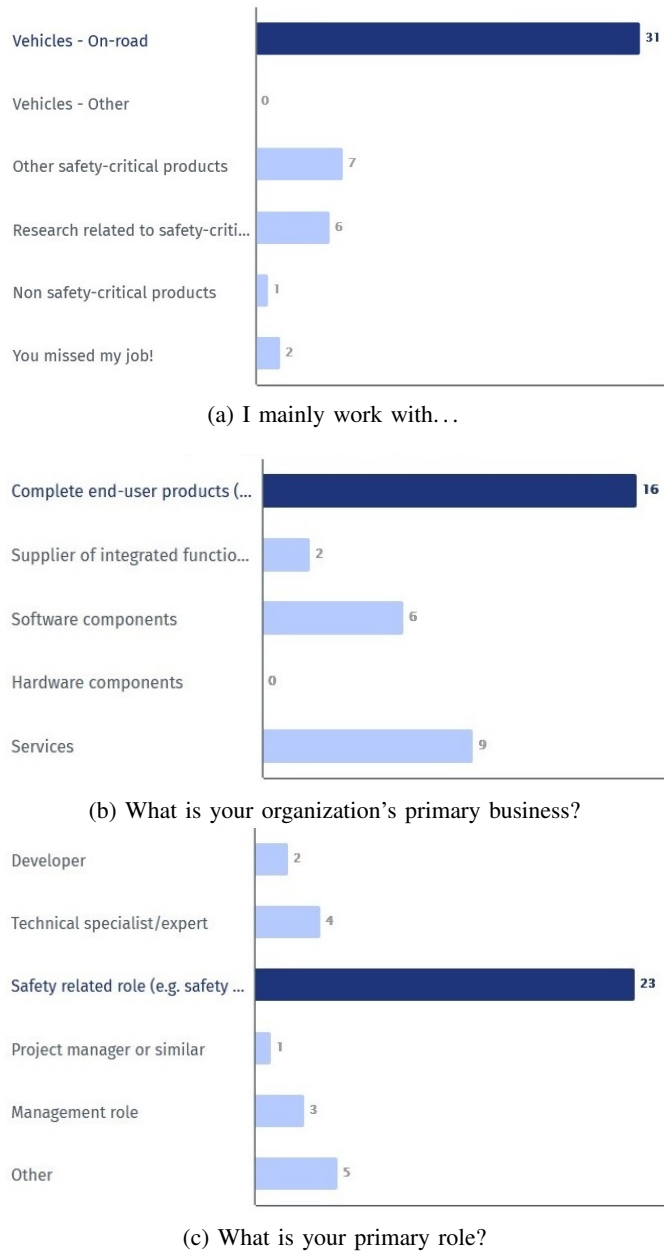
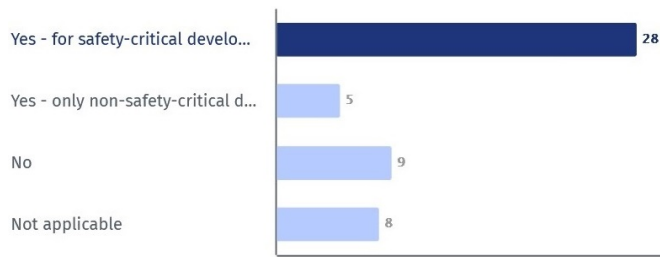


Fig. 2: Background questions to workshop participants.

- How would consistency between changes in safety requirements, architecture, implementation, and different variants be assured in a CI/CD toolchain?
- How to continuously maintain the safety case evidence after a system change or increment?
- How can the safety case be projected to highlight its updated parts after a system change efficiently?
- How the safety claims can be validated against the new safety boundaries or thresholds?
- How do you think CI/CD works with current safety standards and regulations?

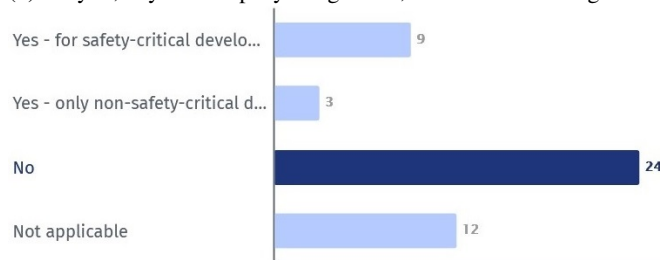
Some (unstructured) notes were taken. Here is a summary



(a) Do you, at your company/assignment, practice agile development?



(b) Do you, at your company/assignment, do continuous integration?



(c) Do you, at your company/assignment, do continuous deployment?



(d) Do you, at your company/assignment, practice DevOps?

Fig. 3: State of practice questions.

of the key points in response to the questions raised³:

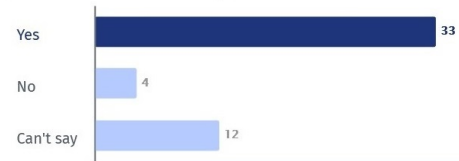
Managing and Reconciling Impacts

Impact analysis is crucial to managing product-line variability and unexpected side effects. Complex systems require proper processes, automation, and modularity. Maintaining separate branches and safety documentation helps manage changes. Continuous integration (CI) can lead to unaccounted side effects, particularly in complex systems. Separate safety cases for each release are necessary.

³ChatGPT 3.5 (version 2023-09-25, <https://chat.openai.com/>) has been used to help summarize the notes. Editing has been done to the generated summary.



(a) How do you think CI/CD changes the total safety assurance work?



(b) Do you believe CI/CD can improve system safety development over traditional development models with less frequent releases?

Fig. 4: Expected effects of CI/CD on safety assurance.

Maintaining a structured process is vital, with teams working on hazard analysis (HARA) and safety goals, followed by the development of safety cases.

Parts of Development Life Cycle in CI/CD

Validation, data collection specifications, and all aspects of the development life cycle need to be considered in a CI/CD pipeline. Tooling gaps and inconsistencies can occur due to different tools and practices in different organizations or different parts of an organization, or lack of interoperability between tools. Safety validation should occur at the vehicle level in the CI/CD chain. There's a split between traditional waterfall and CI/CD approaches, leading to challenges in managing safety requirements.

Ensuring Consistency

Ensuring consistency between changes in safety requirements, architecture, implementation, and variants can be challenging. Automation, traceability, and regression testing help maintain consistency. The frequency of safety case releases depends on company culture and architectural changes.

Continuously Maintaining Safety Case Evidence

Impact analysis, automation, and understanding what is being delivered are key to maintaining the safety case. Automation and simulation are essential, but manual steps such as impact analysis are still required. Safety assurance often occurs in batches of changes.

Projecting Safety Case Updates

Full automation according to e.g., ISO 26262 standard (for automotive safety) is ideal. Automation and detection of unsafe conditions play a crucial role in projecting updated safety case parts.

Validating Safety Claims

Safety performance indicators (SPI) can be used to validate safety claims. The adaptation of safety implementation is critical when implementing CI, particularly in the context of system changes.

CI/CD and Safety Standards/Regulations

CI/CD doesn't necessarily contradict safety standards but poses practical challenges. Adapting CI/CD to traditional safety standards and regulations can be difficult. There's a gap between agile development and traditional safety practices, with unclear guidance in standards like ISO 26262. While CI/CD and safety standards are not inherently incompatible, they may require process enforcement to ensure compliance.

TL;DR⁴

The notes highlight the complexities and challenges associated with integrating CI/CD into the development and maintenance of safety-critical systems. It underscores the importance of impact analysis, automation, and consistency in managing changes while dealing with evolving safety requirements and standards.

B. Live Poll on Challenges

Some live poll questions were asked at the end of the "challenges" section (see Fig. 5 for results). In short, the sentiments were mixed, but with a slight leaning toward a positive belief, when it comes to the question if using a CI/CD workflow will be more capable of reproducing the safety-related work (for new versions) than traditional WoW. Additionally, a notable majority of participants indicated that a freeze point is necessary to align safety assurance before a release. In other words, there's a belief that maintaining an "always ready for release" state, akin to some non-dependability-critical CI/CD WoW, might not be feasible in this context.

IV. SAFETY CONTRACTS AS POTENTIAL SOLUTION

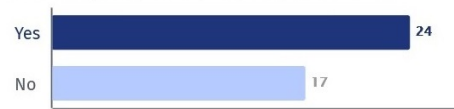
The third part of the workshop delved into the topic of safety contracts for safety assurance, as introduced in the presentation provided in Appendix B. Much like the "challenges" section of the workshop, this portion featured world café-style discussions after the presentation and finally some individual live poll questions.

A. World café on "Safety Contracts as Potential Solution"

Questions for this part of the workshop:

- What things in safety-contract methodology are preventing this methodology from being used in practice?
- If suitable tools existed, how would safety contracts benefit different development organizations at different abstraction levels?

⁴Internet slang for "too long, didn't read", often used to introduce a key message that summarizes a longer discussion or explanation.



(a) Do you think CI/CD is more capable of reproducing the deliverables and all safety-related verification and validation work products than traditional WoW?



(b) Do you think it is feasible to identify and analyse potential failure modes (that system changes might frequently introduce) by following CI/CD WoW?



(c) Do you believe CI/CD shall consider system boundaries and requirements freeze at some point during the development lifecycle?

Fig. 5: Challenges with CI/CD for dependable systems.

- What in the current development methods are most difficult to master when practising agile development and simultaneous engineering at several abstraction levels?
- How would safety assurance measures (e.g. audits, assessments, etc.) benefit from safety-contract-based design?
- What challenges are there to applying component-based design for ADS at all abstraction levels, i.e. from Item level to atomic SW- and HW-component level?
- How to derive safety contracts systematically (e.g. driven by safety analysis outcomes)?
- How to measure and ensure the completeness, correctness, and consistency of safety contracts?
- How to continuously check that the safety contracts capture the guarantees of the desired safe performance?

Summary of notes:

Challenges Preventing Adoption of Safety Contracts

Formalism in safety contracts may not be appealing to software developers. Academia may provide expertise, but there is doubt about industry demand. Lack of available tools for implementing safety contracts. Interface issues when aligning development with suppliers. Difficulty in defining the appropriate level of detail for safety contracts. Challenges in applying safety contracts at different abstraction levels and ensuring compatibility.

Benefits of Suitable Tools for Safety Contracts

Suitable tools could benefit "code-near" development and provide an atomic focus for developers. Safety contracts eliminate assumptions about component interfaces, especially in software develop-

ment. Quality advantages and a clearer definition of requirements and boundaries can be achieved. Machine learning tools might enhance the concept but need careful qualification.

Challenges in Agile Development and Simultaneous Engineering

Mastering impact analysis at all levels. Establishing a comprehensive model of all dependencies. Ensuring traceability from top to bottom. Reconciliation with standards and regulatory requirements. Unclear benefits of safety contracts at certain abstraction levels and ensuring contract assumptions are correct.

Benefits of Safety Contracts for Safety Assurance Measures

Safety contracts could potentially reduce the need for extensive reviews. The possibility of using modular reviews for safety contracts.

Challenges in Applying Component-Based Design in All Abstraction Levels

Challenges are mentioned, but specific details are not provided in the notes.

Systematic Derivation of Safety Contracts

Inclusion of elements like contracts for schedulers, CPU, and memory budgets. Consideration of stochastic, probabilistic, and statistical contracts. Uncertainty in requirements and the need for flexibility in safety contracts.

Ensuring Completeness, Correctness, and Consistency

Challenges in measuring completeness, consistency, and correctness of safety contracts. Suggestions of using tools like Prolog for consistency checks. The need to define complete and consistent safety contracts.

Continuous Checking of Safety Contracts

Suggested inclusion of safety contract validation to check that they capture the desired properties. The importance of creating models at the appropriate level for validation was discussed.

TL;DR

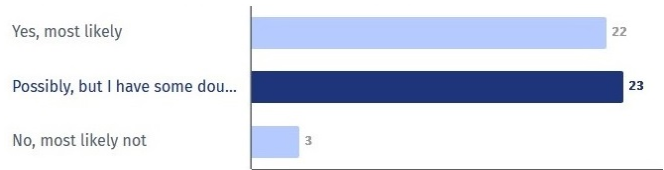
The notes address the challenges and potential benefits of using safety contracts in agile development, emphasize the need for suitable tools, and raise questions about the completeness, correctness, and consistency of safety contracts. It also highlights the importance of continuous validation and the need for flexibility in handling changing requirements.

B. Live Poll on Safety Contracts

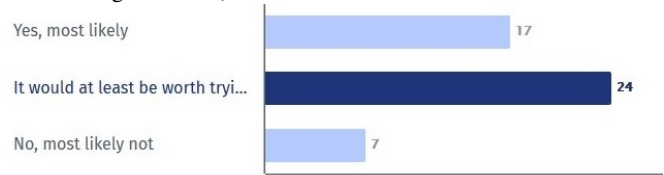
The live poll questions (see Fig. 6 for results) indicated that a majority of participants were not acquainted with the concept of safety contracts prior to the workshop, although many had heard of it. However, after the presentation and subsequent discussions, an overwhelming majority expressed belief in the potential benefits of safety-contract-based design for their organizations, and most were also willing to consider implementing it provided suitable tools were available.



(a) Did you ever hear of safety contracts and their usage in safety-critical system engineering before today?



(b) Do you believe safety-contract based design could benefit your organization (e.g., when negotiating requirements and changes between organizations)?



(c) Would you consider using contract-based design if suitable tools existed?

Fig. 6: Use of safety-contract based design.

V. OPEN DISCUSSION

The last part of the workshop was an open discussion and questions and answers session. Again notes were taken and are summarized below:

The discussion was mainly about the concept of safety contracts in the context of software and hardware development. Key points:

- Safety contracts serve as agreements between separate development teams, defining assumptions and guarantees related to interfaces and interactions between software (SW) and hardware (HW) components.
- The nature of safety contracts is a point of consideration, with questions about what these contracts should look like.
- Existing languages and syntax can express and define the behaviors of safety contracts. Some research has been conducted on these languages, and their suitability for specific applications and problem domains is being examined.

- Safety contracts can be presented in various forms, including formal, informal, or semi-formal. Some tools and architectural description languages (ADLs) can derive safety contracts, making it possible to perform consistency checks for these contracts.
- To apply safety contracts in daily work, they can be implemented at various levels of system abstraction. Implementing them at the architectural level may be particularly beneficial.
- The completeness of derivations in safety contracts can be verified through safety analysis, ensuring that assumptions and guarantees are comprehensive.
- Safety contracts can be seen as a more formal and intense version of requirements, and their benefit lies in the potential for continuous assurance and automated impact analysis.
- Automation, including the use of tools and natural language processing, is considered to be valuable for managing safety contracts. The idea is to streamline the process and provide logical consistency within safety contracts.
- There is a distinction between a contract and a traceability tool, with the possibility of needing a dedicated contract management tool to handle safety contracts effectively.

TL;DR

Safety contracts are agreements that define interactions and expectations between different teams working on software and hardware components. They can be presented in various forms, and automation and tools are considered important for their management and verification. The focus is on ensuring logical consistency within safety contracts to enable continuous assurance.

VI. CONCLUSION

The objective of this workshop was to disseminate knowledge, assess interest, and collect feedback on the topic of continuous assurance for dependable systems, and especially considering the use of safety contracts, which is a topic under consideration within the SALIENCE4CAV research project. As summarized in this report, the topic appears to be highly relevant and aligns with the challenges that many organizations are presently grappling with. The utilization of safety contracts and modular assurance cases is viewed as a promising approach to facilitate continuous assurance for dependability-critical systems. However, there are some hurdles to overcome regarding tooling, ease-of-use, and integration in the WoW. There are also beliefs that CI/CD may lead to more total effort spent on assurance, and that one would need a freeze period to sync development and assurance work before each release, something the authors of this report believe could be addressed with an integrated approach, e.g., as described in [5]. It is clear that there are still issues to solve in order to bring the theories of safety contract-based design to widespread practical use.

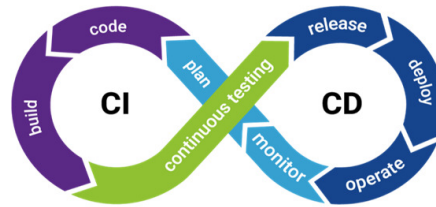
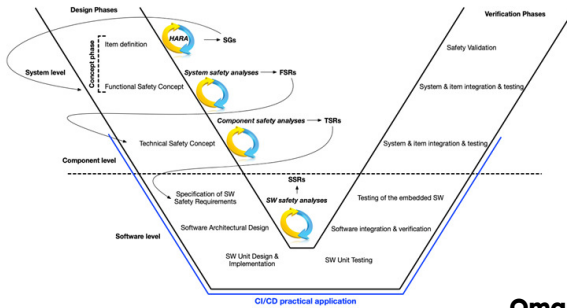
ACKNOWLEDGMENT

We would like to extend our gratitude towards everyone who participated and helped make this a lively and rewarding workshop, and to the organizing committee of the SCSSS conference and all participants in the SALIENCE4CAV project for their support in making the workshop happen.

REFERENCES

- [1] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the "stairway to heaven"—a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software," in *2012 38th euromicro conference on software engineering and advanced applications*. IEEE, 2012, pp. 392–399.
- [2] ISO, "ISO 26262:2018 Road vehicles – Functional safety," 2018.
- [3] ISO/SAE, "ISO/SAE CD 21434:2021 - Road Vehicles – Cybersecurity engineering," 2021.
- [4] T. Stålhane, T. Myklebust, and G. Hanssen, "The application of safe scrum to IEC 61508 certifiable software," in *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference*, 2012, pp. 6052–6061.
- [5] F. Warg, H. Blom, J. Borg, and R. Johansson, "Continuous deployment for dependable systems with continuous assurance cases," in *2019 IEEE Int. Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2019, pp. 318–325.
- [6] C. Fayollas, H. Bonnin, and O. Flebus, "SafeOps: A concept of continuous safety," in *2020 16th European Dependable Computing Conference (EDCC)*. IEEE, 2020, pp. 65–68.
- [7] I. Bate, R. Hawkins, and J. McDermid, "A contract-based approach to designing safe systems," in *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*. Australian Computer Society, Inc., 2003, pp. 25–36.
- [8] J. L. Fenn, R. Hawkins, P. Williams, T. Kelly, M. Banner, and Y. Oakshott, "The who, where, how, why and when of modular and incremental certification," in *2nd IET International Conference on System Safety*. IET, 2007, pp. 135–140.

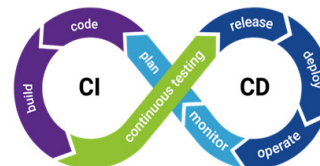
Continuous- Integration & Deployment for Automotive Safety Systems



Omar Jaradat
SCSSS'22 - workshop
Nov 23, 2022
Lindholmen

1

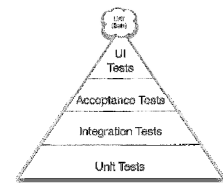
CI/CD



- **Continuous integration (CI)**- short-lived branches that are merged into a shared trunk several times a day where a series of automated tests give feedback about the changes introduced
 - Examples of branching strategies:
 - GitFlow
 - GitHub Flow
 - GitLab Flow
 - Trunk-based development (Our work assumes TBD as the used strategy)
- **Continuous delivery (CD)**- after continuous integration, continuous delivery prepares the software for delivery and ensures that the software can be reliably released at any time.
- **Continuous deployment**- after CI and CD, changes are automatically deployed into production by a fully automated process.

2

Continuous Testing



- **Unit tests**- to verify single parts of the application. This isolated part of the codebase is referred to as a unit.
- **Integration tests**- unit tests focus on an individual unit and thus may be insufficient by themselves, integration tests ensure that multiple components work together correctly and test how parts of the application work together as a whole.
- **Functional tests**- these tests make sure that the feature is working as it should
- **End-to-end tests**- these tests simulate a user experience to ensure that real users have a smooth, bug-free experience.
- **Acceptance tests**- these verify the behaviour of the software under significant load to ensure its stability and reliability.

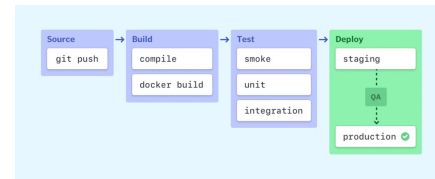
SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

3

CI/CD Pipeline



- **CI/CD pipeline:** a series of steps that should be performed to deliver a new version of the software.
- CI/CD pipelines are focused on improving software delivery via automation.
- A typical pipeline builds the code, runs tests, and then deploys the new software into production in a true replica of the software development lifecycle.
- **Building**, merging then testing the code-continuous integration
- **Preparing** the code for delivery- continuous delivery
- **Deploying** the code automatically- continuous deployment

SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

4

Pipeline's Engine and Stages

- **Source:** the CI/CD pipeline is triggered when a new code is committed to the repository.
- **Build:** this is where developers put their new code changes and compile them so they may pass through the initial testing phase
- **Test:** this is when the new code is tested through automated tests (for example, running unit tests through continuous integration). Depending on the size and complexity of the software, this step could last from seconds to hours. This stage will provide the feedback necessary for developers to fix any issues.
- **Deploy:** this is when the code is deployed to a testing or staging environment to prepare it for final release i.e continuous delivery. Usually, the build will automatically deploy once it passes through a series of automated tests.
- **Deploy to production:** here the code is released into a live production environment to reach end-users, either manually or automatically

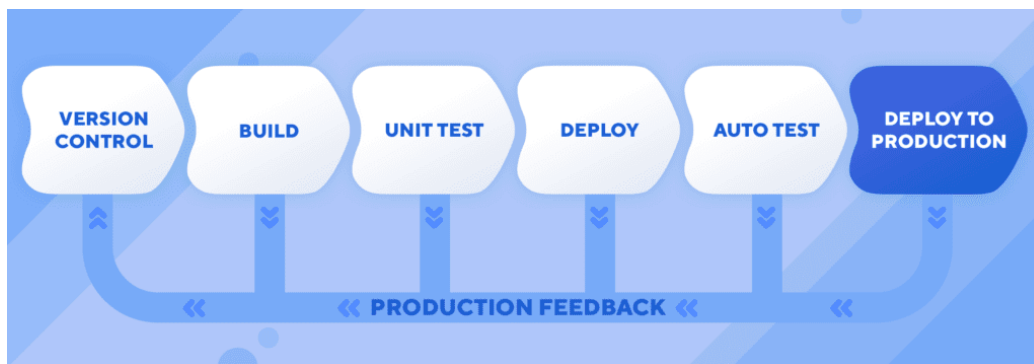
SALIENCE₄CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

5

Pipeline Example



Ref. <https://katalon.com/resources-center/blog/ci-cd-pipeline>

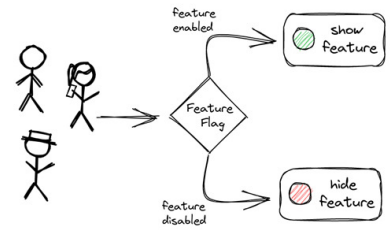
SALIENCE₄CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

6

Feature Flags



- A feature flag is a software development tool whose purpose is to turn certain functionalities **ON** or **OFF** to safely test in production by decoupling code deployment from feature release.
- With feature flags, developers can push their changes without waiting for other developers by simply turning **OFF** the incomplete portions of the code.
- Incomplete changes can be hidden behind a feature flag while the finished changes can be released. Once the incomplete is complete, they can be turned **ON** to become visible to end-users.
- This is important as the whole aim of continuous integration is to integrate changes at least once a day, so **feature flags help maintain the momentum of continuous integration**.

SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

7

Challenges & Quick Thoughts



- The term “feature” in agile methodologies and CI/CD does not exist in ISO26262
- What should the feature be mapped to (e.g., function, requirement, unit, component, etc.)?
- How to fit the CI/CD into the V-model? For example:
 - Shall CI/CD be limited to SW development, or should it cover the entire V-model?
- How to construct and maintain Safety Cases in the CI/CD pipeline?
 - Continuous Safety Assurance (CSA). Maintain already existing items of evidence and highlight the missing ones
 - Can we automate the evolution and maintenance of the safety case after each deployment?
- How to manage (split, group, categorize, and prioritize) the features in the backlog? Based on:
 - their dependencies?
 - Deliveries?
 - Change containment and susceptibility to change?
 - Safety case or evidence modularity
 - Limitation by suppliers?
 - ASIL?

SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

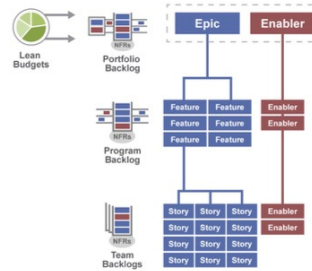
8

The Feature!

Define Features for the Program Backlog

Features are services that fulfill user needs.

- ▶ Feature is an industry-standard term familiar to marketing and Product Management
- ▶ Features are described with a short phrase and a benefits hypothesis, that clearly expresses their value
- ▶ Features are identified, prioritized, estimated, and maintained in the Program Backlog



SCALED AGILE © Scaled Agile, Inc.

5

SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

9

Example Feature

Features represent the work for the Agile Release Train

- ▶ The Feature benefit hypothesis justifies development cost and provides business perspective for decision-making
- ▶ Acceptance criteria are typically defined during Program Backlog refinement
- ▶ Reflect functional and nonfunctional requirements
- ▶ Fits in one PI

In-service software update

Benefit hypothesis

Significantly reduced planned downtime

Acceptance criteria

1. Nonstop routing availability
2. Automatic and manual update support
3. Rollback capability
4. Support through existing admin tools
5. All enabled services are running after the update

Example Feature

SCALED AGILE © Scaled Agile, Inc.

6

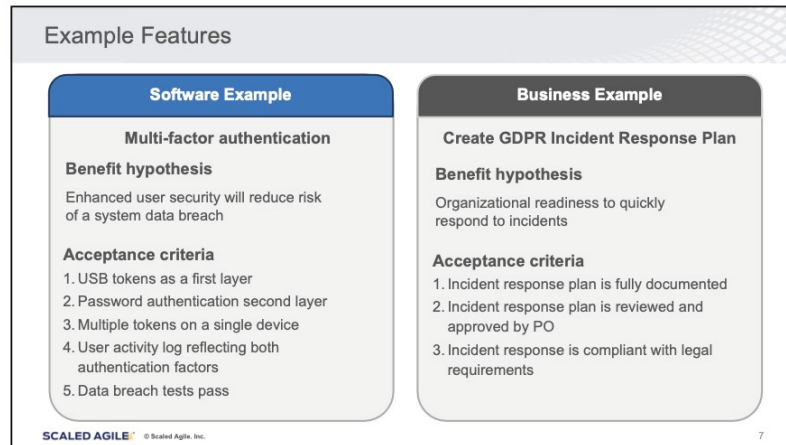
SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

10

Example Feature



SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

11

Function Vs. Item Vs. Feature

- Vehicle function (ISO26262): behaviour of the vehicle (intended by the implementation of one or more items) that is observable by the customer e.g., Autonomous Emergency Brake (AEB)
- Item (ISO26262): System or (combination of systems) that implements a function or part of a function at the vehicle level e.g., AEB can be an item
- In agile methodology, a feature is a service or function of the product that delivers business value and fulfills the customer's need. Each feature is broken down into several user stories, as it is usually too big to be worked on directly
- Fitting the agile's def. of the feature into ISO 26262 context so that it is a building block for a system or systems that implement item(s)
e.g., the brake pedal position is a feature that contributes to accomplishing vehicle functions such as braking
- Hint: CI/CD features can be inspired and derived from the HAZOP functions list

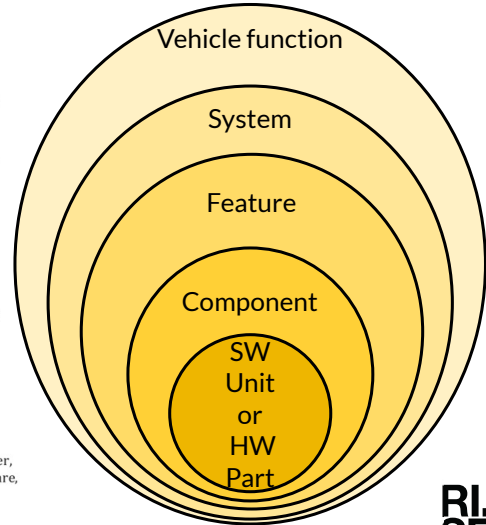
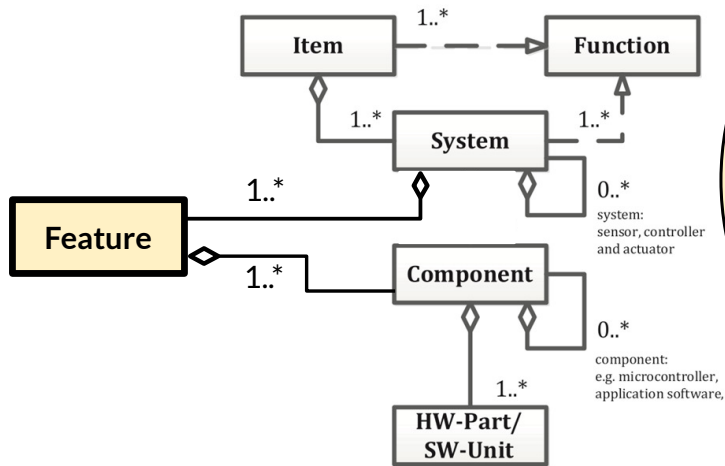
SALIENCE4CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

12

Specific proposal



**RI
SE**

13

Thank you!

SALIENCE₄CAV

Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

**RI
SE**

14



SALIENCE₄CAV
Safety Lifecycle Enabling Continuous Deployment for Connected Automated Vehicles

Safety assurance & Safety contracts

in Continuous Deployment

Anders Cassel
Date: 2022-11-23


SALIENCE₄CAV 

1

Trends for autonomous systems – A challenge

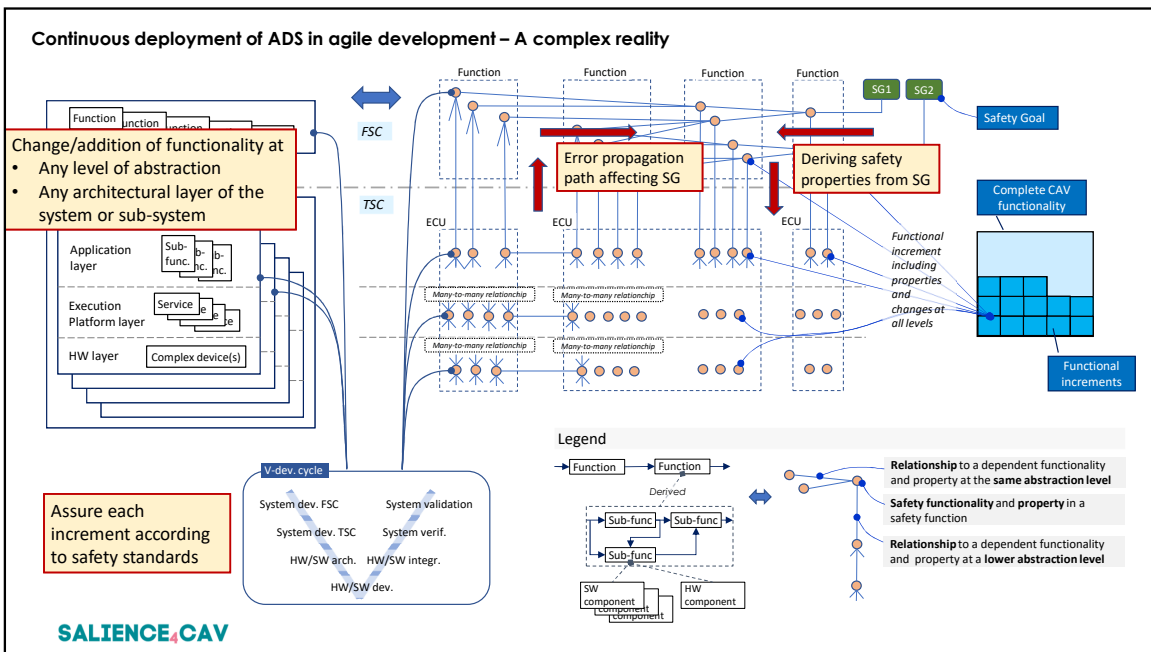
How to master a complex world of

- Agile development methods & simultaneous engineering
- Central compute architectures
- Frequent system release cycles
- Conform to safety & security standards

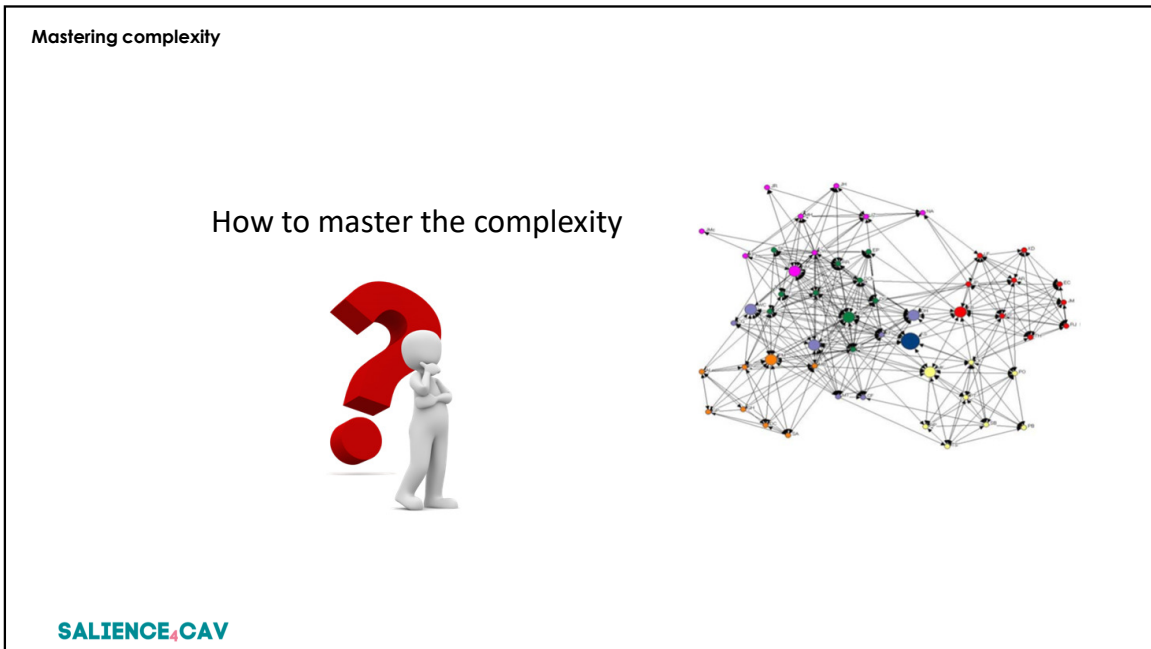


SALIENCE₄CAV

2

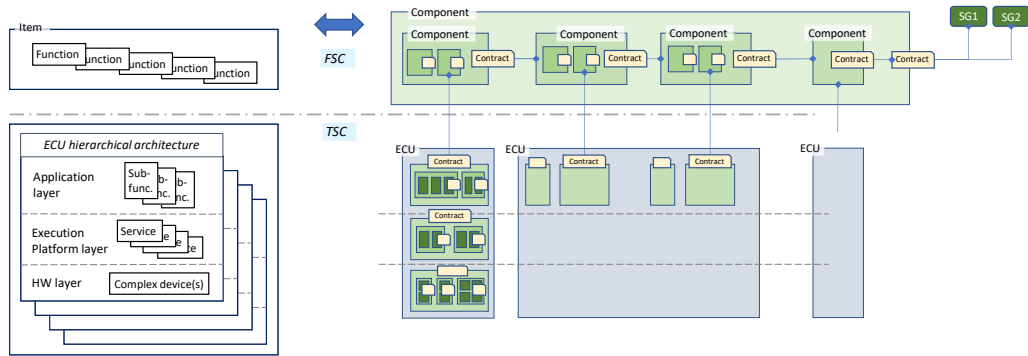


3



4

Mastering complexity – Introduction of Contract-based design



- Separation of safety concerns and modularization
 - Component-based design
 - Hierarchy of components → highest Item level to atomic level
- Safety properties and functionality assured by Safety contracts
 - Component safety property and functional response guaranteed by Safety contract
 - Contracts specified for each component at all abstraction levels
 - Higher level contract assured by fulfillment of lower-level contracts

SALIENCE₄CAV

5

Introduction of safety contract-based design

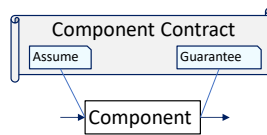
- Safety contracts
 - Methodology focusing on separation of concerns
 - Assuring safety properties and behavior of each component
 - Expressed by formal requirement syntax
 - Enables automatic contract checking
 - Safety contracts part of architecture model at all design levels by e.g. SysML/UML, EAST-ADL,..
 - Safety case compilation based on safety contracts by SW tool support integrated in CI/CD build chain

SALIENCE₄CAV

6

Introduction of safety contract-based design

- **Component Assume-Guarantee (A/G) Contracts**
 - **Contracts** are defined as **Assume-Guarantee** assertion pairs
 - **Guarantee** are the guaranteed functionality that the specific component is able to fulfill.
 - **Assume** are interpreted as a set of assumptions on the signals provided at their input-ports and the operational environment required for the component functionality.
 - Component response and properties are guaranteed under a set of assumptions on the environment, e.g. inputs and dependencies
 - **Top-down & bottom-up**
 - **Global properties** of systems are composed based on local properties of the components
 - **Local properties** of components are decomposed based on properties at a higher abstraction level



SALIENCE₄CAV

7

Types of Contracts

Component Contract

- Pair of assertions of assumptions and guarantees of a specific component

Component-Component Interface Contract

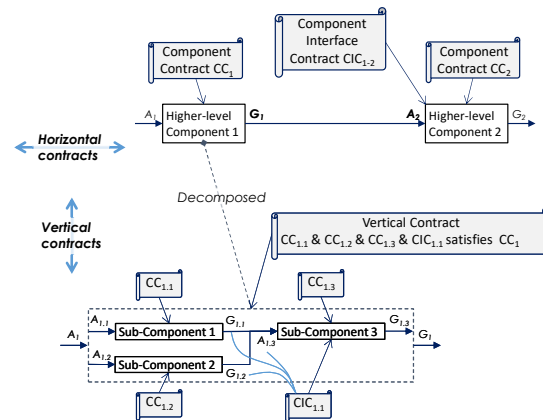
- Relationship between assumptions of a specific component and guarantees of the interfacing components.
- The guarantees of a component output-port must satisfy the assumptions of the signal input-port of the receiving component(s).

Horizontal Contracts

- Component Contracts and Component Interface Contracts defined at the same abstraction level.

Vertical Contracts

- Decomposition of a higher-level component contract to a set of sub-component contracts and component interface contracts
- Sub-component contracts satisfies the higher-level component.

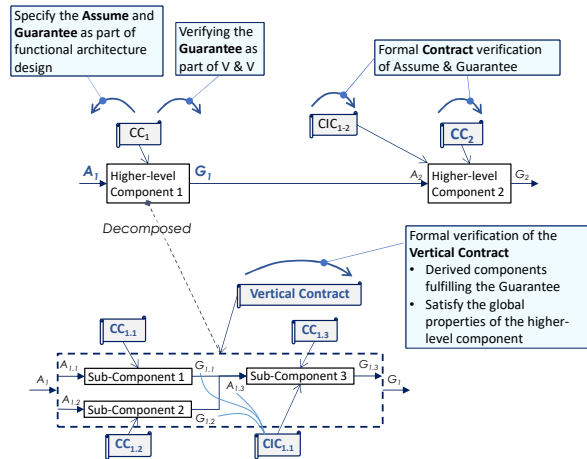


SALIENCE₄CAV

8

Formal checking of contracts

- Contracts are specified by a defined requirement syntax implementing a set of logic expressions
- Result of verification activities feedback into the contract model
- Component contracts are verified that the guarantee is realized and satisfies the assumption
- Formal verification:
 - Checking the formal assertions and verification result of assume and guarantee
 - **Pass:** Behavior and properties of assume and guarantee meets the criteria
 - **Fail:** Behavior and properties of assume and guarantee don't meet the criteria
 - Supports impact and variability analysis



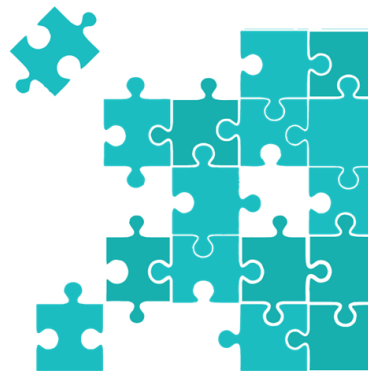
SALIENCE₄CAV

9

Modelling

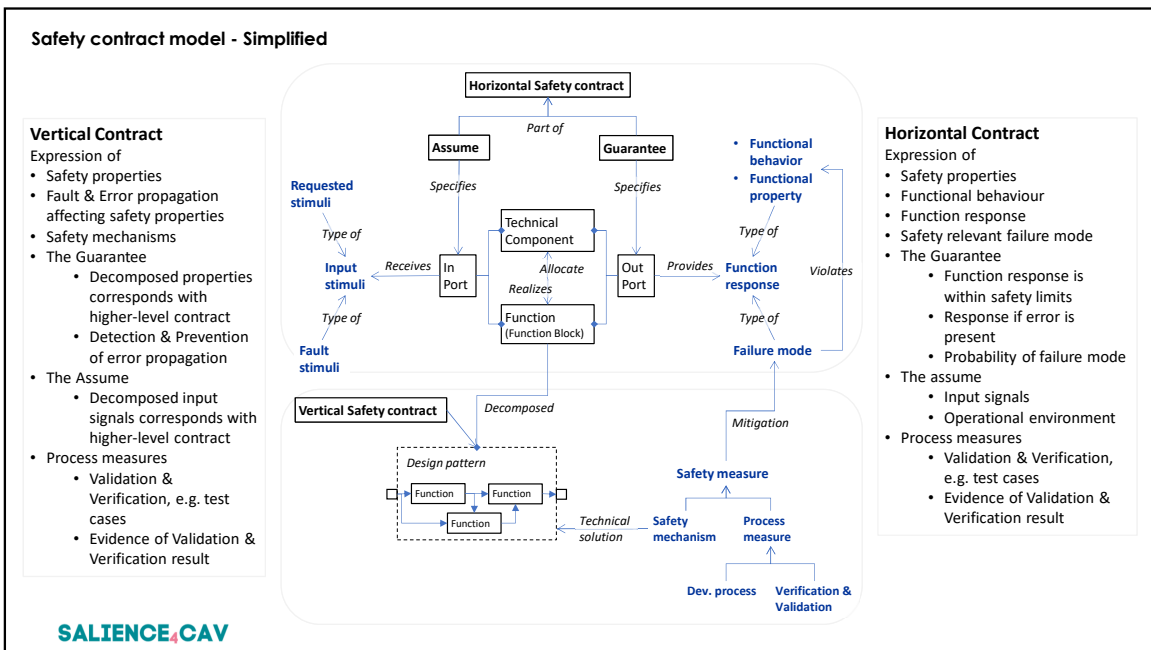
Fitting components & contracts together

- Meta-models

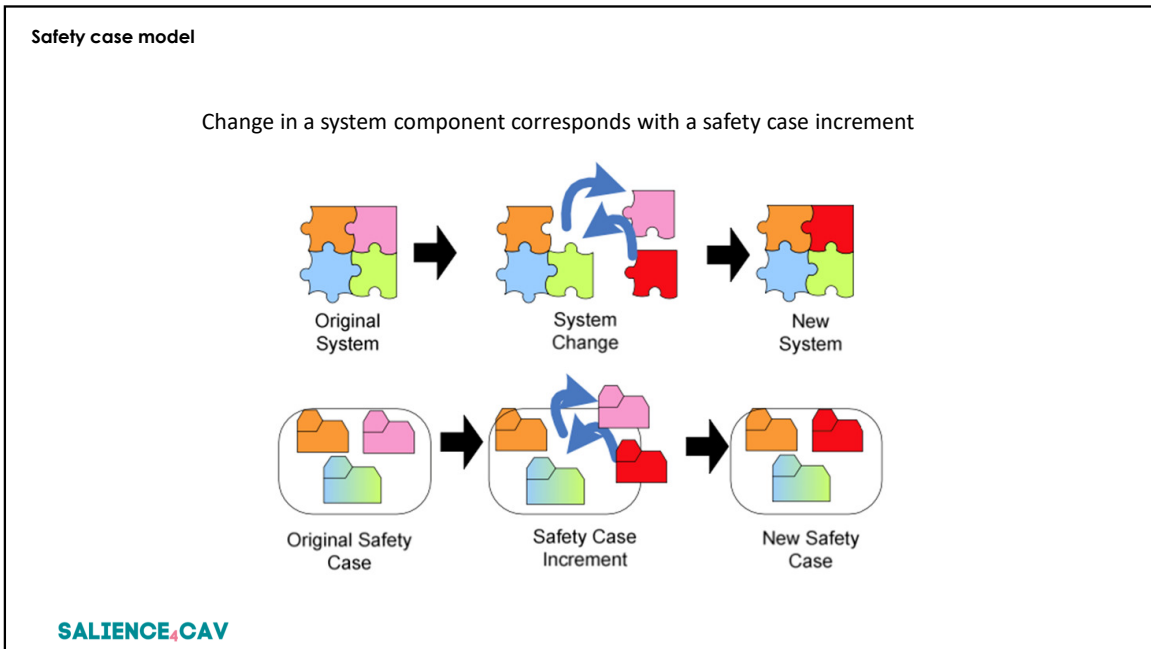


SALIENCE₄CAV

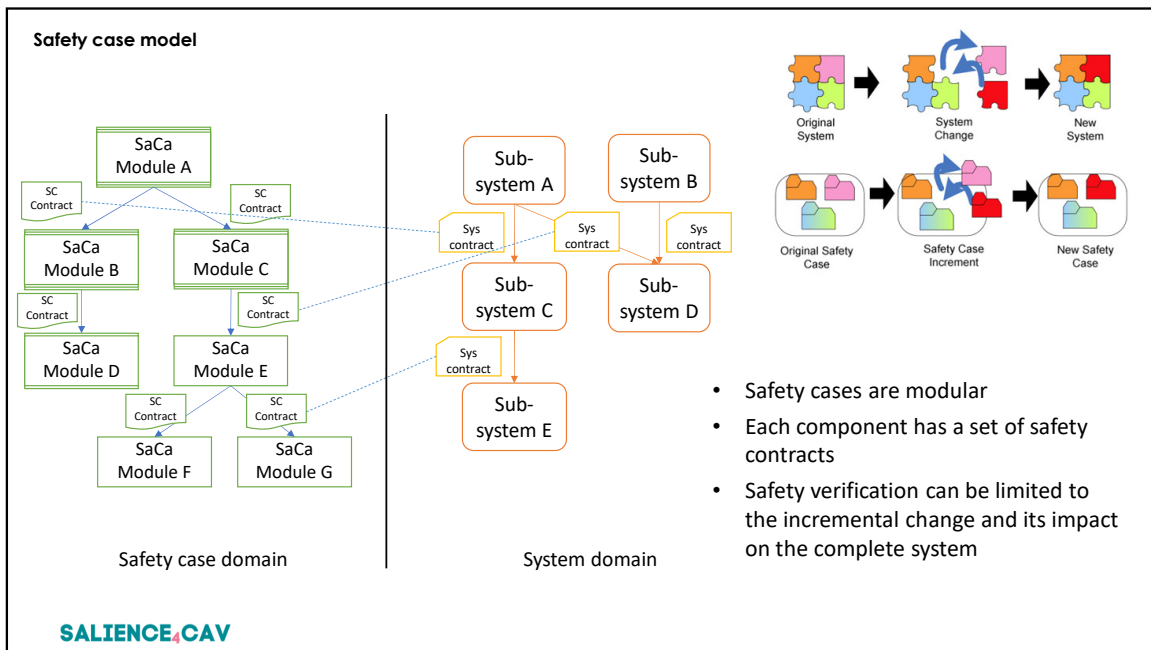
10



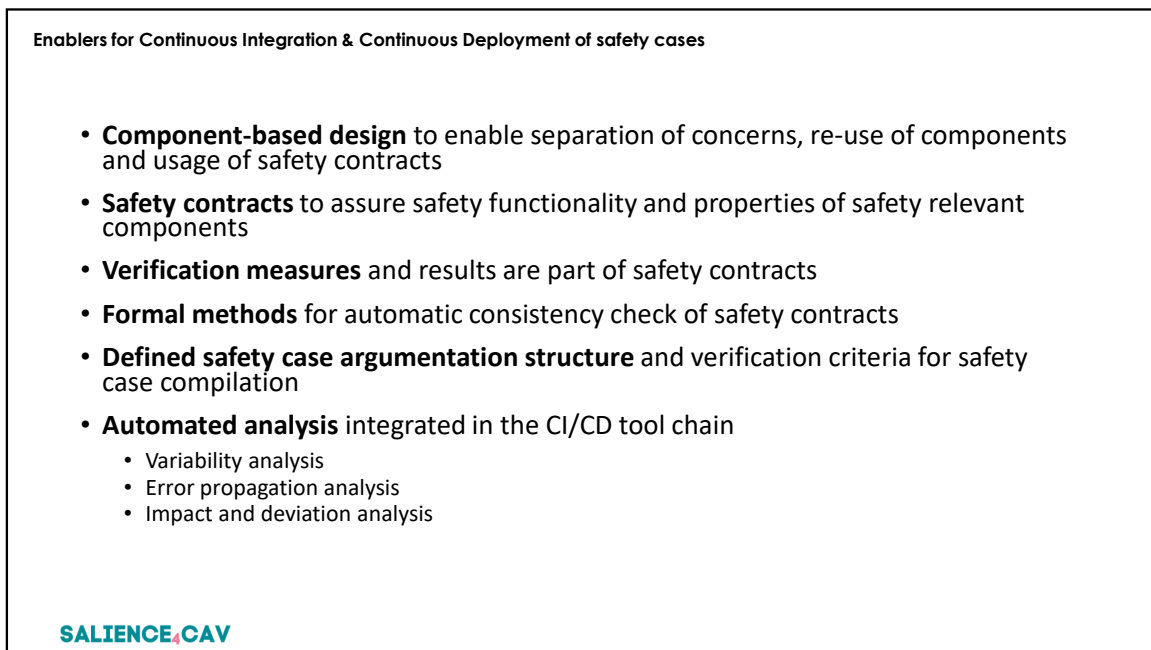
11



12



13



14

References

- Contracts for system design (A Benveniste, INRIA 2012)
- Assurance aware contract-based design for safety-critical systems (I Sljivo, 2018)
- AMASS research project - Baseline and requirements for architecture-driven assurance (AMASS_D3.1_WP3_FBK_V1.1, 2018)
- AMASS research project - Design of the AMASS tools and methods for architecture driven assurance (AMASS_D3.3_WP3_INT_V1.0, 2018)
- Continuous Deployment for Dependable Systems with Continuous Assurance Cases (F. Warg, et al., 2019)


SALIENCE₄CAV

15

End

Thank you

Anders Cassel
Anders.cassel@qamcom.se

 **qamcom**
Qamcom Research & Technology

SALIENCE₄CAV

16

