Continuous Deployment for Dependable Systems with Continuous Assurance Cases

Fredrik Warg\textsuperscript{1}, Hans Blom\textsuperscript{2}, Jonas Borg\textsuperscript{3}, and Rolf Johansson\textsuperscript{4}

\textsuperscript{1}RISE Research Institutes of Sweden, fredrik.warg@ri.se
\textsuperscript{2}Zenuity AB, hans.blom@zenuity.com
\textsuperscript{3}Volvo Penta AB, jonas.borg@volvo.com
\textsuperscript{4}Autonomous Intelligent Driving, rolf.johansson@aid-driving.eu

Continuous Deployment for Dependable Systems with Continuous Assurance Cases

Fredrik Warg  
RISE Research Institutes of Sweden  
Borås, Sweden  
fredrik.warg@ri.se

Hans Blom  
Zenuity AB  
Göteborg, Sweden  
hans.blom@zenuity.com

Jonas Borg  
Volvo Penta AB  
Göteborg, Sweden  
jonas.borg@volvo.com

Rolf Johansson1  
Autonomous Intelligent Driving  
Göteborg, Sweden  
rolf.johansson@aid-driving.eu

Abstract—An assurance case contains a structured argument supported by evidence, demonstrating that a system fulfils a certain quality attribute such as safety, cybersecurity or reliability. The traditional way of building assurance cases is, however, not well suited to continuous deployment, and difficult to maintain with a product structure where many variants and frequent new versions must be managed. By integrating the assurance work with product development in continuous assurance cases, which are updated and assessed iteratively, we claim continuous deployment of dependability-critical products is possible to achieve. In this paper we propose a work process combining the use of component-based design, contracts, modular assurance cases, and continuous assessment to enable continuous deployment in the context of product lines.

Index Terms—Assurance case, Safety case, Safety Contracts, Continuous assessment, Continuous deployment.

I. INTRODUCTION

Safety cases, security cases, reliability cases etc. are quality attribute (QA) specific variants of the more general term assurance case. An assurance case contains a structured argument supported by evidence, demonstrating that a system in a specified context fulfils a specific property. Many industrial domains have explicit demands for assurance cases, e.g. the automotive standard ISO 26262 [1] contains requirements for a safety case providing an argument that functional safety is achieved, and which contains the work products resulting from following the standard’s prescribed safety lifecycle. However, the traditional way of working with development and assessment of dependable systems is reaching its limits. Driving reasons for this are the emergence of highly complex and quickly evolving dependability-critical systems such as automated driving systems (ADS), and increased connectivity, which necessitates a change in how products are managed, e.g. the possibility to do quick updates to fix security holes becomes necessary. Many of these products also come to the market in several variants and new versions at a high pace. For each variant or new version, an updated set of assurance cases is needed to show that the QAs of the product are maintained.

For systems without dependability requirements, methods such as continuous deployment, agile development, and DevOps have evolved and become ever more popular over the last two decades. The main advantage is ability to quickly adapt in a fast-changing world. While there have been efforts to introduce such methods for dependable systems, state of practice is typically far more traditional. In this paper we contribute by outlining a work process where continuous assurance cases, which are updated as part of the development process, i.e. in the same pace as the product, are used to enable continuous delivery (CD) of dependability-critical systems. This process builds upon several practices, which in turn build upon a large body of existing work:

- The use of component-based design and property-specific contracts (e.g. safety contracts) specifying the components’ assumptions on the environment and what it can guarantee provided the assumptions are fulfilled.
- Modular assurance cases where components have their own assurance case fragments relevant to the component context, facilitating reuse of components in several products or variants provided they fit in the actual context.
- Stepwise refinement using feedback of operational data and the practice of creating forward-looking assurance cases as part of planning product changes.
- Reusable patterns for arguments and contracts to promote simplicity and enable automation.
- Continuous assessment integrated with development and combined with confirmation reviews and audits.

We thus propose to create assurance cases in a modular fashion with fragments rather than as a monolith. Since each component is confined by its contract and has its own fragment, it can also have its own independent life cycle. This enables an organization to develop many variants and versions simultaneously at a high pace by reusing these self-contained components. Merge conditions to a product main branch are in our process not only a functioning product which passes its regression tests, but also completeness and consistency of the property contracts, and completeness of the assurance case fragments. Hence assurance becomes part of the continuous integration machinery which is also an enabler for CD.

This work process can enable, for instance, incremental expansion of the operational design domain for an ADS function [2], i.e. frequent software updates where each new version improves the performance or expands the region where the function can operate.

This research has been supported by Vinnova - Sweden’s innovation agency, via the project ESPLANADE (ref 2016-04268).

1Most of the work was done while Rolf Johansson was with Zenuity AB.
II. BACKGROUND AND RELATED WORK

A. Continuous Deployment

The aim of continuous deployment is frequent releases, and therefore requires efficient processes to reduce the time required for developing, testing and releasing software. There is a progressive use of automation in continuous integration, where build and test is automated, continuous delivery, which is the practice of ensuring new software versions can be reliably released at any time, and continuous deployment, which is the practice of deploying every update directly to the customer [3]. Integrating the operations phase with logging and monitoring of the released products with feedback to development, is the practice of DevOps [4]. Continuous deployment is typically done in the context of an iterative development process, e.g. one of the many agile methods, where new working product versions are developed in a predictable and high pace. For these practices to make sense for dependable systems, the methods and tools must be adapted to preserve the advantages despite the need for maintaining an assurance case and providing more comprehensive documentation than may be the case for non-critical software.

Safe Scrum [5] proposes a way to combine scrum with the requirements of IEC 61508. In Safe Scrum, only software development is done with the Scrum methodology, while concept phase, hazard analysis, and hardware development are done with traditional methods. This is based on an observation that safety requirements are quite stable, while functional requirements change more rapidly. A later extension also includes planning and analysis phases to allow for an agile safety case [6]. We focus on systems where safety requirements are not stable since the entire function is inherently safety critical, e.g. an ADS, hence the need for a process where the assurance case and assessment evolve together with the scope and requirements of the product.

B. Component-Based Development and Contracts

In software design, design-by-contract [7] programming - where the interaction of software elements is guided by contracts which specify what each element expects, guarantees and maintains - was pioneered in the 1980s. For safety-critical systems, safety contacts described in works such as [8], [9] have been proposed, for instance for the automotive domain [10]. The main difference from design-by-contract is that safety contracts are focused on the properties of a system that may lead to a breach of safety requirements. Using safety contracts, the safety properties of elements can be matched in an, at least partly, automated way [11]. With formal contracts [12], [13] both the validity of component aggregation on the same level of abstraction and contract refinement can be proven. A drawback is difficulty to express an entire complex safety-critical function in formal semantics, something that has so far prevented wide adoption of such contracts. While we do not exclude the use of formal contracts, our described method instead relies on semi-formal contracts together with argumentation and assessment to provide confidence in their validity.

C. Incremental and Modular Safety Cases

In [14], the authors describe how to build a preliminary safety case with an outline of the argument early in the development process; the case is then evolved with the design. A key advantage is to resolve risks associated with certification as early as possible, e.g. avoid late discovery that the safety strategy does not hold. This is similar to how we use forward-looking assurance cases to plan updates of assurance cases as part of the development work. Stålhane et al. [6] proposes a similar evolutionary approach to the safety case, focusing on reuse of existing artefacts for new product versions, but they do not discuss variants or use with component-based design.

Component-based development and assurance is combined by Fenn et al. [15] in an ambitious effort within the avionics domain. The aim was to reduce re-certification effort for changes in the system and the proposed solution was modular safety cases where the impact of changes can be analyzed and only affected modules need to be updated. This work uses dependency-guarantee relationships, which are like safety contracts. They also stress the importance of creating a modular safety case architecture with containing changes in mind and address the subject of safety case for third-party modules. This is the work most similar to our proposal, though we also stress the integration with continuous deployment.

Sljivo et al. [16] present a method to generate argument-fragments with supporting evidence for reusable components from safety contracts, and has also extended this work to allow for other assurance concerns than safety [17].

D. Argument Patterns

A structured argument can be used to show how safety [18] or any other quality attribute is achieved. Such an explicit way or showing the reasoning can be beneficial to make an assurance case understandable and easier to maintain and modify. However, an argument for a complex product can also become very big, and therefore requires careful management not to become unwieldy. One way to manage complexity is to increase uniformity of the argument with patterns. Much work has been done in this area, some examples for the automotive domain are [19], [20], [21], and for a multi-concern assurance case [22]. We use argument patterns to make the assurance case fragments more uniform and efficient to update.

III. GOALS AND PRINCIPLES

A. Solving a Multi-Dimensional Development Puzzle

Many standards and methods of working with assurance describe the context as a one-shot effort for a single product that is developed from blank slate to released product. This neglects to take the complicated product and development structure for many large systems into account. Often, several features are built using a common platform and include reusable components. There may be a mix of components developed in-house, typically with more freedom for alterations, and components or even larger subsystems developed by suppliers. Each feature built on top of this substrate may exist in several variants, customized for different customers or use-cases, and
must be able to evolve through many versions throughout its lifetime. It is a challenge to create a structure to cope with all these dimensions while maintaining the integrity of an assurance case. Hence, we believe integrating assurance more tightly with development and having the same support for versioning and product lines for both activities is one crucial part of enabling continuous assurance.

B. Separation of Concerns

Separation of concerns is a design principle long used to manage complexity and facilitate re-use, and is a key principle underpinning the practice of component-based design. It is also a useful paradigm for accommodating distributed development, e.g. using third party components or outsourcing part of the development work, which is common in the automotive and other domains. A note in the second edition of ISO 26262 also acknowledges that “In the case of a distributed development, the safety case of the item [i.e. a system or array of systems to implement a function at the vehicle level, to which ISO 26262 is applied] can be a combination of the safety cases of the customer and of the suppliers […]”. Therefore, we use component-based assurance case fragments to align assurance with design and facilitate re-use not only of the actual components, but also of their assurance cases fragments.

C. Simplicity With Recursive Patterns and Automation

It is not uncommon that safety aspects are mainly managed by safety experts, with a lower level of involvement by a majority of the developers. A danger with agile development, which has a strong focus on delivering functionality (the principle of “working software over comprehensive documentation”), is that the developers could be even less likely to get involved with the safety aspects, unless it is an integral part of the development process.

To avoid that assurance is handled separately causing bottlenecks and handover mistakes, the development teams need to be able to keep the assurance cases up to date as part of releasing new functionality. In addition, the assurance case is an important tool for impact analysis when updating a feature. However widely used processes need to be simple, flexible and efficient. Experience tells us that complex or rigid methods, theoretically sound as they may be, are seldom successfully deployed large-scale. Therefore, the underlying argument of the assurance case should be easy to interpret to allow for efficient change impact analysis and modification. In many standards, such as ISO 26262, the underlying argument structure is implicit [23]. In the proposed way of working, the maintainability of assurance cases is improved by making the argument explicit, and to achieve simplicity, patterns should be developed when possible to provide uniformity, which also facilitates automation.

D. Avoiding Big-Bang Releases

A known danger with the traditional long release cycles is problems surfacing close to release, frequently leading to delays in introduction of a new product or updated version, e.g. due to integration problems late in the project, or inability to accommodate changing requirements. For projects requiring assurance, an added uncertainty for waterfall projects is late discovery that the project has failed to fulfil the assurance objectives. The use of progressive safety assessment can alleviate some of the problems and is mentioned in a note in the 2018 revision of the ISO 26262 standard: “To support progressive functional safety assessments […] the safety case can be released progressively as work products are generated”. However, the use of agile development and continuous deployment is a way to get rid of big-bang releases altogether. So rather than progressive assessment spanning from project initiation to release, we advocate making the assessments an integrated part of development, to make it possible to update assessments at the same pace as the product, i.e. at every development iteration.

E. Feedback of Operational Data

In recent years, there has been an increased focus on integrating development and operations [4]. DevOps goes somewhat beyond continuous deployment in also considering monitoring, which enables the use of operational data as input to further development. We argue that the use of operational data can also be a valuable tool for assurance for a system that evolves with frequent updates. For instance, a safety requirement may be formulated in a conservative manner for earlier iterations of a feature if not enough field data is available, but refined to provide improved functionality once sufficient data is available to provide confidence for its use. This is an approach similar to but more conservative than dynamic safety cases [24] where the safety case can include update rules triggered on operational data indicating possible need for safety modifications. We believe the conservative approach is often more appropriate. Forward-looking assurance cases is a way to plan such iterative updates.

IV. CONTINUOUS ASSURANCE CASES

We will use a safety case for ISO 26262 to illustrate our proposed way of working; however, we believe a similar structure can be used for many other standards and QAs as well. ISO 26262 as well as many other functional safety standards use a V-model where a feature is represented in several abstraction and aggregation levels, and organizes the standard requirements around these levels. The V-model of ISO 26262 is illustrated in Fig. [1]. The “left leg” represents design abstraction levels starting with a high-level description (item definition) and hazard analysis at the top; the feature is then first described with a functional/logical architecture which is refined progressively towards the actual implementation at the bottom of the V, providing traceability for safety requirements along the way. A key task for each refinement step is to show that the architecture on the lower abstraction level is complete and consistent with respect to the previous level - verification of refinement - in particular when it comes to the safety requirements. The “right leg”, from bottom and
up, represents increasing levels of aggregation of components making up the feature, where the goal of each level is to verify larger integrated parts of the feature for every step - verification of implementation. The right leg starts from individual software or hardware units at the bottom and ends with validation of the feature in its intended environment (a vehicle) at the top.

Fig. 1. ISO 26262 V-model.

To organize an assurance case to cope with the realities described in Sec. III-A and the component-based nature of typical products, we advocate structuring it as an aggregation of assurance case fragments, where each such fragment will provide all the evidence (both product and process based) needed for one component of the product. These fragments will be updated and re-assessed every time the corresponding component changes. Another important property is that the components should be re-usable for several products or variants, hence a well-defined interface and re-usable format is needed for the assurance case fragments. For a well-defined interface, the concept of safety contracts is well known in literature as discussed in Sec. II-B. Hence we propose to use quality attribute specific (e.g. safety) contracts in a way outlined in Sec. IV-A, together with an assurance case fragment as outlined in Sec. IV-B. Contracts and assurance case fragments should be used for components on all abstraction levels, not just on the implementation level, since a crucial part of the fragment is to contain the refinement argumentation (fulfilling the requirements of the V-model left leg), and integration verification evidence for the implementation (right leg).

In Fig. 2 we illustrate the complete integrated feature as a jigsaw-puzzle where component safety case fragments from different abstraction and aggregation levels contain the argumentation and evidence they need to fulfill the standard requirements. It is important to remember, however, that for an automotive safety case, the safety property is only defined on the vehicle level, i.e. the component safety case fragments themselves do not say anything about the safety of the item. Therefore, there will be a top-level (item) safety case fragment on vehicle level containing item definition, hazard analysis, safety goals (top-level safety requirements) and final validation. The other abstraction levels have safety case fragments for each constituent component. This top-level safety case will reference the highest abstraction-level components. As the safety case fragments on each level will further reference the next-level components as part of their refinement argument, the safety property is only shown with the recursive unfolding of the safety case fragments as support for the top-level safety case. In other words we note that the general pattern of an assurance case looks different depending on whether it is related to the entire feature on the level where the assurance property has a direct meaning, or if it is related to a component in a product structure, where the assurance property is rather related to requirements in an architecture which gives semantics to what the assurance property means.

This configuration into components with separate assurance case fragments, contracts and top-level assurance cases for the item is an enabler for several important goals:

1) Re-usable components - since the components and their assurance case fragments are confined by the contracts and not by the item safety goals.

2) Product lines - since many similar products can be maintained using multiple top-level assurance cases. Proper configuration management for components including their assurance case fragments will ascertain proper change management for the entire product line.

3) Efficient update process - since the contracts also provide impact analysis, i.e. as long as the contract is stable a change will not propagate outside the component, if the contract changes the connections provide impact traceability.

A. Component Contracts and Their Relations

Several proposals of safety contracts exist, but the common denominator is that the component will have attached guarantees (what the component promises) and assumptions (what the component requires from the environment for the guarantees to be valid), and that these guarantees and assumptions somehow express safety-critical properties of the component. How this
is done varies. We propose to use semi-formal descriptions, which depending on type of contract and abstraction level can include information such as failure modes and tolerance margins. While formal contracts do exist [12], the difficulty of expressing such contracts have so far prevented widespread use and would contradict our goals of simplicity and integration in the general development process. A preliminary example of how we envision these contracts is illustrated in Fig. 3. In this example, $r_p$ is an assumption for component $P$, while $q_A$ is an assumption of component $A$; similarly, $p_D$ is a guarantee from component $D$ and $p_A$ of component $A$. When $A$ is implemented as an aggregation of components $P$ and $D$, it is necessary to check if the contracts match. For simple relations, such matching may be done with propositional logic. However, in order to avoid the need for full formal semantics, semi-formal argumentation may also be used for some aspects. This argumentation of contract matching is not part of the contract, but of the argument in the assurance case fragment of the aggregated or composed component.

**Assurance Case Fragments**

The assurance case fragments will differ in content depending on which quality attribute and standard it is for, and also which abstraction and aggregation level it pertains to. Fig. 5 shows how the ISO 26262 V-model can be illustrated in terms of these two dimensions. The content of the assurance case fragment is different depending on which coordinate in this diagram the component is located in, commensurate with the standard requirements from the corresponding position in the V-model. We exemplify with a description of fragments for an ISO 26262 automotive safety case. On a high level, each fragment will contain the same basic aspects as a full safety case according to the standard, however only the parts applicable for each component context are instantiated. Thus, the fragment shown in Fig. 2 contains its own safety plan, safety argument and safety work products, and is confined by the safety contract. The assurance case fragment is the subject of safety assessment further discussed in Sec. V.

1) **Top Safety-Case Fragment:** For an automotive safety case, e.g. an automated driving system feature for a road vehicle, a key part of the argument in the top-level safety-case fragment would be that the specific feature is always safe in its context. Given such a safe feature definition, a functionally safe implementation is also needed; this calls for following ISO 26262. There are many ways to organize the top-level argumentation for a QA, e.g. the QA pattern in [22]. Some parts would be safety process capability, the completeness and correctness of the hazard analysis and risk assessment (HA&RA), and fulfillment of all the Safety Goals produced by the HA&RA. With the component-based strategy we use, fulfillment of the safety goals is shown as a recursive argument that the feature is built as a safe integration of safely implemented parts (i.e. components with their own safety case fragments).

2) **Component Safety-Case Fragment:** A safety case fragment related to a component being part of this feature needs another top structure. What is needed is a claim that the component is implemented and used according to its safety
contract. The argument for implementation will show con-
formance to the ISO 26262 standard requirements from the
abstraction/aggregation level of the component (i.e. design
and/or V&V requirements) and therefore, as explained, there
will be different patterns for each position in the V-model. On
a high level, however, the argument for usage according to
the contract will follow a similar pattern regardless of position
and QA; this pattern is depicted in Fig. 6 and shows that the
different contract types according to Fig. 3 are correctly used.
The argument also needs to show that the organization (team)
developing the component has sufficient process capabilities
for this QA; this is further discussed in Sec. V-B. The pattern
is shown in GSN v2 notation explained in the GSN community
standard [25].

C. Change Management - Forward-Looking Assurance Cases

A typical aspect of agile development is working iteratively
with relatively short loops, followed by delivery of working
software. As a shippable product within the framework we
propose does not only consist of tested and working software,
but also of an up-to-date assurance case which has been
assessed, it follows that these iterations will also include
the required assurance activities. E.g. part of the planning
of the coming iterations in the Scrum framework include
to make sure the content of the work is specified to a
level where it is clear what needs to be done (Definition of
Ready), and there also needs to be an understanding of what
should have happened before the team declares the work to
be completed (Definition of Done). Hence, for Definition of
Ready, there should be an understanding of the impact of the
proposed change not only for the implementation, but also
for the assurance cases. When analyzing the impact on the
assurance cases, a forward-looking assurance case is created,
indicating the necessary changes needed for the proposed
work. Performing these changes and having them assessed
(see Sec V) are part of Definition of Done, together with the
more traditional activities such as testing and integrating the
changes. The Scrum backlog is usually managed so that highly
prioritized items close to the top are more detailed, while items
further down are less detailed. This can also be the case with
forward-looking assurance cases. It is sometimes necessary to
plan more than one sprint ahead to accommodate activities
with longer timeframes, e.g. for hardware dependencies or
interacting with third party suppliers.

Assuming a change of an existing product like for example
the reuse of a developed component in a new context; What
will be the impact on the integrity of the assurance case?
The replaced component has an associated contract giving
guarantees provided assumptions are fulfilled in the context.
The context has arguments on how the component and its
contract matched other components in the context and their
contracts. Clearly, this argumentation needs to be revised.
Either an analysis give that the contracts of the context is
matching the new component contract or there is an impact
on other components. This way of analyzing the assurance
case would give an early indication on required changes to
keep the integrity of the assurance case. Fig. 7 shows three
examples of changes in a model with three abstraction levels
(A-C) and a number of components on each level. Here we
describe what the forward-looking assurance case for safety
consists of for each of these changes:

![Fig. 7. Three examples of changes (dark elements) between releases.](image)

Fig. 6. Assurance case fragment argument pattern on component level.
1) **Release n+1:** The definition of the feature is updated. The previous release was conservative in its definition of a certain parameter (e.g., part of the ODD definition for an ADS). The components of the solution had their contracts limiting possible failure modes. The confidence in the solution has increased during the collection of operational data from the fleet of vehicles, and it was decided that the feature performance shall be increased by modifying this parameter. A forward-looking safety case is developed for release n+1. By using the evidences from the field and revisiting the argumentation for the goals it is found that the tolerances stated when limiting the failure modes are within those of the new parameter. The forward-looking safety case states that safety can be maintained with the same solution.

2) **Release n+2:** A non-safety-related issue with a component in the solution has been found and the component shall be updated. The contracts are stable and the forward-looking safety case states that evidences shall be revisited for the updated component only.

3) **Release n+3:** A different solution is requested by a new customer, this shall be released as a new variant. The feature definition is different, and a new solution is assembled from available components. Each component has its contracts and the argumentation on the matching of related contracts are checked by means of comparing the tolerances given in the requirements limiting possible failure modes. The forward-looking safety case asks for updates of contracts where the argumentation and/or component design is incomplete with respect to this new variant.

We can find other sources that challenges the assurance case argumentation. Kelly [18] exemplifies with changes in goals, evidences and context in a damage phase, corresponding to the described planning leading to a forward-looking assurance case and tasks in the backlog; and a recovery phase, corresponding to the update needed to reach Definition of Done.

**V. CONTINUOUS ASSESSMENT**

**A. Integrating Assessment With Development**

Confidence in continuous and modular assurance cases requires continuous assessment of elements on different levels. The assessors need to continuously be convinced that the contracts hold so that the elements can be trusted in their intended contexts. ISO 26262 [1] requires independent assurance of items and elements through the concept of confirmation measures. These consist of confirmation reviews, functional safety audit and functional safety assessment. The confirmation reviews are intended to check the compliance of selected work products to the corresponding requirements in the standard and judging whether the key work products provide sufficient and convincing evidence of their contribution to the achievement of functional safety. The purpose of the functional safety audit is to check the implementation of the processes required for functional safety, and the functional safety assessment is to judge if functional safety is achieved.

For a continuous safety case we suggest that confirmation reviews and functional safety audits are merged and combined with the functional safety assessment and is performed in a cadence. Cadence is a key construct in all agile development. It provides a predictive heartbeat whether it is a small team or large organization and opens for a continuous execution of the functional safety work through the Shewhart/Deming PDCA (Plan-Do-Check-Adjust) improvement cycle, see Fig. 8.

![Fig. 8. PDCA cycle for continuous safety case](Image)

As mentioned in Sec. [IV.C], a forward-looking assurance case is updated as part of the preparation for the learning cycle. It points out the argument branches in the assurance case that are in focus for the coming iteration or increment, and hence identifies the evidence that needs to be generated. As an assurance case for ISO 26262 would contain both product and process branches, the items in the backlog can also point to the process side of the argument in order to gain more confidence in the team’s abilities to manage functional safety. The interaction between development team and assessor in the improvement cycle will entail:

- In the **plan** phase the team and the assessor reasons about the forward-looking assurance cases in the backlog to gain a common understanding. This enables the team to commit to the backlog items and plan for the iteration.
- In the **do** phase the team produces the agreed evidence and updates the assurance case.
- In the **check** phase the assessor performs an incremental assessment of the assurance case and indicates for the team the level of confidence in the claims.
- In the **adjust** phase the assessor’s level of confidence is jointly discussed with the team. The focus is what can be improved in the next cycle.

**B. Process Capabilities for Development Teams**

A necessary claim is often that the developing organization has sufficient process capabilities. To substantiate such claims, we need a consistent way to measure the process capabilities in the teams with regards to the quality attribute we want to assess, e.g., functional safety. In the automotive industry, the VDA/QMC standard Automotive Software Process Improvement and Capability determination (Automotive SPICE) [26] is widely accepted as a Process Assessment Model (PAM).
covering system and software process quality. However, it does not cover the safety domain. The Swedish standard SS-7740 \cite{27} is a standard based on Automotive SPICE and complemented with safety processes and safety indicators to cover the requirements in ISO 26262. The result from a SS-7740 assessment is a process capability profile covering both quality and safety. For the development teams SS-7740 assessment can propel a continuous improvement cycle leading to higher capabilities to develop automotive software in the safety domain and at the same time provide the evidence that the developing organization has sufficient process capabilities.

VI. CONCLUSIONS AND FUTURE WORK

We believe state of practice for agile development and continuous deployment of products needing assurance is still immature, especially for domains where complex product structures are the norm. On the other hand, we argue that the basic building blocks for achieving a high level of efficiency and frequent deliveries already exist. By combining the use of component-based design, contracts, and modular assurance cases with agile practices, we can achieve continuous assurance cases that evolve and are assessed in the same pace as, and together with, the product. However, some pieces of the puzzle are still missing, one of them is tool support for efficiently integrating assurance and other development work, e.g. for efficiently performing the assessment and communicating the results, and to check the consistency of the assurance cases. While this paper has outlined a proposed process, many details remain. Our future work is on how to formulate and use semiformal assume-guarantee contracts in this context, and on more detailed argument patterns for the continuous assurance cases.

REFERENCES

[26] VDA. Automotive SPICE - Verband der Automobilindustrie e. V. (VDA).