A Safety-Centric Change Management Framework by Tailoring Agile and V-Model Processes

Omar T. Jaradat, Ph.D. Candidate; Mälardalen University; Västerås, Sweden

Abdallah M. Salameh, Ph.D. Candidate; University of Salford, Manchester, UK

Keywords: safety case, contracts, impact analysis, change management, agile software development, agile tailoring, V-model, XP, Kanban

Abstract

Safety critical systems are evolutionary and subject to preventive, perfective, corrective or adaptive changes during their lifecycle. Changes to any part of those systems can undermine the confidence in safety since changes can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. Changes to the software components are no exception. In order to maintain the confidence in the safety performance, developers must update their system and its safety case. Agile methodologies are known to embrace changes to software where agilists strive to manage changes, not to prevent them. In this paper, we introduce a novel framework in which we tailor a hybrid process of agile software development and the traditional V-model. The tailored process aims to facilitate the accommodation of non-structural changes to the software parts of safety critical systems. We illustrate our framework in the context of ISO 26262 safety standard.

Introduction

Many safety critical systems are subject to compulsory or advisory certification process which often necessitates building the systems in compliance with domain-specific safety standards (Jaradat & Bate, 2017). Safety standards are becoming the main guide of the development and maintenance of hardware and software parts of safety critical systems. Safety standards, also, form the basis for the approval and certification of those systems (Denney et al., 2015). Software systems, in general, are subject to different types of changes (e.g., preventive, perfective, corrective or adaptive) during the different stages in their life-cycle. In order to maintain the confidence in safety after accommodating a change, developers are required to update the safety case, which in turn requires identifying, re-analysing, and re-checking the impacted parts of the system and generate a new valid set of evidence (Jaradat & Bate, 2017). Despite the obvious recommendations to adequately maintain and review the systems and their safety cases by different safety standards, the latter offer little or no advice on how such operations can be carried out (T. Kelly & J. McDermid, 1999). There is an increasing need for globally-accepted methods and techniques to enable easier change accommodation in safety critical systems without incurring disproportionate cost compared to the size of the change. However, since broader re-verification and re-validation require more effort and time, it is important for any proposal aims to facilitate system changes to delimit the impact of changes.

Safety standards in many safety critical system domains adopt the traditional V-model as a development process for building the systems. Despite the effectiveness of validation and verification that the V-model provides, in addition to other advantages (e.g., easy to estimate costs, create timeliness, and stick to deadlines), the model has a well-known drawback when it comes to handling system changes. This is particularly true when it comes to changes to software systems and their requirements. Following the V-model implies that changes to software components requires re-visiting the system requirements and all later stages to perform a broad and costly impact analysis process. Hence, accepting software changes while using a V-model based process is not a trivial task.

Unlike the series of isolated phases in the V-model, agile methods depend on iterative and incremental development of software to enable reduction in cost, acceleration of time to market in addition to the focus of providing more maintainable code (Salameh, 2011; Tarwani & Chug, 2016). Software developers who follow agile methods breakdown their project into manageable fragments which enables a rapid responsive way to handle software changes. The Agile way of working minimises the shortcomings of traditional sequential methods and improves the software development process in a more cost-efficient way (Tarwani & Chug, 2016). The alignment of the development process with a dynamic environment is a critical motivation for adopting Agile Software Development (ASD) (Cao et al., 2010). Test Driven Development (TDD) is an important agile process that brings many benefits such as reducing the potential consequences of software defects. TDD protects the system from future failures proactively, which leads to an acceleration of the maintenance process (Knippers, 2011).

The work in this paper does not seek to conduct a comparative study between agile methods and the V-model. The main contribution of this paper, however, is to propose XP-Kan-Safe as a novel maintenance framework to facilitate the accommodation process of software non-structural changes in safety critical systems by utilising the strengths of agile methods and the V-model. More clearly, we reconcile the known effective validation & verification process of the V-model to the known effective practices and the TDD process of agile methods. We exploit the usage of safety contracts (Bate et al., 2003) as: 1) stitches that connect the V-model, Extreme Programming (XP) and Kanban into our tailored process, and 2) means to enable a tri-directional impact analysis process. The hypothesis we make is that ASD can resolve some observed maintenance challenges in the V-model while maintaining software parts of systems.

Background and Motivation

Safety Cases and Safety Arguments

A safety case (also known as assurance or safety assurance case) is: “*A structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment*" (00-56 Standard, 2015). A safety case shall comprise both safety evidence (e.g., safety analyses, software inspections, or functional tests) and a safety argument explaining that evidence (Jaradat et al., 2014). Safety cases might contain an implicit safety argument, but some safety standards require an explicit argument that is usually expressed in terms of a defined hierarchy of safety claims and subclaims that are supported by a body of evidence (00-56 Standard, 2015). There are several ways to represent safety arguments (e.g., textual, tabular, graphical, etc.). In this paper, we use the Goal Structuring Notation (GSN) (GSN Standard, 2011), which provides a graphical means of communicating (1) safety argument elements, claims (goals), argument logic (strategies), assumptions, context, evidence (solutions), and (2) the relationships between these elements (Jaradat et al., 2015a). Figure 1 shows the main notations of the GSN.

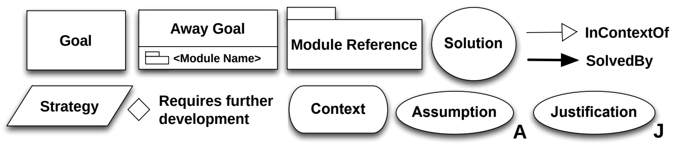


Figure 1— Notation Keys of the GSN

Maintenance of Safety Critical Systems and Their Safety Cases

Change requests should be assessed before decision makers decide whether or not to accept them. The assessment should reveal if the change can cause unreasonable risks, and the required cost to implement the change. Hence, system developers should understand the change and the potential risks that it might carry before they identify the impacted parts. Misunderstanding the change might lead to skip those parts of the system which are dependent on that assumptions. Also, the developers need to understand the dependencies between the system parts to identify the affected parts correctly. For example, the effect of a change can propagate to other parts of the system — creating a ripple effect — and cause unforeseen violations of the acceptable safety limits. If the impact of change is not clear, developers might be conservative and do wider analyses and verification (i.e., check more elements than strictly necessary), and this will exacerbate the cost problem of safety cases. It is also necessary for the developers to describe how the change affects the system parts in order to correctly estimate the cost of the response to that change. Otherwise, the response to a change might generate unplanned further changes to which the system must again respond, and this requires more cost than originally estimated.

ISO 26262 Safety Standard

ISO 26262 (ISO 26262:2011, 2011) regulates the automotive domain and it is intended to be applied to safety-related systems that include one or more electrical and/or electronic systems. The following parts are summarised descriptions of the safety requirements decomposition directly from ISO 26262 guidelines:

1. After identifying hazards, the standard recommends formulating Safety Goals (SGs) to eliminate or mitigate hazards. The standard defines a safety goal as a top-level safety requirement resultant of the hazard analysis and risk assessment. Safety goals are not expressed in terms of technological solutions, but in terms of functional objectives.

2. Identification of SGs leads to the functional safety concept. The objective of the functional safety concept is to derive the Functional Safety Requirements (FSRs) from the SGs, and to allocate them to the preliminary architectural elements. At least one FSR shall be specified for each SG. Derivation of FSRs can be supported by safety analyses (e.g., Failure modes and effects analysis (FMEA), Fault Tree Analysis, Hazard and Operability Study (HAZOP)) in order to develop a complete set of effective functional safety requirements.

3. The functional concept leads to the technical safety concept. The first objective of the latter is to specify the Technical Safety Requirements (TSRs) and their allocation to system elements. The second objective is to verify that the TSRs comply with the functional safety requirements. TSRs are used to derive Software Safety Requirements (SSRs).

Safety Contracts

Contract-based design (Benvenuti et al., 2008) is defined as an approach in which the design process is seen as a successive assembly of components where a component behaviour is represented in terms of assumptions about its environment and guarantees about its behaviour. Hence, contracts are intended to describe functional and behavioural properties for each design component in form of assumptions and guarantees. A contract is said to be a safety contract if it guarantees a property that is traceable to a hazard. Using contracts in development of safety critical systems is not a novel idea since there are many works utilise contracts for building, reusing or maintaining safety critical systems (e.g., (Bate et al., 2003; Jaradat et al., 2015a; Jaradat et al., 2015b)). The cost of maintaining, reusing and changing software components is lessened while using contracts as developers may rework software components with knowledge of the constraints placed upon them (Bates et al., 2003). In this paper, we use contracts to support the maintainability of safety critical systems. We also suggest to include additional information into safety contracts in order to enable effective traceability.

Agile Software Development (ASD)

Compared to traditional software engineering approaches, ASD targets complex systems and product development with dynamic, non-deterministic and non-linear characteristics. ASD methods (e.g., XP, Kanban, Scrum) evolve through collaboration between self-organising and cross-functional teams by sharing the same philosophy and utilising the appropriate practices for their contexts.

Each agile method has its own set of features (e.g., practices, terminologies, and tactics) and those features should reflect ASD values and principles. However, agile methods vary when it comes to the strategies they adopt to reflect those values and principles. For example, Kanban is known to have a rapid response to software requirement changes since it allows the team to instantly postpone some change requests to start with other emergent requests. Scrum might do the same but not after the completion of a sprint planning meeting and team commitment. XP teams are amenable to change within their iterations as long as a team has not started work on a particular feature that needs to be exchanged with the new feature. There is no standard recommendation as to how an agile method should implement its features (Campanelli & Parreiras, 2015).

Organisations, typically, adapt software development methodologies to be in line with their needs and contexts, which covering the full spectrum of the software development life-cycle (Heeager & Rose, 2015; Salameh, 2011). In fact, there is no single agile method that can be adopted for any arbitrary context or to efficiently cover all phases in the development life-cycle.

Hence, organisations might not adopt an entire agile method, but rather they combine different processes from different agile methods based on their needs and contexts.

Agile Tailoring

The process in which an agile method is adapted for a specific project situation in a responsive way to accommodate the encountered challenges and to cover the indented interplay between contexts in a dynamic way, is called Agile Tailoring. There are two main approaches to tailor agile methods: the contingency factors and the method engineering theory (Campanelli & Parreiras, 2015). The first approach handles the tailoring by choosing multiple methods to be on standby in an organisation (i.e., Crystal family (Abrahamsson et al., 2003)). The selection of any standby method is based on project size and criticality, as well as the development context, such as uncertainty level, impact and structure. The second approach is based on meta-method processes and proposes the creation of a new method to be applied on specific contexts based on existing method fragments (a fragment represents a set of practices) (Campanelli & Parreiras, 2015). Despite the flexibility of this approach, it introduces challenges such as how to control the fragments or how to assemble the method for a context specific situation by bringing the appropriate fragments and integrating them into one framework (Campanelli & Parreiras, 2015). In this paper, we tailor our framework using the method engineering approach.

The Kanban Method

Kanban is based on lean principles: it tries to remove the waste of the production process by embracing rules to limit Work In Progress (WIP) and measures the time to finish the tasks (Campanelli & Parreiras, 2015). Kanban does not prescribe a specific set of roles or process steps, but rather it encourages its users to start from the existing context by understanding and emphasising the customers' needs (Ahmad et al., 2013). Kanban is deemed as an approach to process change for organisations by providing sufficient visibility and understanding of the workflow and its progress. Kanban is all about visual signs (aka Kanban Cards) which represent individual work items accompanied with their critical information. Those cards move across a board (aka Kanban board). The latter is partitioned by vertical lanes which are titled, typically, according to the names of the development life-cycle phases (e.g., Analysis, Development, testing). These lanes can be partitioned further to specify the current state of each phase (To Do, Doing and Done). The location of a card on the board indicates the progress of the work and its current state. Kanban shows the assigned work for each team member, communicates priorities and highlights bottlenecks via cycle or lead time and the cumulative flow diagram (Ahmad et al., 2013; Campanelli & Parreiras, 2015).

The XP Method

The XP method intends to improve software quality and responsiveness to the changing customer requirements. XP is considered a lightweight agile method that focuses on cost savings, unit tests before and along code activities, frequent full system integration and frequent releases (Campanelli & Parreiras, 2015). XP comprises five phases: exploration, planning, iterations to release, productionising, maintenance and death (Salameh, 2011). The exploration, planning and iterations to release are the only phases involved in our tailored framework.

During the exploration phase, the customers describe the features they wish to have in the first release of their system by writing each of them into a story card (Abrahamsson et al., 2017). Our tailored framework is designed to deal with changes to a system that has been already built by the V-model. Hence, the features are considered as changes to the software system in our case. More clearly, safety engineers (who represent the customers) write change requests into story cards and discuss them with the team manager. During the planning phase, the story cards should be prioritised, an agreement on the first small release should be made and the time span required to implement the story cards should be estimated (Abrahamsson et al., 2017). In the iteration and release planning phase, each release should be incremented by exactly one iteration. The development team should break down requested features (i.e., requested changes in our case) into several small releases. The customer selects the stories that should be implemented in a specific iteration. XP Planning Game is a close interaction between the customer and the development team. The latter should estimate the effort needed to implement the stories.

A Maintenance Framework to Facilitate Change Management

In this section, we build upon the background section to propose a new framework which aims to streamline the change management process of non-structural software changes in safety critical systems. The framework is referred to as XP-Kan-Safe and it comprises two main processes: The *Preliminary Process* and the *Change Management Process*. Figure 2 provides a conceptual model of the framework. The conceptual model encompasses three phases: 1) Analysis phase to cover the derivation of safety contracts, 2) Planning phase to cover the game planning, and 3) Implementation phase to cover the TDD and other XP practices. The grey background of the model represents the Kanban board.

The Preliminary Process

This process is preparatory and should be performed before handling changes. The main objective of this process is to derive safety contracts and enrich them with additional information to increase the traceability between the requirements (i.e., guarantees) and different related artefacts. The activity of deriving safety contracts should start from the safety analysis phase. Safety analysis, however, is typically performed on different levels such as system, subsystems and components levels. The preliminary process enables system developers to derive contracts from safety analyses on the highest level down to lower levels. The preliminary process is applicable to any approach aims to decompose and specify safety requirements. The work in this paper, however, is designed to comply with ISO 26262 thus the derivation of safety contracts starts from the safety analysis through which SGs are derived.

After completing the safety analysis on the system level, safety contracts should be derived to guarantee the resultant SGs. A safety contract that guarantees a SG is referred to as “SG contract". The assumptions of a SG contract should capture the FSRs that fulfil the guaranteed SG. Furthermore, a contract should be derived to guarantee every assumed FSR in SG contracts after completing the safety analysis on the safety function level. A safety contract that guarantees a FSR is referred to as *“FSR contract"*. The assumptions of a FSR contract should capture the TSRs that implement the guaranteed FSR. Finally, a contract should be derived to guarantee every assumed TSR in FSR contracts; such contracts are referred to as *“TSR* *contracts"*. The assumptions of a TSR contract should capture the SSRs that implement the guaranteed TSR after completing the safety analysis on the software components level.

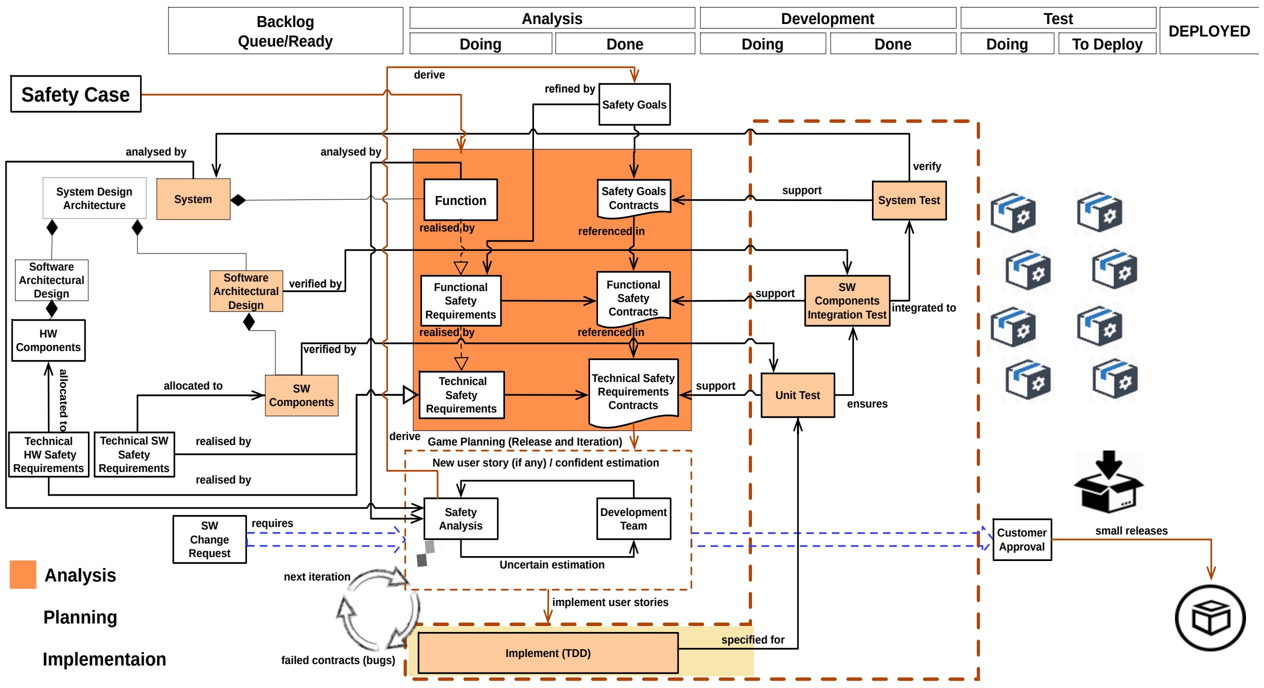


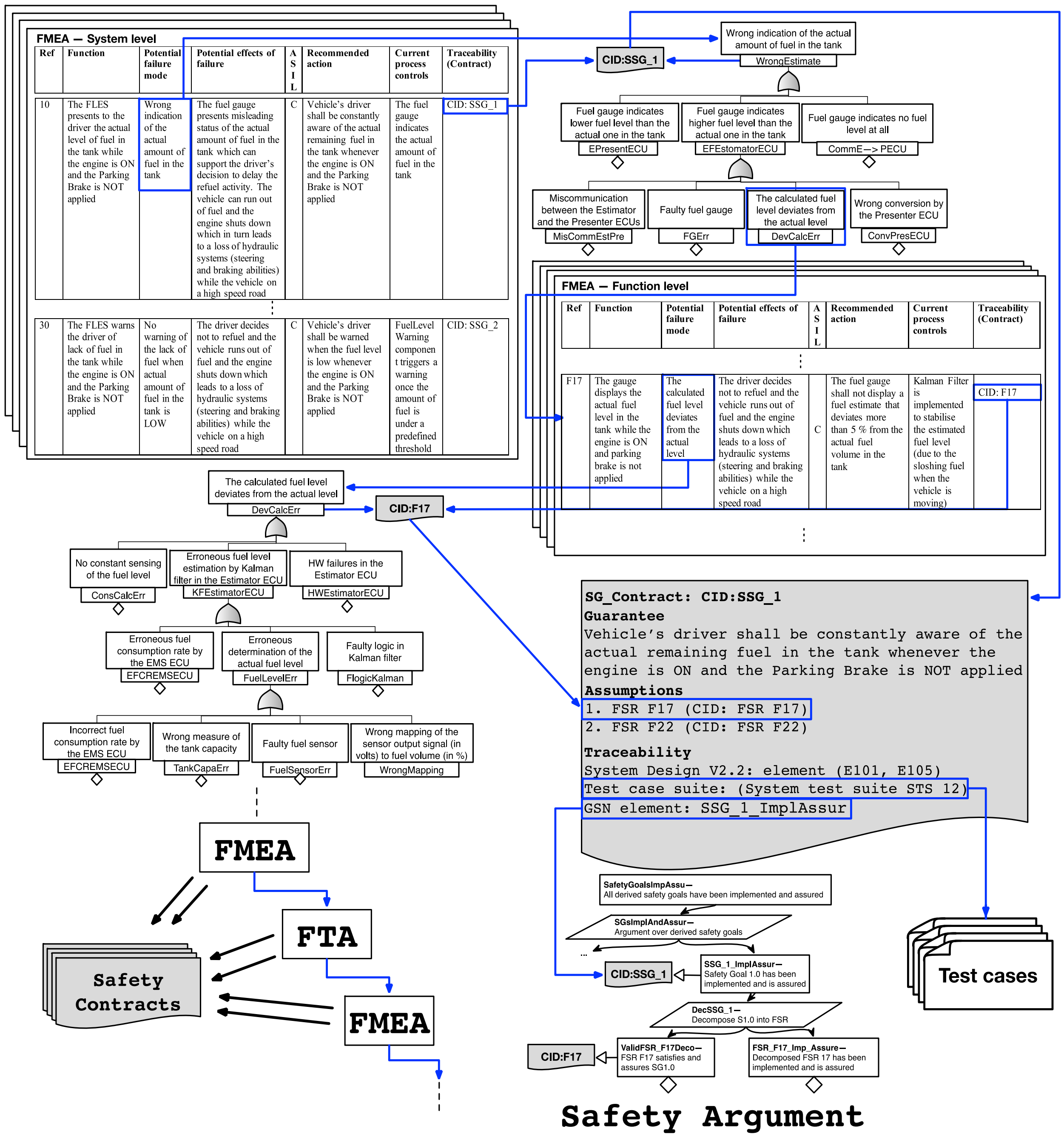
Figure 2— A conceptual model of XP-Kan-Safe framework

Failure modes and effects analysis (FMEA) is recommended by many safety standards (including ISO 26262) as a safety analysis tool to identify potential failures modes. We enable the derivation of safety contracts from FMEAs by adding an extra column to the FMEA table so that safety analysts, together with requirement engineers, should cite their derived contracts in it. FMEA might have a deficiency when it comes to multiple failures investigation. Hence, safety analysts might use different tools, such as Fault Tree Analysis (FTA) to search for the effects of multiple failures. Our preliminary process takes this into account and manages the derivation of safety contracts from FMEAs and FTAs. Figure 3 shows the connection between FTA and FMEA in addition to an example of a derived safety contract.

A guarantee in a contract and its related assumptions are the main elements of the contracts and they help to understand the relationships and the dependencies among the safety requirements. However, they might not be enough for analysts to identify the impacted artefacts and the elements in the GSN safety argument due to changes because they do not provide information as how the different parts are related to each other. For instance, identifying an impacted TSR will not directly lead to the impacted test cases and the items of evidence which need to be replaced. In order to enhance the traceability between the requirements (i.e., guarantees) and other related artefacts as well as GSN elements, safety contracts should be enriched with additional information. To this end, system developers should include additional information into the derived contracts as follows:

1. Elements in the system architecture: all derived safety requirements should be allocated to elements of the system architecture. However, since the changes we are after in this work

are non-structural, we assume that the changes have no effect on the system architecture.



Ref	Function	Potential failure mode	Potential effects of failure	ASIL	Recommended action	Current process controls	Traceability (Contract)
10	The FLES presents to the driver the actual level of fuel in the tank while the engine is ON and the Parking Brake is NOT applied	Wrong indication of the actual amount of fuel in the tank	The fuel gauge presents misleading status of the actual amount of fuel in the tank which can support the driver's decision to delay the refuel activity. The vehicle can run out of fuel and the engine shuts down which in turn leads to a loss of hydraulic systems (steering and braking abilities) while the vehicle on a high speed road	C	Vehicle's driver shall be constantly aware of the actual remaining fuel in the tank whenever the engine is ON and the Parking Brake is NOT applied	The fuel gauge indicates the actual amount of fuel in the tank	CID: SSG_1
30	The FLES warns the driver of lack of fuel in the tank while the engine is ON and the Parking Brake is NOT applied	No warning of the lack of fuel when actual amount of fuel in the tank is LOW	The driver decides not to refuel and the vehicle runs out of fuel and the engine shuts down which leads to a loss of hydraulic systems (steering and braking abilities) while the vehicle on a high speed road	C	Vehicle's driver shall be warned when the fuel level is low whenever the engine is ON and the Parking Brake is NOT applied	FuelLevel Warning component triggers a warning once the amount of fuel is under a predefined threshold	CID: SSG_2

Ref	Function	Potential failure mode	Potential effects of failure	ASIL	Recommended action	Current process controls	Traceability (Contract)
F17	The gauge displays the actual fuel level in the tank while the engine is ON and parking brake is not applied	The calculated fuel level deviates from the actual level	The driver decides not to refuel and the vehicle runs out of fuel and the engine shuts down which leads to a loss of hydraulic systems (steering and braking abilities) while the vehicle on a high speed road	C	The fuel gauge shall not display a fuel estimate that deviates more than 5 % from the actual fuel volume in the tank	Kalman Filter is implemented to stabilise the estimated fuel level (due to the sloshing fuel when the vehicle is moving)	CID: F17

SG_Contract: CID:SSG_1
Guarantee
Vehicle's driver shall be constantly aware of the actual remaining fuel in the tank whenever the engine is ON and the Parking Brake is NOT applied
Assumptions
1. FSR F17 (CID: FSR F17)
2. FSR F22 (CID: FSR F22)

Traceability
System Design V2.2: element (E101, E105)
Test case suite: (System test suite STS 12)
GSN element: SSG_1_ImplAssur

Wrong indication of the actual amount of fuel in the tank
WrongEstimate
CID:SSG_1
Fuel gauge indicates lower fuel level than the actual one in the tank
EPresentECU
Fuel gauge indicates higher fuel level than the actual one in the tank
EFEstomatorECU
Fuel gauge indicates no fuel level at all
CommE—> PECU
Miscommunication between the Estimator and the Presenter ECUs
MisCommEstPre
Faulty fuel gauge
FGErr
The calculated fuel level deviates from the actual level
DevCalcErr
Wrong conversion by the Presenter ECU
ConvPresECU
The calculated fuel level deviates from the actual level
DevCalcErr
CID:F17
No constant sensing of the fuel level
ConsCalcErr
Erroneous fuel level estimation by Kalman filter in the Estimator ECU
KFEstimatorECU
HW failures in the Estimator ECU
HWEstimatorECU
Erroneous fuel consumption rate by the EMS ECU
EFCREMSECU
Erroneous determination of the actual fuel level
FuelLevelErr
Faulty logic in Kalman filter
FlogicKalman
Incorrect fuel consumption rate by the EMS ECU
EFCREMSECU
Wrong measure of the tank capacity
TankCapaErr
Faulty fuel sensor
FuelSensorErr
Wrong mapping of the sensor output signal (in volts) to fuel volume (in %)
WrongMapping
FMEA
FTA
FMEA
Safety Contracts
SafetyGoalsImpAssu—
All derived safety goals have been implemented and assured
SGsImplAndAssur—
Argument over derived safety goals
...
CID:SSG_1
SSG_1_ImplAssur—
Safety Goal 1.0 has been implemented and is assured
DecSSG_1—
Decompose S1.0 into FSR
CID:F17
ValidFSR_F17Deco—
FSR F17 satisfies and assures SG1.0
FSR_F17_Imp_Assure—
Decomposed FSR 17 has been implemented and is assured
Test cases
Safety Argument

Figure 3— An illustration of a contract derivation by the Preliminary Process

2. Test cases: potential failure modes for which a safety requirement is derived should be considered as testing criteria during the verification phase to ensure the prevention of those failures. Including a reference to test cases in safety contracts enables direct traceability between safety analyses (i.e., FMEA and FTA), safety requirements (i.e., guarantees) and test cases. This traceability enables a top-down change impact analysis from the safety analysis down to the test cases. This top-down analysis represents the **first** direction of the tri-directional impact analysis process in our maintenance framework. While documenting the safety contracts, the reference of test cases might not be available as the test cases themselves might not be built yet. System developers are required to revisit each contract and add the corresponding test case references whenever they are made available. Furthermore, given that the test cases are available and complete, system developers can annotate them with the contracts' references. The annotations in the source code of the test cases are important to establish a traceability that enables a bottom-up impact analysis from the test cases up to the safety analysis. This bottom-up analysis represents the **second** direction of the tri-directional impact analysis process in our maintenance framework.

3. Elements of safety arguments: each safety contract should contain a reference to the related goals, contexts or items of evidence from safety arguments. Whenever GSN references are made available, system developers are required to revisit each contract and add the corresponding GSN reference to it. Including a reference to GSN elements in safety contracts enables direct traceability between a system and its safety case. This traceability enables a bi-directional impact analysis from the system to its safety case and vice versa. More clearly, an affected guarantee can lead to an affected GSN element. Since the safety case presents the logic of how different artefacts are related, impact analysts might use it to highlight the change impact in the related system. The bi-directional change impact analysis represents the **third** direction of the tri-directional impact analysis process in our framework.

Figure 3 highlights the suggested traceability information and connects them to specimen artefacts and a GSN element.

The Change Management Process

In this section, we describe the second process of XP-Kan-Safe. This process and its activities represent the result of tailoring ASD and the V-model. The main objective of this tailored process is to guide whoever involved in the change management activities from the arrival of a change until the generation of a new test results report. Figure 4 presents the flow of these activities. The Change Management Process activities are described as follows:

Activity 1: Understand the change and its impact in the system and its safety case. Once a change request is placed, *Activity 1* should be followed in which the safety engineers should understand the nature of the change and determine its potential effects in the system and its safety case. In order to initiate the Kanban management process, safety engineers should create a card that describes the change request in more technical specifications and visualise it as a WIP in the analysis phase. The outcome of this activity should provide plausible data about the impacted parts of the system and its safety case.

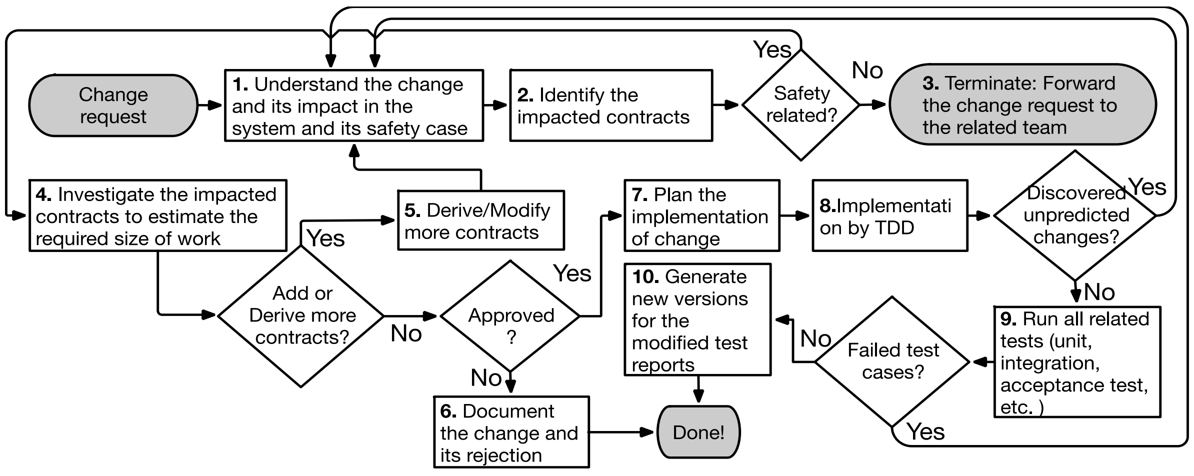

Change request
1. Understand the change and its impact in the system and its safety case
2. Identify the impacted contracts
Safety related?
Yes
No
3. Terminate: Forward the change request to the related team
4. Investigate the impacted contracts to estimate the required size of work
5. Derive/Modify more contracts
Yes
Add or Derive more contracts?
No
Approved ?
Yes
No
6. Document the change and its rejection
Done!
7. Plan the implementation of change
8.Implementati on by TDD
Discovered unpredicted changes?
Yes
No
9. Run all related tests (unit, integration, acceptance test, etc.)
Failed test cases?
No
Yes
10. Generate new versions for the modified test reports

Figure 4— The change management process of XP-Kan-Safe framework

Activity 2: Identify the impacted contracts. In this activity, all related safety contracts to the change should be identified. The benefit of applying the first process of XP-Kan-Safe (i.e., preliminary) will be more realised in this activity since using safety contracts should help to provide a systematic impact analysis through the utilisation of the tri-directional impact analysis. The identified contracts should be listed in the Kanban card.

Activity 3: Terminate: Forward the change request to the related team. If there is no safety contract identified as suspect in *Activity 2*, this implies that the change request has nothing to do with the functional safety in the system (no safety requirements are affected). In this specific case, the change request should be forwarded to the relevant team and no further continuation of the change management process is needed.

Activity 4: Investigate the impacted contracts to estimate the required size of work. There is no perfect impact analysis that can determine the effects of a change in the system and its artefacts at the first glance. That is, it is unlikely that the team will find out what might, precisely, get impacted merely by looking at the documented requirement and without iterating the impact analysis process. Hence, further investigation should be conducted to gain sufficient confidence in the perceived impact of a change. To this end, this activity should be followed to make further investigation of the impacted contracts. During this activity, a preliminary meeting should be carried out in which safety engineers, who represent the on-site customer with respect to XP, together with the development team, should determine the possibility of identifying more impacted contracts. Any additional identification of safety contracts should be added to the Kanban card. Safety contracts should support the collaboration between safety engineers and the development team to delimit the impacted parts of a system through the tri-directional impact analysis process. It is worth mentioning that any need to modify an existing contract or derive a new one will necessitate the application of this activity.

Activity 5: Derive new contracts or modify existing contracts. Since changes might introduce other changes, this might lead to modifying or deriving other contracts (i.e., requirements) that were not thought of earlier in *Activity 2*. In this activity, safety engineers and system developers derive new contracts or modify the existing ones to capture the newly introduced requirements or to update the already captured requirements, respectively. An initial cost of the change accommodation and its timeframe are two among several other factors upon which the approval decision is made. The involvement of the development team in the Activities 4 and 5 should cover the estimation of the initial amount of work and the time needed to complete it. Safety engineers and system developers should agree on: 1) what should be changed or added (i.e., size of the work) and 2) the acceptance of the accompanied potential risk on safety functions. Subsequently, they should submit their agreement to the management where the latter can either decline or accept the change request. Submission of the agreement concludes the Analysis Phase, and this means that the Kanban card should move on the board from (Analysis → Doing) to (Analysis → Done).

Activity 6: Document the change and its rejection. If the change request receives a rejection by the management, the change request, the performed investigation and the management decision should be documented (ISO 26262:2011, 2011). The rejection implies that the Kanban card should be closed.

Activity 7: Plan the implementation of change. If the change request receives an approval by the management, the Kanban card should be available for development. The adopted planning method, in our change management process, complies with XP. This implies that the implementation of the change request is initiated by the planning game. The input of the planning game is the estimated work and the impacted safety contracts. The output are more fine-grained estimated tasks than the earlier estimated tasks in *Activity 4*.

Activity 8: Implementation by TDD. In this activity, the implementation of the change is carried out using TDD. For those contracts that are subject to modification, system developers should find the related test cases (using the parameters that refer to them in the contracts) and modify them accordingly. Since modifying a contract might require creating new test cases, system developers should cite the newly added test cases in the corresponding contracts and vice versa. This is particularly important to support bi-directional traceability between the test cases and the contracts while is deemed as a preparation for future changes. Citing the newly added test cases in the contracts applies to the derived contracts during the impact analysis process — after the preliminary process — as well *Activity 5*. Moreover, after implementing required production code to satisfy the derived test cases, other already existing test cases might get impacted by newly added code. If the solution is to modify or add new requirements, system developers should inform the safety engineers about the suggested changes to the requirements. In this case, the suggested changes by system developers should be declared as unexpected changes. Afterwards, safety engineers and system developers should arrange an on-the-fly meeting to investigate the discovered unexpected changes *Activity 4*. The meeting should reveal 1) whether or not the suggested changes might introduce unreasonable risks (i.e., criticality level) and 2) the size of work required to cope with the suggested changes. The size of work is defined, in this context, based on its influence on the earlier gaming plan *Activity 7* so that big work means a modification of the release planning is required. If the suggested changes are non-critical, system developers should implement them or forward them to the relevant team. If the suggested changes are critical, one of two possible actions should be performed:

1. If the size of work is small, developers should do the fixes on-the-fly and cite the related test cases in the contracts and vice versa.

2. If the size of work is big and critical, developers should either follow the exchange strategy by XP to re-prioritise the tasks within the current iteration of the planned release or plan the tasks for the next release.

Activity 9: Run all related tests. In this activity, system developers should utilise the continuous integration as a first step, according to XP, to avoid delays caused by integration problems. Subsequently, a continuous testing process should be initiated to obtain immediate feedback on the possibility of violating safety countermeasures to prevent unreasonable risks associated with a software release. The scope of testing should be extended from a bottom-up assessment (from test cases to safety requirements) to validate safety goals. In case of any violation of safety requirements after running the continuous testing, system developers should follow *Activity 8*.

Activity 10: Generate new versions for the modified test reports. This activity should be followed once the continuous testing is completed successfully. New reports of the test results should be generated to replace the out-of-date reports in the safety case. It is significant to update the references of these reports in the safety contracts of the system and its safety case.

Discussion and Conclusion

Maintaining safety critical systems due to changes is a challenging process because of: 1) the lack of awareness of the change’s effects and the ripple of these effects on the system, 2) the lack of documentation of dependencies among the generated artefacts during the development process, and 3) the lack of traceability between a system and its safety case. Following the V-model to accommodate system changes might be very strict, which might be justifiable for structural system changes since many parts get impacted and there is no precise clue about the size of work needed to maintain the system. For software non-structural changes, this might not be justifiable. ASD can provide promising methods to maintain software changes. For example, XP puts great emphasis on the technical aspects (e.g., TDD, continuous integration and code refactoring). Also, Kanban brings the visibility of the workflow and improves the communication and collaboration among the stakeholders. Using ASD for maintaining safety critical systems can be promising but it still needs to comply with the current safety standards. In this paper, we introduced XP-Kan-Safe as a novel framework in which we tailor a hybrid process of ASD and the traditional V-model. The tailored process exploits safety contracts to connect ASD and the V-model, and enable a tri-directional impact analysis process. Future work will focus on creating a more in-depth case study to validate both the feasibility and efficacy of the process as well as to fully automate its application.

Acknowledgment

This work has been partially supported by the Swedish Foundation for Strategic Research (SSF) (through SYNOPSIS and FiC Projects) and the EU-ECSEL (through SafeCOP project). Both authors contributed equally to the paper and their names are listed in alphabetical order.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *CoRR, abs/1709.08439*.

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*. Washington, DC, USA.

Ahmad, M., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. In *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*.

Bate, I., Hawkins, R., & McDermid, J. (2003). A contract-based approach to designing safe systems. In *Proceedings of the 8th Australian workshop on safety critical systems and software - volume 33* (pp. 25–36). Darlinghurst, Australia. Australian Computer Society, Inc.

Benvenuti, L., Ferrari, A., Mazzi, E., & Vincentelli, A. L. (2008). Contract-based design for computation and verification of a closed-loop hybrid system. In *Proceedings of the 11th international workshop on hybrid systems: Computation and control* (pp. 58–71). Berlin, Heidelberg: Springer-Verlag.

Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring – a systematic literature review. *The Journal of Systems & Software*, *110*, 85–100.

Cao, L., Ramesh, B., & Abdel-Hamid, T. (2010). Modeling dynamics in agile software development. *ACM Trans. Manage. Inf. Syst.*, *1* (1), 5:1–5:26.

Denney, E., Pai, G., & Habli, I. (2015). Dynamic safety cases for through-life safety assurance. In *Proceedings of the 37th IEEE international conference on software engineering (ICSE).*

GSN Standard (2011). *Goal Structuring Notation working group*.

Heeager, L., & Rose, J. (2015, December). Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, *20*(6), 1762–1784.

ISO 26262:2011 (2011). Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization.

Knippers, D. (2011). Agile software development and maintainability. In *Proceedings of the 15th Twente Student conference.*

Jaradat, O., Bate, I. (2017). Using safety contracts to guide the maintenance of systems and safety cases. In *Proceedings of the 13rd European Dependable Computing Conference (EDCC).*

Jaradat, O., Bate, I. & Punnekkat, S. (2015a). Facilitating the maintenance of safety cases. In *Proceedings of the 3rd International Conference On Reliability, Safety and Hazard - Advances In Reliability, Maintenance and Safety (ICRESH-ARMS).* Luleå, Sweden.

Jaradat, O., Bate, I. & Punnekkat, S. (2015b). Using sensitivity analysis to facilitate the maintenance of safety cases. In *Proceedings of the 20th International Conference on Reliable Software Technologies (Ada-Europe)*.

Jaradat, O., Graydon, P. & Bate, I. (2014). An approach to maintaining safety case evidence after a system change. In *Proceedings of the 10th European Dependable Computing Conference (EDCC).* Newcastle, UK.

Bates, S., Bate, I., Hawkins, R., Kelly, T., McDermid, J., & Fletcher, R. (2003). Safety case architectures to complement a contract-based approach to designing safe systems. In *Proceedings of the 21st International System Safety Conference (ISSC).*

Salameh, A. (2011). *On Process Tailoring - An Agile Example.* Master Thesis. Chalmers University.

Kelly, T., & McDermid, J. (1999). A systematic approach to safety case maintenance. In *Proceedings of the Computer Safety, Reliability and Security (SAFECOMP)* (Vol. 1698, p. 13-26). Springer Berlin Heidelberg.

Tarwani, S., & Chug, A. (2016). Agile Methodologies in Software Maintenance: A Systematic Review. *Informatica*, 40(4), 415.

00-56 Standard (2015). Defence Standard — Issue 6. Safety Management Requirements for Defence Systems — Part 1: Requirements and Guidance. U.K. Ministry of Defence.

Biographies

Omar T. Jaradat, Ph.D. candiate, School of Innovation, Design and Engineering, Mälardalen University, Högskoleplan 1, SE-72123, Västerås, Sweden, Tel: +46 21101369, Fax: +46 21101460 e-mail – omar.jaradat@mdh.se.

Omar Jaradat is a Ph.D. candidate in the Innovation, Design and Engineering department at Mälardalen University. His research interests include safety argumentation for safety critical systems, where the main focus is on maintenance of safety-critical systems and safety cases.

Abdallah M. Salameh, Ph.D. candidate, School of Computing, Science & Engineering, University of Salford, Manchester, UK, Tel: +46 721844015, e-mail – a.salameh@edu.salford.ac.uk.

Abdallah Salameh is a senior developer at Bambora Group AB - Sweden and a Ph.D. candidate in the School of Computing, Science and Engineering at the University of Salford, U.K. His research interests include agile software development, where the main focus is on tailoring the processes in large-scale software intensive organisations.