A Methodological Approach for Rigorous Assessment of Software Architectures within ISO26262

- TECHNICAL REPORT RCL130401 -
(UNIVERSITY OF FIRENZE, DIP. SISTEMI E INFORMATICA)
(VERSION 1.0 – MARCH, 2012)

Valentina Bonfiglio
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
viale Morgagni 65,
50134, Firenze, Italy
email: valentina.bonfiglio@unifi.it

Leonardo Montecchi
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
viale Morgagni 65,
50134, Firenze, Italy
email: lmontecchi@unifi.it

Francesco Rossi
ResilTech s.r.l.
Piazza Iotti 25, I-56025, Pontedera, Italy
francesco.rossi@resiltech.com

Andrea Bondavalli
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
viale Morgagni 65,
50134, Firenze, Italy
email: bondavalli@unifi.it

ABSTRACT

Safety analysis is becoming more and more important in a wide class of systems, and especially when normative standards are present. In the automotive field, the recent ISO26262 foresees safety analysis to be performed at different levels: system, software and hardware. The assessment of architecture with respect to safety is typically better understood at system and HW levels, while an equivalent analysis at SW level has not such an established background. In literature, approaches exist to handle specific activities related to the safety assessment of software, but they are typically not so well integrated within a more general assessment and certification process. Recent safety standards put more and more emphasis on software-level safety analysis, therefore calling for a precise methodology for the assessment of software architectures. While ISO26262 requirements prescribe safety analysis of the software architecture, clear guidelines on how it should be performed are not provided, thus leaving an important gap for its industrial adoption. The aim of this paper is to fill this gap in current industrial practices, by defining a precise workflow composed of well defined and repeatable activities. While we are not providing new theoretical research results, we believe that the work in this paper provides an useful contribution to a timely problem of great industrial relevance in the automotive domain. Moreover, part of the workflow can be automated, thus improving the efficiency of the assessment process.

1 Introduction

Safety analysis supports the production of convincing evidence that the operation of the system is safe, i.e., even in presence of failures, catastrophic consequences on the user(s) and the environment are avoided [1]. To this purpose, the system architecture is typically analyzed using systematic techniques like Failure Modes and Effects Analysis (FMEA) [2] and Fault Tree Analysis (FTA) [3] to identify possible violations of safety requirements. Within several domains (e.g., automotive, railway) the analysis of functional safety is mandatory and it is regulated by specific standards.

The importance of rigorous methodologies to perform safety analysis is increasing, since the complexity of modern safety-critical systems and their dependence on electronic components are growing. A prominent example is the automotive industry, where traditional mechanical components have been replaced by electronic sensors and actuators, and new electronic functionalities to assist the driver are being implemented

(e.g. ESP, cruise control). As a consequence, software is becoming more and more important in the design of safety-critical systems, as more and more safety requirements are assigned to it. Indeed, recent safety standards are putting more and more emphasis on software-level safety analysis. The recent standard ISO26262 [4] for the functional safety of road vehicles foresees safety analysis to be performed at different levels: system, hardware, and software. It is worth to specify that within the standard, the term "safety analysis" identifies a precise activity: the study of faults, their correspondent effects and the possible mitigations to be introduced.

While in performing safety analysis it is common practice to consider both hardware and software, the assessment of architecture with respect to safety is typically better understood at system and hardware levels, while an equivalent analysis at software level has not such an established background. Safety analysis of software introduces significant challenges with respect to the hardware counterpart: failure modes and related statistics aren't typically available as datasheets. Moreover, even small changes to the software architecture or to its components can produce significant effects on the propagation or mitigation of failures.

While ISO26262 requirements prescribe safety analysis of the software architecture to be performed, clear guidelines on how such analysis should be performed are not provided, thus leaving an important gap for its industrial adoption. In literature, approaches exist to handle specific activities related to the safety assessment of software architectures, but they are typically not contextualized within a more general assessment and certification process. The aim of this paper is not to present new theories and techniques for the safety analysis of software, but rather to clarify how such analysis should be performed in order to fulfill the requirements of ISO26262. The methodology defines a workflow composed of well defined and repeatable activities, which are therefore suitable to be supported by automatic facilities. While we are not providing new theoretical research results, we believe that the work in this paper provides on useful contribution to a timely problem of great industrial relevance in the automotive domain.

The paper is organized as follows. The relevant state of the art is discussed in Chapter 2, focusing on safety analysis of software architectures, as well as previous publications on the ISO26262 standard. Chapter 3 briefly introduces the relevant requirements of the ISO26262 standard and then describes our workflow for the safety analysis at software level. Finally, concluding remarks are reported in Chapter 4.

2 State of the Art

In literature, the topic of safety analysis of software architectures has been addressed in different ways. Most work focuses on methods and tools to support the application of FMEA at software level (SW-FMEA). A well-known approach to the safety analysis of software architectures is based on failure propagation and transformation annotations. The design specification of the software architecture is annotated with information about the failure behavior of the architectural components. Different notations supporting such approach exist: Failure Propagation and Transformation Notation (FPTN) [12], Component Fault Trees (CFTs) [13], and Fault Propagation and Transformation Calculus (FPTC) [14] are some notable examples. Using such an approach, the failure behavior of the entire system can be automatically calculated starting from the failure behavior of its components and the design of the software architecture; supporting tools also exist and are freely available (e.g., see [8]). Additional details and comparison of such approaches can be found in [11].

Other approaches focus on detailed software FMEA, i.e., they take into account a code-level representation of software components and perform a qualitative analysis of software, based on tracing the dependencies across variables through a body of source code. Although this technique can be used to automate the analysis of large code trees [10], it has the main drawback of needing an implementation of the software component under analysis.

The work in [7] presents an ontology-based approach for efficient manipulation of data involved in the SW-FMEA process, formalizing the concepts involved in the analysis, thus giving a precise semantics to concepts collected in the artifacts of an industrial documentation process. Other works focus on the formalization of safety requirements, e.g., the work in [5], which addresses the formalization of requirements in a model-based design setting, focusing on asymmetric failures.

One of the most comprehensive methods for safety analysis of system and software architectures is the Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) methodology [9]. HiP-HOPS modifies and integrates classical safety analysis techniques, guiding the analysis from the functional level through low levels of its hardware and software implementation, and provides support for the automation of certain tasks (e.g., the construction of fault-trees).

In this paper we focus on a set of requirements dictated by the ISO26262 standard to define a systematic methodology that is able fulfill them during the assessment process. Our objective is not to introduce novel analysis methods, but rather to precisely define a set of activities, together with their inputs and outputs, and organize them in a structured workflow. In this perspective, our proposal is complementary to other works mentioned above, which can be used to carry out specific activities within the workflow.

A number of recent publications have targeted the ISO26262 standard, including introductions to the standard itself [16], experience reports [17], support tools [18]. Other publications focus on specific aspects of system development and assessment according to ISO26262. The work in [6] introduces a set of best practices for model review of software models with the aim of ensuring safety-related objectives and adherence to ISO26262, using a combination of automated and manual reviews. As mentioned above, the work in [5] addresses the formalization of requirements, targeting the EAST-ADL language within the ISO26262 context. A more comprehensive survey on recent publications related to ISO26262 can be found in [15].

Despite the relatively large number of publications on the new automotive standard, a proven workflow to properly support the safety evaluation of software architectures according to ISO26262 requirements is still missing from industrial practice. In this paper we provide our contribution in filling this gap.

3 Assessment of Software Architectures according to ISO26262

3.1 Main Involved ISO26262 Requirements and Motivation

The system lifecycle described in the ISO26262 standard foresees safety analysis to be performed at different levels: system, SW and HW. The importance of carrying out a safety analysis at the SW level is highlighted in Part 6 of the standard, “Product development at the software level”. In particular, within the SW lifecycle (Fig. 1), the activity of software safety analysis can be contextualized within the “Software architectural design” phase, described in Clause 7 of Part 6.

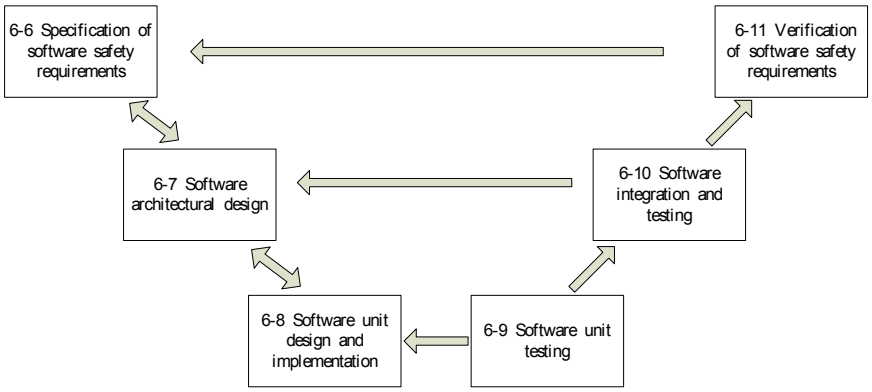

Fig. 1. Architecture design phase within the SW lifecycle.

According to requirement 7.4.13 (Part 6), safety analysis of the software architecture has the aim to identify or confirm the software components on which are instantiated requirements or that otherwise have an impact on them, and to support the specification and verify the efficiency of safety mechanisms. In this regard the standard suggests some mechanisms for error detection and error handling that should be defined at the software architectural level; the appropriate mechanisms should be selected based on results of safety analysis. Furthermore, requirement 7.4.16 (Part 6) requires that if new hazards are introduced by the software architectural design, which are not already addressed by an existing safety requirement, they shall be considered and evaluated as part of maintaining up-to-date the results of the hazard analysis and risk assessment activity.

Part 6 explicitly states that safety analysis should be performed at the software architectural level (requirement 7.4.13), and references Part 9 for further guidance on this topic. Part 9 of the standard, "Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses", however, defines the requirements and recommendations at a generic level and does not provide specific recommendations regarding software.

As an example, requirement 8.4.9 requires safety analysis to include a systematic identification of faults that could lead to the violation of safety requirements, the evaluation of the consequences of each identified fault, and the identification of potential safety-related problems, including the ineffectiveness of safety mechanisms under specific circumstances (e.g., cascading failures). As it is evident this is too generic to really help the safety engineer in evaluating and upgrading the SW architecture to properly address safety requirements in presence of failures.

Summarizing, while the standard explicitly requires that safety analysis is performed on the software architecture, no details on how it can be performed from a practical point of view are provided, favoring ambiguity on activities to be performed from a practical point of view. From an industrial perspective, a well-defined and repeatable methodology is paramount, since it allows reducing efforts and costs in the assessment and certification process. The approach presented in this article is aimed precisely to meet this industrial need.

3.2 The Overall Workflow

In this section we introduce our methodology for the assessment of software architectures according to requirements of the ISO26262 standard. The methodology clearly identifies a flow of activities which are summarized in Fig. 2. Each activity requires precise input and output information; a more detailed view of the workflow is shown in Fig. 3

Mandatory inputs in order to be able to perform the analysis are, of course, the *software safety requirements*, and the *software architecture information*; such information is the input of the overall workflow. In our methodology, the analysis is performed by analyzing some kind of model of the software, obtained based on requirements and architecture information.

First of all, it is necessary to identify the properties of the software architecture that should be taken into account during the safety analysis in order to be able to successfully demonstrate that the violation of the SW safety requirements is prevented. Such properties are closely related to the requirement assigned to the

software architecture: as a simple example, if requirements related to execution delays exist, then the model of the software should take into account execution delays. Accordingly, the first activity in the workflow, **Safety Analysis Modeling**, receives as input the safety requirements of software, and produces as output a set of properties that should be represented in the model (SW model properties).
Based on software safety requirements, and properties to be represented in the model, this activity also defines the fault models to be considered for software components. Sources of potential failures are systematic software faults; in general, resulting failures can be categorized as service provision failures, value failures, or timing failures (e.g., see [9]).

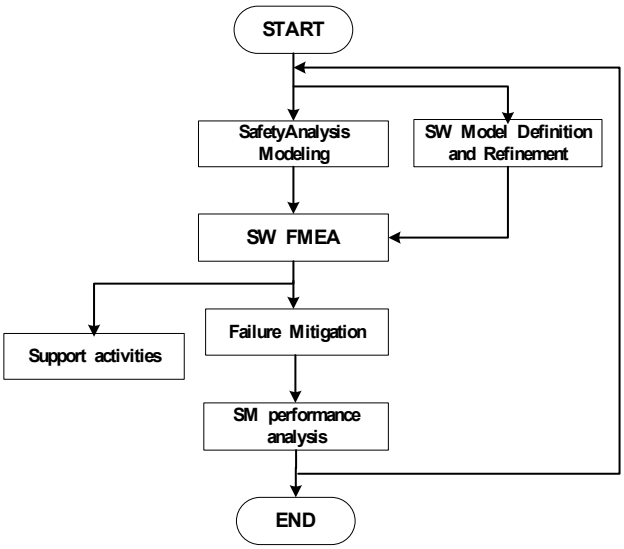

Fig. 2. Our workflow for safety analysis of software architectures according to ISO26262.

The Safety Analysis Modeling activity covers requirement 8.4.6 in Part 9, which expresses the need to define fault models consistent with the appropriate design phase, and part of requirement 8.4.9, which dictates the use of a systematic identification of faults and requires the evaluation of the consequences of each identified fault to determine its potential to violate safety requirements. It should be noted however that identifying and defining the correct fault models and model properties to be considered for each specific application is a challenging task, and, most importantly, not all faults could be reproducible, or worse yet, not all faults could be observable.

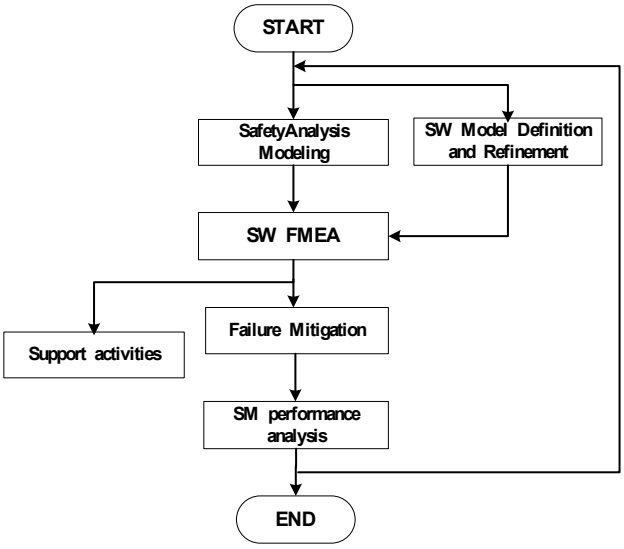

Fig. 3. Our workflow for safety analysis of software architectures according to ISO26262.

To reduce the effort and improve reuse, our methodology foresees two databases, *Fault Models DB* and *Model Properties DB*, which can be used as repositories of commonly needed fault models and model properties. The use of a repository of fault models is also suggested by section 8.3.2 (Part 9), which explicitly mentions the possibility to consider fault models from external sources as input to the safety analysis.

Requirement 8.4.9 in Part 9 requires that both the software component itself and the interaction between different software components are considered. Our methodology for safety analysis is designed to support both the case in which some kind of software model is provided by the software manufacturer, and the case in which such model is not available. In both cases, the model may need to be created, or enriched to include the relevant information; therefore an additional activity of **SW Model Definition and Refinement** is needed. The workflow we are proposing does not require the use of a specific model for the software architecture. Actually, different approaches can be used, based also on what is provided by the customer: if some kind of model exists, you may need to integrate (refine) the model with additional information based on the model properties previously identified; in the second case the entire model has to be built, taking into account of such properties.

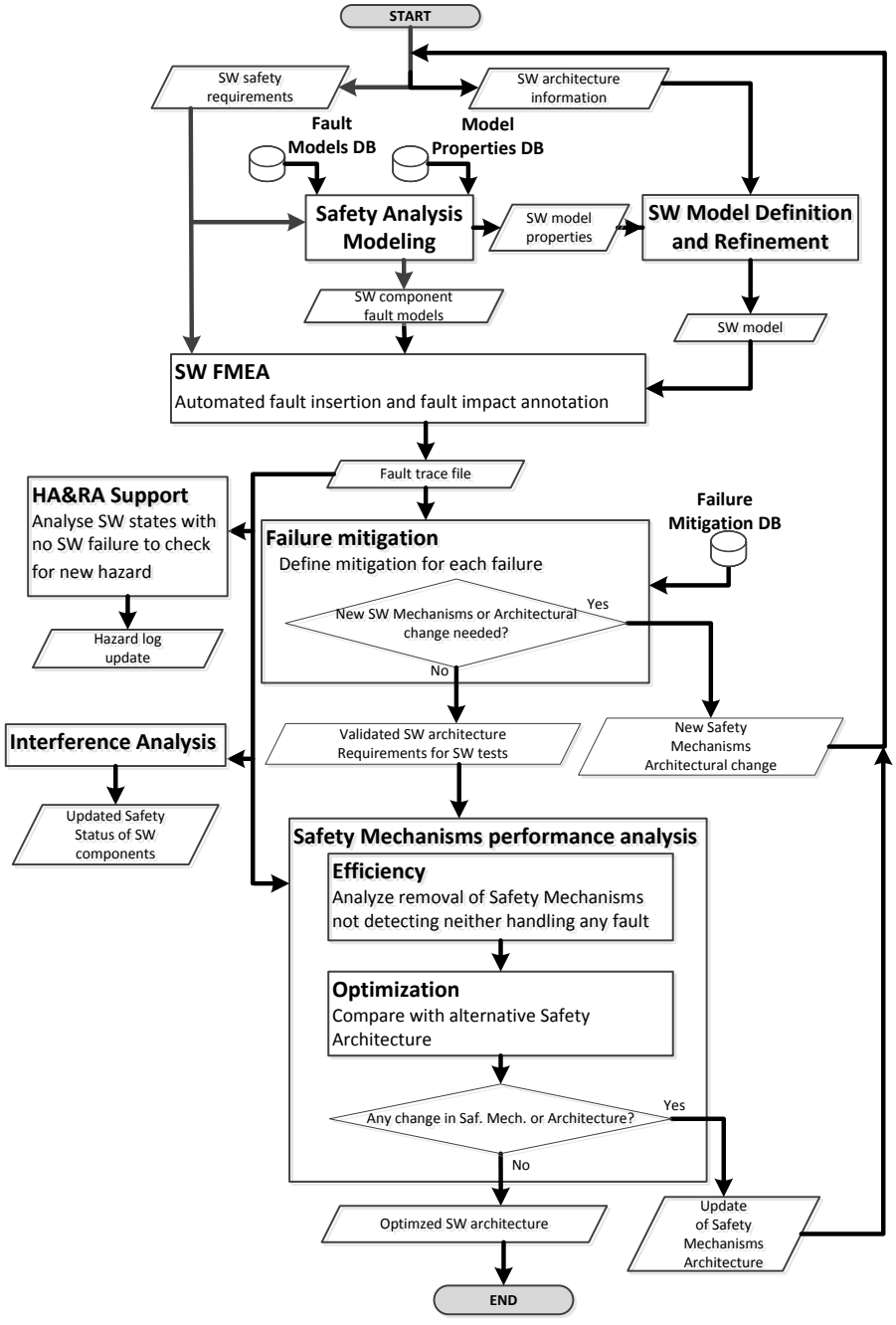

Fig. 4. Our detailed workflow for safety analysis of software architectures according to ISO26262 requirements.

A flexible approach that is typically applied in the safety analysis of software architecture is that employed in Failure Propagation and Transformation Calculus (FPTC) [8,14] and other similar approaches [11]. Such approach does not require knowing the details of the various components that make up the software architecture: each software component is annotated with a table that describes the failure behavior of the component, i.e., how failures occurring at the input of the component are propagated at the output of the component. Failures can be propagated without any transformation (e.g., a "late" failure in input generates a "late" failure at the output), or they can be transformed into other kind of failures. The mitigation of a failure is a particular case of failure transformation. Based on the failure propagation table obtained for each SW component, it is possible to analyze the effect on the entire system of any possible fault, i.e., what is the effect on the output of the system, and verify if a violation of requirements is present or not.
Once the requirements, the SW model, and the fault models to apply at the software components are known, it is possible to proceed to the next activity, which consists in performing an architectural-level **SW FMEA**, in which software elements are considered as black boxes that contain unknown code, but which implement the requirements assigned to them. The objective of SW FMEA is to evaluate the impact of software faults, estimating the ability of the software to provide protection from the effects of software failures. The results of the SW FMEA describe the possible effects of software faults on the system, and indicate if safety goals or safety requirements assigned to software are complied with, as required by requirements 8.4.2 and 8.4.9 in Part 9. The kinds of faults to be considered at this stage are obtained from the fault models produced as output by the *Safety Analysis Modeling* phase.
The output of this activity is a *Fault trace file*, which stores information about faults that trigger a Safety Mechanisms, as well as faults leading to a SW failure (i.e., a violation of a SW safety requirement). Based on the content of the *Fault trace file*, we can distinguish two cases.
If, according to the *Fault trace file*, no faults are able to produce a violation of software safety requirements, then the introduced safety mechanisms are able to mitigate the possible SW failures, and no additional failure mitigation mechanisms are needed. In this case the flow continues directly to the *Safety Mechanisms performance analysis* activity.
On the contrary, if SW failures are still present, a **Failure Mitigation** activity is required in order to derive, based on results of the *SW FMEA* activity, measures that are able to prevent, mitigate, or reduce the effect of the potential safety requirement violation. This activity is necessary to satisfy several requirements present in Part 6 and Part 9 of the ISO26262 standard. On one hand, the specification of safety mechanisms is one of the main objectives of safety analysis, according to requirements 7.4.13, 7.4.14 and 7.4.15 in Part 9. On the other hand, requirement 8.4.3 in Part 9 explicitly states that, if a safety goal or safety requirement is not satisfied, the result of the analysis should be used to derive prevention, detection, or mitigation measures. Additionally, based on the results of the analysis, it may be necessary to determine additional safety-related test cases (requirement 8.4.7, Part 9) in order to provide evidence of correct behavior.
Our methodology foresees one database, *Failure Mitigation DB*, which can be used as repositories of commonly used mechanisms for failure mitigation. For each failure to be mitigated, one of the following mitigation strategies should be chosen:

1. change/improve specification of existing Safety Mechanisms;
2. add new Safety Mechanisms for error detection and/or error handling, this could be also in the form of ad-hoc SW tests to be run on-line;
3. propose changes in the architecture in order to avoid SW failure by removing specific fault propagations;
4. specify requirements for SW tests for subsequent SW verification activities.

If changes are made to the architecture or safety mechanism are added (or changed if they already exist), the entire workflow must be repeated from the beginning: the architecture and/or the model have been modified, so it is necessary to take this into account in all the subsequent activities: defining the necessary properties for the SW model, definition/ refinement of the SW executable model, identification of fault models.
Subsequently, a support activity, **Safety Mechanism performance analysis**, is started. Such activity allows us to provide evidence of the performance of safety mechanisms that are currently defined and if necessary, propose to evaluate alternative solutions as expected from requirement 7.4.13 in Part 6 and requirement 8.4.9 in Part 9. This activity is composed of two sub-activities that evaluate, respectively, the **Efficiency** and **Optimization** of the safety mechanisms.
The first activity, *Efficiency*, consists of analyzing the possibility to remove a safety mechanism that doesn't contribute to the detection or mitigation of any fault. The decision of eliminating a safety mechanism is made considering various parameters, including cost and impact of the modification. It is also necessary to take

into account the fact that removing a safety mechanism that in the considered analysis doesn't provide any contribution could, however, have harmful consequences in other situations. As already said, not all faults could be observable and reproducible, so this operation of removing a safety mechanism have to be carefully considered. In the worst case, it could be possible that a safety mechanism that seems doesn't have any healthy contribute, could have a role in mitigating a failure that we have not been able to identify.
The second activity, *Optimization*, is related to the possibility to consider cost-effective changes to the safety architecture. An optimized, i.e. less expensive, safety architecture is proposed, based on the analysis of fault trace file and the SW safety requirements.
Any change introduced in these two activities, either related to a safety mechanism which is removed, or to a change in the software architecture, triggers a re-evaluation of the whole software architecture. In both cases, in fact, it is required to restart the workflow from the beginning, in order to update the requirements and/or the architecture and therefore the model SW resulting from them.
A further support activity can be identified in the flow diagram with the name of **HA&RA Support** in order to satisfy two requirements, 7.4.16 in Part 6 and 8.4.5 in Part 9: they cover the possibility to identify new hazards, not covered by an existing safety requirements, that should be introduced and evaluated in the hazard analysis and risk assessment. This can be carried out in cooperation with the responsible for the hazard analysis, and allows understanding whether the injected faults that do not violate any requirement could lead to: i) a known hazard, or ii) the identification of a new hazard. In these cases is probable that some SW safety requirement is missing and needs to be added and annotated. The output of this activity is therefore a feedback to other phases of the safety lifecycle, and it is not shown in the figure.
Finally, the **Interference Analysis** activity consists in verifying that failures of lower integrity modules do not have impact on higher integrity modules, i.e., they do not interfere with them. If there is interference, it is necessary to increase the integrity of the level that generates it. This activity is related to the task of identifying (or confirming) the safety-related parts of the software, as specified by requirement 7.4.13 in Part 6.
In conclusion, the considered workflow for safety analysis of software architectures allows us to obtain:

1. *New SW requirements and/or architecture change for failure mitigation;*
2. *Evidence of failure mitigation;*
3. *Evidence of faults effect to support HA&RA in the identification of new functional or non-functional hazards.*

As described throughout this section, each activity of our workflow has been tailored accordingly to a set of requirements of the ISO26262 standard. Table 1 summarizes the activities of our workflow, and the requirements to which each of them is related.

Table 1. Mapping between activities in our workflow and requirements related to the safety assessment of the software architecture in ISO26262.

Activities	**Requirements**
All	ISO26262-9 Req. 8.4.1 ISO26262-6 Req. 7.4.13
Safety Analysis Modeling SW Model Definition and Refinement	ISO26262-9 Req. 8.4.6 ISO26262-9 Req. 8.4.9
SW FMEA	ISO26262-9 Req. 8.4.2 ISO26262-9 Req. 8.4.9
Failure Mitigation	ISO26262-6 Req. 7.4.13 ISO26262-6 Req. 7.4.14 ISO26262-6 Req. 7.4.15 ISO26262-9 Req. 8.4.3 ISO26262-9 Req. 8.4.7 ISO26262-9 Req. 8.4.9
Safety Mechanisms performance analysis	ISO26262-6 Req. 7.4.13 ISO26262-9 Req. 8.4.9
HA&RA Support	ISO26262-6 Req. 7.4.16 ISO26262-9 Req. 8.4.5
Interference Analysis	ISO26262-6 Req. 7.4.13

It should be noted that part of the workflow can be supported by automated facilities. In particular, techniques to perform (semi-)automated SW FMEA analysis exist (e.g., see [8]) allowing an automated injection of fault in the software model, and automated annotation of their impact. Indeed, the injection of faults inside the SW architecture model could be automated by allowing an exhaustive application of the fault models identified in the *Safety Analysis Modeling* phase, and the logging of their fault impact across the SW components. Moreover, using repositories of fault models, properties, and mitigation means, we could (semi-)automatically select parts already used for similar software components or system or situation.

4 Concluding Remarks

The importance of safety analysis of software architectures is continuously growing, since more and more functionalities of safety-critical systems are being implemented by electronic devices. In particular, the recent ISO26262 standard comprises several requirements on the safety analysis of software; however it does not provide clear guidelines on how such requirements should be fulfilled. Defining a precise workflow for the assessment of software architectures is therefore of great industrial relevance.
In this paper we have developed a systematic methodology to perform the safety analysis of software architectures according to ISO26262 requirements. We have precisely defined a set of activities, together with their inputs and outputs, and organize them in a structured workflow. The workflow supports the entire software safety analysis, including the identification of software failures and their effects, the production of evidence of failure mitigation, and the prescription of architecture and/or requirements modifications in order to satisfy software safety requirements.
Although we did not introduced new theoretical research results, we believe that the work in this paper provides a useful contribution to a timely problem of great industrial relevance in the automotive domain. This approach can find concrete application in the automotive industry and help in meeting the recommendations of the ISO26262 concerning safety analysis of software.
Part of the activities in the workflow is suitable to be automated or supported by automatic facilities. In the future we plan to further develop this methodology, improving its degree of automation (e.g., semi-automated SW-FMEA), and integrating it into a supporting tool to support the safety analysis in the automotive domain.

Acknowledgment

This work has been partially supported by the RACME-MAAS project [19], funded by the Tuscany Region within the framework POR CReO FESR.

References

1. Avižienis, A., Laprie, J.-C., Randel, B, Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing, 1, 11-33 (2004).
2. Palady P.: Failure modes and effects analysis. PT Publications, West Palm Beach, FL, (ISBN: 0-94545-617-4) (1995).
3. Vesely W.E.: Fault tree handbook, U.S. Nuclear Regulatory Committee Report NUREG-0492, US NRC, Washington DC, United States, p. X.15–8, (1981).
4. ISO: International Standard 26262 Road vehicles -- Functional safety (2011).
5. Bergenhem, C., Johansson, R., Lönn, H: A Novel Modelling Pattern for Establishing Failure Models and Assisting Architectural Exploration in an Automotive Context. In: Computer Safety, Reliability, and Security, LNCS, vol. 7612, pp 247-257 (2012).
6. Stürmer, I., Salecker, E., Pohlheim, H.: Reviewing Software Models in Compliance with ISO 26262. In: Computer Safety, Reliability, and Security, LNCS, vol. 7612, pp 258-267, (2012).
7. Bicchierai, I., Bucci, G., Nocentini, C., Vicario, E.: An Ontological Approach to Systematization of SW-FMEA. In: Computer Safety, Reliability, and Security, LNCS, vol. 7612, pp 173-184 (2012).
8. Paige, R.F., Rose, L.M., Ge, X., Kolovos, D.S., Brooke, P.J.: FPTC: Automated Safety Analysis for Domain-Specific Languages. In Models in Software Engineering, LNCS, vol. 5421, pp 229-242 (2009).
9. Papadopoulos, Y., McDermid, J., Sasse, R., Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. Reliability Engineering & System Safety, vol. 71, Issue 3, Pages 229-247 (2001).
10. Snooke, N., Price, C.: Model-driven automated software FMEA. In: Proceedings of Reliability and Maintainability Symposium (RAMS) 2011, pp.1-6, 24-27 Jan. (2011).

11. Grunske, L.; Han, J.: "A Comparative Study into Architecture-Based Safety Evaluation Methodologies Using AADL's Error Annex and Failure Propagation Models". In 11th IEEE High Assurance Systems Engineering Symposium (HASE 2008). pp. 283-292, 3-5 Dec. (2008).
12. Fenelon, P., McDermid, J., Nicholson M., Pumfrey D.J.: Towards integrated safety analysis and design. ACM Computing Reviews, 2(1):21–32 (1994).
13. Kaiser, K., Liggesmeyer, P., Mackel, O.: A new component concept for fault trees. In Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03), pages 37–46, Adelaide (2003).
14. Wallace, M.. "Modular architectural representation and analysis of fault propagation and transformation. Electr. Notes Theor. Comput. Sci., 141(3):53–71 (2005).
15. Izosimov, V., Ingelsson, U., Wallin, A.: Requirement decomposition and testability in development of safety-critical automotive components. In: Computer Safety, Reliability, and Security. LNCS vol. 7612, pp 74-86 (2012).
16. Dittel, T., Aryus, H.-J.: How to "survive" a safety case according to ISO 26262. In: Computer Safety, Reliability, and Security, LNCS vol. 6351, Springer, pp. 97-111 (2010).
17. Schubotz, H.: Experience with ISO WD 26262 in Automotive Safety Projects, SAE Tech Paper (2008).
18. Makartetskiy, D, Pozza, D., Sisto, R.: An Overview of software-based support tools for ISO26262. In: Intl. Workshop Innovation Inf. Tech. – Theory and Practice (2010).
19. RACME-MAAS: Progettazione di plug-ins RACME per la Modellizzazione ed Analisi di Architetture di Sistemi, Bando Regione Toscana POR CREO FESR, Linea d'intervento 1.3b (2013)