Safety Case Impact Assessment in Automotive Software Systems: An Improved Model-Based Approach

Sahar Kokaly[1(✉)], Rick Salay[2], Marsha Chechik[2], Mark Lawford[1], and Tom Maibaum[1]

[1] McMaster Centre for Software Certification, McMaster University, Hamilton, Canada
{kokalys,lawford,maibaum}@mcmaster.ca
[2] Department of Computer Science, University of Toronto, Toronto, Canada
{rsalay,chechik}@cs.toronto.edu

Abstract. Like most systems, automotive software systems evolve due to many reasons including adding, removing or modifying features, fixing bugs, or improving system quality. In this context, safety cases, used to demonstrate that a system satisfies predefined safety requirements, often dictated by a standard such as ISO 26262, need to co-evolve. A necessary step is performing an impact assessment to identify how changes in the system affect the safety case. In previous work, we introduced a generic model-based impact assessment approach, that, while sound, was not particularly precise. In this work, we show how exploiting knowledge about system changes, the particular safety case language, and the standard can increase the precision of the impact assessment, reducing any unnecessary revision work required by a safety engineer. We present six precision improvement techniques illustrated on a GSN safety case used with ISO 26262.

1 Introduction

Safety engineers in various domains, including automotive, experience difficulties with safety case maintenance. As stated in [11], the main reason for this is that they do not have a systematic approach by which to examine the impact of change on a safety argument. The authors of [2] performed a study which suggested that engineers spend 50–100 h on Change Impact Assessment (CIA) per year on average. The second most commonly mentioned CIA challenge is related to information overload. The three most senior engineers in the study reported that obtaining a system understanding is hard due to the complexity of the systems. The sheer number of software artifacts involved makes traceability information highly complex. Based on the results of [2], determining how a change impacts the product source code seems to be less of a challenge than determining impact on non-code artifacts, e.g., requirements, specifications, and test cases. In [14,17], the authors further discuss the problem of CIA being a

S. Tonetta et al. (Eds.): SAFECOMP 2017, LNCS 10488, pp. 69–85, 2017.
DOI: 10.1007/978-3-319-66266-4_5

challenge in safety-critical systems. Specifically, Leveson [14] mentions that inadequate CIA has been among the causes of accidents in the past. Thus, the current state of practice can clearly benefit from improved CIA techniques, especially to help perform safety assurance more cost-effectively.

In this paper, we build on our earlier work [13] which proposed using a model-based approach to perform impact assessment on an assurance case due to system changes. Our technique is applicable to assurance cases in general and ensures soundness, i.e., it does not miss any elements that are impacted. Yet, the approach is conservative. i.e., it can flag elements as impacted when they are not, resulting in "false positives". Using knowledge about the system models, the safety case language and the standard under consideration, the precision of our approach can be improved, thus reducing unnecessary effort by the safety engineer. The contributions of this paper are as follows: (1) we provide a model-based approach for impact assessment on GSN safety cases used with ISO 26262, and (2) we identify and describe six techniques for improving the precision of the impact assessment approach.

The rest of the paper is organized as follows: Sect. 2 introduces the power sliding door system used as a running example in the paper. Section 3 presents background material on ISO 26262. Section 4 describes how our model-based approach can be used for GSN safety cases linked to ISO 26262. Section 5 presents the techniques that can be used to improve the precision of our model-based impact assessment approach. Section 6 discusses related work, and Sect. 7 summarizes the paper and outlines problems for future work.

2 Running Example: Power Sliding Door (PSD) System

Consider an automotive subsystem that controls the behavior of a power sliding door in a car. The system has an `Actuator` that is triggered on demand by a `Driver Switch`. This example is presented in Part 10 of ISO 26262 [8]. Figure 1 shows the system models comprised of a Class Diagram (to model structure), a Sequence Diagram (to model behavior) and a relationship between them. This can be visualized at a high-level as the *megamodel* [16] in Fig. 4a, which includes other parts of the system such as results of model checking and testing. In practice, the system megamodel could include other system models, e.g., SysML representations, FMEA and FTA results.

The `Driver Switch` input is read by a dedicated Electronic Control Unit (ECU), referred to as `AC ECU` which powers the `Actuator` through a dedicated power line. The vehicle equipped with the item is also fitted with a control unit able to provide the vehicle speed, referred to as `VS ECU`. The system includes a safety element, namely, a `Redundant Switch`. Including this element ensures a higher level of integrity for the overall system.

The `VS ECU` provides the `AC ECU` with the vehicle speed. The `AC ECU` monitors the driver's requests, tests if the vehicle speed is less than or equal to 15 km/h, and if so, commands the `Actuator`. Thus, the sliding door can only be opened or closed if the vehicle speed is not higher than 15 km/h. The `Redundant Switch` is

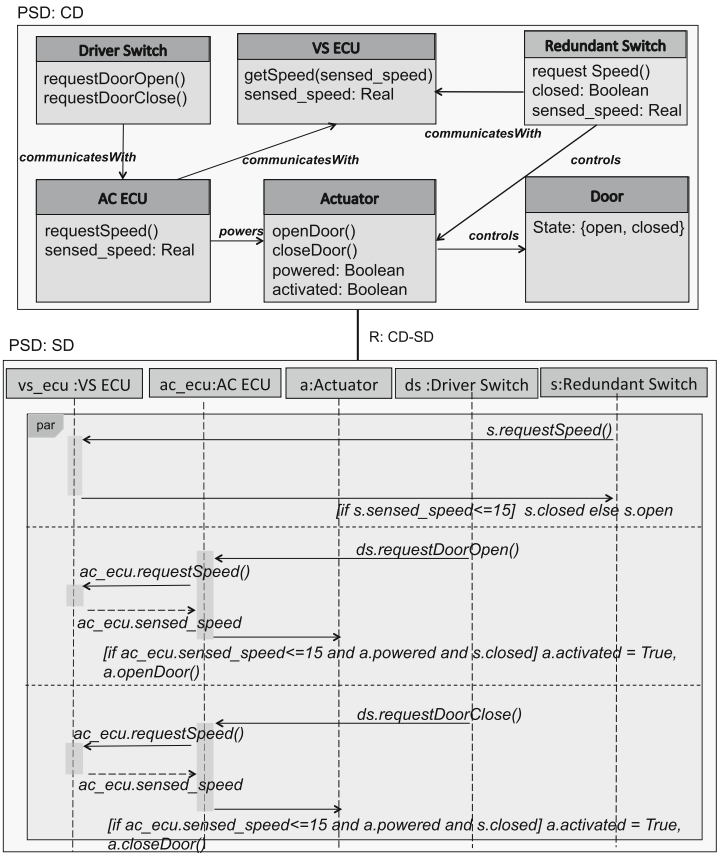

Fig. 1. Power sliding door class diagram-sequence diagram.

located on the power line between the `AC ECU` and the `Actuator` as a secondary safety control. It switches on if the speed is less than or equal to 15 km/h, and off whenever the speed is greater than 15 km/h. It does this regardless of the state of the power line (its power supply is independent). The `Actuator` operates only when it is powered.

Suppose that the PSD system changes such that the redundant switch is removed. In the new system, only the `AC ECU` checks the vehicle speed before commanding `Actuator`. Given a safety case for the original system (refer to Fig. 7 ignoring the annotations[1]), it is desirable to reuse as much as possible of its content in structuring a new safety case. An important prerequisite for this

[1] Note that the ASIL assignments are given in the example in Part 10 of ISO 262626, and we selected assignments for requirements B2 and B4 based on the possible ASIL C decompositions for redundancy as shown in Fig. 2 – ASIL decomposition schemes, in Part 9 of the standard.

is performing an assessment to identify the impact of the changes made in the system on the safety case components.

3 Background: ISO 26262

ISO 26262 is a standard that regulates functional safety of road vehicles. It recommends conducting a Hazard Analysis to identify and categorize hazardous events in the system and to specify safety goals and integrity levels related to the mitigation of the associated hazards. The standard has 10 parts, and we focus on one of them, "Product Development at the Software Level" (Part 6), and refer to Part 9 which explains Automotive Safety Integrity Levels (ASILs).

ASIL Allocation and Propagation. An ASIL refers to an abstract classification of inherent safety risk in an automotive system or elements of such a system. ASIL classifications are used within ISO 26262 to express the level of risk reduction required to prevent a specific hazard, with ASIL D representing the highest and ASIL A the lowest. If an element is assigned QM (Quality Management), it does not require safety management. The ASIL assessed for a given hazard is then assigned to the safety goal set to address that hazard and is then inherited by the safety requirements derived from that goal following ASIL propagation rules. The higher the ASIL, the more rigorous the application of ISO 26262 has to be. i.e., the more requirements need to be fulfilled.

ASIL Decomposition. The method of ASIL tailoring during the design process is called "ASIL decomposition". When allocating ASILs, benefit can be obtained from architectural decisions, including the existence of sufficiently independent architectural elements (as in the redundancy in the original PSD system). This offers the opportunity to implement safety requirements redundantly by these independent architectural elements, and to assign a potentially lower ASIL to these decomposed safety requirements[2].

Furthermore, ISO 26262 requires the production of over 100 work products, achieved via various requirements and methods used in the different phases of software development. For example, Sect. 9 of Part 6 of ISO 26262 discusses Software Unit Testing, and Sect. 9.5 outlines the required work products for it. One

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

Fig. 2. Methods for software unit testing - ISO 26262 Part 6 (cropped for space).

[2] Refer to Fig. 2 in Part 9 of the standard for ASIL decomposition schemes.

of these work products is 9.5.1: Software Verification Plan which results from requirements 9.4.2–9.4.6 in the same section. Consider one of these requirements, 9.4.3, which describes which software testing methods can be used. These methods clearly link to ASILs. Specifically, Fig. 2, lists various methods for software unit testing and how they relate to the four ASILs. The degree of recommendation to use the corresponding method depends on the ASIL and is categorized as follows: "++" indicates that the method is highly recommended for the identified ASIL (we interpret this as "required"), "+" – that the method is recommended for the identified ASIL, and "o" – that the method has no recommendation for or against its usage for the identified ASIL. For example, methods 1a, 1b, 1e in Fig. 2 are required for unit testing for ASIL C. An increased ASIL D, now requires methods 1c and 1d which were only recommended for ASIL C.

4 GSN Safety Case Impact Assessment

In this section, we present our generic safety case impact assessment approach [13] specifically instantiated for GSN Safety Cases [10]. First, we define the GSN metamodel and the result of the impact assessment algorithm. Then, we describe the algorithm, which we name GSN-IA (GSN Impact Assessment), and the supporting model transformations. We have shown our algorithm to be sound [4] but do not replicate the argument due to space restrictions.

4.1 GSN and Annotation Models

Figure 3 gives a fragment of the GSN metamodel extended with state information. A `Goal` has a truth state and in this paper we assume that the truth state is two-valued truth (*true*, *false*) and that every goal represents a claim about the system for which the truth can be determined (e.g., claim expressed as a temporal logic statement). Thus, for the time being, we preclude more fine-grained measures of truth (e.g., degrees of confidence) and goals that have fuzzy truth conditions, and leave as future work. A `Solution` represents some kind of evidence about the system and has a validity state that indicates whether the evidence is applicable or it is "stale" and must be regenerated (e.g., old test results). A `Strategy` is used to decompose goals (conclusions) into subgoals (premises), and its validity state indicates whether the strategy is a valid one for connecting its premise goals to its conclusion. Finally, a `Context` element describes assumptions on the elements it connects to, and also has a validity state.

We consider two ways that a change to the system can impact the elements of the safety case: (1) `revise` – the *content* of the element may have to be revised because it referred to a system element that has changed and the semantics of the content may have changed, and (2) `recheck` – the *state* of the element must be rechecked because it may have changed. For example, the goal "The power sliding door opens when the function `DriverSwitch.RequestDoorOpen`() is invoked and the vehicle speed is not greater than 15 km/h." (see the class diagram in Fig. 1)

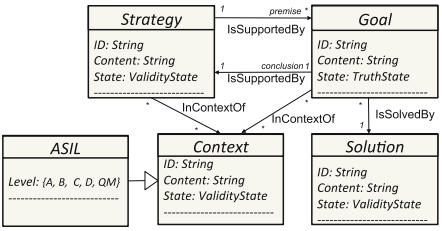

Fig. 3. Fragment of GSN metamodel extended with validity states.

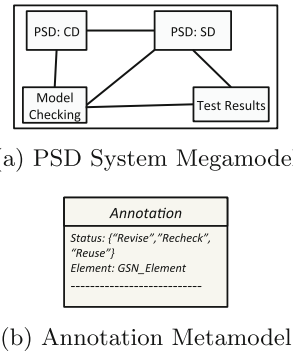

Fig. 4. PSD system megamodel and annotation metamodel.

must be revised if the function name is changed to `CommandDoorOpen`() since the goal now refers to an element that does not exist. However, if some aspect of the system that affects door opening functionality changes, then the goal must be rechecked because it may no longer hold. We assume that after a revision, a recheck must take place; thus, at most one of these impacts can apply to an element. If an element is not impacted by a system change we say that it can be reused and mark it as `reuse`.

The purpose of executing our impact assessment algorithm, GSN-IA, on a safety case is to determine the impact type for each safety case element and to "mark" the element accordingly. This marking is stored in a simple annotation model with the metamodel shown in Fig. 4b. Thus, an annotation model consists of an `Annotation` element for each GSN element that contains the marking as its `Status` attribute.

4.2 GSN-IA: GSN Impact Assessment Algorithm

Figure 5 shows the GSN-IA algorithm both in pseudocode and diagrammatically. The input to GSN-IA is the initial system model S and a safety case A connected by a traceability mapping R, the changed system S' and the delta D recording the changes between S and S'. Specifically, D is the triple $\langle C0a, C0d, C0m \rangle$ where of $C0a$ is the set of elements added in S', $C0d$ is the set of elements deleted from S and $C0m$ is the set of modified elements that appear in both S and S'. These are shown in the top part of the diagram. GSN-IA is parameterized by the model slicer $\texttt{Slice}_{Sys}$ used to determine how change impact propagates within the system model – that is, we consider this slicer to be given as an input to GSN-IA. Note that our approach readily applies not only to singleton models but also to more realistic cases where the system is described by a heterogeneous collection of related models as a megamodel. We have defined a sound slicing approach for this case [16]. The output of GSN-IA is the model K that annotates A to indicate which elements are marked for `revise`, `recheck` or `reuse`.

GSN-IA uses several *model transformations* described below. In line 1, the `Restrict` transformation extracts the subset R'_A of traceability links from R that are also valid for S'. Lines 2 and 3 use the model slicer $\texttt{Slice}_{Sys}$ to expand the combined (using `Union`) set of changed elements in S and S', respectively, to all elements *potentially impacted* by the change. Then, in line 4, these potentially impacted elements are traced to A across the traceability relationships using the `Trace` transformation and combined to identify the subset of elements in A that must be rechecked. The subset of safety case elements for revision is identified in line 5 by tracing the deleted and modified elements of S to A. Note that the elements of A marked `revise` is a subset of those marked `recheck`. Only those that are directly traceable to changed elements of S may require revision; others only need to be rechecked. In lines 6 and 7, the appropriate GSN slicer $\texttt{Slice}_{GSN_V}$ ($\texttt{Slice}_{GSN_R}$) is invoked to propagate each of the revise (recheck) subsets to dependent elements in A which are added to the recheck subset. Finally, line 8 invokes `CreateAnnotation` to construct the annotation model K from the identified subsets of A. The elements of the subset $C2_{\text{revise}}$ are marked `revise`; the remaining elements in the subset $C3_{\text{recheck2}}$ are marked `recheck`, and all other elements are marked `reuse`.

Algorithm: GSN-IA
Params: $\langle \texttt{Slice}_{Sys} \rangle$
Input: initial system model $S : Sys$, safety case $A : GSN$,
traceability map R, changed system megamodel $S' : Sys$,
delta $D = \langle C0a, C0d, C0m \rangle$
Output: Annotation K

1: $R'_A \leftarrow \texttt{Restrict}(R, D)$
2: $C1dm \leftarrow \texttt{Slice}_{Sys}(S, \texttt{Union}(C0d, C0m))$
3: $C1am \leftarrow \texttt{Slice}_{Sys}(S', \texttt{Union}(C0a, C0m))$
4: $C2_{\text{recheck}} \leftarrow \texttt{Union}(\texttt{Trace}(R, C1dm), \texttt{Trace}(R'_A, C1am))$
5: $C2_{\text{revise}} \leftarrow \texttt{Trace}(R, C0d)$
6: $C3_{\text{recheck1}} \leftarrow \texttt{Slice}_{GSN_V}(A, C2_{\text{revise}})$
7: $C3_{\text{recheck2}} \leftarrow \texttt{Slice}_{GSN_R}(A, \texttt{Union}(C2_{\text{recheck}}, C3_{\text{recheck1}}))$
8: $K \leftarrow \texttt{CreateAnnotation}(A, C3_{\text{recheck2}}, C2_{\text{revise}})$
9: **return** K

Fig. 5. Algorithm for assessing impact of system changes on a GSN safety case.

Our $\texttt{Slice}_{GSN_V}$ slicer uses the dependency rules in Table 1 adapted from the set of propagation rules described in [11] to identify elements to be marked for rechecking. For example, $\texttt{GSN}_{1.1}$ says that all goals and strategies linked to a goal G on either end of the `IsSupportedBy` relation are dependent on G (and are therefore marked "recheck"), if G is marked for revision. On the other hand, $\texttt{Slice}_{GSN_R}$ only uses two dependency rules to identify elements to be marked for rechecking: (1) Conclusion goals depend on premise goals they are indirectly linked to by the same strategy, and (2) Goals depend on solutions they are linked to by the `IsSolvedBy` relation.

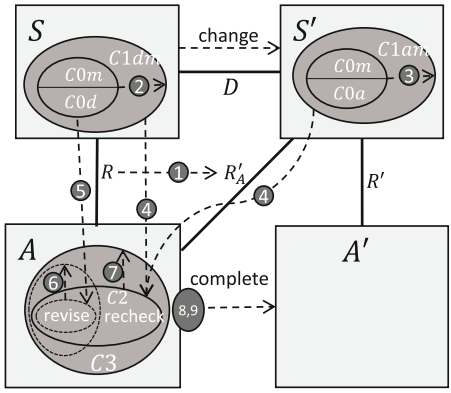

Fig. 6. Visualization of GSN-IA algorithm.

Table 1. $\texttt{Slice}_{GSN_V}$ dependency rules.

Rule	Element	Dependent Element(s)
GSN_1	`Goal` G	1. All goals/strategies linked to G on either end of the `IsSupportedBy` relation.
		2. All solutions linked to G via the `IsSolvedBy` relation
GSN_2	`Strategy` S	All goals linked to S on either end of the `IsSupportedBy` relation
GSN_3	`Context` C	1. All goals, strategies and solutions A that introduce C as the context via the `InContextOf` relation
		2. All goals, strategies and solutions that inherit C as the context (i.e., all children of A)
GSN_4	`Solution` S	All goals related to S via the `IsSolvedBy` relation

While $\texttt{Slice}_{GSN_V}$ only performs a one-step slice to find the revised elements' direct dependencies, $\texttt{Slice}_{GSN_R}$ works by continuously expanding a subset of elements in a GSN model to include its dependent elements until no further expansion is possible.

4.3 Illustration: Power Sliding Door Example

In our PSD example, the change in the system is the removal of the redundant switch, so the delta D is $\langle \emptyset, (\texttt{RedundantSwitch}), \emptyset \rangle$. The change directly affects goals B3-6 shown in Fig. 7, which refer to the Redundant Switch, and are therefore marked as `revise` by GSN-IA. The change also affects solutions SN3-6 which would include information about the Redundant Switch. Goal B2 refers to the AC ECU which is traced to the Redundant Switch in the PSD Class Diagram. $\texttt{Slice}_{Sys}$ would have detected that; therefore, B2 is marked `recheck`. Goal B1 does not link to any system components, so it does not appear in the result

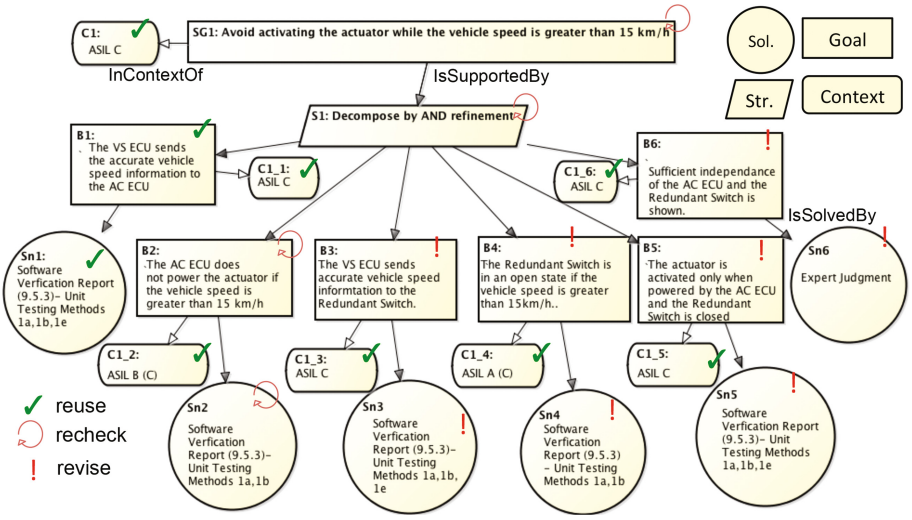

Fig. 7. An annotated GSN safety case for PSD system after running GSN-IA.

of $\texttt{Slice}_{Sys}$, and is therefore marked `reuse`. The remaining parts of the safety case elements are not traced directly to elements in the delta, but get marked using $\texttt{Slice}_{GSN_V}$ and $\texttt{Slice}_{GSN_R}$ described earlier. The result of GSN-IA is the annotation given on top of the original safety case and shown in Fig. 7.

5 A More Precise Impact Assessment

The algorithm GSN-IA, presented in in Sect. 4, is conservative, i.e., more elements are marked `recheck` and `revise` than potentially necessary to still be sound. In this section, we present six different techniques, T1-T6, aimed to improve the precision of GSN-IA. Together, they form a variant of GSN-IA, called GSN-IA-i (improved). The improvements in assigning annotation can be both at the level of safety case elements (goals, strategies, contexts and solutions), or finer, at the level of element identifiers. In order to validate GSN-IA-i, we use a metric $Cost_{IA}$ to compute the cost associated with revision and rechecking after impact assessment. The equation for $Cost_{IA}$ is shown in Fig. 8. For each technique, we describe the current state of GSN-IA, show how to improve the precision in each case (GSN-IA + Ti), present the prerequisites to ensure its soundness, and illustrate it on the PSD example. The techniques are summarized in Table 2.

$$
\begin{aligned}
Cost_{IA} &= Cost_{Revise} + Cost_{Recheck} \\
&= (Cost_{G_V} + Cost_{C_V} + Cost_{Sol_V} + Cost_{Str_V}) + (Cost_{G_R} + Cost_{C_R} + Cost_{Sol_R} + Cost_{Str_R}) \\
&= (\sum_{g \in G_V} K_V(1 + n(g)) + \sum_{c \in C_V} K_V(1 + n(c)) + \sum_{s \in Sol_V} K_V + \sum_{s \in Str_V} K_V) + \\
&(\sum_{g \in G_R} K_R + \sum_{c \in C_R} K_R + \sum_{s \in Sol_R} K_R + \sum_{s \in Str_R} K_R) \\
&= K_V(\sum_{g \in G_V} (1 + n(g)) + \sum_{c \in C_V} (1 + n(c)) + |Sol_V| + |Str_V|) + K_R(|G_R| + |C_R| + |Sol_R| + |Str_R|) \\
&= K_V(\sum_{g \in G_V} (1 + n(g)) + \sum_{c \in C_V} (1 + n(c)) + |Sol_V| + |Str_V|) + K_R(|E_R|), \text{ where:}
\end{aligned}
$$

- $Cost_{Revise}$ ($Cost_{Recheck}$): Cost of all revisions (rechecks).
- E_V (E_R): Number of total elements marked for revision (rechecking).
- G_V (G_R): Number of goals marked `revise` (`recheck`).
- C_V (C_R): Number of contexts marked `revise` (`recheck`).
- Str_V (Str_R): Number of strategies marked `revise` (`recheck`).
- Sol_V (Sol_R): Number of solutions marked for `revise` (`recheck`).
- $n(x)$: Number of identifiers in x marked for `revise`.
- K_V (K_R): Cost of performing a revision (a recheck).

Fig. 8. Cost equation for effort incurred after an impact assessment.

Table 2. GSN-IA + Ti techniques and improvements.

Technique	1	2	3–6
Improvement	$n(g) \downarrow, n(c) \downarrow$	$\lvert G_V \rvert \downarrow, \lvert C_V \rvert \downarrow$	$\lvert E_R \rvert \downarrow$

5.1 T1: Increasing the Granularity of Traceability Between the System and the Safety Case

GSN-IA: Trace links between the system and safety case provided to GSN-IA are assumed to link entire safety case elements to system elements. That is, if a change occurs in any of the linked system elements, the entire safety case element is marked for revision.

GSN-IA + T1: Trace links between the system and safety case connect *identifiers* in safety case elements to corresponding system elements. Annotations are then assigned to safety case element identifiers rather than to entire elements.

Improvement: With more fine-grained trace links, GSN-IA + T1 can identify which specific identifiers in a safety case element should be marked for revision, allowing the safety engineer to focus on revising only those parts instead of the entire element. This in turn decreases the number of unnecessary identifier revisions, i.e., $n(g)$ and $n(c)$, since only goals and context nodes are assumed to have identifiers traceable to the system, thus decreasing the overall cost.

Prerequisites: A safety case language that clearly distinguishes identifiers from other text, ensuring that the finer-grained trace links cover at least all the originally covered links in order to preserve soundness of the technique.

Example: In the PSD system, the goal B3 "The VS ECU sends accurate vehicle speed information to the Redundant Switch" can be traced to both VS ECU and Redundant Switch components. Currently, when either VS ECU or Redundant Switch changes, GSN-IA marks the entire goal `revise`. A more fine-grained traceability would link the identifier "VS ECU" to VS ECU in the system and the identifier "Redundant Switch" to the Redundant Switch in the system. Now, if Redundant Switch changes in the system but VS ECU does not, then only the identifier "Redundant Switch" in goal B3 needs to be marked for revision, while the rest of the goal can be reused.

Discussion: Traceability between the system and its safety case can be established at different levels of granularity. Formal safety case languages have clearly defined *identifiers*, thus they can easily be traced to the appropriate system elements. For example, the author of [12] defines a six-step approach for creating well-formed GSN goal structures that in turn aid in a finer-grained system traceability. For languages that only use natural language to describe goals, this fine grained traceability may not be feasible.

5.2 T2: Identifying Sensitivity of Safety Case to System Changes

GSN-IA: Any change to a system element will cause its associated element in the safety case to be marked for revision.

GSN-IA + T2: We mark the safety case element for revision only if it is required by the type of system change.

Improvement: Unnecessary revisions of safety case element are minimized by identifying cases where a system change should actually impact the element, and where it can be ignored. This in turn decreases the number of goal ($|G_V|$) and context ($|C_V|$) elements marked for revision, decreasing the overall cost.

Prerequisites: For each model type in the system megamodel, a sensitivity table that lists all element types of that model and the kinds of changes that they can undergo, and, for each trace link between the system and the safety case, the type of change the link is sensitive to. We assume that the *types* of changes that occur as part of the system evolution are captured with each of the corresponding changes in the *Delta* we are provided. Since the assignment of sensitivity to change is performed by the domain expert, we require these assignments to be correct to preservation of soundness.

Example: In the PSD System, the class Door in the Class Diagram model has an attribute *state*, which is an enumeration with possible values *open* and *closed*. Assume a goal such as "If the door state is open and the speed is greater than 15 km/h, the driver is notified.". Currently, if we add a new option to the door state (e.g., "stuck"), that is considered a change in the door state, which marks the goal for revision. However, such a change (an attribute enumeration extension) should not impact the goal which is only concerned with the door state being open. If we do not add that type of change in the sensitivity list of

that particular trace link between system and goal, we are able to ignore it and allow the goal to be `reused`.

Discussion: In the example above, if the goal had been "If the door state is *not closed* and the speed is greater than 15 km/h, the driver is notified.", then the change should have impacted this goal, as "stuck" is considered "not closed". We assume that goals are structured in a way that specific states are identified; if they are not, T2 cannot be used. Interestingly, in such a case, the goal would have to be marked `revise`, which may allow detecting missing test cases or other evidence for the "stuck" state.

5.3 T3: Understanding Semantics of Strategies

GST-IA: Any truth valuation change of the premise goals of a strategy lead to rechecking the conclusion goal.

GSN-IA + T3: Here, we use semantic knowledge, i.e., which changes in truth values of the premises do not affect the truth value of the conclusion.

Improvement: We limit the unnecessary propagation of `recheck` annotations across the safety case, thus $|E_R|$ decreases, causing the overall cost to decrease.

Prerequisites: Semantics of the strategies connecting premise and conclusion goals. This applies to a fixed set of known strategies and not to strategies expressed in natural language. Soundness is preserved since we are using sound semantics of logical connectives to make decisions.

Example: Assume in the PSD system that SG1 was connected to its subgoals B1-B6 via an "OR" decomposition strategy (as opposed to an "AND"). Also assume that currently all of B1-B6 have *true* states. This means that SG1 is also evaluated to *true*. If the system changes so that B5-B6 are marked `recheck`, we don't need to mark SG1 `recheck` since, due to disjunction, it must still be *true*.

Discussion: Marking a premise of an "OR" strategy `recheck` (while other premises are marked `reuse`) can impact the overall confidence in the argument, as the premise can become *false* after the recheck is performed. We do not take confidence into account at this point and consider it future work.

5.4 T4: Decoupling Revision from Rechecking

GSN-IA: Forces a recheck every time an element is marked `revise`.

GSN-IA + T4: By knowing circumstances under which revising a goal will not impact its truth value, we require a recheck after a revision only when necessary.

Improvement: Eliminating unnecessary rechecks after revisions leads to possibly decreasing $|E_R|$ and, therefore, the overall cost.

Prerequisites: An extra column in the sensitivity table described in T2 that lists if a particular type of change affects the truth value of a goal. We require

correctness of assignments of *changes* to their *effect on goal truth values* as well as completeness of trace links to ensure soundness of the approach.

Example: In the PSD system, changing the name of a system element such that it does not conflict with other names (e.g., Redundant Switch is renamed to Extra Switch) will cause the goals referring to that element (e.g., goal B3) to be marked for revision. However, since changing the name does not impact the truth state of the goal, rechecking can be skipped. Other examples include capitalization of names, spelling corrections or language translations, such that the renaming is done consistently in both the system and the safety case.

5.5 T5: Strengthened Solutions Do Not Impact Associated Goals

GSN-IA: If a piece of evidence that a solution points to changes, the goal supported by that solution is always marked `recheck`.

GSN-IA + T5: A change to a solution that strengthens it should not affect its support for associated goals.

Improvement: Understanding which changes in solutions do not necessitate a rechecking of associated goals can reduce the unnecessary goal rechecks. Thus, $|E_R|$ decreases causing the overall cost to decrease.

Prerequisites: A sensitivity table (similar to T2) that identifies, for each type of evidence, the types of changes it can undergo, and for each "isSupportedBy" link between a solution node pointing to this kind of evidence and a goal, whether or not it is sensitive to each kind of change. Assignments of *changes* to their *effect on goal truth values* need to be correct to guarantee soundness.

Example: Assume that B1 was "The VS ECU sends accurate vehicle speed information to the AC ECU *90% of the time*" and that it was linked to a solution with test cases which showed accuracy 90% of the time. If the system changes so that the test cases can now demonstrate accuracy 100% of the time, this does not affect goal B1, meaning that it should not be marked for rechecking.

5.6 T6: Exploiting Knowledge About ASIL Work-Product Dependencies and ASIL Propagation and Decomposition Rules

GSN-IA: Does not take into account how changes in the system impact ASILs.

GSN-IA + T6: Determine how ASILs should change due to system changes by using knowledge about ASIL work-product dependencies and ASIL propagation and decomposition rules.

Improvement: Increase in precision due to distinguishing between changes to the goals and changes to the ASILs, potentially decreasing the number of required goal rechecks. This decreases $|E_R|$, thereby decreasing the overall cost.

Prerequisites: Dependency tables from ISO 26262 Part 6 that describe the types of methods for each work product required to achieve certain ASILs, and

ASIL decomposition and propagation rules as presented in ISO 26262 (refer to Sect. 3). We assume the soundness of the tables and ASIL propagation and decomposition rules in order to guarantee soundness of our approach.

Example: We present two examples in the PSD system:

1. If method `1e` (Back-to-back comparison test between models and code) used for unit testing as part of the Software Verification Report work product for goal B1 is deleted, the ASIL for B1 supported by Sn1 changes from ASIL C to ASIL B based on Table 1. This would in turn impact the ASIL on SG1, since the ASIL propagation rule no longer holds. In this case, claims B1 and SG1 themselves are not impacted, only their ASIL levels are.

2. With redundancy present in the PSD system, ASIL decomposition was used to allocate ASIL B to B2 and ASIL A to B4 (decomposed from ASIL C). B6 was added to demonstrate sufficient independence of the Redundant Switch element from the AC ECU as required by ASIL decomposition. When the system changes and the redundant switch is deleted, requirements B4 and B6 are marked for revision, causing the original decomposition rule to be impacted. B2 is only marked `recheck`, but its ASIL level will be marked for revision (from ASIL B to ASIL C) to respect ASIL propagation rules from SG1. The impact assessment now flags both C1_2 and Sn2 for revision. Ideally, the safety engineer will revise Sn2 to be strengthened (e.g., unit testing method `1e` is added) to increase the ASIL on B2 to level C.

5.7 PSD Example Cost Comparison

Assume that the revision cost K_V is 2 units and the rechecking cost K_R is 1 unit[3]. On the PSD example, GSN-IA produced an annotation with 8 elements marked `revise` (4 goals, 4 solutions) and 4 marked `recheck`. Goals marked `revise` have the following number of identifiers: B3 has 3 (VS ECU, vehicle speed, Redundant Switch), and similary, B4, B5 and B6 each respectively have 3, 6 and 2 identifiers. The cost incurred after GSN-IA is $2 \times ((1+3)+(1+3)+(1+6)+(1+2)+4)+1 \times (4) = 48$ units.

Using T1, changes to the redundant switch link only to the *Redundant Switch* identifier in goals B3-B6 (as opposed to the entire goals), dropping the number of revised elements in each of these goals to only 1 (as opposed to marking all the identifiers in the goal for revision). The cost after running GSN-IA + T1 is $2\times((1+1)+(1+1)+(1+1)+(1+1)+4)+1\times(4) = 28$ units, representing a clear improvement. Due to space limitations, we do not demonstrate the application of the other techniques.

6 Related Work

Model-Based Approaches to Safety Case Management. Many methods for modeling safety cases have been proposed, including goal models and requirements models [3,6] and GSN [10]. The latter is arguably the most widely used

[3] In practice, $K_V > K_R$, since revision requires more effort than rechecking.

model-based approach to improving the structure safety arguments. Building on GSN, Habli et al. [7] examine how model-driven development can provide a basis for the systematic generation of functional safety requirements and demonstrates how an automotive safety case can be developed. Gallina [5] proposes a model-driven safety certification method to derive arguments as goal structures given in GSN from process models. The process is illustrated by generating arguments in the context of ISO 26262. We consider this category of work complimentary to ours; we do not focus on safety case construction but instead assume presence of a safety case and focus on assessing the impact of system changes on it.

Safety Case Maintenance. Kelly [11] presents a tool-supported process, based on GSN, that facilitates a systematic safety case impact assessment. The work by Li et al. [15] proposes an assessment process to specify typical steps in the safety case assessment. The authors develop a graphical safety case editor for assessing GSN-based safety case and use the Evidential Reasoning (ER) algorithm to assess the overall confidence in a safety case. Jaradat and Bate [9] present two techniques that use safety contracts to facilitate maintenance of safety cases. As far as we are aware, none of the approaches provide a structured model-based algorithm for impact assessment, or consider methods for improving its efficiency. In the context of safety case maintenance, Bandur and McDermid [1] present a formalization of a logical subset of GSN with the aim of revealing the conditions which must be true in order to guarantee the internal consistency of a safety argument. This provides a sound basis for understanding logical relationships between components of a safety case and thus to enhance impact assessment.

7 Conclusion

In this paper, we showed how using various sources of knowledge about the system changes, the particular safety case language and the safety standard can increase the precision of the previously proposed impact assessment technique [13], thus reducing the work required by the safety engineer. We presented six precision improvement techniques and illustrated our ideas using a GSN safety case used with ISO 26262. In the future, we aim to address the following problems:

Addressing Additions. Currently, our impact assessment approach addresses the effect of adding components in the system on the existing parts of the safety case. However, it currently cannot address how adding components can potentially require additions to the safety case. We plan to study this further and propose approaches for addressing this in the future.

Exploiting System Design Patterns. We would like to understand whether our approach can detect certain changes in the system design which change not the functionality of the system but its level of integrity. For example, consider the "redundancy pattern", where a component such as the redundant switch in our PSD example is added. We would like to study if it is possible to syntactically identify this case as a redundancy change by witnessing two paths to the actuator

(one via the VS ECU and one via the redundant switch), and how this can be exploited for impact assessment.

Design Space Exploration. We believe that our approach can be used for impact assessment in general, and not just for safety case co-evolution due to system changes. One application for this is in design space exploration, to enable answering what-if questions about the impact of changes on safety cases. In this context, we would like to study the effect of changes, other than just system changes, on the safety case.

Constructing an Assurance Case for Change. Our impact assessment approach can guide the creation of a Change Argument: an argument for the changes made to the original safety case, providing evidence for such an argument. For example, our approach can support a `revise` marking in a safety case by linking the element to the appropriate counterparts in the system megamodel that caused this marking to be computed.

Confidence. We would like to augment our approach to handle a confidence model on top of safety cases. That is, we would like to assess the impact of changes not just on the safety case elements themselves but on the confidence level we assign them and on the safety case as a whole.

Tool Support and Validation. We are actively working on extending our model management framework MMINT [4] to include safety cases and model management operators for them (e.g., slice). We are also implementing our impact assessment approach using the workflow language defined in MMINT. We plan to incorporate the improvement techniques discussed in this paper and validate their effectiveness on a large scale industrial example.

Acknowledgements. This work is funded by Automotive Partnership Canada and NSERC in collaboration with General Motors.

References

1. Bandur, V., McDermid, J.: Informing assurance case review through a formal interpretation of GSN core logic. In: Proceedings of SafeComp 2015, pp. 3–14 (2015)
2. Borg, M., Vara, J.L., Wnuk, K.: Practitioners' perspectives on change impact analysis for safety-critical software – a preliminary analysis. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9923, pp. 346–358. Springer, Cham (2016). doi:10.1007/978-3-319-45480-1_28
3. Brunel, J., Cazin, J.: Formal verification of a safety argumentation and application to a complex UAV system. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7613, pp. 307–318. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33675-1_27
4. Di Sandro, A., Salay, R., Famelis, M., Kokaly, S., Chechik, M.: MMINT: a graphical tool for interactive model management. In: Proceedings of MoDELS 2015 (Demo) (2015)
5. Gallina, B.: A model-driven safety certification method for process compliance. In: Proceedings of ISSRE 2014 Workshops, pp. 204–209. IEEE (2014)

6. Ghanavati, S., Amyot, D., Peyton, L.: A systematic review of goal-oriented requirements management frameworks for business process compliance. In: Proceedings of RELAW 2011, pp. 25–34. IEEE (2011)
7. Habli, I., Ibarra, I., Rivett, R.S., Kelly, T.: Model-Based Assurance for Justifying Automotive Functional Safety. Technical report, SAE (2010)
8. International Organization for Standardization: ISO 26262: Road Vehicles - Functional Safety, 1st version (2011)
9. Jaradat, O., Bate, I.: Systematic maintenance of safety cases to reduce risk. In: Skavhaug, A., Guiochet, J., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2016. LNCS, vol. 9923, pp. 17–29. Springer, Cham (2016). doi:10.1007/978-3-319-45480-1_2
10. Kelly, T., Weaver, R.: the goal structuring notation - a safety argument notation. In: Proceedings of DSN 2004 (2004)
11. Kelly, T.P., McDermid, J.A.: A systematic approach to safety case maintenance. In: Felici, M., Kanoun, K. (eds.) SAFECOMP 1999. LNCS, vol. 1698, pp. 13–26. Springer, Heidelberg (1999). doi:10.1007/3-540-48249-0_2
12. Kelly, T.: A Six-Step Method for the Development of Goal Structures. York Software Engineering, Flixborough (1997)
13. Kokaly, S., Salay, R., Cassano, V., Maibaum, T., Chechik, M.: A model management approach for assurance case reuse due to system evolution. In: Proceedings of MoDELS 2016, pp. 196–206 (2016)
14. Leveson, N.: Engineering a Safer World: Systems Thinking Applied to Safety. MIT Press, Cambridge (2011)
15. Li, Z.: A Systematic Approach and Tool Support for Assessing GSN-Based Safety Case. Master's thesis, Technische Universiteit Eindhoven (2016)
16. Salay, R., Kokaly, S., Chechik, M., Maibaum, T.: Heterogeneous megamodel slicing for model evolution. In: Proceedings of ME@MoDELS 2016, pp. 50–59 (2016)
17. de la Vara, J.L., Borg, M., Wnuk, K., Moonen, L.: An industrial survey of safety evidence change impact analysis practice. IEEE TSE **42**(12), 1095–1117 (2016)