Research Reports in Software Engineering and Management 2012:04 ISSN 1654-4870

A Light-Weight Defect Classification Scheme for Embedded Automotive Software Development

Niklas Mellegård,
Miroslaw Staron, Fredrik Törner

CHALMERS | UNIVERSITY OF GOTHENBURG
Department of Computer Science and Engineering

A Light-Weight Defect Classification Scheme for
Embedded Automotive Software Development

Report no 2012:04
ISSN: 1651-4769
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden

Telephone + 46 (0)31-772 1000

Göteborg, Sweden 2012

Research Reports in Software Engineering and Management No. 2012:04

A Light-Weight Defect Classification Scheme for Embedded Automotive Software Development

Niklas Mellegård
Miroslaw Staron
Fredrik Törner

Department of Computer Science and Engineering
CHALMERS | University of Gothenburg

Gothenburg, Sweden 2012

A LIGHT-WEIGHT DEFECT CLASSIFICATION SCHEME FOR EMBEDDED AUTOMOTIVE SOFTWARE DEVELOPMENT

Niklas Mellegård, Miroslaw Staron
Software Engineering Division
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
{niklas.mellegard, miroslaw.staron}@chalmers.se

Fredrik Törner
Volvo Car Corporation
ftorner@volvocars.com

Abstract–
Objective: **Systematic software defect documentation is an essential part of software development process models as a means of early identification of patterns in defect inflow. Such documentation, however, may often be a tedious task requiring analysis work in addition to what is necessary to resolve the issue. Furthermore, generic defect documentation approaches often have a strong focus on source-code aspects making them unsuitable for development contexts with supplier-side implementation. To increase documentation efficiency in a development context with limited access to source-code, adapted schemes are needed. In this paper a light-weight defect classification scheme adapted to automotive software development is presented.**
Method: **A case study was conducted at Volvo Car Corporation to adapt the IEEE Std. 1044 for the development of embedded automotive safety features.**
Results: **The results consist of a detailed description of a defect classification scheme that complies with the IEEE Std. 1044. The main adaptations to the scheme consisted of raising the level of abstraction of the captured data items, shifting the focus from source-code to other artefacts and activities, and by conforming to the terminology of the company.**
Conclusions: **We conclude that the IEEE Std. 1044 can be successfully adapted to a development context where source-code is not the main development artefact. Furthermore, initial evaluation showed that the adapted classification scheme captures what is currently tacit knowledge and has the potential of revealing patterns in the defects detected in different project phases. As a result we are currently in the process of incorporating the classification scheme into the company's defect reporting system.**

Keywords: Software engineering; Defect analysis; Modelling; Process

1 INTRODUCTION

Software reliability is of central importance in modern cars as software controlled systems are becoming increasingly pro-active – recent safety functions are, for instance, able to automatically apply brakes to avoid crashes or mitigate their effects. Car manufacturers (OEMs) need, in order to achieve reliability, effective ways to manage defects during development (in-process) and during run-time (e.g. fault tolerance mechanisms). For the in-process defects it is important to identify, analyse and remove defects which could compromise the reliability of the cars. Furthermore, identifying patterns in the in-process defects enables effective detection and removal of defects, for instance by indicating which test activities to focus on. In order to identify such patterns, however, systematic and structured defect documentation is required.

Defect documentation and analysis is common practice in most software development organizations. Its benefits are further emphasised through the inclusion in process maturity models – such as CMMI [1] and SPICE [2] – as they require systematic defect documentation, analysis and follow-up. Neither CMMI nor SPICE, however, specifies how such defect documentation and analysis is to be done. Companies thus have their own interpretations resulting in varying quality of defect documentation; for instance, ambiguous interpretation of data or subjective opinions of the reporter. Hence, there is a need for a structured approach to defect documentation.

There have been several approaches proposed on how to perform structured collection and analysis of defect information; e.g. defect taxonomies [3], root cause analysis (RCA) [4] as well as various defect

classification schemes such as Orthogonal Defect Classification (ODC) [5], the HP scheme [6] and IEEE Std. 1044 [7]. Although shown to be useful these approaches were designed for specific contexts [8] causing the need for adaptations [9]; such adaptations have been identified as one of the major challenges in applying a defect classification scheme [8], [9]. Specifically, defect classification approaches often assume full knowledge of the defects, i.e. have a source-code focus and assume ownership of the software components [8]. Consequently, such defect classifications schemes need adaptations to be applicable to organizations where software is developed by suppliers – a situation common in the automotive software domain: even though software components (e.g. ABS or collision warning system) are often developed by suppliers, the quality of the complete product – the car – is the responsibility of the OEM. The need to systematically analyse and follow-up on the quality of the supplied software components is, nevertheless, important.

Furthermore, defect documentation – however important – may be seen as a mainly administrative task that does not directly contribute to the end-product. Thus, the defect documentation approach taken should require a minimum of analysis effort in addition to what is needed to identify and remove the defect, while still providing the additional benefit of characterizing the quality of the product and development process [10].

In this paper we address the challenges of efficient defect classification by pursuing the following research question: "*How to efficiently support defect identification and resolution time by classifying in-process defects?"*. The research question is addressed by investigating how a defect classification scheme can be adapted to the automotive software development context by studying the development of active safety features. The aims of our adapted classification scheme – the Light-weight Defect Classification scheme (LiDeC) – include: (1) as existing classification schemes have a strong source-code focus, how can such a scheme be adapted to a development setting with limited insight into the source-code; (2) as adding additional workload on development teams may reduce the likelihood of adoption, how can a classification scheme be adapted to minimize its process foot-print in terms of required learning and classification time.

LiDeC was developed as part of a case-study at Volvo Car Corporation (VCC[1]) and initially evaluated with a sample of problem reports from a project finished a year prior to the study. As a result we present a defect classification scheme, compliant with the IEEE Std. 1044 [11], [12], specifically adapted for the development of automotive safety-critical software. Furthermore, an initial evaluation showed that developers quickly learned to apply the classification scheme, and that the required time to classify a defect was substantially lower than with other approaches to defect documentation.

The rest of the paper is structured as follows: section 2 provides the study with background, section 3 describes the method used, section 4 summarizes the results and the final sections conclude the paper and outline future work.

2 BACKGROUND

This section provides the research presented in this paper with background: first a summary of the terminology used is outlined, then related work is presented, and finally aspects of developing software at the case company is presented.

2.1 Terminology

In this report the terminology defined in the IEEE Std. 1044-2009 [7] is used. Specifically the following terms are used in the report:

- *Defect*– An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced [7].
- *Failure*– (A) Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits. (B) An event in which a system or system component does not perform a required function within specified limits.
- *Fault*– A manifestation of an error in software.

Figure 1 shows how these terms relate and what is within the scope of the IEEE Std. 1044.

[1] http://www.volvocars.com

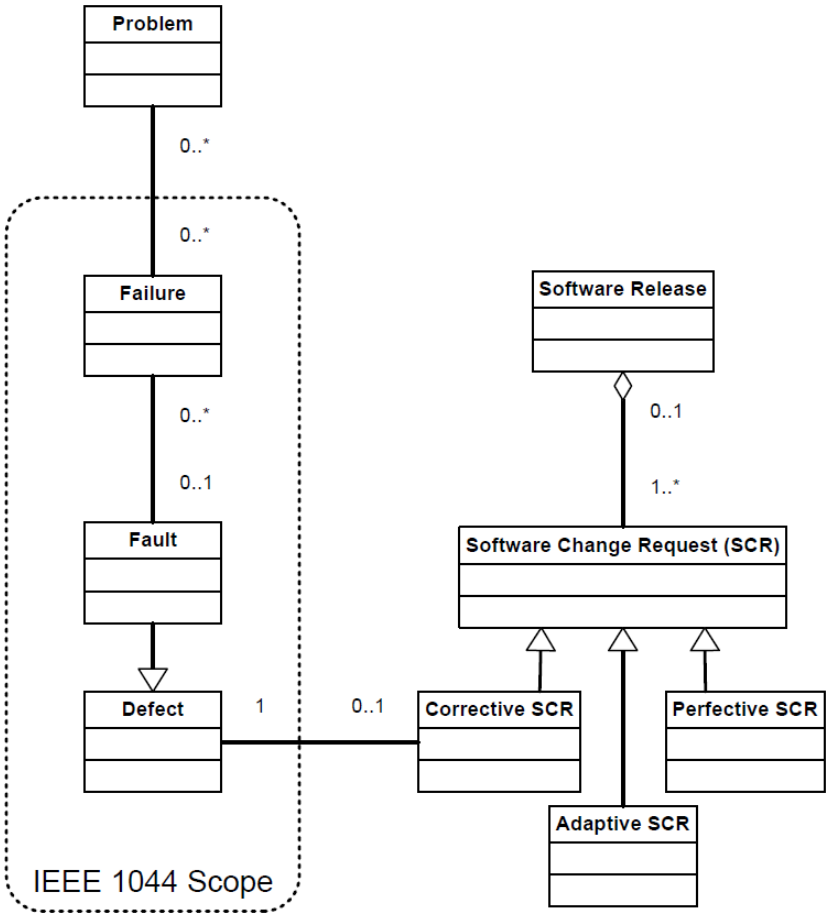

Figure 1. IEEE Std. 1044 concepts and their relationships (from [7])

As shown in Figure 1, problems are sufficient but not necessary conditions for the recognition that the software is failing to behave in a desirable manner. The failure may be caused by faults in the software, which in turn can be corrected by a software change request. These relationships are described in Table 1.

TABLE 1. RELATIONSHIPS BETWEEN IEEE STD. 1044 CONCEPTS (FROM [7])

Class/Entity pair	Relationships
Problem - Failure	A problem may be caused by one or more failures. A failure may cause one or more problems.
Failure - Fault	A failure may be caused by (and thus indicate the precense of) a fault. A fault may cause one or more failures.
Fault – Defect	A fault is a subtype of the supertyp defect. Every fault is a defect, but not every defect is a fault. A defect is a fault if it is encoutered during software execution (thhus causing a filaure). A defect is not a fault if it is detected by inspection or static analysis and removed prior to executing the software.
Defect – Change Request	A defect may be removed via completion of a corrective change request. A corrective change request is intended to remove a defect. (A change request may also be initiated to perform adaptive or perfective maintanance.)

2.2 Related Work

This report is an extension of our previous work [13]. Whereas our previous work focused on the evaluation of LiDeC this report instead focus on the description of LiDeC; more specifically:

i. The Background section has been extended with a terminology subsection
ii. The Background section has been extended with more related work
iii. The results has been extended with a new subsection with a comparison between LiDeC and IEEE Std. 1044
iv. The description of the attributes in the results section has been extended with more details
v. A full description of LiDeC has been added as Appendix A
vi. A classification guide has been added as Appendix B
vii. Two example classifications using LiDeC has been added as Appendix C
viii. An IEEE Std. 1044 compliance matrix has been added as Appendix D
ix. A mapping between attributes of IEEE Std. 1044 and LiDeC has been added as Appendix E

Defect reports are a valuable source of information about issues that arise in development: defect reports can reveal information about systematic problems with the development process such as the activities most prone to generating defects, or the efficiency of testing activities with respect to the number and type of defects they detect. Defect reports, however, are often used as a means to track and resolve the identified defect. But in order to systematically collect and analyse defect data there is a need to formalize the information collected about each defect. There are several proposed approaches which Wagner [9] identifies as belonging to three main categories:

- *Defect taxonomies* which are categorizations of faults mainly related to the implementation. Wagner mentions examples categories such as wrong variable declarations and wrong variable scope;
- *Root cause analysis* (RCA) which is a more detailed approach. RCA not only analyses the fault itself, but also why the fault was introduced. The goal of RCA is to identify the root causes and eliminating them, thereby preventing similar faults from being introduced in future projects;
- *Defect classification* is an approach in which data is collected about the defect in a similar manner to both defect taxonomies and root cause analysis, but does so in a more coarse-grained manner.

Defect taxonomies are focused on the implementation and do not provide support for analysing what measures to take to prevent or mitigate any systematic issues it may reveal. RCA, in contrast, is focused on identifying why the identified defect was introduced into the system. RCA, however, is considered to be effort intensive and its cost/benefit is unclear [9]. Defect classification, on the other hand, aims at reducing the effort required to analyse a defect while still retaining the power of analysis – such as what types of defects are most common, which artefacts are most prone to defects. The approach taken is to gather a wider but more coarse-grained range of data. In this paper we have chosen to adapt a defect classification scheme given our goal of small process foot-print. Defect classification schemes are discussed in more detail in the following subsection.

In their paper Li et al. [14] present experience from adapting existing issue tracking systems at two companies. The adaptations resembles our work as the pre-existing issue tracking system were mainly intended for in-process progress tracking of defect resolutions and resource management. Their justification for adapting the issue tracking systems included inadequately designed attributes and attribute values which made the collected issue data poorly equipped for use as software quality assessment and software process improvement – data was entered inconsistently or omitted, resulting in the assembled data "behaving largely as an information graveyard" [14]. By redesigning the issue tracking systems – incorporating parts of ODC [5] and the IEEE Std. 1044 – Li et al. were able collect higher quality data and use that data to point out improvement targets in both companies studied. Furthermore, follow-up analyses conducted after the process changes were able to detect improvement in terms of lower number of defects.

The work presented in this paper complements the scheme presented by Li et al. in that our classification scheme targets a development context in which code to a large extent is written by sub-contractors and where the sub-contractors own the source-code; thus, limiting the possibilities to analyse the exact nature of the defects. Furthermore contrary to the work presented by Li et al., where the classification scheme had to comply with legacy issue tracking systems – attributes were added to or modified in already existing defect databases -- we had the opportunity to work alongside the team

setting up a new issue management system. As a result, LiDeC is compliant with the IEEE Std. 1044. In addition, Li et al. identifies a number of lessons learned that are of great interest in our work, as we currently are in the process of incorporating LiDeC in a new issue tracking system at our industrial partner.

Dubey [15] reports on a case-study applying ODC scheme to a project developing embedded systems in which a substantial amount of software was developed by suppliers. They concluded that the classification scheme's focus on source-code required it to be adapted. Specifically, the attribute defect type needed adaptation. In their case there were many defects classified as "*Functional*" defects leading them to propose additional attribute values related to the design phase (as ODC originally only contained one value: "*Functional defect*").

In [4], [16] Lezsak et al. report that conducting RCA on up to a year old defect reports in a distributed, component-based development process required on average 19 minutes per defect. Leszak et al. also concluded that analyses conducted in-process when detailed knowledge about the defects can easily be recalled would further reduce the required effort.

Cavalcanti et al. [17] investigated the problem of identifying duplicate defect reports in a number of private and open source projects. Specifically, they examined the amount of time required to analyse a defect to determine whether it was a duplicate or not. The time required varied in the projects they examined from 5-10 minutes per defect to 20-30 minutes, with an average of 12.5 minutes. Furthermore, based on the size of staff and amount of defects reported, Cavalcanti et al. calculated that on average 48 man-hours per day was spent in search of duplicate defects in the examined projects.

Software reliability growth models (SRGM) [18] estimate software reliability by statistically correlating the cumulative number of defects discovered to a known function [19]. SRGMs can be used to predict the number of residual defects in a product. SRGMs, however, do not provide any data about the type of defects, or in which part of the product they are likely to occur. Consequently, SRGM provides limited guidance as to which testing activities should be focused on to detect the yet unknown defects. Defect classifications, on the other hand, provide more detailed information – e.g. about detection and injection phase, and type of defect – and can be more precise than traditional SRGM [20].

There have been many methods on fault prediction proposed. Liparas et al. [21] examine a statistical method for analysing what factors in a multivariate data set that are best suited for predicting the number of defects contained in a software module. In their paper, Liparas et al. use a set of complexity metrics – such as McCabe's cyclomatic complexity, lines of code and branch count – to predict the fault-proneness of the modules. The method described predicted whether a module is within a normal cluster or not – where a module not in the normal cluster would contain more than the standard amount of defects. Such information is valuable when assigning resources; more resources can be assigned to the modules that are most likely to contain defects. While the number of defects may be used to indicate the modules in most need to testing, it does not, however, provide the testers with any indications of what to test for, nor which modules contain the most sever defects. In fact, in the comprehensive systematic literature review where Hall et al. [22] examined 36 studies (from a selection of 2,073) on fault prediction models published between January 2000 and December 2010, they found that few studies differentiate between the faults predicted; for instance, only one [23] of the 36 studies used fault severity in their prediction model. In order to differentiate between defects, more nuanced data about the defects are needed; our work aims at contributing to a model for assembling such defect data, thus enabling prediction of additional defect attributes.

2.2.1 Defect classification schemes

Defect classification schemes define a set of attributes, where each attribute captures a specific aspect of the defect – e.g. how the defect was detected, its severity and type. Moreover, for each attribute the schemes typically provide a set of values that can be chosen from; this contributes to the efficiency as well as to the reliability of the classification. The most commonly referred [10] classification schemes in literature are ODC from IBM [5], the HP approach [6] named *Defect Origins, Types and Modes* [9] (here referred to as the HP scheme) and the IEEE Std. 1044 [7].

Regardless of which classification scheme is applied, the main challenge is to select the attributes and attribute values which are relevant to the specific development context [8]. In [10], Freimut provides a framework for developing and using classification schemes; this framework has been followed in the work presented in this paper. Furthermore, in [10] Freimut provides a comparison between the ODC, HP and IEEE Std. 1044 classification schemes and a mapping between the attributes of the different classification schemes.

2.2.1.1 The HP Scheme

The approach taken by the HP scheme is to define only three attributes: Origin, Type and Mode. The *Type* attribute is dependent on the value chosen for the Origin attribute. This first requires analysis of when the defect was injected into the system before its type can be established. Furthermore, the HP scheme does not explicitly capture data about how a defect was detected (its trigger [5], [24]); there is thus no attribute available to identify which testing activities are effective in detecting particular defect types. Moreover, applying the HP scheme makes it difficult to identify effective testing techniques and investigate how late and severe defects can be identified earlier as the scheme does not include attributes such as:

- Severity of the defect from an end-user perspective
- The method by which the defect was detected
- Timing of defect detection
- The cause of the defect

Such issues considered important for VCC, thus a wider range of attributes than provided by the HP scheme needed to be collected.

TABLE 2. IEEE STD. 1044 ATTRIBUTES (ADAPTED FROM [10])

Life-cycle Phase	Attribute Name	Attribute Meaning
Recognition	Project activity	What were you doing when the defect occurred?
	Project Phase	In which life-cycle phase is the product?
	Suspected Cause	What do you think might be the cause?
	Repeatability	Could you make the defect appear more than once?
	Symptom	How did the defect manifest itself?
	Product Status	What is the usability of the product with no changes?
Investigation	Actual Cause	What caused the anomaly to occur?
	Source	Where (part of the system and its documentation) was the origin of the defect?
	Type	What type of defect/enhancement at the code level?
Action	Resolution	What to do to prevent the defect from happening again?
	Corrective action	What action to take to resolve the defect?
Impact Identification	Severity	How bad was the defect in more objective engineering terms?
	Priority	Rank the importance of resolving the defect?
	Customer value	How important is a fix to the customer?
	Mission safety	How bad was the defect wrt. project objectives or human well-being?
	Project schedule	Relative effect on the project schedule to fix?
	Project cost	Relative effect on the project budget to fix?
	Project risk	Risk associated with implementing a fix?
	Project Quality/Reliability	Impact to the product quality or reliability to make a fix?
	Societal	Impact of society of implementing the fix
Disposition	Disposition	What actually happened to close the anomaly?

2.2.1.2 IEEE Std. 1044

The IEEE Std. 1044 and ODC, in contrast, define a set of failure and defect life-cycle phases each containing a number of attributes (independent of each other) that are to be recorded; the life-cycle defined by IEEE Std. 1044 is shown in Table 2. The phases represent the states the defect can be in: initially a failure is recognized, then investigated, which might lead to discovering the cause of the failure (fault), an action to resolve the defect is then planned and the possible impacts of the chosen action/resolution are analysed, and finally what was actually done to close the defect. The attributes that are to be recorded in each of the phases represent information about the defect that is relevant for that particular phase; the information would be required to understand/resolve defects regardless of whether a defect classification is applied or not. This matched our requirements for LiDeC in that we – in order to minimize the process foot-print – intended to capture what was currently tacit knowledge in the defect analysis process.

2.3 Study context – Automotive Software Development

The case-study presented in this paper was conducted at the department developing active safety features – such as collision warning, lane departure warning and driver alert control – at Volvo Car Corporation (VCC). In the following subsection we describe the development of software for such features at VCC.

2.3.1 Development process

The development process of active safety features at VCC [25], [26] can at a high level of abstraction be visualized by the V-model [27]. As shown by the left leg in Figure 2: product requirements are specified on vehicle-level and then refined through the development process into (sub-)system requirements and design. The system specifications are further refined into requirements and design of the individual hardware and software components that will realize them.

The bottom of Figure 2 shows the implementation of the components which is often done by suppliers; VCC commissions a component from a supplier based on requirement and design specifications. As VCC may have limited insight or control over the implementation phase – the in-house development activities are to an extent limited to design, specification and integration testing – applying defect classification schemes that have code focus consequently presents a challenge.

Furthermore, the test phases are shown by the right leg in Figure 2: components delivered by the suppliers are tested on unit level, subsystems are integrated and tested and finally the whole car is tested; it is in this phase that defects are reported.

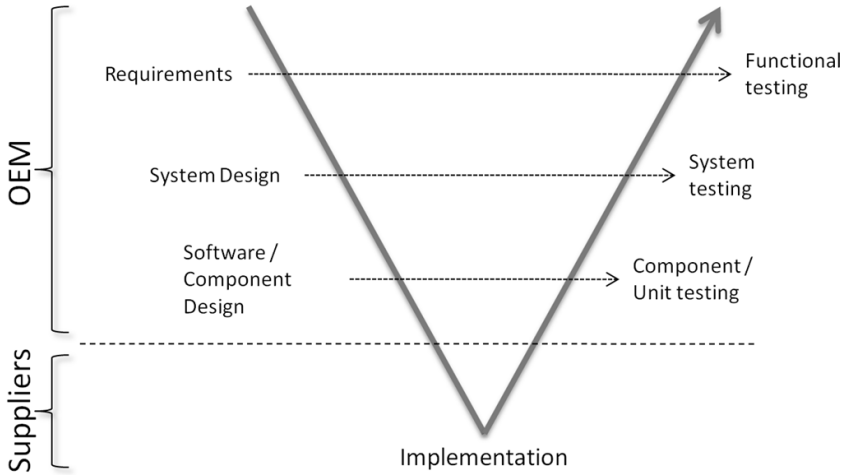

Figure 2. The V-model

Even though the overarching development process can be visualized by the V-model, in practice the process is better described as a federated development process [28]. As shown in Figure 3 development is iterative in three stages. In the first stage, corresponding to "*System Design*" in Figure 2, a system – e.g. collision detection or driver alert control – is designed and specified. The main focus of this stage is

to develop algorithms that fulfil the high-level requirements. In addition to system requirements and design, the results from this stage may include executable models (e.g. Simulink models) that can be validated in simulated environments, e.g. using a test rig, or cars equipped with simulated hardware (e.g. dSPACE[2]).

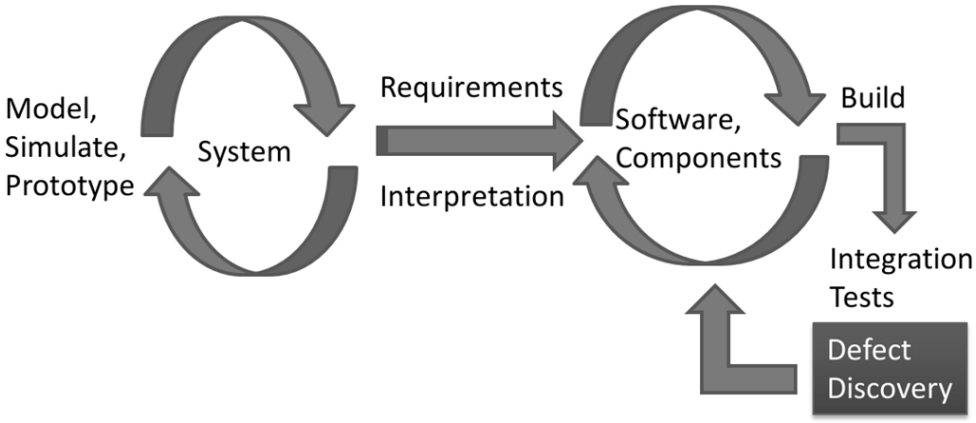

Figure 3.Federated development process (adapted from [28])

The second stage of development is shown in Figure 3 as "*Software, Component*" and corresponds to "*Software / Component Design*" in Figure 2. The focus of this stage is to decompose a system into individual software and hardware components that realize the system. The result of this stage is a set of component specifications that are used as base when commissioning components from suppliers.

In the final loop in Figure 3 – corresponding to the right leg of Figure 2 – software testing is done. As shown in Figure 3, the component design and test stages are intertwined. In practice, this means that the suppliers are involved from an early stage delivering a number of revisions of the components; each revision is tested by VCC, defects are discovered and corrected, and a new iteration is started. More specifically, the testing procedure done by VCC at each iteration of the third loop can be separated in three categories:

- *Component/Unit tests*. The algorithms – often in the form of executable models [29] – are tested on unit level before being provided to the supplier. The supplier provides VCC with an implementation in the form of a component (e.g. optimized binary software component, or a hardware component with the software installed). VCC verifies that the component complies with the requirements;
- *System tests*. System tests are done on simulations of the system on a test rig using recorded data. The focus of this phase is on initial integration testing;
- *Functional tests*. Finally, function tests are done on builds of the system in a real car. Initial functional tests are run on test tracks, while in later project phases expeditions on roads are done. The focus of this test phase is on the whole vehicle, i.e. that high-level requirements are met (e.g. that features behave as intended from an end-user perspective)

In addition to the test activities done by VCC, suppliers conduct units test which may not be reported to the OEM.

In this case-study the focus was on defects discovered during the last two stages in Figure 3 (indicated in the figure by "*Defect Discovery*"). The reasons for this delimitation include the challenges associated with supplier implemented software – it is in this stage that the suppliers get involved in the development. Furthermore, initial analysis of the defect inflow – shown in Figure 4 as the defect backlog[3] (number of open defects over time) – revealed that there was a considerable spike in defects during the component development and integration testing phase (the start of which is shown in the figure as "*Software Phase*"). The increasing inflow of defects is expected as testing of supplier developed hardware and software – specifically integration testing – is conducted during this phase.

[2] http://www.dspace.com

[3] The total number of defects has been scaled to 100 and the time scale has been removed due to confidentiality reasons. In addition, the time scale has been cropped (indicated by the ellipsis in the star and end of the curve) and does therefore not include the last phase leading up to start of production.

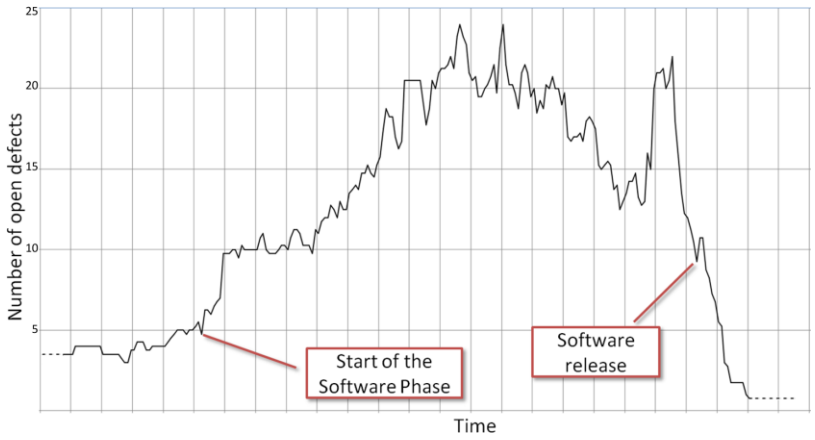

Figure 4. Defect backlog from the studied project

However, the increase in open defects (i.e. unresolved defects) during the software phase, and especially the peak close to software release (a major in-development milestone), raised the interest of our case company. Therefore, in the evaluation of LiDeC we analysed a sample of defects from the last peak shown in Figure 4.

3 METHOD

The research presented in this paper followed the case-study method described by Yin [30]. The case-study method was considered appropriate as the applicability of the adapted classification scheme was of importance; adapting a classification scheme in the same context as it will be deployed would increase the chances of it being useful in that context. Specifically, we have used a single-case design by applying the classification scheme to defect reports from one project at our case company. The rationale of this was that the defect inflow profile from the project (shown in Figure 4) was considered representative by the developers – similar inflow profiles had been observed in other projects. Using the defect reports from the project, we developed an adapted defect classification scheme; in particular, we addressed the following main research question:

RQ: How to efficiently support defect identification and resolution time by classifying in-process defects?

As the development context under study have specific properties (described in section 2.3), the main research question was broken down into:

RQ 1 In an automotive safety feature development context where source-code is often not available, how can a standard defect classification scheme be suitably adapted?

RQ 2 As defect classification may often be considered an administrative task, how can the adaptation of a standard defect classification scheme be done to minimize required learning and classification time?

As a practical guideline to adapting a defect classification scheme we followed [10]. The study was conducted in the following three stages:

Stage 1: Establish terminology. The aim of the first stage of the study was to establish a set of classification attributes using terminology aligned with the case company.

As a base for developing the classification scheme we used the IEEE Std. 1044 [7] and its guide [12]. In this stage we began by choosing attributes and the set of values available for each attribute from the IEEE Std. 1044 that we – based on our previous research [25], [29], [31] – found relevant for the specific development context at the company; for instance attributes related to customer value was not considered relevant for the development phase under study.

During two one hour-long interviews, we explained the initial classification attributes to the interviewee (project leader), and asked the interviewee to relate these attributes to the case company. We took notes during these interviews and refined the classification scheme according to these notes.

Stage 2: Tune feasibility. The second stage of the case-study aimed at streamlining the set of values each attribute could be assigned.

The stage consisted of a two hour long interview with a developer in which a number of defects were classified. The set of values available for each attribute was evaluated during the classification session, where for each attribute:

i. If an attribute value was never used and the interviewee could not think of an example when the value would be used, the value was considered for removal;
ii. If the interviewee did not consider any of the available values described the defect sufficiently, a new value was considered for addition.

As a result the final classification scheme was defined and depicted in the form of flowcharts with short questions providing a guide to arrive at the correct attribute value. Each attribute value was provided with a short illustrative example.

Stage 3: Evaluate scheme. The final stage of the study aimed at evaluating the efficiency and effectiveness of the classification scheme.

In this stage defect reports were classified according to the scheme. Four subjects involved in the project participated in six separate two hour long classification sessions (two subjects participated in two consecutive classification sessions). Three of the subjects were not involved in the previous two stages of the project.

The project used as case had finished one year prior to the study and contained over 100 problem reports[4]. All subjects involved in the study had been part of the project with the following roles: two developers, one tester and one project leader.

During the third stage of the study we were able to classify 22 defects. Of the 22 defects 12 were randomly selected from the last peak (as shown in Figure 4) and the remaining 10 from the rest of the project. This selection was done because of an expressed interest by members of the project to gain more insights in the defect peak. The results of the evaluation are reported in [13].

3.1 Validity Evaluation

We have identified and grouped the threats to validity in our study according to recommendations of Yin [30]:

- *Construct validity*– By basing our defect classification scheme on the IEEE Std. 1044 and by keeping careful notes on how to map concepts specific to our case to the standard, we consider that the threat to construct validity to have been minimized. Furthermore, as both the adaptation of the classification schemes and the evaluation was done using real defect data from an industrial project with the assistance of the developers involved in the project, we consider the threat to construct validity to have been further reduced.
- *Internal validity*– As any interview study we anticipated some personal bias in the answers from the interviewees. In order to minimize this threat we triangulated the results by including multiple subjects in our interviews. In addition, a set of defects were classified by multiple subjects thereby allowing evaluation of the repeatability of the classification scheme; section 4.2 reports the results from this evaluation.
- *External validity*– There is a risk that the results are too specific to Volvo Car Corporation. However, as we documented and justified the modifications done to the IEEE Std. 1044 as well as described the particular development context of our case, we believe that our results can be generalized to similar contexts outside our specific case. Moreover, we consider the mapping between attributes of the classification schemes provided by Freimut [10] to contribute to the generalizability of our classification scheme; e.g. the mapping between classification schemes enables analysis methods utilized with other schemes to be applicable to LiDeC as well, and therefore we believe that results are also comparable.

[4] Exact number cannot be disclosed due to confidentiality reasons

– *Reliability*– As part of the case study design, we have created a case study protocol which ensured that we conducted the study and collected the data in a consistent manner. By using this protocol, we believe that the study can be reliably reproduced.

4 RESULTS

The main challenge of adapting the IEEE Std. 1044 included tailoring the attributes relating to the fault and its resolution; specifically attributes in the phases *Investigation*, *Action* and, *Impact Identification* as the attributes in these phases have a strong focus on source-code aspects.

The results are reported below in two parts; first, the classification scheme is presented, and second, a comparison with the IEEE Std. 1044 is presented; for results from the initial industry evaluation of LiDeC, see [13].

4.1 LiDeC

LiDeC captures – as shown by the scheme overview in Figure 5 – attributes from four phases of the defect life-cycle [12] (described in more detail below as well as in Appendix A and Appendix B): the first phase captures information about the recognition of the defect, i.e. observing a deviation (failure) from intended or specified requirement [32]; the second phase captures information about the underlying cause of the defect (referred to as *Investigation* in [12]); in phase three information about the defect resolution is captured (referred to as *Action* in [12]); and the last phase captures information about what was actually done about the defect (referred to as *Disposition* in [12]. Table 3 shows a comparison between the life-cycle phases of IEEE Std. 1044, ODC and LiDeC.

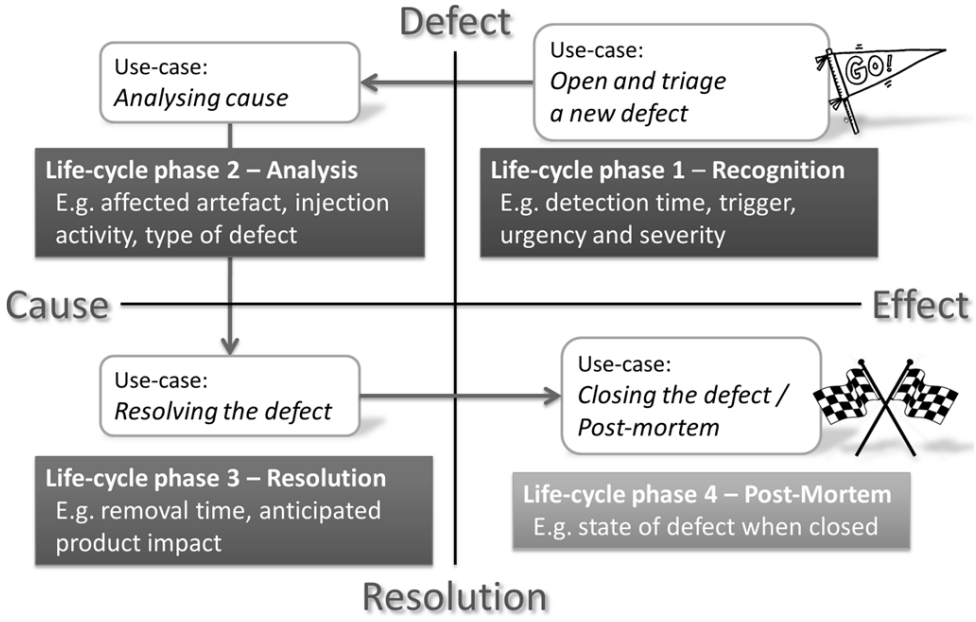

Figure 5. Overview of the LiDeC Scheme

TABLE 3 MAPPING OF LIFE-CYCLE PHASES

Defect life-cycle phase	IEEE 1044	ODC	LiDeC
1	Recognition	Open	Recognition
2	Investigation		Analysis
3	Action	Close	Resolution
4	Impact Identification		
5	Disposition		Post-mortem

As can be seen by the number of attributes in each phase (described in tables Table 4–Table 7 in the subsections below), the main focus of LiDeC is on recognition and analysis of a defect. The justification is that the later a defect is discovered the more costly its resolution tend to be [33]. In addition, as implementation is done mainly by suppliers, the most promising areas of process improvement lies in more efficient verification and validation. Consequently, the main focus of the classification scheme is on how defects are discovered, how the product is affected by them, and what types of defects they are. Analysing this information will contribute to understanding which phases of the development process contain the most improvement potential.

The phases of the classification scheme are aligned with the defect management process at the case company and directly correspond to the states a defect can be assigned: recognized, analysed, resolution proposed and post-mortem. The following sections describe the attributes of each phase. Appendix A provides an exhaustive list of attributes and their description and Appendix B provides a classification guide that was used in the case-study. The classification guide in Appendix B contains a more detailed description of each attribute value along with typical examples expressed in the terminology of the company; the purpose is to maintaining consistency of the classification over time and between reporters. In addition, Appendix C contains classification examples.

4.1.1 Recognition

The attributes in the first phase of the defect life-cycle (shown in Table 4) relate to data about the discovery of a failure and its effects on the system in question, i.e. the manifestation of the defect. The attributes capture project related information:

- Timestamp of detection
- Resolution urgency
- End-user perceived severity of the defect, and
- How the defect affects the product, including whether ASIL requirements are affected (ISO/IEC 26262 [34]).

This information will be used in analysis, for example, to assess how timely the most serious defects are detected, or which activities are more effective in detecting defects.

TABLE 4 SUMMARY OF ATTRIBUTES IN THE RECOGNITION PHASE

Attribute	Question	Values
Timing / Detection	When was the defect detected?	Date and project phase
Timing / Preferred	When should the defect have been detected (subjective)?	Project phase if different from Timing / Detection
Affects S/W	Does the defect affect software?	Yes / No
Detection Activity	What was done to detect the defect?	Inspection/Requirements, Inspection/Design, Unit test/In-house, Unit test/Supplier, System test/bench, Functional test/Test track, Functional test/Expedition Production/Manufacturing Production/Customer report
Urgency	How urgently does the defect need to be addressed?	Immediately, Next development release, Before start of production, Deferrable
Severity	How severe is the defect with respect to product quality?	None, Nuisance, Limited Functionality, Show-stopper
Effect	How does the defect primarily affect the product?	Capability/Undesired activation, Capability/Inactive on true positive, Capability/Other, Function Safety, Maintainability, Usability, Testability, Configurability
Functional Safety Impact	Does the defect have impact on a software compontent with ASIL-classified requirements?	Yes, No

4.1.2 Analysis

The attributes in the second phase of the defect life-cycle (shown in Table 5) aim at capturing data about the cause of the failure; e.g. in what work product and product component contained the defect, what type of defect it was, and which process step caused the problem.

TABLE 5 SUMMARY OF ATTRIBUTES IN THE ANALYSIS PHASE

Attribute	Question	Values
Artefact	Which software work product contained the defect?	Req./Internal, Req./Cross-function, Req./External, Design model, Impl./Executable model, Impl./Code, Impl./Configuration params, Tool
Injection activity	When was the defect injected?	Specification, Design, Impl./In-house modelling, Impl./Suppl. mdl. transform., Impl./Supplier coding, Configuration
Component / Asset	Which design component contained the defect?	Internal (product) module name also identifying its version
Type	What type of defect was it?	Description, Data, Interface / Timing, Logic / Algorithm, Tooling, Tuning

4.1.3 Resolution

The attributes in the third phase of the defect life-cycle (shown in Table 6) aim at capturing data about the proposed resolution. As implementation specific details of the resolution may not be available, the attributes in this phase focus on capturing the cost of resolving the defect in terms of development effort. More specifically, to capture what impact a resolution would have on the product and on the process; the impact on the product is captured in terms of how much of the product would be affected by the modification, and impact on the process in terms of amount of regression testing needed.

TABLE 6 SUMMARY OF ATTRIBUTES IN THE RESOLUTION PHASE

Attribute	Question	Values
Removal time	When was the defect report closed?	Date and project phase
Product impact	What would the impact of a proper resolution be on the product?	None, Local (unit) modification, Multiple components, Funct. changes (re-design)
Required Verification Level	What level of regression testing would a proper resolution require?	None, Inspection, Unit test, System test, Expedition

Furthermore, the attributes of the *Resolution* phase capture data about a proper resolution of the defect. In practice, a defect could be resolved by means of workarounds (this data is captured in the final phase of LiDeC).

4.1.4 Post-mortem

In the last phase of the defect life-cycle the single attribute (shown in Table 7) records what was finally done to close the defect; to what extent the defect was resolved.

TABLE 7 SUMMARY OF ATTRIBUTES IN THE POST-MORTEM PHASE

Attribute	Question	Values
Resolution state	What was the final state of the defect when the problem report was closed?	Corrected (proper resolution applied), Workaround/Fix, Workaround/Product de-scoped, No Action/Deferred, No Action/Referred, No Action/Not found, No Action/No action

4.2 Comparison with IEEE Std. 1044

In the process of adapting IEEE Std. 1044 compliance with the standard was considered an important requirement (Appendix D shows the compliance matrix as proposed in [12]). Retaining compliance with the standard contributes to the generalizability of the results – e.g. data collected with LiDeC and analyses conducted on that data should be comparable with IEEE Std. 1044 compliant data from other companies.

The main differences between the IEEE Std. 1044 and LiDeC are described below. The full mapping of attributes between the IEEE Std. 1044 and LiDeC is presented in Appendix E.

4.2.1 General Modifications

The main adaptation made to LiDeC consists of raising the abstraction level of the attributes, i.e. choosing attribute values that are less fine-grained than their IEEE Std. 1044 counterpart. Furthermore, the set of values available for each attribute expressed in the terminology of the company and provided with examples; for example the attribute "*Type*" has been shown to be problematic (e.g. "*to me everything is a logic problem*" [8]); in LiDeC typical examples for each available value (see for instance Figure 15 in Appendix B) are provided.

Furthermore, attribute values that in IEEE Std. 1044 consisted of "Low, Medium and High" (e.g. *Project Risk* and *Priority*) has been replaced by more descriptive values (see, for instance, the LiDeC attributes *Urgency* and *Severity* in Appendix A and figures Figure 10 and Figure 11 Appendix B), also contributing to making values less ambiguous by limiting the amount of interpretation needed by the reporter.

Moreover, supporting data items from the standard, e.g. cost and time estimations, defect reporter, developer assigned to the defect) has been omitted from LiDeC unless they have a specific purpose related to the analysis of the defect data (e.g. the LiDeC attribute *Component/Asset* is originally a supporting data item in the IEEE Std. 1044 *Action* phase). Such supporting data items, however, are assumed to be part of the company's normal issue tracking process as needed.

4.2.2 *Added Attributes*

The attributes described in the following subsections were added in LiDeC.

4.2.2.1 Timing/Preferred

The attribute was added to the *Recognition* phase in order to capture – at the time of detection – the reporter's subjective opinion of whether there was a previous test phase in which this type of failure should have been uncovered. The intention of the attribute is to be able to gauge the fault-slip-through rates of the test activities.

4.2.2.2 Affects Software

The attribute was added to the *Recognition* phase in order to capture whether the defect has an impact on software. As the products developed are software intensive mechatronic systems, there may be defects

that are not related to software; the purpose of the attribute is thus to be able to filter defects based on whether they affect the software. The definition of whether a defect affects software, however, is broad; see the example given in the classification guide in Figure 8 in Appendix B.

4.2.2.3 Component/Asset

The attribute was added to the *Analysis* phase in order to be able to evaluate the distribution of faults among components in the system. The attribute is originally part of the supporting data items of the IEEE Std. 1044 *Action* phase – thus whereas the IEEE Std. 1044 records the component(s) in need of modification, LiDeC records the component(s) containing the fault. This redefinition was made as the main focus of LiDeC is on capturing data about the defects rather than their solution; as the majority of implementation is done by suppliers, it is of more interest to the company to identify which components contain the defects rather than which components need modification (e.g. a workaround may require modifications to other components than the one containing the defect).

4.2.2.4 Removal Time

The attribute *Removal Time* was to the *Resolution* phase added in order to allow evaluation of defect longevity. The attribute is originally part of the supporting data items in the IEEE Std. 1044 *Action* phase.

4.2.3 Omitted Attributes

In this section the attributes available in the IEEE Std. 1044 that were omitted in LiDeC are listed. All omitted attributes are listed as optional in IEEE Std. 1044.

4.2.3.1 Suspected cause

While the *Suspected cause* attribute may provide valuable input during the process of analysis a failure, it was not considered important from LiDeC's point of view. LiDeC aims at capturing attributes of the defect itself, rather than speculations done as part of the defect analysis process.

4.2.3.2 Repeatability

The *Repeatability* attribute was omitted (as a separate attribute) in the LiDeC for the same reasons as the *Suspected cause* attribute (see 4.2.3.1). In LiDeC, the attribute is instead partly represented as an attribute value in the *Disposition* attribute of the *Post-mortem* phase; non-repeatable defects would be reported as *No action / Not found*.

4.2.3.3 Corrective action

The IEEE Std. 1044 attribute *Corrective action* records detailed data about what action was taken in order to resolve the issue. The attribute values proposed by the standard – such as revising the developing process or implementing a training program – would not generally be applicable to the context of VCC based on a single observed defect; rather such actions would instead be taken on the basis of analysing defect data from a number of projects.

Furthermore, the exact action taken may not easily be identified as a substantial amount of defects relate to code, and code is generally produced by suppliers. Instead, LiDeC captures data about the estimated repercussions of the corrective action(s) by the attributes *Resolution Impact* and *Required Verification Level*.

4.2.3.4 Customer value, *Mission/safety* and *Project risk*

As the roles responsible for reporting defects may not have the necessary insight in, for instance, product planning the (explicit) attributes *Customer value* and *Mission/Safety* were omitted. Instead the data are in LiDeC implicit in the attribute *Severity* – i.e. how severely the defect affects the product from a customer perspective.

The attribute *Project risk* was omitted in LiDeC as a defect reporter may not have necessary knowledge of project planning (defects may be reported by suppliers, as well as various in-house testers) to be able to assess project risk. The risk can, however, be assessed by analyzing the LiDeC attributes *Resolution impact* and *Required verification level* together with *Severity* and *Time of detection*; late sever defects that have a large impact on the product and/or require the more costly verification types would constitute a higher risk.

4.2.3.5 Project quality/reliability

The intention of the IEEE Std. 1044 attribute *Project quality/reliability* is to appraise the impact on the project quality if a defect is addressed. The attribute was omitted in LiDeC as the data was not considered to be relevant to study context – the subjects in the case-study had difficulty relating the

attribute to the defects analysed, indicating that the roles responsible for defect reporting may not have the necessary insight in project planning to be able to reliably assess the attribute.

The LiDeC field *Required Verification Level*, however, captures data with similar intentions: when addressing the defect how much additional re-verification effort would be required (also see *4.2.4.7 Project schedule* below).

4.2.4 Redefined Attributes

A number of IEEE Std. 1044 attributes have in LiDeC been redefined. The redefinitions have been done with care to retain the intention of the original attribute, but adapted to the context of VCC; e.g. by changing the available attribute values (for instance the *Symptom* attribute) or by using a different measurement unit than the IEEE attribute originally specified (for instance the *Project cost* attribute).

The following subsections describe the attributes that have been redefined.

4.2.4.1 Symptom

The IEEE Std. 1044 attribute *Symptom* has been redefined in LiDeC in that, whereas the original attribute captured more detailed data about the behaviour of the system, the LiDeC attribute *Effect* captures less fine-grained data about what quality aspect of the product – i.e. the product's "-ablities" – is affected.

Furthermore, the attribute values chosen for the *Effect* attribute has been tailored specifically for the context of active safety systems; e.g. the *Capability* attribute values (which would correspond to the values available in the *Symptom* attribute) has been subdivided into the two categories that are most relevant (the function triggering on a false positive and the function being inactive despite a true positive) as well as a catch-all third capability related value, see Figure 12 in Appendix B.

4.2.4.2 Product Status and *Severity*

Both the IEEE Std. 1044 attributes *Product status* and *Severity* are included in the LiDeC attribute *Severity*. Whereas the IEEE attribute *Severity* captures the severity of the fault, the LiDeC attribute *Severity* captures the severity of the failure (i.e. the manifestation of the fault) – as does the IEEE *Product Status* attribute. LiDeC's focus on the failure aspect of a defect rather than the fault causing it is due to the limited in-house implementation done, and the specific interest in the ability to evaluate test activities.

Furthermore, the attribute values available in the LiDeC attribute *Severity* have been defined to describe the impact of the failure in more objective terms than the original values proposed by the IEEE Std. 1044 (which are "Urgent, High, Medium, Low and None), see Figure 11 in Appendix B.

4.2.4.3 Societal

The attribute *Societal* is in LiDeC covered by the attribute *Functional Safety Impact*. Whereas the IEEE Std. 1044 does not define the *Societal* attribute clearly, in the attribute Functional Safety Impact is specific with respect to defects that may cause harm. In particular, LiDeC attribute captures whether the defect have impact on a software component that has ASIL-classified requirements (as defined by ISO/IEC 26262 [34]).

4.2.4.4 Actual cause

Whereas the IEEE Std. 1044 captures data about the artefact that caused the defect, LiDeC captures data about in which project phase the fault was injected. This redefinition was made as it was, in the case-study, found that the attributes *Actual cause* and *Artefact* was treated identically. By referring to the activity causing the defect it was clearer to the subject how to use the attribute (see Example 2 Appendix C).

4.2.4.5 Type

The IEEE Std. 1044 attribute *Type* has a strong focus on source code, and provides a detailed set of attribute values. In the LiDeC case-study the attribute was found difficult to assign of two main reasons: i) as the access to source-code may be limited, identifying detailed data about type of defect is often not possible, ii) the detailed set of values proposed by the IEEE Std. 1044 were shown to make distinction between values difficult (this is also corroborated in [8] by the quote "to me everything is a logic problem").

The approach taken in LiDeC is to substantially reduce the resolution in the available attribute values, and to provide each attribute value with a typical example in the terminology of the company (see Appendix A for a description of the LiDeC attribute and Figure 15 in Appendix B for examples of each attribute value).

4.2.4.6 *Resolution* and *Priority*

The IEEE Std. 1044 attribute *Resolution* captures data about both the urgency of the resolution and what type of resolution that will be applied [12]. In LiDeC, the urgency of resolving a defect is captured by the *Urgency* attribute while data about the type of resolution is not explicitly captured by LiDeC (instead, it is partly covered by the *Disposition* attribute).

Furthermore, whereas the IEEE Std. 1044 *Resolution* attribute relates to the urgency of applying a specific resolution, the LiDeC attribute *Urgency* relates to the urgency of removing a failure from the system; thus, the focus is on detecting and prioritizing the removal of the manifestation of defects rather than on details of the actual resolution (as the resolution may be developed and applied by the supplier, the OEM may not have the necessary insight).

4.2.4.7 *Project schedule*

The IEEE Std. 1044 attribute *Project schedule* aims at capturing data about the direct impact of the resolution on the project schedule. In LiDeC this is instead captured in terms of amount of re-testing needed after applying a resolution; as verification activities constitutes a substantial amount of the development efforts, the attribute captured data with the same intention as the *Project schedule* attribute. Furthermore, the amount re-testing needed for a resolution is more straight-forward to assess for an engineer reporting the defect than objectively estimating the impact on project schedule.

4.2.4.8 *Project cost*

Whereas IEEE Std. 1044 attribute *Project cost* captures data about the cost of a resolution in terms of real money, LiDeC instead makes the estimate in terms of how much of the product will be affected by the resolution in the attribute *Product Impact* – the assumption is that the more of the product that is affected the more expensive the resolution will be. This redefinition was made as the engineer reporting a defect may not have sufficient insight into the budget or may have limited ability to make a reliable estimation of the cost of applying a resolution. Thus, LiDeC captures data with the same intention as the *Project cost* attribute but in more objective engineering terms.

5 CONCLUSIONS

In this report we have described the adaptation of IEEE Std. 1044, the IEEE standard for defect classification, to the automotive safety feature development context. The specific properties of the development context that influenced the adaptation include a strong reliance on supplier side implementation, which may limit the access and insight into the source-code. Furthermore, as defect classification does not directly contribute to the development of the end-product, it was considered important to adapt the classification scheme to minimize the time required to use the scheme while still providing the additional benefits of characterizing the defects. More specifically, we have addressed the research questions:

RQ 1 *In an automotive safety feature development context where source-code is often not available, how can a standard defect classification scheme be suitably adapted?*

RQ 2 *As defect classification may often be considered an administrative task, how can the adaptation of a standard defect classification scheme be done to minimize required learning and classification time?*

We addressed RQ 1 by:

- Shifting the focus of the classification scheme from detailed aspects of the fault and its resolution to aspects of the discovery of the defect. As the implementation is mainly done by suppliers, most in-house process improvement potential lies in more efficient defect discovery activities. LiDeC reflects this by providing more detailed attributes in the *Recognition* phase (e.g. *Detection activity*, *Urgency*, *Severity* and *Effect*), while granularity of the attributes in subsequent phases have been reduced; e.g. the *Type* attribute is less granular than in IEEE Std. 1044 and detailed aspects of the resolution (captured by the IEEE attribute *Resolution*) has been omitted;
- Adapting attributes for safety specific purposes. In LiDeC the attribute *Functional Safety Impact* (which maps to the IEEE attribute *Societal*) records whether a defect impacts ASIL-classified requirements [REF: ISO 26262]. In addition, the values of the Effect attribute (which maps to the IEEE *Symptom* attribute) was adapted specifically to provide a high-level characterization of safety feature problems (e.g. unintentional activation of a feature).

We addressed RQ 2 by:

- Raising the level of abstraction of attributes. For instance, by providing only higher-level categories as values for the Type attribute
- Providing more descriptive attribute values (Low, medium high)
- Providing attribute descriptions and values phrased using the terminology of the company
- Providing a classification guide with a flow-chart structure, and including typical examples for each attribute value
- Streamlined the attributes by removed or redefining attributes that required insights that the typical reported may not have (project schedule and risk). Attributes were redefined with care to retain the intentions of the original attribute but measured in more specific engineering terms; for instance, whereas the IEEE Std. 1044 attribute *Project schedule* aimed at appraising the direct impact on the project plan, LiDeC instead appraises the estimated amount of re-verification (an activity that may have a large impact on the project schedule).

In conclusion, by putting focus on capturing data about the discovery of defects, streamlining the available attributes with respect to the expertise of the engineers reporting defects, and conforming to the terminology of the company, we have developed an IEEE Std. 1044 compliant classification scheme well suited to the development of automotive safety features at VCC. As a result, we are now in the process of incorporating LiDeC into the issue tracking system of the company.

6 Future Work

The next step in our research concerns analysis recipes – guidelines on how to analyse the collected data. We envision two distinct types of analysis recipes: post-mortem and in-process – post-mortem recipes will mainly be support for organizational learning, whereas in-process would serve as a tool for project control.

Post-mortem recipes would concentrate on analysing the collected data in order to learn about a finished project; e.g. evaluating whether changes in the way of working in the project yielded any noticeable effects, or analysing weak spots in the way of working as promising candidates for future improvements.

In contrast, in-process recipes would be guidelines on how to use collected defect data from previous project phases in order to predict future phases within the project. As part of this we will need to identify relevant predictors. The challenges include the absence of source-code predictors; instead there is a need to identify predictors based on specification and design artefacts, e.g. requirements and design model complexity.

By establishing a defect profile baseline per development phase, we aim at developing a way to predict future defect inflow. Such a prediction model would, for instance, assist in resource planning in a similar way as presented by Bijsma et al. [35] where mainly source-code related predictors were used to estimate the time it would take to resolve a defect – a metric that can be used to assist in prioritizing defects.

Acknowledgements

This research is partially sponsored by The Swedish Governmental Agency for Innovative Systems (VINNOVA) under the Intelligent Vehicle Safety Systems (IVSS) programme. In addition, the authors would like to thank all the people at VCC who participated in the study, and people who generously took the time to review this manuscript.

REFERENCES

[1] CMMI Product Team, "CMMI for Acquisition," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-2010-TR-032, 2010.

[2] The SPICE User Group, "Automotive SPICE," *Automotive SPICE*, 20-Jun-2011. [Online]. Available: http://www.automotivespice.com/. [Accessed: 20-Jun-2011].

[3] B. Beizer, *Software Testing Techniques, 2nd Edition*, 2 Sub. Intl Thomson Computer Pr (T), 1990.

[4] M. Leszak, D. E. Perry, and D. Stoll, "A case study in root cause defect analysis," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, 2000, pp. 428–437.

[5] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *Software Engineering, IEEE Transactions on*, vol. 18, no. 11, pp. 943–956, 1992.

[6] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992.

[7] "IEEE Std. 1044-2009. Standard Classification for Software Anomalies." IEEE, 2010.

[8] B. Freimut, C. Denger, and M. Ketterer, "An industrial case study of implementing and validating defect classification for process improvement and quality management," in *Software Metrics, 2005. 11th IEEE International Symposium*, 2005, p. 10 pp.–19.

[9] S. Wagner, "Defect classification and defect types revisited," in *Proceedings of the 2008 workshop on Defects in large software systems*, New York, NY, USA, 2008, pp. 39–40.

[10] B. Freimut, "Developing and using defect classification schemes," Fraunhofer IESE, IESE- Report No. 072.01/E, Sep. 2001.

[11] "IEEE Std. 1044-1993 Standard Classification for Software Anomalies." IEEE, 1994.

[12] "IEEE Std. 1044.1-1995 -- Guide to Classification for Software Anomalies." IEEE, 1996.

[13] Niklas Mellegård, Miroslaw Staron, and Fredrik Törner, "A Light-weight Defect Classification Scheme for Embedded Automotive Software and its Initial Evaluation," presented at the Manuscript submitted for publication in IEEE International Symposium on Software Reliability Engineering, Dallas, Tx USA, 2012.

[14] J. Li, T. Stalhane, R. Conradi, and J. M. W. Kristiansen, "Enhancing Defect Tracking Systems to Facilitate Software Quality Improvement," *Software, IEEE*, vol. 29, no. 2, pp. 59 –66, Apr. 2012.

[15] A. Dubey, "Towards adopting ODC in automation application development projects," in *Proceedings of the 5th India Software Engineering Conference*, New York, NY, USA, 2012, pp. 153–156.

[16] M. Leszak, D. E. Perry, and D. Stoll, "Classification and evaluation of defects in a project retrospective.," *Journal of Systems and Software*, vol. 61, no. 3, pp. 173–187, 2002.

[17] Y. C. Cavalcanti, P. A. Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. Almeida, and S. R. Lemos Meira, "The bug report duplication problem: an exploratory study," *Software Quality Journal*, Oct. 2011.

[18] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 1st ed. Addison-Wesley Professional, 1995.

[19] A. Wood, "Software reliability growth models," Tandem Computers Inc.,, Technical Report 96.1, 1996.

[20] J. Ploski, M. Rohr, P. Schwenkenberg, and W. Hasselbring, "Research issues in software fault categorization," *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 6, Nov. 2007.

[21] D. Liparas, L. Angelis, and R. Feldt, "Applying the Mahalanobis-Taguchi strategy for software defect diagnosis," *Automated Software Engineering*, vol. 19, no. 2, pp. 141–165, Jul. 2011.

[22] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A Systematic Review of Fault Prediction Performance in Software Engineering," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 0.

[23] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1868–1882, Nov. 2008.

[24] R. Chillarege and K. Ram Prasad, "Test and development process retrospective - a case study using ODC triggers," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 669–678.

[25] N. Mellegård and Miroslaw Staron, “Use of Models in Automotive Software Development: A Case Study,” in *The First Workshop on Model Based Engineering for Embedded Systems Design*, Dresden, Germany, 2010.
[26] N. Mellegård, “Method and Tool Support for Automotive Software Engineering,” Licentiate Thesis, No 74L, Chalmers University of Technology, 2010.
[27] S. L. Pfleeger, *Software Engineering: Theory and Practice*, 2nd ed. Prentice Hall, 2001.
[28] T. L. Bennet and Paul W. Wennberg, “Eliminating embedded software defects prior to integration test,” *CrossTalk*, vol. 18, no. 12, pp. 13–18, 2005.
[29] N. Mellegård and M. Staron, “Characterizing Model Usage in Embedded Software Engineering: A Case Study,” in *8th Nordic Workshop on Model Driven Software Engineering (NW-MoDE)*, Copenhagen, Denmark, 2010.
[30] R. K. Yin, *Case Study Research: Design and Methods, Third Edition, Applied Social Research Methods Series, Vol 5*, 3rd ed. Sage Publications, Inc, 2002.
[31] N. Mellegård and M. Staron, “Distribution of Effort Among Software Development Artefacts: An Initial Case Study,” in *Exploring Modelling Methods for Systems Analysis and Design*, Hammamet, Tunisia, 2010, vol. LNCS.
[32] “ISO 9000:2005 — Quality management systems - Fundamentals and vocabulary,” International Organization for Standardization / Technical Committee 176 (ISO/TC 176), Geneva, Switzerland, 2005.
[33] B. W. Boehm, *Software Engineering Economics*, 1st ed. Prentice Hall, 1981.
[34] “ISO/DIS 26262 - Road vehicles — Functional safety,” International Organization for Standardization / Technical Committee 22 (ISO/TC 22), Geneva, Switzerland, Jul. 2009.
[35] D. Bijlsma, M. A. Ferreira, B. Luijten, and J. Visser, “Faster issue resolution with higher technical quality of software,” *Software Quality Journal*, May 2011.

Appendix A. LIDEC ATTRIBUTE DESCRIPTION

Life-cycle phase 1 – Recognition

TABLE 8 LIDEC SCHEME – ATTRIBUTES IN THE RECOGNITION PHASE

Attribute	Attribute Description	Values	Value Description
Timing/Detection [RTD]	When was the defect discovered?	Date *[RTD1]*	Date/project phase of detection
Timing/Prefered *[RTP]*	Was the defect discovered in the proper test phase according to the goals of the phase?	Yes *[RTP1]*	The defects discovery was timely ; there was no previous test phase in which the specified goal included detection of this type of defect
		No *[RTP2]*	This value is only provided if it is apparent that the defect should have been caught in an earlier test phase. Also specify which test phase, e.g. *E1-Ex, M, VP, TT*
Affect SW *[RAS]*	Does the defect affect software? "Affect" here is used to denote either defects that are caused by anomalies in the software or whose resolution have an impact on software – an example of the latter may be that dirt causes a sensor to degrade, but if the system fails to detect this (through diagnostic software) and notify the driver of degraded performance, it still affects software. This attribute will be used as a way to filter the defect reports as we are mainly interested in defects that are either caused by software or where the resolution may affect the software	Yes *[RAS1]*	The defect affects software
		No *[RAS2]*	The defect has no relationship with software at all, e.g. testing of vibration resilience of the physical component failed
Detection activity *[RDA]*	What was done when the defect was discovered? Detection activity captures what was done when the defect was detected.	Inspection / Requirements *[RDA11]*	The defect was detected during inspection of requirements specification
		Inspection / Design *[RDA12]*	The defect was detected during inspection of design specification, e.g FMEA
		Component test / VCC *[RDA21]*	The defect was detected while running a unit test in-house, e.g. unit testing of an isolated component using a SimuLink model.
		Component test / Supplier *[RDA22]*	D.o but the defect was detected by the supplier
		System test / System-bench *[RDA31]*	The defect was detected while runing an integration test (multiple cooperating components realizing functionality) on a simulation of the target platform done in-house. As "simulation" of the target platform are considered "box-car" (early E-series) as well as "mules" (M-series)

Attribute	Attribute Description	Values	Value Description
		System test / Whole car-bench *[RDA32]*	This refers to system testing done on system simulation on rigs with the whole electrical system present; though not necessarily with the final hardware present E.g bench tests with the whole electrical system
		Functional test / Test track *[RDA41]*	Functional testing of system using a car-build – a mule of a test build of the final hardware. Test-track refers to testing isolated scenarios on a test-track i.e.with real sensor input, but a simulated test-setup (e.g. with balloon cars or dummies)
		Functional test / Expedition *[RDA42]*	Functional testing of the whole car using a test-build – mule or the final hardware build. The defect was detected on an expedition with real data
		Production platform / Manufacturing *[RDA51]*	Defect was detected during manufacturing, e.g. calibration or configuration of a function, such as calibration of sensors in the production line
		Production platform / Customer reported *[RDA52]*	The defect was detected by customer post-release, e.g.car owner or maintanence staff
Urgency *[RU]*	How urgently does the defect need to be addressed? Denotes how urgent the defect needs to be removed from the product – thus urgency is related to the *project*. In late stages of the project defect will naturally be more urgent than in earlier stages. However, defects in early project phases that are blockers (i.e. blocking other functionality from being testable), should be considered urgent	Immediate *[RU1]*	The defect should be removed in the current development cycle; i.e. before the next development release (e.g. detected in E3.1 and should be removed in E3.2). E.g. defects that are blocking vital functionality should be classified as “immediate” (as they are inhibiting testing of that functionality)
		Next major release *[RU2]*	The defect should be removed before the next major development release. E.g. detected in E3.x and should be removed by E4.
		Before SoP *[RU3]*	The defect should be removed before the software is released. E.g. minor flaws or functionality that can be adjusted using tunable parameters, or documentation issues
		Deferrable *[RU4]*	Defects that are not considered to have much of an impact on product, and can be deferred until later versions or revisions of the product
Severity *[RS]*	How severly does the defect affect the product? Severity denotes the end-user perceived impact on the product if the defect is left in the released product – thus severity is related to the *product (i.e. vehicle)* not the *project*. Note, this attribute shall not consider the timing of the defect detection, i.e. regardless of when a defect is detected during the project it shall receive the same severity (whereas its Urgency may vary)	None *[RS1]*	The defect would not be noticed by the end-user
		Nuisance *[RS2]*	The defect would be limited to a nuisance for the end-user – though the product would still realize the full functional specification. E.g. a warning system would still be able to function in all scenarios originally specified, but may give an increase amount of false warnings
		Limited functionality *[RS3]*	The defect would limit the functionality of the product – e.g. the product would still function but not to the extent originally specified.
		Show-stopper *[RS4]*	A “show-stopping defect” is one that would prevent the product from being released; e.g. defect that would result in increased risk of injury, or that block other functions from performing according to specifications.

Attribute	Attribute Description	Values	Value Description
Effect *[RE]*	How does the defect primarily affect the product? Note that these may be overlapping to some extent but classification should be done on the effect, not the cause (as life-cycle phase 1 is focussed on the detection of defects) – e.g low performance of the software in a sensor may affect the functionality of the system, thus defect should be classified as 'Functionality'	Capability / Undesired activation *[RE11]*	The defect causes the function to trigger on a false positive
		Capability / Inactive despite True Positive *[RE12]*	The defect inhibits activation of functionality despite presence of a true positive
		Capability / Other capability related defect *[RE13]*	The defect affects the capability of the product; i.e. the product does not behave as intended or to the extent intended
		Maintainability *[RE4]*	The defect would affect the maintainability of the system; e.g. documentation issues, too complex design, cryptic internal error codes, wrong or missing diagnostic codes.
		Usability *[RE5]*	The defect affects the systems ease of use; e.g. complex user interface, missing or wrong visual cues to driver
		Configurability *[RE6]*	The defect affects configuration or calibration of the function; e.g. configuration of vehicle model variations or calibration of components during manufacturing
		Testability *[RE7]*	The defect affects the testability of the product; e.g. radar software the fails to detect a balloon car at the test site
Functional Safety Impact *[RFS]*	Does the defect have an impact on a software component with ASIL-classified requirements (IS 26262)?	Yes *[RFS1]*	The defect have an impact on ASIL-classified requiremetns according to the ISO 26262 standard [34].
		No *[RFS2]*	The defect does not affect ASIL-classified requirements.

Life-cycle phase 2 – Analysis

TABLE 9 LIDEC SCHEME – ATTRIBUTES IN THE ANALYSIS PHASE

Attribute	Attribute Description	Values	Value Description
Artefact [AA]	Which software work product contained the defect? This attribute relates to the work product in which the fault causing the failure was contained. Note that the underlying reason for introducing the fault may lie in another work product (see Injection activity)	Requirement / Internal [AA11]	Defect was contained in a requirement for the module itself
		Requirements / Internal cross-function *[AA12]*	The defect was contained in a requirement the module posed on an external module. E.g wrong required resolution posed on a sensor which is not part of the module itself.
		Requirements / External *[AA13]*	The defect was contained in a requirement posed on the module by another module. E.g an external module required wrong resolution of a sensor which is part of the module itself.
		Design model *[AA2]*	The defect was contained in a design model, e.g logical design (class diagram)
		Implementation / Executable model *[AA31]*	The defect was contained in a simulation model, e.g. Simulink
		Implementation / Code *[AA32]*	The defect was contained in code, either written in-house or by supplier, or code generated from models. Note, if code was correctly generated from a defective model, the defect should be classified as *[AA31]*
		Configuration Parameters *[AA33]*	The defect was contained in the tuning parameters for the function
		Tool *[AA4]*	Defect was contained in tools used during development; e.g. simulation environments for sensors
Injection activity *[AI]*	When was the defect injected? This attribute shall capture the reason why the defect was contained in the work product (as specified in *[AA]*); i.e. what caused the defect to have been introduced in the system. Note, it may defer from the artefact the defect was contained in , e.g. a design defect may have been injected due to a poor or missing requirement (artefact='Design model', Injection='Requirement')	Specification *[AI1]*	The defect was injected in the requirements phase; e.g. a missing, faulty, misrepresented, ambiguous requirement caused the defect
		Design *[AI2]*	The defect was injected in the design phase eventhough the specification was stated correctly; e.g. missing or faulty signal between modules, problems with the modularization
		Implementation / In-house model *[AI31]*	The defect was injected when constructing the simulation model in-house. Requirements and design were correctly specified, but mistake was made in an implementation model.

Attribute	Attribute Description	Values	Value Description
		Implementation / Supplier Auto-coding *[AI32]*	The defect was injected when transforming an executable model into code by supplier. Specification and simulation model were correct, but mistake was made in code generation by the supplier.
		Implementation / Supplier implementation *[AI33]*	The defect was injected in implementation at the supplier side (code not generated from simulation model by VCC); implementation based on correct specification and design from VCC
		Configuration *[AI4]*	The defect was injected in the configuration of the function (specification, design and implementation is correct); i.e. a faulty value of a tuning parameter
Component /Asset *[AC]*	Which design component contained the the defect? This attribute shall identify the component (at code level) at the lowest level available. The purpose is the attribute is to identify which components are most likely to contain defects.	Component name or ID *[AC1]*	A unique ID of the component containing the defect. The higher the resolution the better
Type *[AT]*	What type of defect was it? The type attribute describes the character of the defect. The values of the type attribute may depend on which artefact/component it affects	Data *[AT1]*	Defect in data definition, initialization, mapping, access, or use, as found in a model, specification, or implementation [11]. E.g. initialization of a variable, incorrect assignment of a value, incorrect carinality in data model, using wrong variable, assuming wrong variable type (e.g. assuming vehicle speed in km/h when it is stored as mph)
		Interface / Timing *[AT2]*	Defect in specification or implementation of an interface between two design components, e.g. missing or wrong signals specified or errors in the timing of communication
		Tooling *[AT4]*	The defect is present in tools used in development; e.g. simulation environments that are used in development (e.g. simulating external components such as sensors etc.)
		Logic / Computation *[AT3]*	A defect in the logic of execution; eg. an algorithmic defect either because of a faulty implementation of a correct specification or a faulty specification (or any combination thereof)
		Tuning *[AT5]*	The defect relate to tuning parameters of the function. E.g. missing or misinterpreted tuning parameters (errors in design) or faulty values assigned to tuning parameters (implementation)
		Description *[AT6]*	Defect in specification, e.g. missing or wrong description (such as a requirement)
		Standards *[AT7]*	Non-conformity with a defined standard

Life-cycle phase 3 – Resolution

TABLE 10 LIDEC SCHEME -- ATTRIBUTES IN THE RESOLUTION PHASE

Attribute	Attribute Description	Values	Value Description
Timing/Removal [ET]	When was the defect removed? Note, this does not necessarily mean that the defect was fixed (see attribute 'Resolution state' in life-cycle phase 4)	Date / Project phase [ET1]	The date/project phase when the defect was considered removed from the system, i.e. when the defect report was closed.
		Defect not yet closed *[ET2]*	The defect has not yet been closed. NOTE! This refers to the defect/problem report not the fault or failure; i.e. a defect report can be closed without having the underlying fault addressed.
Product Impact *[EI]*	What is/would be the impact on the product of a proper resolution? Note, this is an estimate of a *proper* resolution of the defect; i.e. an issue that would require major redesign to resolve, but that can be worked around with a small local fix shall be classified as a "Re-design' (the scope of the change made is captured by 'Resolution state' in life-cycle phase 4)	None *[EI1]*	There is no resolution (e.g. the reported defect was intended behaviour) or the resolution has no impact on the product
		Local modification *[EI2]*	The resolution is limited to a fixing a local module; other modules are not affected E.g. modification of a tuning parameter or code modifications to a single module that does not affect other modules
		Multiple components *[EI3]*	The resolution requires changes in multiple existing modules
		Functional Changes *[EI4]*	The resolution require a redesign, e.g. adding, removing or redefining modules
Required Verification Level *[EV]*	What level of regression testing would a proper resolution require?	None *[EV5]*	No re-verification needed
		Inspection *[EV1]*	Review of documentation, report or code is sufficient means of verification/validation of the modified component/system
		Component test *[EV2]*	Re-verification of the modified component using recorded data (Resim) is sufficient. E.g. running an executable model in Simulink with the recorded data
		System test *[EV3]*	Re-verification at system level using recorded data (Resim) is sufficient E.g. a new software build of all components of the system is needed in order to validate the changes; it is sufficient to re-verify the system on bench with recorded data (Resim)
		Expedition *[EV4]*	The resolution would need re-validation with real data, e.g. in a full car-build on test-site or on an expedition ("Breddprovning")

Life-cycle phase 4 – Post-mortem

TABLE 11 LIDEC SCHEME -- ATTRIBUTES IN THE POST-MORTEM PHASE

Attribute	Attribute Description	Values	Value Description
Resolution state [PS]	What was the state of the resolution when the defect was marked as closed? This attribute is meant to track how the defect was eventually handled	Corrected [PS1]	A proper resolution, addressing the root cause, was applied
		Workaround / Fix *[PS21]*	The underlying fault remains, but workarounds were made to avoid failure. The workaround retains the intended capability of the original specification
		Workaround / Product de-scope *[PS22]*	D.o. but the workaround forced de-scoping of the system, e.g. limiting the functionality or quality of service
		No Action / Deferred *[PS31]*	The defect was left in the system, and resolution deferred to a later revision
		No Action / Referred *[PS32]*	The source of the defect lies in another system. Defect was referred and closed in this system
		No Action / Not Found *[PS33]*	The defect was not found again; e.g. the failure could not be reproduced or the defect was not observed in a later revision of the software
		No Action / No Action *[PS34]*	No action taken, defect remains in system

Appendix B. LiDeC Classification Guide

Figures Figure 6 to Figure 18 below show the classification guide used during the classification sessions, as described in section *3 Method*. Initially, LiDeC was presented to the person responsible for defect classification by an overview of the classification scheme. The overview was described by using Figure 6, in which all available LiDeC attributes are represented and grouped into the phases of the defect life-cycle (further described in *2.2.1 Defect classification schemes*).

During the classification sessions, each defect was classified according to the attributes in LiDeC. For each attribute, the developer was shown the corresponding image (shown below). In each image, the question that guides the reporter is shown at the top in an orange rectangle (e.g. “What was done when the defect was detected?”). The values that can be assigned the attribute is shown in blue rectangles below the question. The white boxes, shown e.g. in the attribute “Detection Activity”, represent categories of values and serve only to provide the values with a clearer structure. For instance, for the attribute “Detection activity” the categories serve to make an initial separation on types of activities, and then breaks down those into the sub-activities that are to be chosen as the value for the attribute.

Note, in this appendix the guides for trivial attributes (i.e. attributes with simple Yes/No values or dates) have been omitted.

Attribute overview

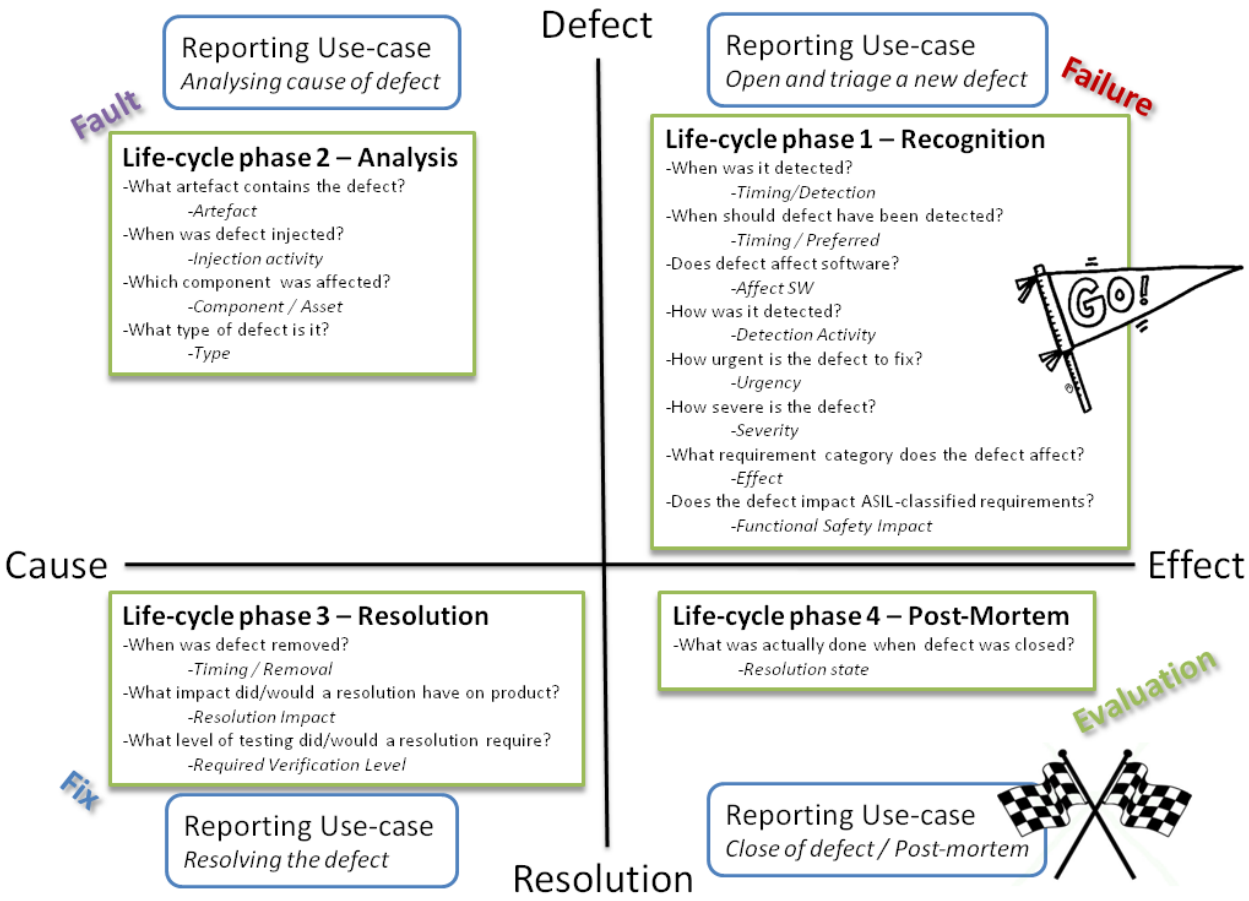

Figure 6 Detailed LiDeC Scheme attribute overview

Life-cycle phase 1: Recognition – Detection of the defect

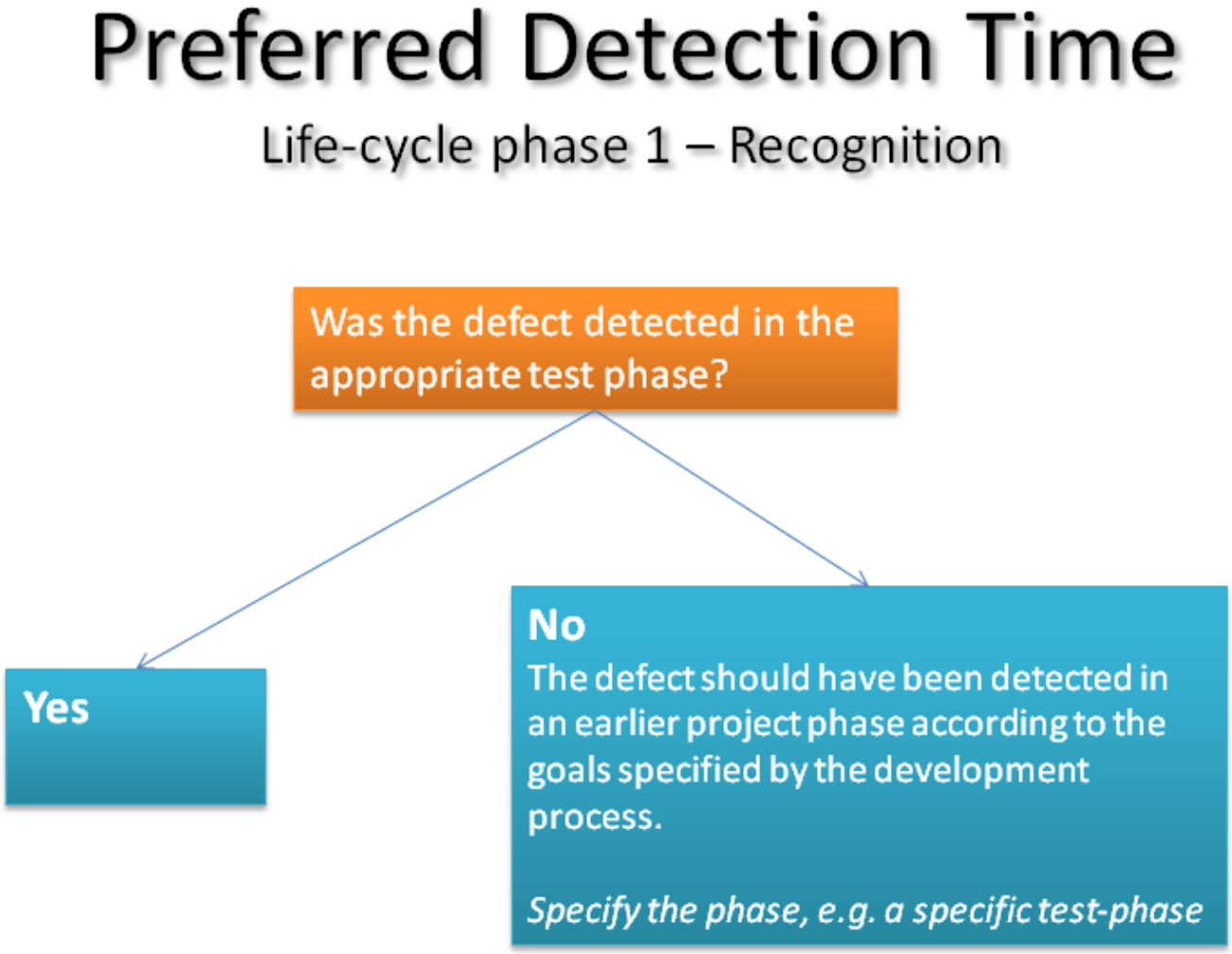

Figure 7 Classification guide for the attribute *Preferred detection time*

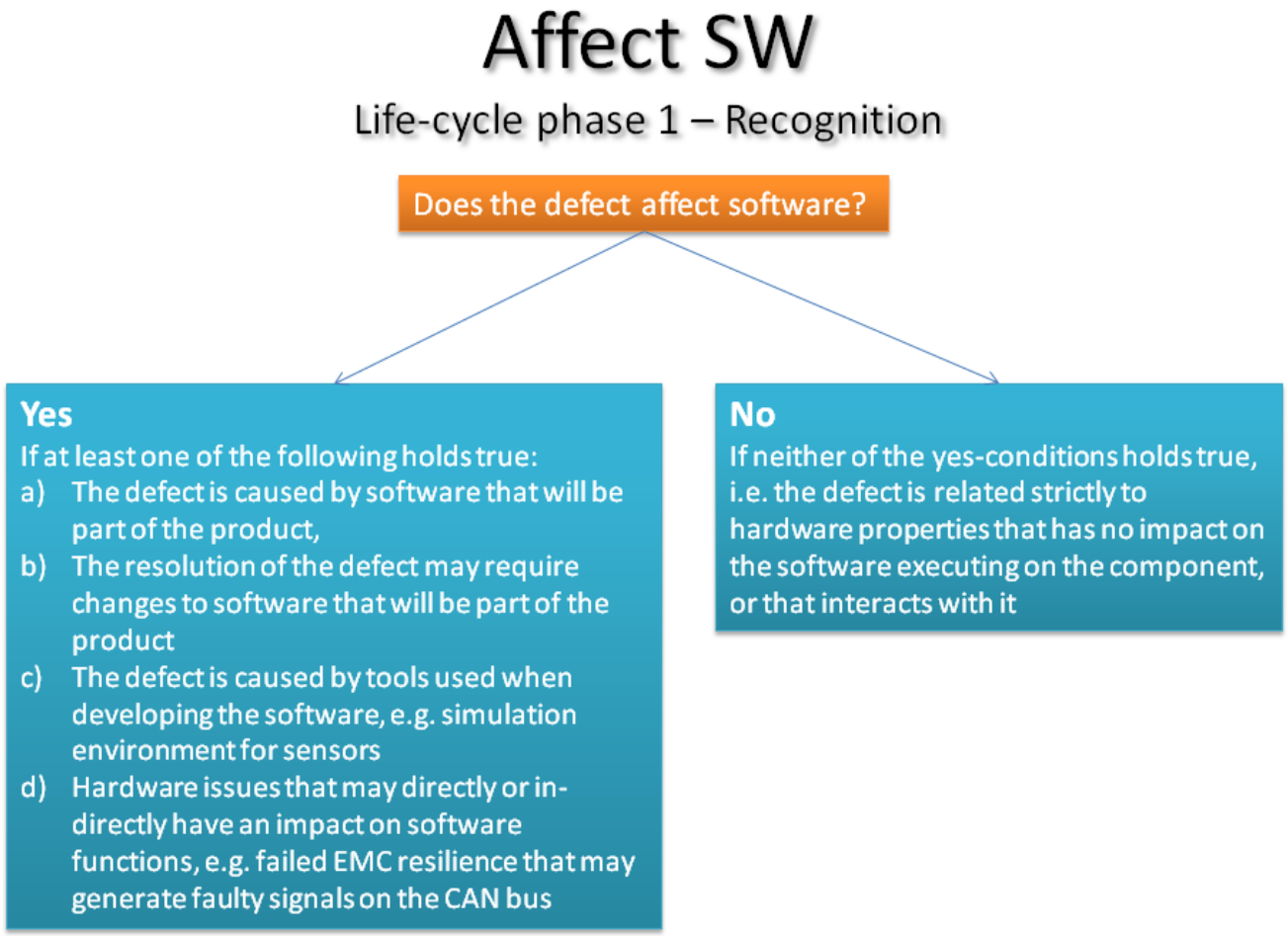

Figure 8 Classification guide for the attribute *Affect Software*

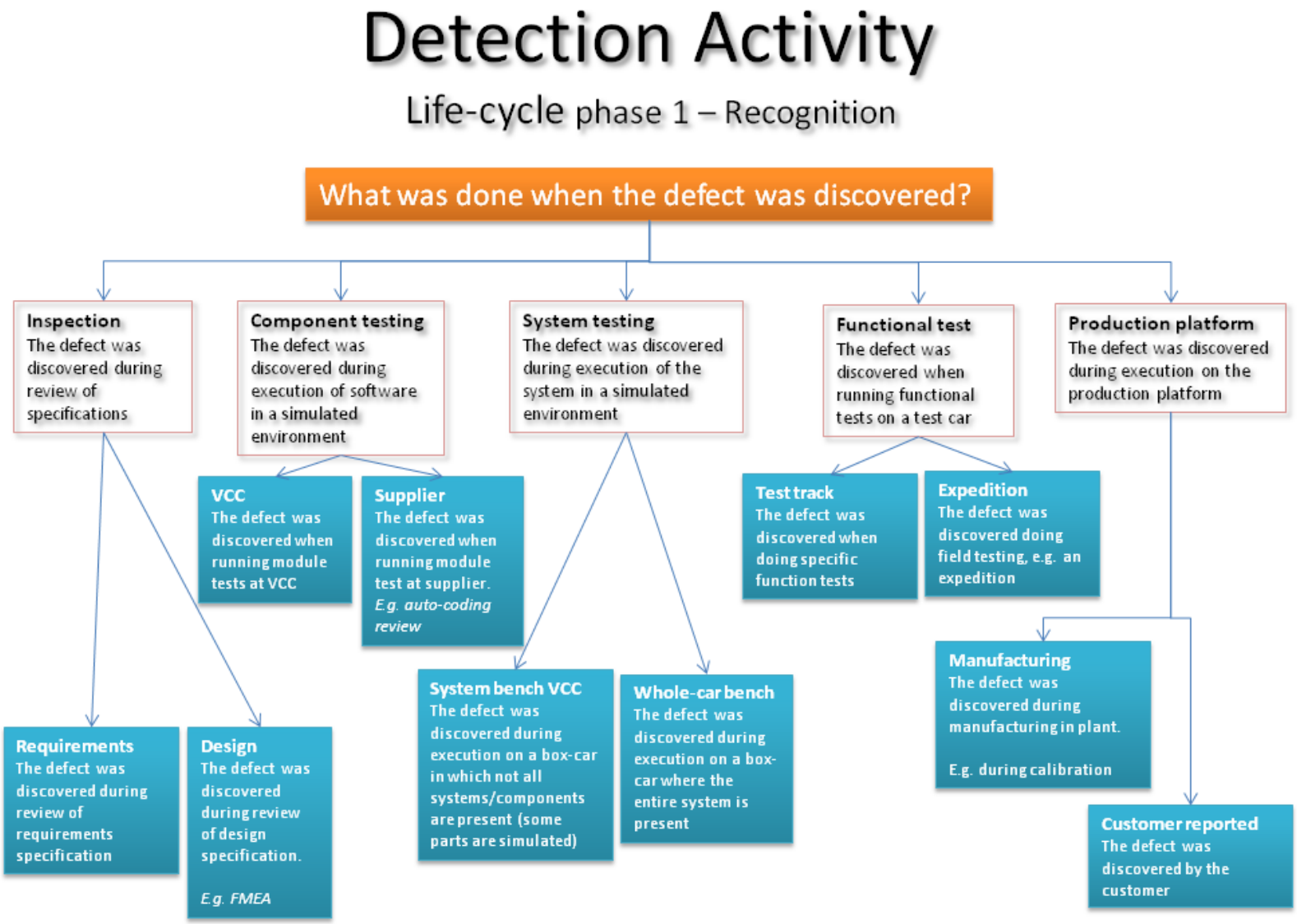

Figure 9 Classification guide for the attribute *Detection activity*

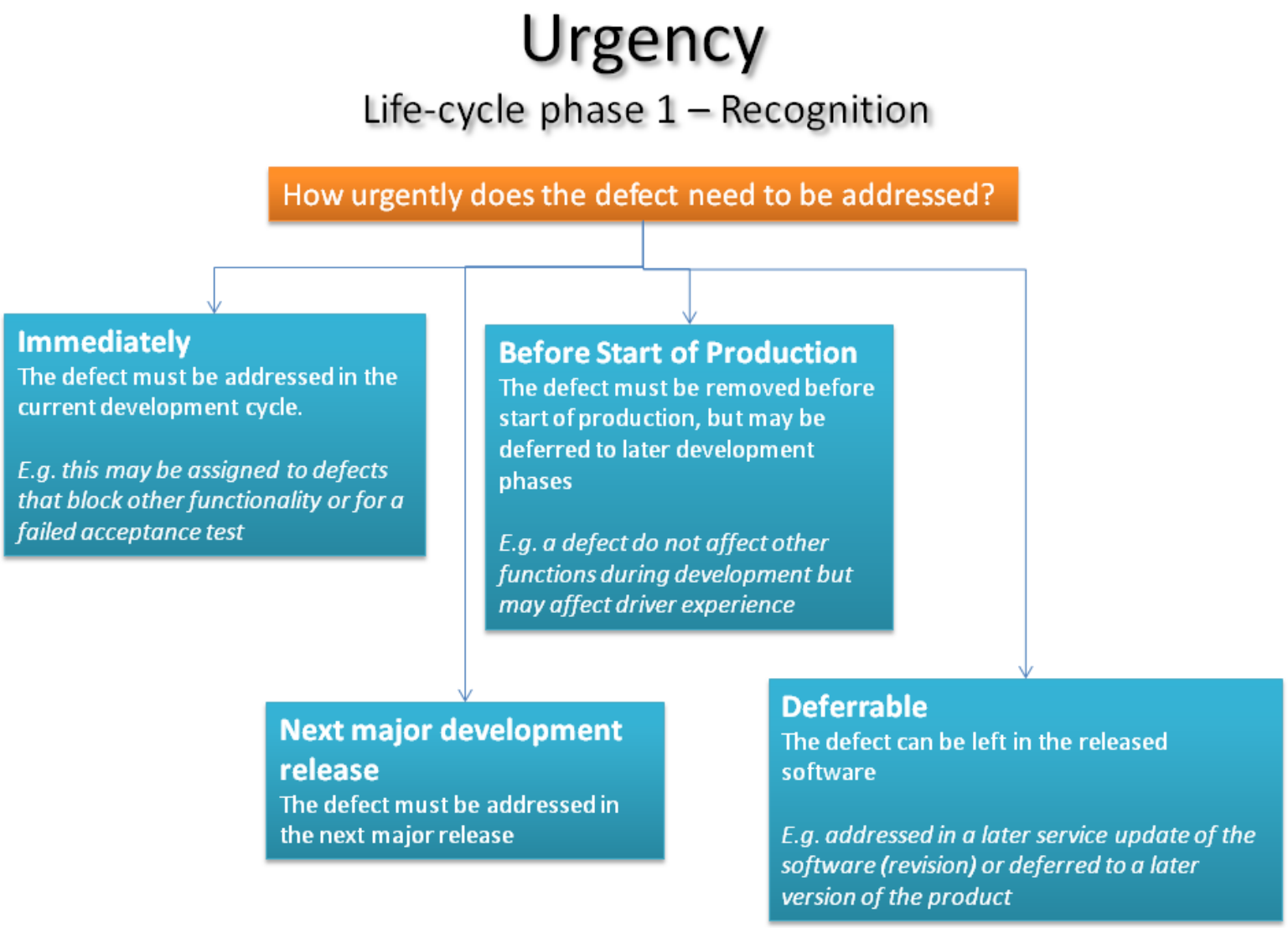

Figure 10 Classification guide for the attribute *Urgency*

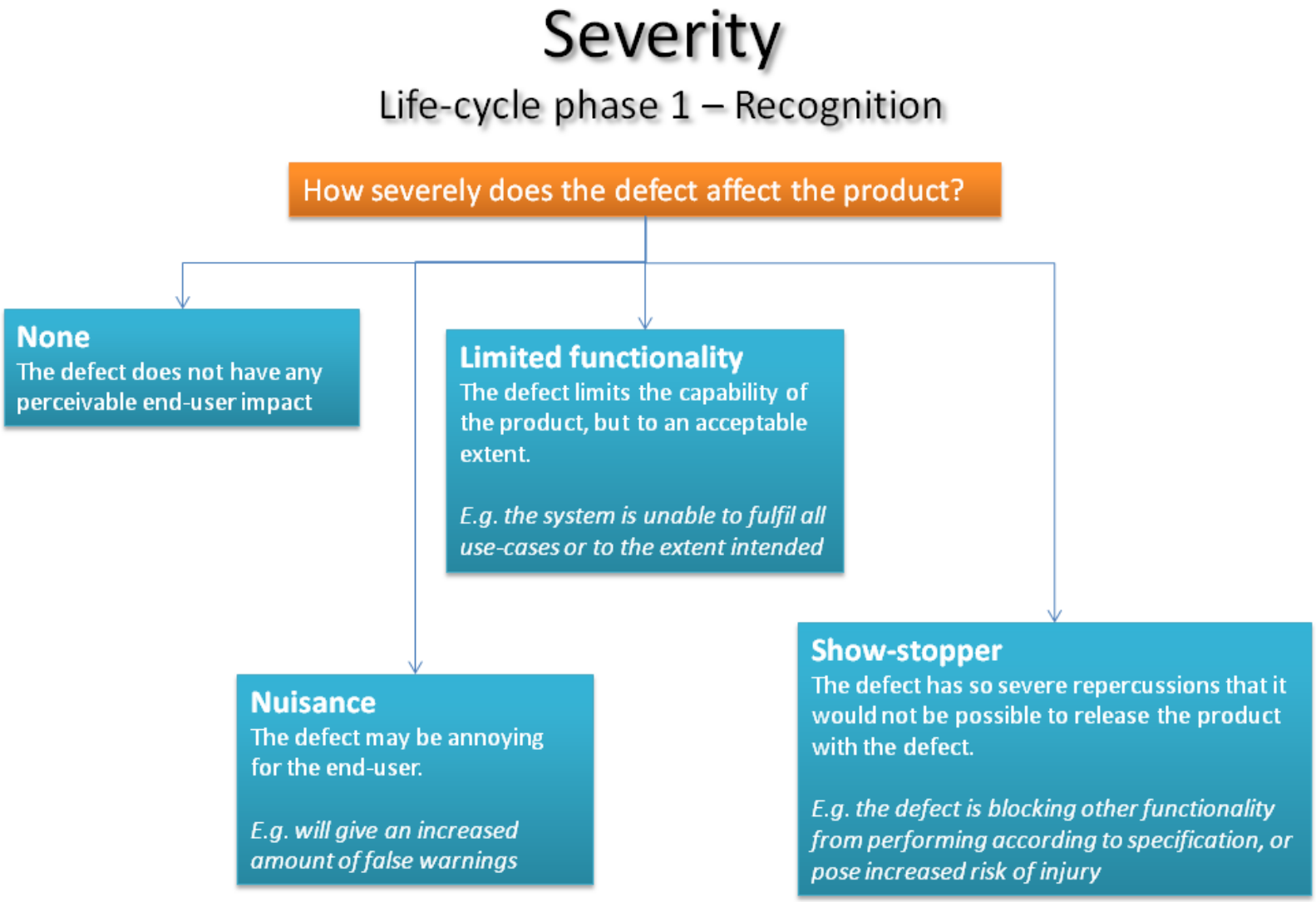

Figure 11. Classification guide for the attribute *Severity*

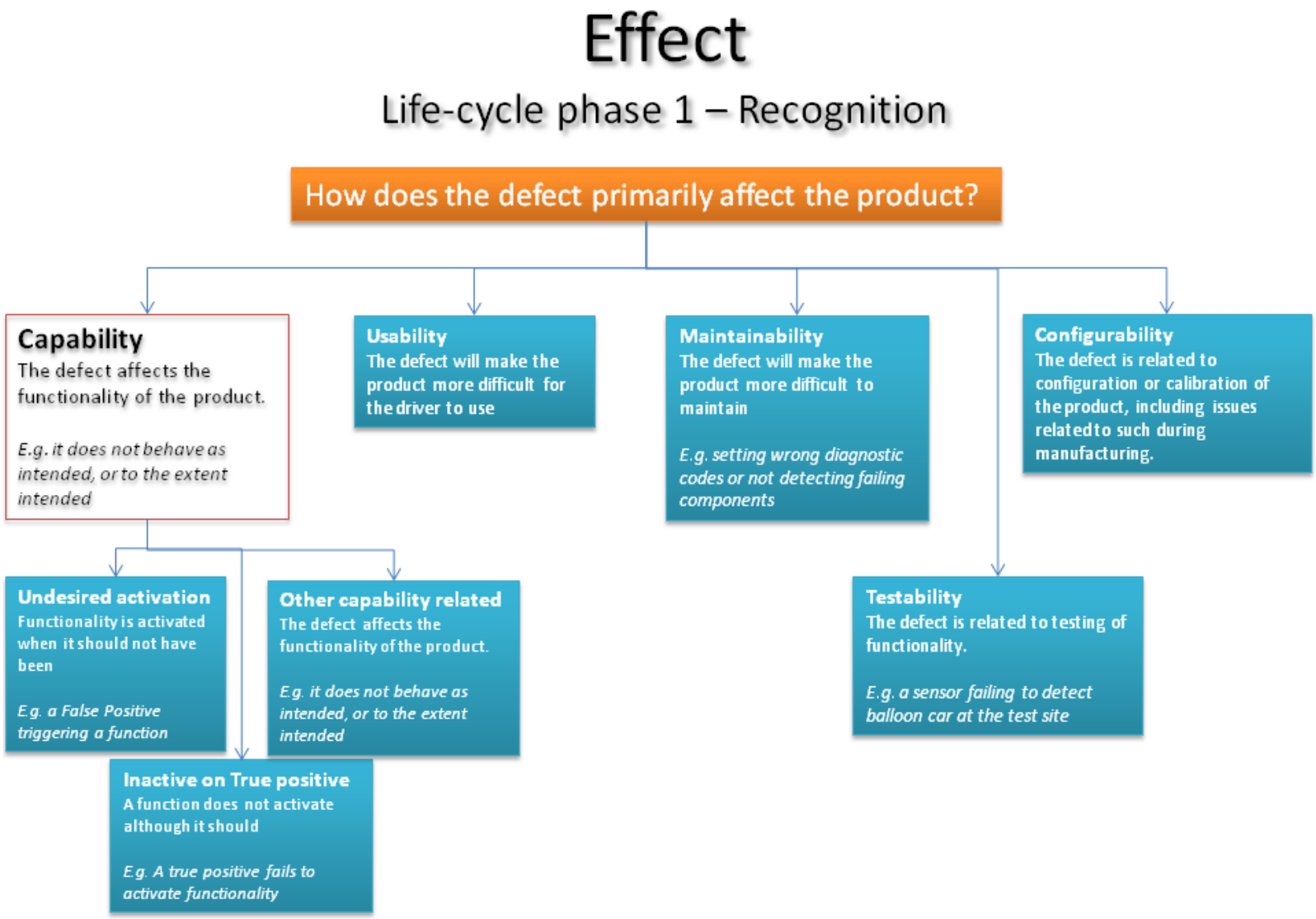

Figure 12 Classification guide for the attribute *Effect*

Life-cycle phase 2: Analysis – Investigating the cause of the defect

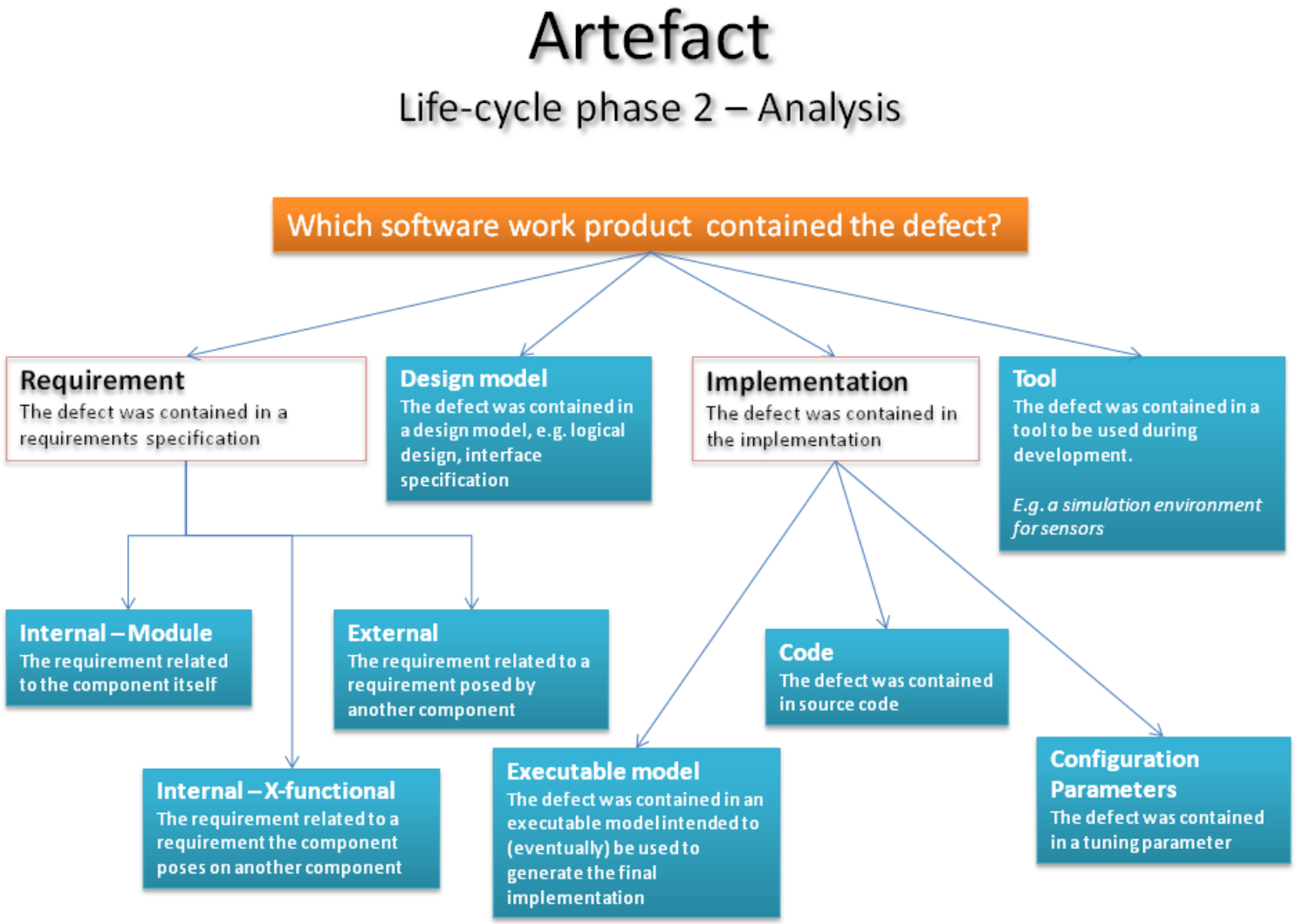

Figure 13 Classification guide for the attribute *Artefact*

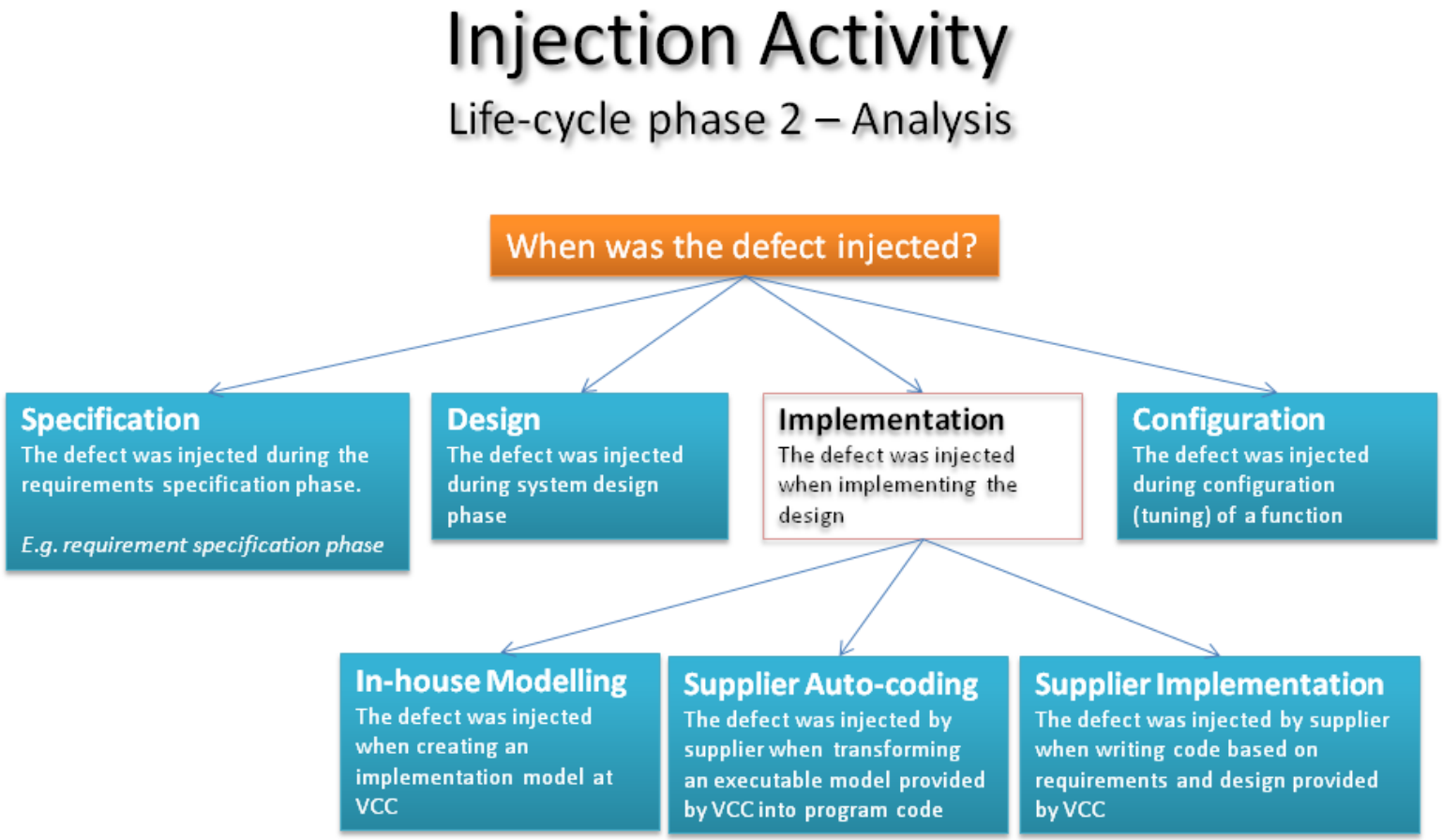

Figure 14 Classification guide for the attribute *Injection activity*

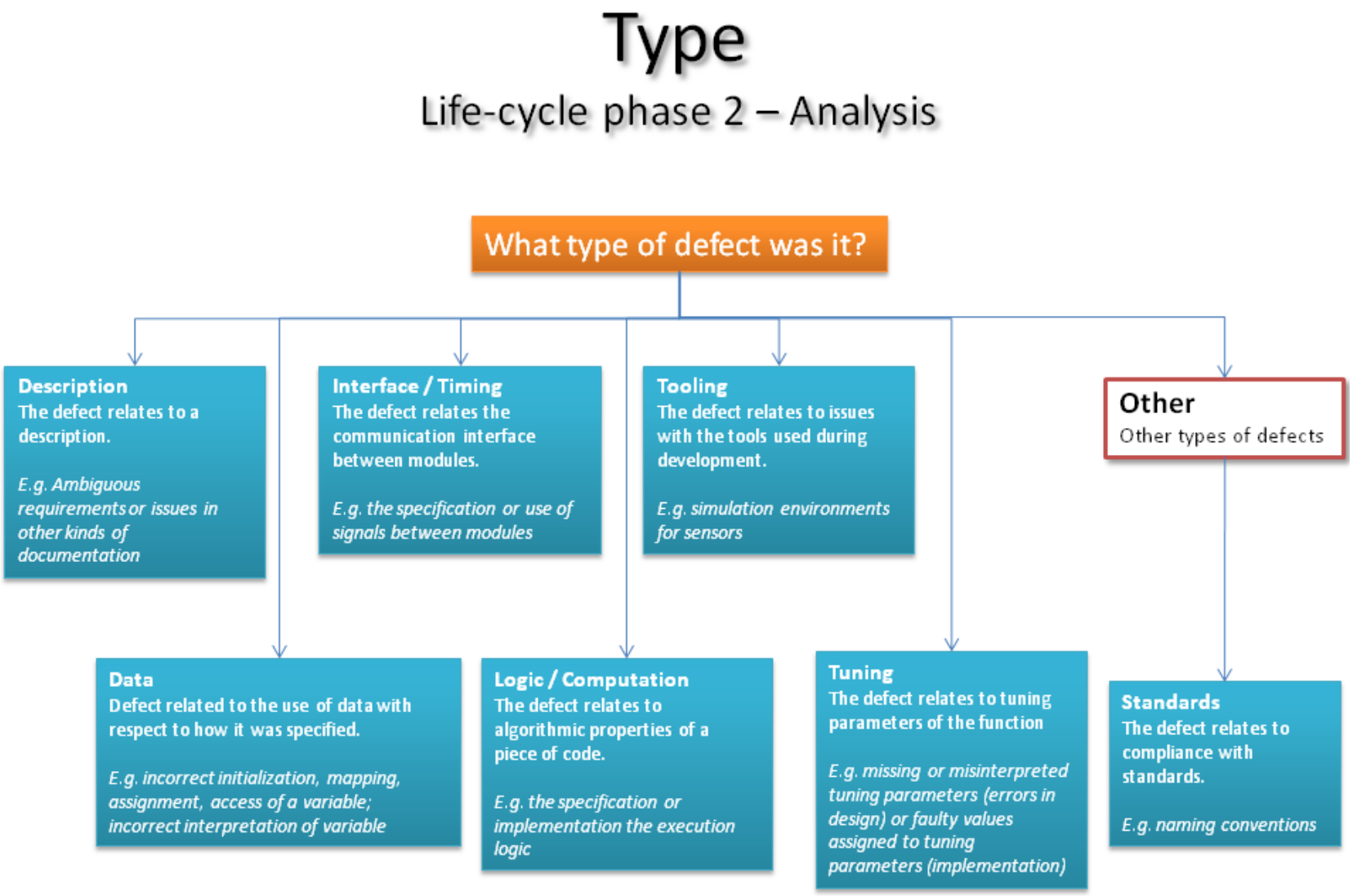

Figure 15 Classification guide for the attribute *Type*

Life-cycle phase 3: Resolution – The action leading to defect removal

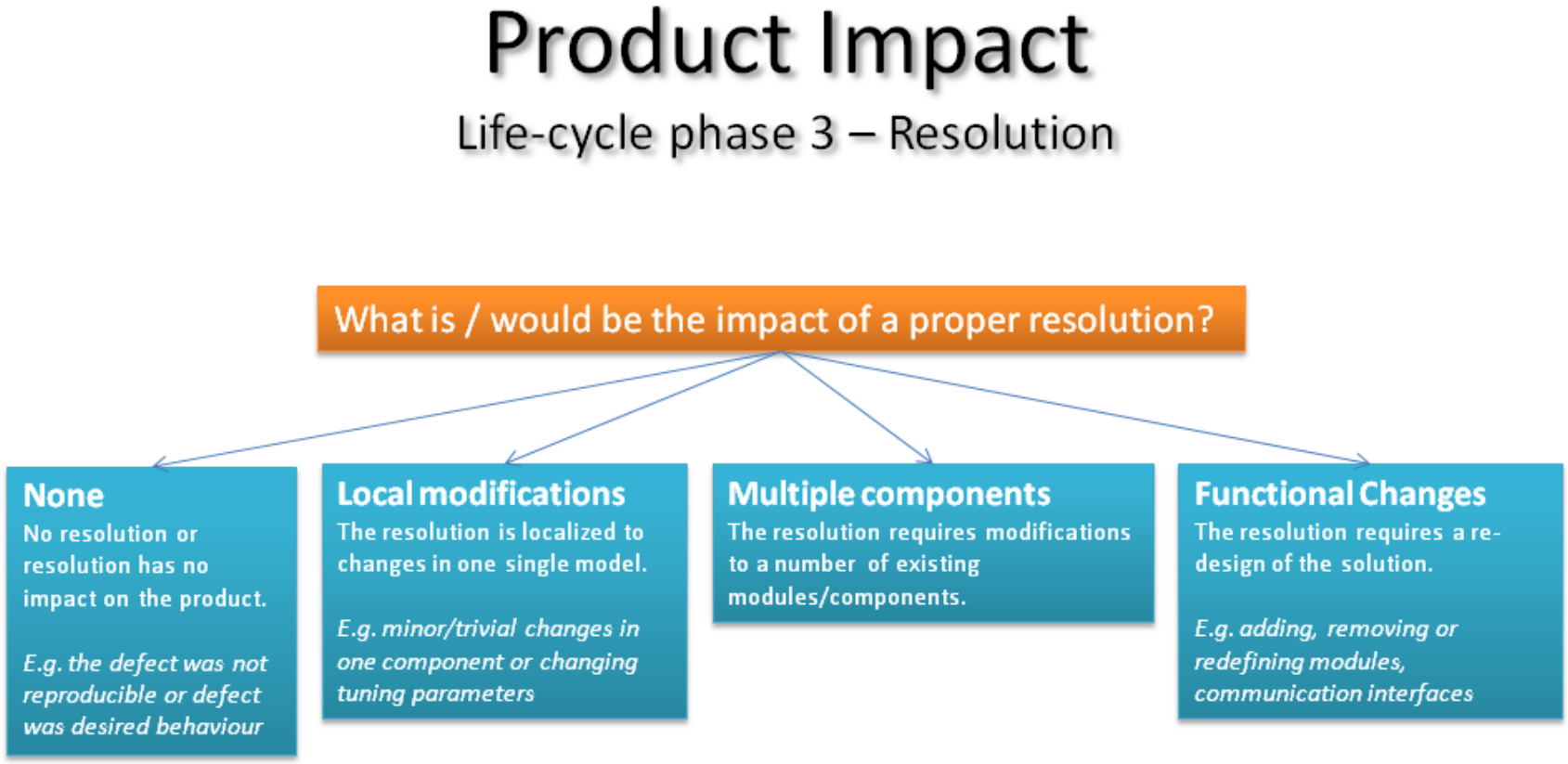

Figure 16 Classification guide for the attribute *Product impact*

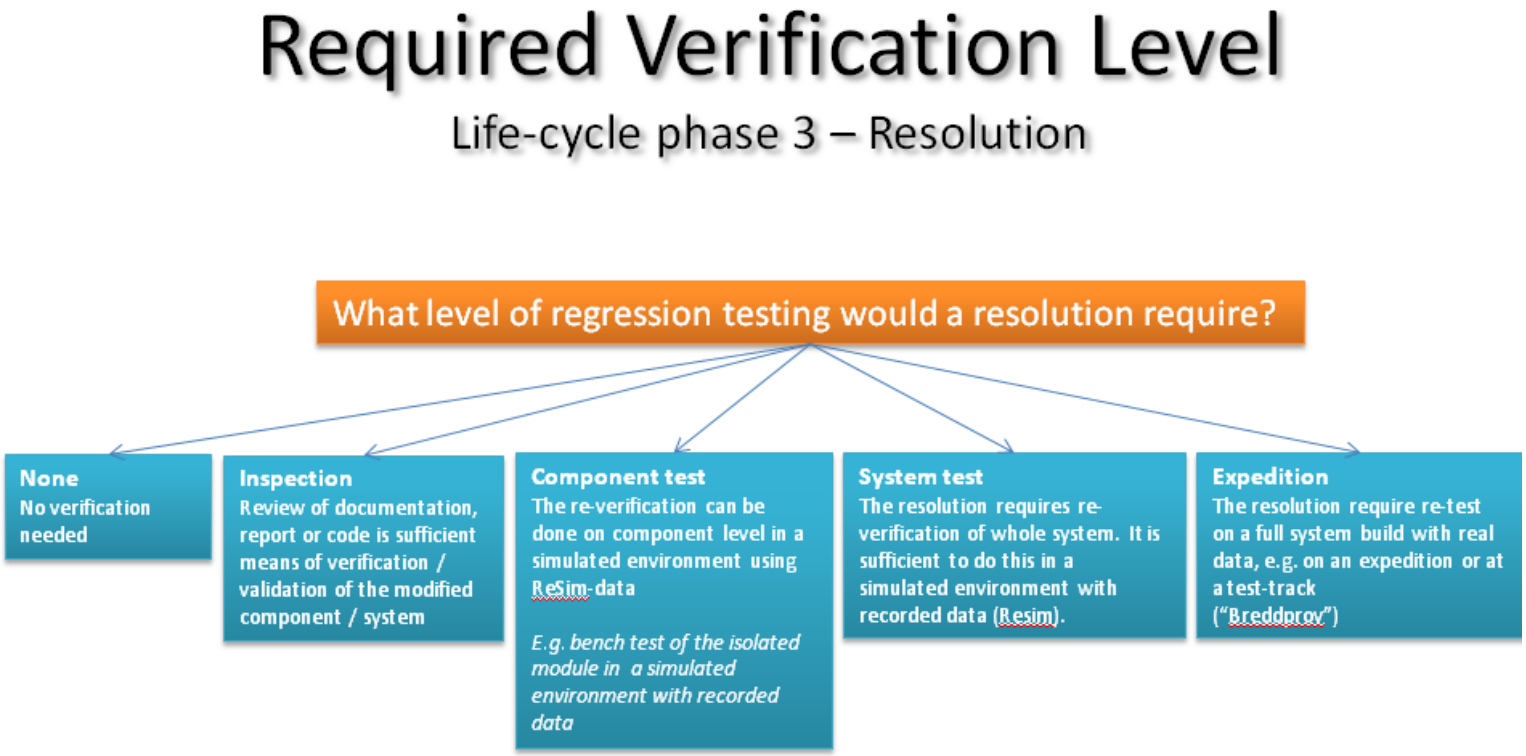

Figure 17 Classification guide for the attribute *Required verification level*

Life-cycle phase 4: Post-mortem – Final state of defect

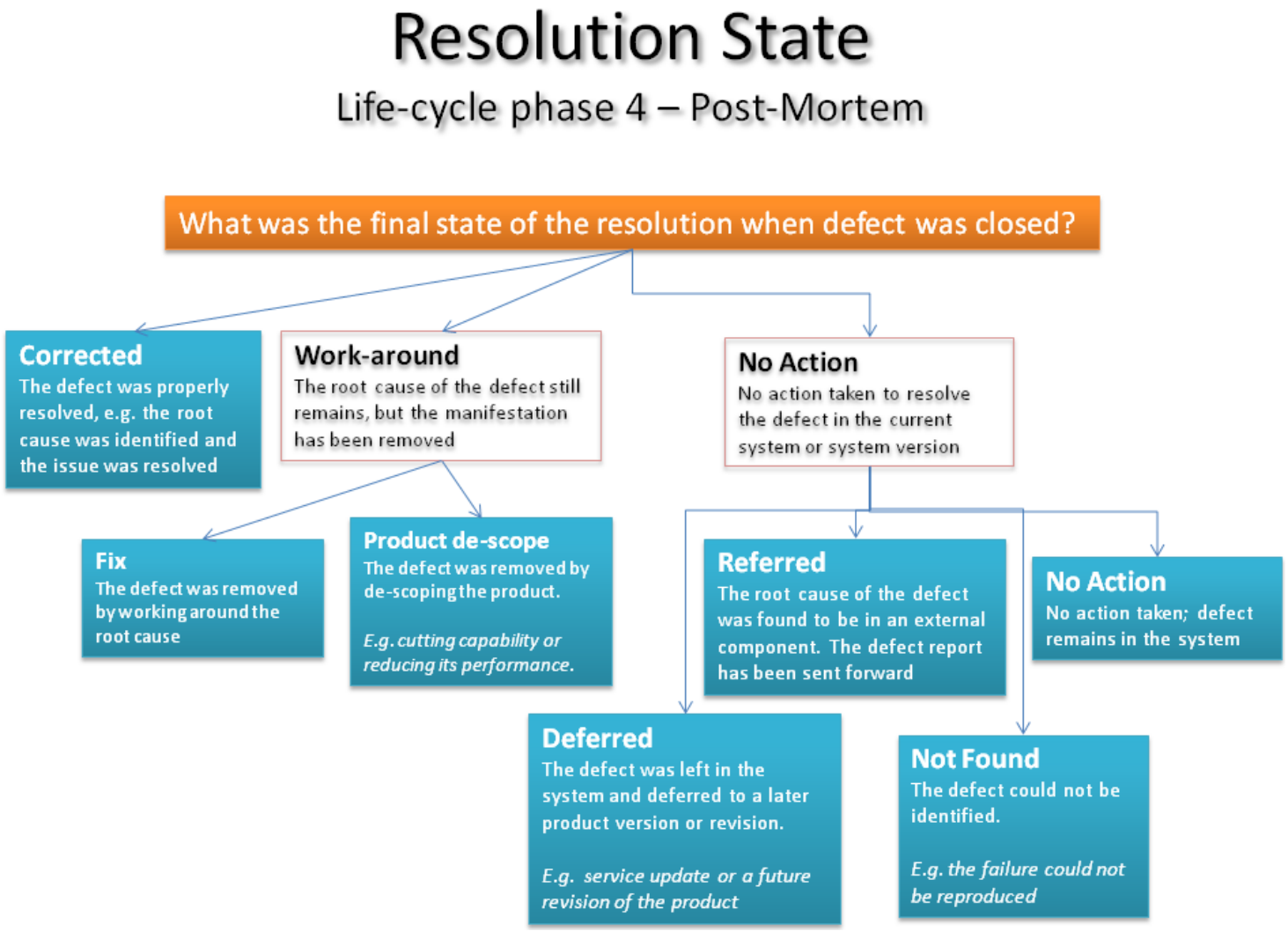

Figure 18 Classification guide for the attribute *Resolution state*

Appendix C. EXAMPLE CLASSIFICATION

In this section the classification of two example defects is described. Due to confidentiality reasons, the defects described below are construed examples. The example defects, however, are inspired by defect reports encountered during the case-study.

Example 1

The first defect caused a faulty diagnostic flag to be set indicating that part of software installed on an ECU failed. The defect was detected during testing of the vehicle at the factory manufacturing line late in the project. The problem occurred when updating the software and its configuration parameters on the ECU. In the process of deploying software and configuration parameters, the ECU was first set into a programmable mode in which a diagnostic routine is executed; it was in the diagnostic routine that the faulty error-mode was set.

During the root cause analysis it was found that the flag indicating component's (programmable) mode was stored to an incorrect output port which caused the programmable mode to be interpreted as a faulty error mode. Although the defect manifested itself during manufacturing (when software was first deployed to the ECU), it could occur during the software maintenance phase when software updates are deployed to the ECUs. It would, however, have had no effect on the normal operational mode of the component, the system or the complete vehicle (e.g. the driver would not have been affected by the defect).

Recognition phase

The date on which the defect report was submitted in the issue management system was used for the *Timing/Detection* attribute (2009-04-14). In addition, the development phase in which the detection was made was also noted (though this is redundant information, as the development phase can be derived from the date using the project plan, but it was convenient to have that information readily available). The detection time was considered (subjectively by the reporter) as significantly late in the project. The reporter considered it to be a software unit problem and that it should have been detected in an earlier test phase (either during unit testing at the supplier side, or during unit testing at the OEM side; the internal name of the preferred test phase was *U2*). The defect was, furthermore, considered to affect software, as (according to the reporter) it was probably an implementation error that led to the wrong diagnostic flag being set.

The defect was discovered during the testing of the manufacturing line when the car was assembled in factory, thus the value *Manufacturing* was selected for the *Detection Activity* attribute. As the defect in this case made it difficult to assess whether deploying the software to the ECU was successful it was considered impossible to release the software into production in its current state, and its resolution to be very urgent; the value *Immediately* was chosen for the *Urgency* attribute (as the defect was discovered late, a fix was promptly needed) and the value *Show-stopper* for the *Severity* attribute (as the software could not be released while containing the defect).

The *Effect* attribute was set to *Maintenance* as the main effect of the defect related to problems when software and configuration parameters were to be updated. The defect would, furthermore, have affected any future maintenance updates to both software and configuration parameters. Finally, the defect was considered to have no impact on any ASIL-classified requirements (if the software update had truly failed, it would have successfully triggered other diagnostic functions indicating that the component was not operating properly), it was thus considered not to have any impact on functional safety (as defined by ISO 26262).

Table 12 summarizes the classification for the Recognition phase.

TABLE 12. EXAMPLE CLASSIFICATION; EXAMPLE 1, RECOGNITION PHASE

Attribute	Value
Timing / Detection	2009-04-14 (Manufacturing test)
Timing / Preferred	U2 (unity testing)
Affect SW	Yes
Detection activity	Manufacturing
Urgency	Immediate
Severity	Show-Stopper
Effect	Maintenance
Functional Safety Impact	No

Analysis phase

During the root cause analysis of the failure it was found that the defect was contained in the binary code deployed on the ECU. It was, furthermore, found that the requirements clearly stated to which port the diagnostic flag should be written, and the executable model which served as base for the binary code was correctly implemented according to the requirements. Consequently, the fault was introduced during transformation from executable model to source-code; an activity done by a supplier. Therefore, the attribute *Artefact* was assigned the value *Implementation / Code*, and the attribute *Injection Activity* was assigned the value *Implementation / Supplier auto-coding*. The *Component/Asset* attribute was assigned the company's internal code identifying the software module as well as the software version.

Finally, the *Type* attribute was assigned the value *Data*, because the underlying cause of the defect related to data being written to the wrong location. Note that the *Type* attribute in IEEE Std. 1044 has a higher resolution which allows for more precision in defect analysis. In our case, however, such resolution is not possible, as the source-code (which carries the necessary information to allow for more detailed classification) is owned by the supplier. In effect, LiDeC's *Type* attribute captures a black-box alternative to the *Type* attribute in IEEE Std. 1044.

Table 13 summarizes the classification for the Analysis phase.

TABLE 13. EXAMPLE CLASSIFICATION; EXAMPLE 1, ANALYSIS PHASE

Attribute	Value
Artefact	Implementation / Code
Injection Activity	Implementation / Supplier auto-coding
Component / Asset	XYZ-1256
Type	Data

Resolution

The urgency of the defect (due to its late discovery) resulted in the resolution being applied in the same development phase in which it was discovered; the date (and development phase) of successful verification of the resolution was noted as the *Removal time*.

The necessary resolution was determined to be confined to a single software module, as no other components would need any modifications; thus the *Product Impact* attribute was set to *Local modification*. Finally, a test report from the supplier showing successful test of the binary component was considered sufficient means of verification; thus *Inspection* was set as value for the attribute *Required Verification Level*.

Table 14 summarizes the classification for the Resolution phase.

TABLE 14. EXAMPLE CLASSIFICATION; EXAMPLE 1, RESOLUTION PHASE

Attribute	Value
Timing / Removal	2009-05-09 (Manufacturing test)
Product Impact	Local modification
Required Verification Level	Inspection

Post-mortem

In the final defect life-cycle phase, the single attribute *Disposition* records what finally was done to resolve the defect. In this example, a proper resolution was applied; thus the value *Corrected* was assigned to the *Disposition* attribute.

Table 15 summarizes the classification for the Post-mortem phase.

TABLE 15. EXAMPLE CLASSIFICATION; EXAMPLE 1, POST-MORTEM PHASE

Attribute	Value
Disposition	Corrected

Example 2

The defect was found in a feature that issues an audible warning if the driver is unintentionally drifting off-lane. The function monitors the vehicle's position by tracking the lane markers in the road and gives a warning signal when a lane is about to be crossed and the driver does not use the turn signals. During the first field test (also called *expedition*) – where a mature build of the full vehicle is tested on a large variety of road types – it was found that the sensitivity of the warning was too high, resulting in frequent false alarms. The problem was detected on specific road types were the lanes were narrower than what had been anticipated.

Recognition phase

The date on which the defect report was submitted in the issue management system was used for the *Timing/Detection* attribute (2008-10-04) and noted along with the development phase in which it was detected. The detection time was considered (subjectively) by the reported as appropriate. The defect was, furthermore, considered to affect software, as (according to the reporter) it was the behaviour of the software that caused the problem.

The defect was discovered during the functional testing of the vehicle on an expedition, thus the value *Functional Test / Expedition* was selected for the *Detection Activity* attribute. As the defect in this case caused considerable nuisance to the driver on specific road types, it was considered to impossible to release the software into production with the defect remaining (the value was thus set to *Show-stopper*). As the resolution of the defect was considered to need further testing (on the problematic road type, as well as other types to ensure no regressions had been introduced) and that additional development releases had already been planned for testing, the *Urgency* attribute was set to *Next Major Release*.

The *Effect* attribute was set to *Capability / Undesired Activation* as its main effect related to false warnings. Initially, there was some confusion whether the defect should be classified as having effect on the *Usability*. However, as the problem was not related to the *way* the user was warned (i.e. through an audible cue), the problem did not have impact on usability.

Finally, the defect was considered to have no impact on any ASIL-classified requirements, it was thus not considered to have any impact on the functional safety (as defined by ISO 26262). If, on the other hand, the feature had been able to autonomously intervene in steering or braking, it would indeed have had impact on functional safety.

Table 16 summarizes the classification for the Recognition phase.

TABLE 16. EXAMPLE CLASSIFICATION; EXAMPLE 2, RECOGNITION PHASE

Attribute	Value
Timing / Detection	2008-10-04 (First full vehicle functional test)
Timing / Preferred	-
Affect SW	Yes
Detection activity	Expedition
Urgency	Next major release
Severity	Show-Stopper
Effect	Capability / Undesired activation
Functional Safety Impact	No

Analysis phase

The problem occurred in running code (i.e. in the binary code of on one of the software components realizing the feature). However, as the feature was designed to be configurable (using tuning parameters) with respect to the width of the lanes, the *Artefact* attribute was set to *Implementation / Configuration Parameters*. It was, furthermore, found that the requirements specification for the feature did not take the particular road type into consideration. Additionally, the design as well as the executable model and the binary code were found to have been correctly derived and implemented from the requirements specification. Consequently, the attribute *Injection Activity* was assigned the value *Specification*. The *Component/Asset* attribute was assigned the company's internal code identifying the software module as well as the software version.

Finally, the *Type* attribute was assigned the value *Description* because the requirements did not take the particular road type into consideration.

Table 17 summarizes the classification for the Analysis phase.

TABLE 17. EXAMPLE CLASSIFICATION; EXAMPLE 2 ANALYSIS PHASE

Attribute	Value
Artefact	Implementation / Configuration Parameters
Injection Activity	Specification
Component / Asset	ABC-5431
Type	Description

Resolution

The resolution was applied in next major release of the software; the date (and development phase) of successful verification of the resolution was noted as the *Removal time*.

The necessary resolution was determined to be confined to one component (specifically, the configuration parameters of a software module). However, as the particular module also provided other features in the vehicle with data, and thus might be affected by the modification, the *Product Impact* attribute was set to *Multiple Components*. Finally, it was considered necessary to verify the resolution in a full vehicle build, where all features dependent on the modified software model were tested; thus *Expedition* was set as value for the attribute *Required Verification Level*.

Table 18 summarizes the classification for the Resolution phase.

TABLE 18. EXAMPLE CLASSIFICATION; EXAMPLE 2, RESOLUTION PHASE

Attribute	Value
Timing / Removal	2009-02-09 (Second full vehicle functional test)
Product Impact	Multiple modules
Required Verification Level	Expedition

Post-mortem

The resolution of the defect was finally done in two parts: first, the configuration parameters of the software module were modified, and; second, the requirement specification was updated with a description of the road type that caused the problem. Thus, the value *Corrected* was assigned to the *Disposition* attribute. If, on the other hand, the requirements specification had not been updated, it should have been set to *Work-around / Fix* as the problem would have been mitigated, but the root cause not properly removed.

Table 19 summarizes the classification for the Post-mortem phase.

TABLE 19. EXAMPLE CLASSIFICATION; EXAMPLE 2, POST-MORTEM PHASE

Attribute	Value
Disposition	Corrected

Appendix D. IEEE STD. 1044 COMPLIANCE MATRIX

Table 20 shows the IEEE Std. 1044 compliance matrix (see table 3 in [12]). In the table green cells indicates attributes that map between the IEEE 1044 and LiDeC schemes, red cells indicate IEEE 1044 attributes not available in LiDeC, while white cells indicate IEEE 1044 attributes that are implicit in other LiDeC attributes. Furthermore, an asterisk indicates that the LiDeC attribute has a different definition than its IEEE Std. 1044 correspondence.

TABLE 20 IEEE STD 1044 COMPLIENCE MATRIX

IEEE Std 1044-1993 Category	**LiDeC equivalent category**	**Mandatory in IEEE Std. 1044-1993**	**Comment**
Actual cause	Injection activity*	√	Whereas IEEE records actual cause as the artefact that caused the defect, LiDeC records in which activity it was injected
Corrective action			
Customer value			Implicit in LiDeC.Severity
Disposition	Resolution state	√	
Mission/safety			Implicit in LiDeC.Severity
Priority			Implicit in LiDeC.Urgency
Product status	Severity		
Project activity	Detection activity	√	
Project cost	Product impact*	√	The level of impact on the product – in terms of required change – is considered a better estimate of "cost" than money
Project phase	Timing / Detection	√	
Project quality/reliability			
Project risk			

IEEE Std 1044-1993 Category	LiDeC equivalent category	Mandatory in IEEE Std. 1044-1993	Comment
Project schedule	Re-verification level*	√	Whereas IEEE.ProjectSchedule is described as "an appraisal of the amount of effort required to address the defect", LiDeC instead expresses it in terms of the amount of re-verification required. This is because verification is a costly activity, and it is a more convenient way for individual teams to estimate impact on schedule
Repeatability			
Resolution	Urgency*	√	Whereas the LiDeC.Urgency records how quickly the defect needs to be removed, the attribute IEEE.Resolution also records the type of resolution applied
Severity	Severity*	√	The IEEE attribute also includes whether a solution exist in the severity attribute. LiDeC, however, assesses "Severity" on the observed failure, and does thus not take the availability of a fix into consideration
Societal	Functional Safety Impact		The attribute *Functional safety impact* captures whether the defect may risk causing harm to persons (as defined by the ISO 26262 [34]). This maps to the IEEE Std. 1044 attribute *Societal* in that it captures data about the impact of the defect on environment (e.g. driver, passenger or other persons in the vehicle's surroundings).
Source	Artefact	√	
Suspected cause			
Symptom	Effect	√	
Type	Type	√	

Appendix E. MAPPING BETWEEN IEEE STD. 1044 AND LIDEC

TABLE 21 MAPPING BETWEEN IEEE STD. 1044 AND LIDEC

Life-Cycle Phase	*IEEE Std. 1044*	*Description*	*LiDeC*	*Description*	*Mapping comment*
Recognition	**Project Activity**	What were you doing when the anomaly occurred?	**Detection activity**	How was the defect detected?	
	Project Phase	In which life cycle phase is the product?	**Timing / detection**	When was the defect detected?	
	Suspected cause	What do you think might be the cause?	n/a		Speculation of the cause would mainly be of interest when analysing the fault. This is done as part of the defect management process at the company, but is not of interest for our analysis
	Repeatability	Could you make the anomaly happen more than once?	n/a		This attribute is captured by Disposition
	Symptom	How did the anomaly manifest itself?	**Effect**	What requirements category does the defect affect?	The IEEE Std. 1044 has a very detailed symptom classification. In our approach we analyse instead the type of impact the symptom would have on the product; e.g. Capability, maintainability etc
	Product Status	What is the usability of the product with no changes?	**Severity**	How severely does the defect affect the product?	
	n/a		**Timing / Preferred**	When should the defect have been detected?	The attribute records the developers (subjective) opinion on whether this defect's discovery was timely or if there was an earlier project phase in which it reasonably should have been detected. No such attribute exist in IEEE 1044
	n/a		**Affects Software**	Does the defect affect software?	As the development of automotive software is a hybrid of hardware and software development, and that our main interest lies in studying aspects related to software development, we use this attribute to make an initial coarse filtering of the defects

Life-Cycle Phase	*IEEE Std. 1044*	*Description*	*LiDeC*	*Description*	*Mapping comment*
Investigation	**Actual cause**	What caused the anomaly to occur?	**Injection Activity**	When was the defect introduced in the product?	Closely related to IEEE.Source and LiDeC.Component. Where as the IEEE maps this on product parts LiDeC captures the activity in which the defect was injected; i.e. a defect discovered in code may have been introduced due to ambiguous requirements.
	Source	Where was the origin of the anomaly?	**Artefact**	Which software work product contained the defect?	
	Type	What type of anomaly/enhancement at the code level?	**Type**	What type of defect was it?	Directly mappable, though LiDeC use a much higher abstraction level of the selection of types. There were still cases where the distinction between types was not straight-forward – mainly because the types were not easily understandable (rather than lack of understanding of the defect itself)
	Action supporting data item		**Component /Asset**	Which design component contained the defect?	The attribute captures which part of the product contained the defect. This relates to IEEE.ActualCause and is also part of the supporting data items in the Action life-cycle phase, although that data item captures which part of the product will need changing (which may not be the same as the one containing the defect!)
Action	**Resolution**	What action to take to resolve the anomaly?	**Urgency**	How urgent is it to resolve the defect?	LiDeC.Urgency also maps to the IEEE.Priority attribute. However, investigating the defect to arrive at the priority requires resources; we have in LiDeC chosen to record the Urgency attribute on failure-level instead of on fault level. Consequently, 'Urgency' relates to how urgent it is to remove the manifestation of the fault rather than the fault itself (which the IEEE.Priority attribute specifies)
	Corrective action	What to do to prevent the anomaly from happening again	n/a		Whereas IEEE records the exact resolution we have chosen instead to record the extent of impact the resolution would have on the product (see IEEE.ProjectCost).

Life-Cycle Phase	*IEEE Std. 1044*	*Description*	*LiDeC*	*Description*	*Mapping comment*
		Action supporting data item	**Removal Time**	When was the defect closed?	LiDeC captures the time of closing the problem report (regardless of the state of the resolution) in order to be able to measure the longevity of defects and the project workload. This information is interesting as it serves as a measurement of the pressure on the project – assuming mistakes are more likely to be made under pressure one would like to keep the number of open defects to a minimum
Impact Identification	**Severity**	How bad was the anomaly in more objective engineering terms?	**Severity**	What would the impact on the product be if defect remain in system on release?	Also see IEEE.ProductStatus
	Priority	Rank the importance of resolving the anomaly (subjective)	(Urgency)	How urgent is it to resolve the defect?	See the IEEE.Resolution attribute
	Customer value	How important a fix is to customers?		n/a	This is implicit in the LiDeC.Severity attribute
	Mission / Safety	How bad was the anomaly with respect to project objectives or human well-being?		n/a	This is implicit in the LiDeC.Severity attribute
	Project schedule	Relative effect on the product schedule to fix	**Required Verification Level**	What level of regression testing would a proper resolution require?	Required effort to apply a resolution is not only captured by the amount of necessary modification to the product. As automotive software have very high reliability requirements, V&V activities require substantial amount of resources. This attribute records the estimated level of regression testing that a proper resolution would require (as order of magnitude)
	Project cost	Relative effect on the project budget to fix	**Product Impact**	What would the impact on the product be if a proper resolution was applied? Value is intended as order of magnitude – from no impact, local modification to a system re-design	Whereas IEEE.ProjectCost specifies to record an appraisal of the cost of a resolution in dollars, LiDeC instead records an estimation of the impact a resolution would have on the product (in terms of the amount of modification needed). We stipulate that the impact of a resolution on the product will correlate with the cost of applying it; the impact, however, is easier to estimate by the person reporting the defect

Life-Cycle Phase	*IEEE Std. 1044*	*Description*	*LiDeC*	*Description*	*Mapping comment*
	Project risk	Risk associated with implementing a fix	n/a		
	Project quality / reliability	Impact to the product quality or reliability to make the fix	n/a		
	Societal	Impact to society of implementing the fix	**Functional Safety Impact**	Does the defect have an impact on a software component with ASIL-classified requirements (ISO 26262)?	The attribute *Functional safety impact* captures whether the defect may risk causing harm to persons (as defined by the ISO 26262 [34]). This maps to the IEEE Std. 1044 attribute *Societal* in that it captures data about the impact of the defect on environment (e.g. driver, passenger or other persons in the vehicle's surroundings). Note, the *Functional Safety Impact* attribute is captured in the recognition phase
Disposition	**Disposition**	What actually happened to close the anomaly	**Resolution State**	What was the final state of the resolution when defect was closed?	Directly mappable – values modified