Collaborative Large-scale Integrating Project

**OPENCOSS**

Open Platform for EvolutioNary Certification Of Safety-critical Systems

**Specification of the transparent certification service infrastructure**

**D7.4**

<table>
<thead>
<tr>
<th>Work Package:</th>
<th>WP7: Transparent Certification and Compliance-aware Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level:</td>
<td>PU</td>
</tr>
<tr>
<td>Status:</td>
<td>Final</td>
</tr>
<tr>
<td>Date:</td>
<td>27 May 2013</td>
</tr>
<tr>
<td>Responsible partner:</td>
<td>Xabier Larrucea (TECNALIA)</td>
</tr>
<tr>
<td>Contact information:</td>
<td><a href="mailto:xabier.larrucea@tecnalia.com">xabier.larrucea@tecnalia.com</a></td>
</tr>
</tbody>
</table>

**PROPRIETARY RIGHTS STATEMENT**

This document contains information proprietary to the OPENCOSS Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the OPENCOSS consortium.
## Contributors

<table>
<thead>
<tr>
<th>Names</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jerome Lambourg</td>
<td>AdaCore</td>
</tr>
<tr>
<td>Martijn Klabbers, Luc Engelen</td>
<td>Eindhoven Univ. of Technology</td>
</tr>
<tr>
<td>Thierry Ventadour, Michel Ilkiewicz</td>
<td>Inspearit (before DNV ITGS)</td>
</tr>
<tr>
<td>Janusz Studziszba, Michał Siemaszko, Magdalena Gniewek</td>
<td>Parasoft SA</td>
</tr>
<tr>
<td>Jose Luis de la Vara, Sunil Nair, Rajwinder Kaur Panesar-Walawege</td>
<td>Simula Research Laboratory</td>
</tr>
<tr>
<td>Xabier Larrucea, Huáscar Espinoza, Ángel López</td>
<td>TECNALIA Research &amp; Innovation</td>
</tr>
<tr>
<td>Eric Verhulst</td>
<td>Altreonic</td>
</tr>
<tr>
<td>Jan Mauerberger</td>
<td>ikv</td>
</tr>
</tbody>
</table>

## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>2013-05-27</td>
<td>Document creation and initial ToC by TEC</td>
</tr>
<tr>
<td>V0.2</td>
<td>2013-06-07</td>
<td>Added 5.1 chapter regarding Certware project by PSF</td>
</tr>
<tr>
<td>V0.3</td>
<td>2013-07-01</td>
<td>Contribution to Background by TEC</td>
</tr>
<tr>
<td>V0.4</td>
<td>2013-09-10</td>
<td>Complete sections general framework and solutions</td>
</tr>
<tr>
<td>V0.5</td>
<td>2013-09-30</td>
<td>Survey by SIM</td>
</tr>
<tr>
<td>V0.6</td>
<td>2013-10-18</td>
<td>Ready for Peer-review</td>
</tr>
<tr>
<td>V0.7</td>
<td>2013-11-01</td>
<td>Ready for PB review</td>
</tr>
<tr>
<td>V1.0</td>
<td>2013-11-01</td>
<td>Released to the EC</td>
</tr>
<tr>
<td>V1.1</td>
<td>2013-12-01</td>
<td>Updates after release to EC</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

Abbreviations .................................................................................................................. 6

Executive Summary .......................................................................................................... 7

1 Introduction ..................................................................................................................... 8
   1.1 Scope and Purpose ...................................................................................................... 8
   1.2 Relationship with other Deliverables ........................................................................ 9
   1.3 Structure of the Document ........................................................................................ 9

2 Background ...................................................................................................................... 11
   2.1 Measurements and Metrics ..................................................................................... 11
      2.1.1 Notions and definitions ............................................................................... 11
      2.1.2 Why do we want to measure? ........................................................................ 13
   2.2 Approaches for Metrics Management ........................................................................ 14
   2.3 Metrics considered as requirements ........................................................................ 15

3 Conceptual Framework ................................................................................................. 16
   3.1 Transparent Certification Approach ......................................................................... 16
   3.2 Metrics Conceptual Framework ............................................................................... 16
   3.3 Interviews on Metrics/Estimations in Safety-critical Product Certification .............. 17

4 Component Specification ............................................................................................... 21
   4.1 Data Manager Server ............................................................................................... 21
      4.1.1 Metric Storage ............................................................................................... 21
      4.1.2 Measurements Storage ................................................................................. 22
   4.2 Measurement and Transparency Server .................................................................... 22
      4.2.1 Metric Engine .................................................................................................. 22
      4.2.2 Metric Library ................................................................................................ 22
   4.3 Measurement and Transparency Client ..................................................................... 22
      4.3.1 Metric Editor ................................................................................................... 22
      4.3.2 Metric Visualizer .......................................................................................... 22

5 Candidate Technologies ................................................................................................. 23
   5.1 The Certware Project ............................................................................................... 23

6 Conclusion ....................................................................................................................... 32

References .......................................................................................................................... 33

Appendix A. Metrics for Safety-Critical Systems ................................................................. 35
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>Functional decomposition for the OPENCOSS tool platform</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>Elements involved in a measurable model [18]</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Measurement and transparency package</td>
<td>16</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Conceptual classes and relationships among them</td>
<td>17</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>SMM and Metrics manager relationships</td>
<td>17</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Components specifications</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>SMM fundamental approach</td>
<td>24</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Certware snapshot: Measure properties</td>
<td>24</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Certware snapshot: CriticalDefectChangeOrderCount</td>
<td>25</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>Certware snapshot: Direct Measurement</td>
<td>27</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>Certware snapshot: Software Project Management Metrics</td>
<td>28</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>Certware snapshot: SCO.ecore</td>
<td>29</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>Certware snapshot: SCOTest</td>
<td>30</td>
</tr>
</tbody>
</table>
List of Tables

No se encuentran elementos de tabla de ilustraciones.
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCL</td>
<td>Common Certification Language</td>
</tr>
<tr>
<td>CENELEC</td>
<td>Comité Européen de Normalisation Electrotechnique (European Committee for Electrotechnical Standardization)</td>
</tr>
<tr>
<td>DoW</td>
<td>OPENCOSS Description of Work</td>
</tr>
<tr>
<td>DX.Y</td>
<td>OPENCOSS deliverable X.Y</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Framework</td>
</tr>
<tr>
<td>GSN</td>
<td>Goal Structuring Notation</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Comission</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>RTCA</td>
<td>Radio Technical Commission for Aeronautics</td>
</tr>
<tr>
<td>SACM</td>
<td>Structured Assurance Case Metamodel</td>
</tr>
<tr>
<td>TX.Y</td>
<td>OPENCOSS task X.Y</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td>WP</td>
<td>OPENCOSS work package</td>
</tr>
</tbody>
</table>
Executive Summary

Safety assurance and certification are amongst the most expensive and time-consuming tasks in the development of safety-critical embedded systems. European innovation and productivity in this market is curtailed by the lack of affordable (re)certification approaches. Major problems arise when evolutions to a system entail reconstruction of the entire body of certification arguments and evidence. Further, market trends strongly suggest that many future embedded systems will be comprised of heterogeneous, dynamic coalitions of systems of systems. As such, they will have to be built and assessed according to numerous standards and regulations. Current certification practices will be prohibitively costly to apply to this kind of embedded systems.

This document is related with the “Design of the transparent certification service infrastructure” task. The transparent certification term means that the process for certifying components or elements\(^1\) is clearly defined and it can be measured and tracked. Thus the basic elements identified to be defined in our service infrastructure are related to the definition of metrics, and specifically metrics creation, configuration and maintenance. In order to visualise all these metrics we will use a dashboard. Therefore stakeholders will access the information in a proper format.

The core idea is that the OPENCOSS infrastructure, as a whole, shall be able to calculate, from the set of certification artefacts, the information required to apply continuous and incremental certification. Therefore this deliverable is just focused on the service infrastructure that makes a certification process explicit through the use of metrics. While there are a number of specific requirements related to this service in OPENCOSS (see deliverable D7.2), the conceptual basis for metrics is something that can be reused from existing approaches. In this deliverable, we particularly describe one widely accepted specification approach (OMG SMM (Structured Metrics Metamodel)).

The technical aspect is being considered in this deliverable by refining the deliverable D2.3 (OPENCOSS Architecture Design), by implementing requirements from deliverable D7.2 (Detailed requirements for Transparent Certification section), as well as by evaluating the reuse of existing technologies.

The results of this deliverable will serve as basis for T7.5 (Implementation of the process-specific service infrastructure). D7.5 will also be used as input for future versions of D3.2 (Integration requirements and test plan) in order to specify integration and unit test cases.

\(^1\) In our context all elements are those defined inside CCL.
## 1 Introduction

### 1.1 Scope and Purpose

Safety assurance and certification are amongst the most expensive and time-consuming tasks in the development of safety-critical embedded systems. European innovation and productivity in this market is curtailed by the lack of affordable (re)certification approaches. Major problems arise when evolutions to a system entail reconstruction of the entire body of certification arguments and evidence. Further, market trends strongly suggest that many future embedded systems will be comprised of heterogeneous, dynamic coalitions of systems of systems. As such, they will have to be built and assessed according to numerous standards and regulations. Current certification practices will be prohibitively costly to apply to this kind of embedded systems.

The OPENCOSS project aims at devising a common certification framework that spans different vertical markets such as the railway, avionics and automotive domains, and to establish an open-source safety certification infrastructure (hereafter referred to as OPENCOSS tool platform). The infrastructure is being realised as a tightly integrated solution, supporting interoperability with existing development and assurance tools. The ultimate goal of the project is to bring about substantial reductions in recurring safety certification costs, and at the same time increasing product safety through the introduction of more systematic certification practices. Both will boost innovation and system upgrades considerably.

This document is related with the “Design of the transparent certification service infrastructure” task. A service infrastructure term has been used in [1], [2] and it refers to a set of services (a.k.a. functionalities, features) that should be defined in order to provide the required basic elements for a transparent certification. In this sense, the transparent certification term means that the process for certifying components or elements is clearly defined and it can be measured and tracked. Thus the basic elements identified to be defined in our service infrastructure are related to the definition of metrics, and specifically metrics creation, configuration and maintenance. In order to visualise all these metrics we will use a dashboard. Therefore stakeholders will access the information in a proper format.

During our project execution we have realised that there is a scarce of or even no consensus on terms used to define metrics, indicators and measures. This fact has been studied in literature [3] stressing the same misinterpretation of terms. In fact some stakeholders use the term “indicator” instead of “metric”. There are some relevant research efforts such as [4] where authors used the term indicator for categorising a set of metrics. For this reason we have included a subsection called “Notions and definitions” where we clarify the terminology used from our perspective.

The core idea is that the OPENCOSS infrastructure, as a whole, shall be able to calculate, from the set of certification artefacts, the information required to apply continuous and incremental certification, for example by highlighting missing or invalid certification artefacts and obsolete traceability links. Therefore this deliverable is just focused on the service infrastructure that makes a certification process explicit through the use of metrics. There are two general questions guiding our work:

1. What information is required by each stakeholder to achieve the required level of transparency and trust?
2. What is the best way to represent such information given the existing standards, practices, and technologies?

---

2 In our context all elements are those defined inside CCL.
D7.4 corresponds to a refinement of the overall OPENCOSS tool platform architecture (Figure 1) for transparent certification.

### 1.2 Relationship with other Deliverables

D7.4 is related to other OPENCOSS deliverables, which have served as input, with which consistency must be kept, or that will use its results. These deliverables are the following ones:

- D2.3 (OPENCOSS platform architecture) presents the overall OPENCOSS tool platform architecture, whose evidence management-related components are specified in detail in WP6.
- D2.4 (Detailed specification of usage scenarios) complement this deliverable by presenting details about how the OPENCOSS platform will be used and some initial user interface mock-ups.
- D3.2 (Integration requirements and test plan) will specify the test cases for validation of the components specified in this deliverable.
- D7.2 defines a set of low level requirements for transparent certification categorised into three groups.
- D7.3 defines a compliance aware service infrastructure which is the basis for D7.4.
- D7.5 provides an implementation of the process-specific service infrastructure.
- D7.6 contributes to the methodological guide of the process-specific service infrastructure.

### 1.3 Structure of the Document

The rest of the deliverable is structured as follows. Section 2 introduces definition and a general approach for measurements. Section 3 outlines a conceptual framework and section 4 components specification
meeting the specifications. Section 5 describes a technology candidate to be used in our service infrastructure. Finally we provide a conclusion.
2 Background

This section presents the background work on which the creation of this deliverable has been based. Such work corresponds to the results of other OPENCOSS deliverables and to activities that have been specifically performed for task 7.3. In fact we realise that a certification process is a hard task involving human decisions during the whole certification process. There are several aspects to be considered during a certification life cycle, especially in safety environments. Some of these aspects such as human aspects are defined previously. Therefore our scope is limited to the definition of an initial infrastructure for providing support to a transparent certification process. The following subsections outline an infrastructure definition to make the certification process explicit. In this sense an explicit certification process involves several aspects such as the definition of specific criteria, artefacts and hence metrics as defined in [18]. However not all aspects are required to be defined in our service infrastructure at this stage.

2.1 Measurements and Metrics

2.1.1 Notions and definitions

While software engineering has quite well defined metrics allowing to assess a software engineering project, such metrics are not well defined for e.g. safety engineering. However, we can reuse metrics concepts from the software domain as there is some similarity. Software measurement methods [14] have been discussed in literature and are required when putting into practice an improvement program, a software development process, an assurance infrastructure or just to provide a feedback [15] amongst others. One of the key elements is the collection of specific data [16] in order to provide a diagnosis about a system whether critical or not. This diagnosis is based on quantitative and qualitative attributes, and all of them represent some of the building blocks of a framework. This is not a straightforward activity: “For safety indicators with their long-term effects, diagnosis is even more difficult” [17].

The measurement process has been defined by the IEEE as showed in the following figure:
In our case, a measurement process and framework [11] should be defined for safety-critical systems. This is especially relevant for achieving transparency across certification processes whereby we need to have a clear idea about the system’s status at any point in time.

There are different measurement frameworks such as [19] where authors define precisely software measurements concepts and a life cycle. From a more general point of view, a Goal/Question/Metric approach [15], [20], or a Quality Function Deployment approach [21] have been used to organise approaches in the context of quality assurance. Both approaches consider using metrics, and these metrics should include quantitative as well as qualitative aspects.

From a qualitative point of view, safety critical systems includes a wide variety of options and conditions to be taken into account [22]–[26] in order to assure the safety of a system. Normally, these qualitative criteria are related to safety cases concepts [27], and different solutions include approaches such as MDE [26] or petrinets [28].

From a quantitative point of view, measurable metrics have been used in the context of safety critical systems [29]. In this sense we can point out which metrics are the most appropriate and related to the transparency of the certification process.

Figure 2. Elements involved in a measurable model [18]
The notion of “Measurement” is the relationship between a measure and a “measurand” [30]. In our context the “measurand” will be an element defined in the CCL. “Measure” is a term referring to “metric” and provides a method to calculate a value with a set of potential constraints. The assigned value can be numerical or not depending on the domain and on the type of measure. This value can be in a range of potential values. Each measure has a set of measurements. These measurements are calculated automatically or manually by using human interactions. Ideally an automated support [10] will be more appropriate. However we consider that users should be able to interact in order to calculate the appropriate values. On this process, users can make use of external tools. The measurement process as it is defined by the IEEE [18] they distinguish the following elements:

- **Indicator**: measure that provides an estimate or evaluation of specified attributes derived from a model with respect to defined information needs
- **Base measure**: measure defined in terms of an attribute and the method for quantifying it. A base measure is functionally independent of other measures.
- **Derived measure**: measure that is defined as a function of two or more values of base measures

These three main elements are defined as a kind of measure. In addition we are aware of the current misinterpretation of terms [3], and some stakeholders use the term “indicator” instead of “metric” or “measure”[4]. Relevant industrial efforts such as Structured Metrics Metamodel (SMM) [30] use “measure” term for referring to metrics and indicators. Therefore from a conceptual point of view, we are going to use the concept measure for representing any kind of measure.

2.1.2 Why do we want to measure?

Poor size and complexity estimation is one of the main reasons major software-intensive acquisition programs ultimately fail. Size and complexity are is the critical factors in determining cost, schedule, and effort. The failure to accurately predict (usually underestimated) will result in budget overruns and late deliveries undermining the confidence and eroding support for the project. Size and complexity estimation is a complicated activity, the results of which must be constantly updated with actual counts throughout the life cycle. Size measures include source lines-of-code, function points, and feature points. Complexity is a function of size, which greatly impacts design errors and latent defects, ultimately resulting in quality problems, cost overruns, and schedule slips. Complexity must be continuously measured, tracked, and controlled. Another factor leading to size estimate inaccuracies is requirements creep which also must be baselined and diligently controlled.

Any human-intensive activity, without control, deteriorates over time. It takes constant attention and discipline to keep software acquisition and development processes from breaking down — let alone improving them. If one does not measure, there is no way to know whether processes are on track or whether they are improving. Measurement provides a way to assess the status of the project to determine if it is in trouble or in need of corrective action and process improvement. This assessment must be based on up-to-date measures that reflect current program status, both in relation to the project’s plan and to models of expected performance drawn from historical data of similar projects. If, through measurement, the project is diagnosed to be in trouble, on should be able take meaningful, effective remedial action (e.g., controlling requirements changes, improving response time to the developer, relaxing performance requirements, extending the schedule, adding financial and other resources, or any number of options).

Measurement provides benefits at the strategic, program, and technical levels. A good measurement program is an investment in success by facilitating early detection of problems, and by providing quantitative clarification of critical development issues. Metrics give the ability to identify, resolve, and/or curtail risk issues before they surface. Measurement must not be a goal in itself. It must be integrated into
the total project life cycle — not independent of it. To be effective, metrics must not only be collected — they must be used.

2.2 Approaches for Metrics Management

Ariane V launch failure, the losses of the Mars Polar Lander, the Mars Climate Orbiter, are some of the examples mentioned by John Knight in one of his seminal papers [31]. As mentioned [31] reliability is one of the major concerns for this kind of software-intensive systems but it is not enough for assuring safety in critical systems. Other factors should be considered such as defective software specifications, software engineering and systems engineering interplay, development time and effort, and so forth. Therefore all these systems require setting up techniques for measuring and estimating failures. Metrics area has been and it is still an area of research as well as on the SWEBOK v3. Its relevance is especially important for safety critical systems where “metrics are not always precisely defined, limiting the reproducibility of results, and lacking a directional quality” [32].

Therefore in our safety-critical context, we need to define an appropriate measurement framework [19] for managing metrics for assuring safety and especially metrics related to certification. Ideally, this topic covers a wide set of approaches and complex implementations [33]. A certification process is normally related to the certification of products, processes and personnel [6]. Our approach is focused on products and processes, and a certification process in safety environments [5] is not a straightforward activity. We need to define and use a set of metrics or criteria appropriately. In fact our aim is to concretize a set of services for metrics creation, configuration and maintenance. It will be suitable to use a specific dashboard in order to allow stakeholders access the information in a proper format.

What kinds of metrics are really needed? There are plenty of metrics that can be potentially included and implemented in our approach. Our focus is mainly on metrics providing data for transparent certification. Traditionally complexity metrics [7] can be divided on “static metrics” which measurements are taken at one particular point in time, and “history metrics” which include a set of measurements. Our safety related metrics are applied to products or/and processes from a general point of view [7], [15], and from a safety critical systems point of view [9]. On the process side, process metrics can have automated support [10] complexity metrics in safety critical [34]

Metrics can be classified into three categories: product metrics, process metrics, and project metrics [35]. As stated previously and aligned to [6], our approach for certification process will be based on products and processes. Therefore we are going to use metrics related to products and processes. Size, complexity, design features, performance, are some of the products metrics. Effectiveness of defect removal, MTTF, activity duration are some examples for process metrics.

All these metrics should be shown through an appropriate dashboard. Part of the existing approaches on metrics management such as FMEA approach [36], can be used for visualising metrics and measurements. Implementing a software metrics program such as in Nokia [37] it is not an easy task, and we even need to take into account architectural considerations on certification [38].

At this point we would like to remark that our intention is not to provide a set of management metrics such as in [39] where metrics are means for managing at high level an organisation. However we need to clearly state that our certification metrics can be related to management metrics in somehow because it is information required by managers. As an improvement in a future, this approach should incorporate an additional hazard analysis in order to avoid an accident before it occurs [40].
2.3 Metrics considered as requirements

D7.2 provides a set of low level requirements for the OPENCOSS approach which are inputs for D7.3 and for this D7.4. In fact this deliverable D7.4 is related to requirements for transparent certification, and therefore we need to identify which are the requirements for this transparent certification. Additionally we define the conceptual framework to be used on these systems, outlining some of the most appropriate metrics. In order to define this conceptual framework we identified on D7.2 the following three groups of low level requirements from the six identified sets:

- **Safety Assurance and certification process metric**: as stated in [5] safety cases approach lacks of a highly prescriptive and domain specific nature existing in other engineering domains. Therefore some metrics related to safety assurance are required for certifying critical systems [6]. The identified metrics are:
  - Time efficiency metrics
  - Resource efficiency metrics
  - Support metrics estimation
  - Completeness metrics estimation
  - Accuracy metrics estimation
  - Assurance/certification metrics presentation
  - Compliance coverage metrics estimation
  - Certification costs specification
  - Claims coverage metric estimation
  - Safety goals coverage metric estimation

- **Safe Product Metrics**: product metrics [7] and a validation of product metrics framework such as [8] are considered to be included in our approach. The identified metrics are:
  - Software Complexity metrics
  - PFD/PFH metrics
  - Product architecture metrics
  - Safe Product metrics presentation

- **Safe Process Metrics**: process related metrics are some of the metrics to be considered such as[9]–[13]. The identified metrics are:
  - Process metrics
  - Process metrics presentation

All these elements are going to be refined through a survey or a set of semi-structured interviews on Metrics/Estimations in Safety-critical Product Certification that is performed iteratively during the project. Some of the preliminary results are included in the following sections, and some metrics in Annex A.
3 Conceptual Framework

This section presents a conceptual framework for achieving transparency in the certification approach in safety critical contexts. This framework enables users to define, use and relate metrics to artefacts used during a certification process. The following questions are some of the foundations for developing the proposed framework:

(1) What information is required by each stakeholder to achieve the required level of transparency and trust?
(2) What is the best way to represent such information given the existing standards, practices, and technologies?

Our approach is aligned with the Goal/Question/Metric approach[15]. However, first of all, we need to define an infrastructure for defining and representing metrics and how to relate them to CCL.

3.1 Transparent Certification Approach

Transparency in the certification process is needed to provide a clear status and knowledge of an on-going certification process

Some of the current problems encountered during process assessments are amongst others [13]:

- auditors have a partial view of the certification process;
- a lack of objective criteria for determining the project’s status.

In addition, it should include a clear assessment method with respect to a wide variety of standards.

3.2 Metrics Conceptual Framework

The approach is based on the current SMM metamodel [30] for representing and storing metrics. Figure 1 represents conceptually the measurement and transparency package and highlights some interfaces for providing metrics information.

![Figure 3. Measurement and transparency package](http://www.tuev-sued.de/uploads/images/1156764217583933590252/klima_goldstandard_e.pdf)

---

4 http://www.emci.nl/index.php/id_structuur/10593/general-information.html
Figure 4 represents the four main classes to be considered and implemented. MetricLibrary shall contain a set of predefined metrics and measurements to be used and to be related to artefacts/evidences for transparent certification. MetricEngine shall provide the means for calculating values to each metric and controlling metrics at any stage. MetricInfo implements the interfaces for showing metrics and measurements. AssuranceProject contains information about the current safety critical certification process.

![Conceptual classes and relationships among them](image)

As our approach is based on SMM Figure 5 describes the main components to be used by OPENCOSS approach, and its relationships to CCL elements and the metrics engine.

![SMM and Metrics manager relationships](image)

### 3.3 Interviews on Metrics/Estimations in Safety-critical Product Certification

This section presents the design of an interview schema for T7.3. An interview is an empirical method for collecting information from or about people to describe, compare, or explain their knowledge, attitudes, and behaviour. An interview is often conducted either in an informal or formal way. The primary means of gathering data are interviews or questionnaires through taking a sample from the population to be studied. The results are then analysed to derive descriptive and explanatory conclusions.
The purpose of the survey for T7.3 is to identify the main elements that characterize the development of safety-critical systems and their safety assessment. Within the scope of this study, we have formulated the following research questions (RQ).

**RQ1) What metrics do practitioners need for planning and/or monitoring the development of a safety-critical system?**

The objective of this RQ is to determine what information is relevant to evaluate the status of the development of a safety-critical system (e.g., cost and schedule control metrics). This information is expected to allow a company to know the actual progress of the safety activities within a project. The following questions can be performed during the scope of this RQ:

1. Is the effort spent in the development of a safety-critical system measured? How?
   - Example: Hours

2. Is the cost of the development of a safety-critical system measured? How?
   - Example: Euros

3. How accurate do you consider the current effort and cost measurement to be? What hinders a more accurate measurement?

4. What metrics specific to the different system lifecycle phases are relevant for safety-critical systems? Are there any specific metrics, which might not be important for other types of systems?
   - NOTE: The phases to ask about must be agreed upon: system inception, safety analysis, requirements, design, coding, V&V, change management, maintenance, etc.
   - Example: number of defects

5. Do you manage metrics that refer to several system lifecycle phases? Which ones?
   - Example: Outcomes from the requirements process that have been subject to V&V

6. In relation to the tools used to support the development of a safety-critical system, what metrics are related to these tools used?
   - Example: ratio of qualified tools

7. Do you make any measurement regarding reuse of existing elements (e.g., a software component) for a safety-critical system?
   - Example: number of reused components

8. Can you think of metrics that would be relevant for the development of safety-critical systems for a given domain but not for others? What distinguishes these metrics?

9. What aspects do you think that are currently not measured but should be to increase the transparency of the development of a safety-critical system?

**RQ2) What metrics related to a system’s properties do practitioners need?**

The objective of this RQ is to determine what information is relevant to evaluate the status of a safety-critical system under development (e.g., technical performance metrics). This information is expected to allow a company to compare the actual behaviour of a product with the expected behaviour, as requested in the applicable standard and requirements. The following questions can be performed during the scope of this RQ:

1. Are metrics related to the hazard of a system used? Which ones?
   - Example: Number of open hazards

2. Is any measurement done regarding the safety requirements of a system? How?
   - Example: Safety requirements whose fulfillment has been successfully tested

3. What metrics are used that require traceability between different system properties?
   - Example: see the previous one

4. What aspects are taken into account when reusing a component for a system?
   - Example: some environmental condition

5. Are any metrics related to past operation of reused components used?

6. Example: hours of operation without failure What other metrics related to software aspects are measured?
   - Example: code lines of a function
o Example: Is the size of safety-critical software measured? What metric is used? - KLOC
7. Are metrics specific to hardware aspects measured? Which ones?
8. What other metrics related to system’s properties are relevant for assessing system’s safety? (Check safety standards)
   o Example: MTBF
9. What aspects do you think that are currently not measured but should be to increase the transparency of the about the properties of a system?

RQ3) What metrics related to adherence to project processes do practitioners need?
The objective of this RQ is to identify those process metrics that are more important for transparency of safety assurance and certification (e.g., system engineering process metrics). This information is expected to allow a company to know if the enacted process will enable safety assurance and certification, according to a pre-defined process. The following questions can be performed during the scope of this RQ:
1. Do you track the degree of compliance with a safety standard? For what elements of the standard?
   o Example: Work products to create
2. Is any state defined for these elements when tracking the degree of compliance?
   o Example: internally approved
3. Why is the degree of compliance with some elements of a safety standard not tracked?
   o Example: Requirements
4. What are the elements of a safety standard for which it is more important to track the degree of compliance? Why?
   o Example: Work products and activities
5. Do you have to create structured safety cases as a means to assure safety or compliance with safety standards? If yes, do you maintain specific metrics for safety cases?
   o Example: number of claims whose evidence has to be approved
6. Are you following a specific strategy? 5
   1. Are you following one or more of the following specific strategies?
      o Strategy 1: determine the project certification scope early
      o Strategy 2: determine feasibility of certification
      o Strategy 3: select an independent assessor (if used)
      o Strategy 4: understand your assessor’s role (if used)
      o Strategy 5: assessment communication is key
      o Strategy 6: establish a basis of certification
      o Strategy 7: establish a “fit and purpose” for your product
      o Strategy 8: establish a certification block diagram
      o Strategy 9: establish communication integrity objectives
      o Strategy 10: identify all interfaces along the certification boundary
      o Strategy 11: identify the key safety defensive strategies
      o Strategy 12: define built in test (BIT) capability
      o Strategy 13: define fault annunciation coverage
      o Strategy 14: define reliance and expectation of the operator/user
      o Strategy 15: define plan for developing software to appropriate integrity level
      o Strategy 16: define artifacts to be used as evidence of compliance
      o Strategy 17: plan for labor-intensive analyses
      o Strategy 18: create user-level documentation
      o Strategy 19: plan on residual activity
      o Strategy 20: publish a well-defined certification plan
   2. Which of the following strategies for implementation are you implementing?
      o Strategy 1: have a well-defined, repeatable peer-review process

7. What aspects do you think that are currently not measured but should be to increase the transparency of the adherence of a project to safety standards?

The action plan for conducting the survey is as follows.

1. Design of the interview schema
   The interview schema will correspond to a questionnaire in which the interviewees will have to provide information about their preference and their current practices in relation to different aspects or metrics of the development and safety assessment of safety-critical systems.
   The instrument is currently under design, and we are in the process of selecting the set of aspects and metrics about which the questionnaire will finally ask. We have searched for metrics in the literature, other deliverables (e.g., D3.1), and the data provided by the industrial partners. Details about these metrics are provided in Appendix A. We also plan to study the possibility of defining metrics based on the CCL concepts and ask experts about other possible metrics.

2. Validation of the instrument
   Once a first version of the instrument is available, it will have to be validated by asking experts about the questionnaire. The purpose will be to get feedback on the understandability and reliability of the questionnaire, as well as on the time that filling the questionnaire will take. Refinements on the instrument might be performed as a result of this activity.

3. Data collection
   Afterwards, practitioners will be kindly asked to fill the questionnaire in order to get information from them. The data sampling strategy will have to be defined before contacting practitioners.

4. Data analysis
   Next, the data collected will have to be analysed in order to answer the RQs formulated.

5. Results report
   Finally, the results of the survey, as well as the conclusions drawn from them, will be presented in a document, probably in some new version of this deliverable. The results will mainly consist in a set of aspects and metrics whose measurement must be supported by the OPENCOSS tool platform.
4 Component Specification

This section presents a first overview of measurements and transparency modules specifications. The following subsections present a high level description of each module to be implemented. Figure 6 describes a Model/View/Controller approach for the transparency certification service infrastructure. GUI will provide mechanisms for defining and visualising metrics. Each metric will be connected to an artefact/evidence playing a key role on the certification process. Core components are mainly dedicated to the metrics engine and to the metric library management. Data Management will store metrics, measurements and the relationship to artefacts used for certification.

![Figure 6. Components specifications](image)

4.1 Data Manager Server

4.1.1 Metric Storage

The Metric Storage component shall manage a repository to store metrics or measures. At a first stage basic definitions are allowed not including derived metrics.
4.1.2 Measurements Storage

The Metric Storage component shall store the numeric values for each measures. Several measures can be stored, and a relationship to the artefacts used for the measurements are also included.

4.2 Measurement and Transparency Server

4.2.1 Metric Engine

The Metric Storage component shall provide the means to compute each metric. Each metric is related to a set of CCL elements. These values should be introduced by users, and thus it will require an intensive human machine interaction for providing the required data.

4.2.2 Metric Library

Metrics are defined and stored in a library. Metric Engine makes use of these metrics.

4.3 Measurement and Transparency Client

4.3.1 Metric Editor

Metric Editor component shall be able to edit a measure, and to relate it to a CCL element or to a set of them. Each measure can be associated to the elements of the OPENCROSS baseline. Each assurance project can define its own set of metrics. CRUD (Create, Read, Update, Delete) functionalities can be performed for metrics.

4.3.2 Metric Visualizer

This component is mainly used to CRUD functionalities for measurements.
5 Candidate Technologies

This section introduces a candidate technology to be adapted to our OPENOCSS service infrastructure. Another candidate is to develop the aforementioned components in order to do not include additional elements that they are not required and which inclusion will be dauntingly complex.

5.1 The Certware Project

5.1.1 Introduction:
CertWare is a tool which helps to collect Management and Quality indicators metrics of software projects. Walker Royce in “Software Project Management: A Unified Framework” propose the following Management and Quality metrics:

Management Indicators:
- work and progress
- budget cost and expenditures
- staffing and team dynamics

Quality Indicators have:
- change traffic and stability
- breakage and modularity
- rework and adaptability
- mean time between failures and maturity

CertWare supports almost all indicators from the above list, it does not support staffing and team dynamics from management indicators.

Walker's Software Project Management statistics depend on different software change order (change requests) types.

Software change order (SCO) types could be:
- critical defects – type 0
- normal defects – type 1
- improvements – type 2
- new features – type 3

In order to store metrics CertWare uses the SPM (Software Project Management) metamodel which is built upon the SMM metamodel (Software Metrics Metamodel).

SMM is a definition of common metamodel which can be used to express metrics of software project. Goal of SMM is to facilitate the interoperability of measurements of software artifacts.

SMM - “Structured Metric Metamodel (SMM) is a publicly available specification from the Object Management Group (OMG). Measures are comparable evaluations and, as such, are critical to software engineering and architecture driven modernization. Software measurement methods produce comparable evaluations of software or application artifacts. Counts such as number of screens, lines of code, number of methods, etc. all quantify the size of artifacts along a single dimension ... ”[30]
There are following main classes in SMM:

- **Measure** – represents metrics type, for example: critical defect change order count, maturity measure (mean time between defects)
- **Measurement** – holds a value, the measurement result, for example: number of type 0 SCOs in SCO model, value of maturity measurement
- **Observation** – represents who, where and when done the measurement

**SMM metamodel** in Certware:

SPM adds Management and Quality indicators metrics except the budget cost and expenditures.
CertWare supports also another metamodel: **SCO** (Software Change Order) metamodel to collect in change-order statistics. In CertWare, SPM model gathers the metric values from SCO model and with its own raw statistics can compute the metrics in SPM. The below paragraphs present SPM and SCO metamodels in details.

### 5.1.2. CertWare SPM (Software Project Management) metamodel

In SMM metamodel there is SmmModel class which represents the aggregation of all the elements of the SMM.

In SPM metamodel there are: ProjectModel and ProjectCommit classes which inherit from SmmModel. ProjectModel has model elements which are belong to it, for example: ProjectCommit belongs to ProjectModel.

#### SPM metamodel in Certware:

![Figure 9. Certware snapshot: CriticalDefectChangeOrderCount](image)

The following measures have been defined in SPM:

1) The **statistics** measures:
   - order counts (measures) – representing number of change orders (change requests) in software project:
CriticalDefectChangeOrder, NormalDefectChangeOrderCount, ImprovementChangeOrderCount, NewFeatureChangeOrderCount, TotalChangeOrderCount, CriticalAndNormalChangeOrderCount

- SLOC counts (measures) – representing statistics about code or safety case, could be also function points, object points, scenarios, test cases: BrokenCaseSizeMeasure, FixedCaseSizeMeasure, TotalCaseSizeMeasure, BaselineCaseSizeMeasure
- hours counts (measures) – representing statistics about time of work UsageTimeMeasure, DevelopmentEffortMeasure, RepairEffortMeasure

In SMM metamodel there is CollectiveMeasure class defined. It represents measure which accumulates measurements from the BaseMeasure in accordance to defined accumulator (here accumulator is a sum – AdditiveMeasure).

Almost all of them inherited from CollectiveMeasure.

Some measures can accumulate two basic measures. In SMM metamodel there is BinaryMeasure which accumulates measurements of two entities related to the given entity. Measurement for BinaryMeasure is calculated in accordance to the given attribute: “functor” - it identifies the binary function which combines two base measurements.

For example in CertWare SPM there is:

CriticalAndNormalChangeOrderCount which is BinaryMeasure and it has:
- BaseMeasure1: Critical Defect Change Order Count
- BaseMeasure2: Normal Defect Change Order Count
- functor: sum

II) The metrics measures – they are computed based on statistics described in #I)

In CertWare SPM there are two characteristics defined: EndProductQuality and InProgressIndicator. They are inherited from Characteristic class from SMM – representation of a hierarchy of measure. These characteristics (EndProductQuality, InProgressIndicator) are groups of measures:

EndProductQuality – group of metrics
InProgressIndicator – group of the metrics from EndProductQuality plotted over time

In CertWare, if they are not defined in the ProjectCommit they will be created by “Compute Metrics” function (described below).

These characteristics indicates static (EndProductQuality) and dynamic (InProgressIndicator) metrics.

**EndProductQuality** characteristic group contains the following measures:

a) AdaptabilityRatioMeasure - average effort per SCO
b) ReworkRatioMeasure – percentage of rework effort
c) ScrapRatioMeasure – percentage of product scrapped
d) ModularityMeasure – average breakage per SCO
e) MaturityRatioMeasure – mean time between defects
f) MaintainabilityMeasure – maintenance productivity

All of these measures inherit from: RatioMeasure -> BinaryMeasure

**InProgressIndicator** characteristic group contains the following measures:
a) ReworkStabilityMeasure – BinaryMeasure -> DimensionalMeasure – breakage minus fixes plotted over time
b) ReworkBacklogMeasure – RatioMeasure -> BinaryMeasure – currently open rework
c) ModularityTrend – TrendMeasure -> CollectiveMeasure – modularity plotted over time
d) AdaptabilityTrend - TrendMeasure -> CollectiveMeasure – adaptability plotted over time
e) MaturityTrend - TrendMeasure -> CollectiveMeasure – maturity plotted over time

Running metrics calculation in CertWare

All metric measures are created by “Compute Metrics” function in CertWare GUI if they do not exist in InProcessIndicator and EndProductQuality characteristics.

Measures from EndProductQuality and InProgressIndicator groups are described in document: “CertWare Workbench Metrics” - SafetyCaseMetrics.pdf.
In order to compute metrics in CertWare user should go to “ProjectCommit” and press the option: “CertWare -> Compute Metrics”. This action produces the following results for each measure groups:

- “Aggregated Measurement Computed” - measurement instance is created containing the calculated measure value
- “Observation”: Observation Computed – observation instance is created representing who and when computed the metrics.

Additionally, after the above metrics calculation, “Commit Metrics” and “Raw Statistics” sections are presented in Software Project Management Metrics view. They can be opened by pressing “Open in SPM View”. This view shows metrics recently calculated and their raw statistics. Metrics are calculated in accordance with “CertWare Workbench Metrics”. Raw statistics are the current, last measure data.

![Certware snapshot: Software Project Management Metrics](image)

CertWare presents computed metric statistics in tree representation layout, it is not drawn as graphical charts.

5.1.3. SCO in CertWare

SCO (Software Change Order)

CertWare supports the SCO metamodel, which can be used to collect change order (change request) statistics, for example related statistics for the repair effort, broken lines, and fixed lines associated with each model artifact.
SCO metamodel defines ArtifactIdentifier class which has:
- criticalDefectChangeOrders
- normalDefectChangeOrders
- improvementChangeOrders
- newFeatureChangeOrders
- totalChangeOrders

Change order types are:
0 (Critical) – for critical defects
1 (Normal) – for normal defects
2 (improvements) – for improvements
3 (NewFeatures) – for new features
4 (Total) – total defects

**SCO metamodel in Certware:**

![Certware snapshot: SCO.ecore](image)

Data (metrics values) can be inserted into SCO model in the following ways:
• in CertWare automatically from the version control (Egit) - “Gather change history to SCO file” option from the selected project
• manually - in CertWare EMF Editor
• from external application lifecycle management (ALM) tools – it has to be implemented

We can collect the data for the statistics in SCO model. From SPM ProjectCommit model we can run CertWare -> “Gather Change Order Count” and show the SCO file with SCO model to gather raw statistics in SPM model. SPM then can compute static and dynamic metrics together with its own statistics and the gathered statistics from the file.

After analyzing the SCO model (by manually work) we can see that for the SPM measures the measurement is created:

“Direct Measurement Computed” which has value from SCO file. This Measurement has “Observation Computed” which is telling which observer is for current measurement. In this case for “Observation Computed” is Observer: CertWare.

Certware action: “Gather Change Order Counts” adds new measurement to the measures: CriticalDefectChangeOrder, NormalDefectChangeOrderCount, ImprovementChangeOrderCount, NewFeatureChangeOrderCount, TotalChangeOrderCount, CriticalAndNormalChangeOrderCount and TotalCaseSizeMeasure, BaselineCaseSizeMeasure and UsageTimeMeasure.

5.1.4. Conclusions – possible use in OPENCOSS platform

We can use SMM, SPM metamodels in OPENCOSS in order to have common, interoperability format of metrics (SMM is specified by OMG).

We can see the following possible ways to leverage SPM, SCO and CertWare functionality on OPENCOSS platform:
• We can use SPM metamodel in OPENCOSS platform to store metrics data. The actual metrics values need to be calculated by OPENCOSS implementation, as SPM provides the metamodel only. We could enter the actual metrics data in the following ways:
  • enter the statistics values in EMF editor
  • we can use JAVA API to create the statistic instances with the data from OPENCOSS system
  • we can load the file with the actual metrics value (unfortunately the xml format is not well specified).
• In OPENCOSS we could use the CertWare function of measures calculation from EndProductQuality and InProgressIndicator groups (described above - running the CertWare “Compute Metrics” option)

• In OPENCOSS we could alternatively use the following metamodels for metrics:
  • both SPM and SCO metamodels – SCO model to hold raw data and then pass it to SPM model for further statistical calculations, so we should:
    ▪ Write the tool to collect OPENCOSS data in SCO model (we need to develop this) – SCO metamodel given by CertWare
    ▪ Gather change order counts in SPM model – read SCO model to gather raw statistics – action given by CertWare
    ▪ Compute metrics – together with SPM own raw statistics compute the static and dynamic metrics – action given by CertWare
  • SPM (without SCO) – we can store the calculated metrics value directly in SPM, and have further statistical calculation there as well
  • SMM alone - define our new metamodel built upon SMM which will store statistics and metrics supported by OPENCOSS platform
6 Conclusion

As described during the introduction we have the following questions guiding our efforts:

(1) What information is required by each stakeholder to achieve the required level of transparency and trust?

Obviously this question is focused on users’ perspectives and the results are coming from the interviews we are carrying on. However our consortium has helped us on identifying the basic elements for the infrastructure, and the metrics identified in Annex A.

(2) What is the best way to represent such information given the existing standards, practices, and technologies?

Our conceptual model identifies the main elements for supporting a measurement process. “Measure” is the main element and its relationship with CCL elements are the basic element in our service infrastructure. By using these building blocks we will be able to define metrics, derived metrics and indicators.

As result this document describes the OPENCOSS tool infrastructure for metrics management. The OPENCOSS approach for metrics management is based upon a wide accepted approach from OMG. This is the SMM (Structured Metrics Metamodel) standard specification, which is used as the backbone of the OPENCOSS metrics and estimations conceptual framework. As part of the conceptual framework, we also present a semi-structured interview schema to be deployed in upcoming months to better understand the kinds of solutions and metrics that can be used in industry. The purpose of the survey is to identify metrics that characterize the development of safety-critical systems and their safety assessment.

Our ultimate goal is that the OPENCOSS platform allows users to check such metrics. The lack of performance metrics and certification efficiency limits the capability to assess long-term costs, savings and benefits associated with safety-critical system development and subsequent recertification activities. The core goal of the OPENCOSS service infrastructure for transparent certification is to be able to calculate, from the set of certification artefacts, the information required to apply continuous and incremental certification, for example: missing or invalid certification artefacts and obsolete traceability links. This is being concretized into a set of services for metrics creation, configuration and maintenance. Specific dashboards shall be created to allow stakeholders to access the information in a proper format.

The results of this deliverable will serve as basis for T7.5 (Implementation of the process-specific service infrastructure). D7.5 will also be used as input for future versions of D3.2 (Integration requirements and test plan) in order to specify integration and unit test cases.
References


Appendix A. Metrics for Safety-Critical Systems

This appendix presents a set of metrics for the development of safety-critical systems and their safety assessment. They have been collected from the literature, other deliverables, and data provided by the industry partners. The metrics are currently under study to decide upon their inclusion in the survey for T7.3 (Section 3.4).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Expected/preferred value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Hazards</td>
<td>A hazard is any real or potential condition that can cause: injury, illness, or death to personnel; damage to or loss of a system, equipment, or property; or damage to the environment.</td>
<td></td>
</tr>
<tr>
<td>Number of risks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of software hazards</td>
<td>A hazard that contains one or more software causes (this can be expressed as a percentage of total hazards)</td>
<td>Number or percentage of total hazards</td>
</tr>
<tr>
<td>Number of causes per hazard</td>
<td>The root or symptomatic reason for the occurrence of a hazardous condition</td>
<td></td>
</tr>
<tr>
<td>Number of software causes</td>
<td>A cause of a hazard that is rooted in software</td>
<td></td>
</tr>
<tr>
<td>Number controls per cause</td>
<td>A control is an attribute of the design or operational constraint of the hardware/software that prevents a hazard or reduces the residual risk to an acceptable level.</td>
<td></td>
</tr>
<tr>
<td>Number of verification actions per control</td>
<td>A verification action is a method for assuring that a hazard control has been implemented and is adequate through test, analysis, inspection, simulation, or demonstration.</td>
<td></td>
</tr>
<tr>
<td>Number of software controls</td>
<td>A control that is part of the software of a system.</td>
<td></td>
</tr>
<tr>
<td>Percentage of software safety requirements</td>
<td>The number of software safety requirements divided by the number of software requirements.</td>
<td>Reasonable value based on heuristics and experience</td>
</tr>
<tr>
<td>Number of high-level safety requirements</td>
<td>One or more high-level safety requirements may exist for mitigating or eliminating a hazard having unacceptable risk. These requirements are sometimes called safety goal.</td>
<td></td>
</tr>
<tr>
<td>Number of low-level safety requirements</td>
<td>One or more low-level safety requirements may exist for each high-level safety requirement.</td>
<td></td>
</tr>
<tr>
<td>Process compliance (e.g., SEI, CMMI +SAFE, safety standards like DO178B etc.)</td>
<td>A process for developing safety critical products and a measure of how close to the standard a developer is.</td>
<td>100% meaning that the organization complies with all aspects of the preferred process</td>
</tr>
<tr>
<td>Percentage level of rigor (or analysis and testing effort)</td>
<td>It compares the level of rigor employed to what is expected of software with a certain level of autonomy or control categorization (e.g., SIL or DAL)</td>
<td>100%</td>
</tr>
<tr>
<td>Software requirements unlinked to a test case</td>
<td>The number of safety requirements unlinked to test cases</td>
<td>0 or approaching zero</td>
</tr>
<tr>
<td>Specification of the transparent certification service infrastructure D7.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Open software safety defects</strong></td>
<td>The number of open software safety defects with priority</td>
<td>0</td>
</tr>
<tr>
<td><strong>Defect rate (density)</strong></td>
<td>Number of defects per thousand lines of code in a given time unit</td>
<td>0</td>
</tr>
<tr>
<td><strong>Controls with causes</strong></td>
<td>The number of causes with a control divided by the total number for all hazards.</td>
<td>1</td>
</tr>
<tr>
<td><strong>Verifications with controls</strong></td>
<td>The number of controls for which there is a verification divided by the number of controls for all hazard causes.</td>
<td>1</td>
</tr>
<tr>
<td><strong>Hazard cause/control closure evolution</strong></td>
<td>Three point moving average of the set of open causes and controls at three consecutive time intervals.</td>
<td>1 or less. Greater than 1 means hazard causes or controls are opening faster than they are closing</td>
</tr>
<tr>
<td><strong>Software requirements demonstration metric</strong></td>
<td>The total number of demonstrated safety requirements divided by the total number of safety requirements.</td>
<td>Reasonable value determined by past experience</td>
</tr>
<tr>
<td><strong>Test metrics</strong></td>
<td>Test case criteria such as statement, decision, condition, and define-use pair and usage-based coverage and the results.</td>
<td>Type of coverage and a successful test.</td>
</tr>
<tr>
<td><strong>Key node safety metric</strong></td>
<td>A metric for predicting the relative safety between different versions of software modules using heuristics analysis of fault tree structures, based on fault tree properties such as key node height, size of key node sub-trees, and the number of key nodes.</td>
<td>0 to 1. The higher the value the better.</td>
</tr>
<tr>
<td><strong>Number of safety claims in a safety case</strong></td>
<td>Safety claims are statements indicating how relevant safety hazards are being mitigated</td>
<td></td>
</tr>
<tr>
<td><strong>Number of uninstanitiated safety claims in a safety case</strong></td>
<td>An uninstanitiated safety claim is one that is a place holder in a safety case, where future claims will be added into an ongoing safety argument</td>
<td></td>
</tr>
<tr>
<td><strong>Number of undeveloped safety claims</strong></td>
<td>An undeveloped claims is one where the exact claim in known but the strategy to decompose in down to sub-claims and eventually tie it to specific evidence is not yet know or has not been included into the safety case</td>
<td></td>
</tr>
<tr>
<td><strong>Number of uninstanitiated and undeveloped safety claims in a safety case</strong></td>
<td>It refers to claims that require both instantiation and further development so that they can be linked to evidence.</td>
<td></td>
</tr>
<tr>
<td><strong>Number of problem reports related to system safety</strong></td>
<td>Problem reports can be categorized based on the consequence that they will have on the system (e.g., cause a failure that has a significant functional consequence)</td>
<td></td>
</tr>
<tr>
<td><strong>Percentage of software hazards</strong></td>
<td>It is a direct indicator of the sufficiency of hazard Identification. By comparing the number of software hazards identified against historical data, it indicates the validity of the software safety requirements through identified hazards.</td>
<td></td>
</tr>
<tr>
<td><strong>Software hazard analysis</strong></td>
<td>Hazardous software, or safety-critical software, allocated as high- or medium-risk requires</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>depth</td>
<td>analysis of second- and third-order causal factors (should they exist). The depth of hazard analysis, indicated by this metric, will contribute to the sufficiency of overall software hazard analysis and hence validity of derived requirements. The metric will be an indicator of whether hazards have been analysed to a sufficient depth.</td>
<td></td>
</tr>
<tr>
<td>Percentage of software safety requirements</td>
<td>It is an indicator of how sufficient hazard analysis has been performed, and hence the validity of the derived safety requirements.</td>
<td></td>
</tr>
<tr>
<td>Percentage of high-risk software hazards with safety</td>
<td>It reveals whether any high-risk software hazards have not resulted in applicable safety requirements through hazard analysis. This indicates sufficiency of the process (through artefacts), and hence validity of the requirements.</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage of medium-risk software hazards with safety</td>
<td>This metric is simply an extension to “Percentage of high-risk software hazards with safety requirements” by considering medium-risk software hazards.</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage of moderate-risk software hazards with safety</td>
<td>This metric is an extension to “Percentage of high-risk software hazards with safety requirements” by considering moderate risk software hazards.</td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage of software safety requirements traceable to</td>
<td>Ensuring traceability to system hazards increases the validation case. This metric is simply a percentage indicator of traceability of requirements. All derived software safety requirements must be traceable to system hazards.</td>
<td></td>
</tr>
<tr>
<td>hazards.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety requirements traceability metric</td>
<td>The purpose of the safety requirements traceability metric is to measure the adherence of the software products to their safety requirements. During the software requirements phase the products to be measured are the Software Requirements Specification and Interface Requirements Specification to the system level requirements specified in the System Segment Specification and the identified hazards. This ensures the system requirements have been properly interpreted by the software engineering organization.</td>
<td></td>
</tr>
<tr>
<td>Safety requirement stability metric</td>
<td>The safety requirements stability metric is used to indicate the degree to which changes in the software requirements to minimize hazards from occurring affect the development of the safety features. It can be used to determine the cause of the requirements change.</td>
<td></td>
</tr>
<tr>
<td>Hazard analysis stability metric</td>
<td>The hazard analysis stability metric is used to identify potential problems in the hazard analysis process. During the software requirements inspection, any additional hazards identified should be evaluated to determine if the hazard analysis process can be improved. It can be graphically represented by plotting the cumulative hazards identified.</td>
<td></td>
</tr>
<tr>
<td>Hazard isolation metric</td>
<td>The hazard isolation metric is used to measure the number of defects from non-safety critical functions that may cause a hazardous event. Collection of data for this metric can begin during the software requirements inspection.</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Safety design stability metric</td>
<td>The safety design stability metric is a subset of the design stability metric. The design stability metric provides an indication of the volatility (i.e., changes) of the design of critical and non-critical software requirements, which can impact the cost, schedule, and quality of the development effort. Tracking the design changes gives an overall indication of the design problems as well as an indication of the understanding of the requirements.</td>
<td></td>
</tr>
<tr>
<td>Safety complexity metric</td>
<td>The safety complexity metric gives an indication of the structure of the critical software by identifying the complexity of the internal structure of the critical module. It is generally accepted that the number of errors the software is likely to contain, and its difficulty to test and maintain, is directly proportional to the complexity of the software.</td>
<td></td>
</tr>
<tr>
<td>Safety breadth of testing metric</td>
<td>The purpose of the safety breadth of testing metric identifies the amount of coverage or the amount of safety requirements tested and the amount of success of the tests. This testing is considered black box testing, since the outputs are verified given a prescribed set of inputs.</td>
<td></td>
</tr>
<tr>
<td>Safety depth of testing metric</td>
<td>The safety depth of testing metric provides an indication of the extent of coverage within the module (i.e., coverage of all possible path conditions) and the success of testing. This test is considered white box testing, since there is visibility in the paths/conditions of the software.</td>
<td></td>
</tr>
<tr>
<td>Safety fault profile metric</td>
<td>The safety fault profile metric provides insight into the quality of the software that provides the system fault tolerance by tracking the number of faults detected during testing. Equally important it is the ability to fix these faults. These measurements can be used to indicate the quality of the interfaces between coding activity, design activity, and requirements activity.</td>
<td></td>
</tr>
<tr>
<td>Number of risks that need to be mitigated identified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of artefacts produced per safety requirement.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of artefacts produced per safety component</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of activities for compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean time to failure</td>
<td>How long the software can run before encountering a &quot;crash&quot;</td>
<td></td>
</tr>
<tr>
<td>The value depends on domain (e.g., the probability of catastrophic failure must be no worse than $10^{-9}$ per hour for civilian airliners)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst case memory usage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst case execution time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory allocation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processing time allocation</td>
<td>Timing such as initialization timing of software, partition timing, and time between reset</td>
<td></td>
</tr>
<tr>
<td>Function point metric</td>
<td>A function point is a weighted total of five major components that comprise an application based on their complexity: external inputs, external outputs, logical internal files, external interface files and external. There is a complex formula for calculating function points. This is then used for either measuring productivity (function points per person-year) or quality (defects per function point).</td>
<td></td>
</tr>
<tr>
<td>Customer problems metric</td>
<td>Number of reported problems per user-month after the software is released to market. This may be of use internally when the software is being developed based on acceptance tests where the users are involved.</td>
<td></td>
</tr>
<tr>
<td>Defect removal effectiveness</td>
<td>There are several ways to calculate it. The main idea is to see how any defects are fixed per phase of the software and how many escape and are found later. This metric is also used to help identify where increases of focus is need during the development.</td>
<td></td>
</tr>
<tr>
<td>Number of complexity metrics defined (e.g., Cyclomatic complexity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use case points</td>
<td>The use case point measurement addresses UML-based software modelling and implementation. The use case points (UCP) are computed as: $UCP = TCP \times ECF \times UUCP \times PF$, where: TCP stands for the technical complexity factors which evaluate by weights the technological type of the system such as distributed system, reusability, concurrent etc.; ECF the environmental complexity factors which characterize the system background like stability of the requirements, experience in OO and UML etc.; UUCP the unadjusted use case points which counts the different use case diagram components, and; PF the productivity factors which weight the UCP considering person hours per use case. These are to be used for effort estimation.</td>
<td></td>
</tr>
<tr>
<td>Mark II function points</td>
<td>This method is a modification of the function point method by changing the viewpoint to the database system approach. The counting characteristics are input, entities referenced, and output. The weight factors are quite different from the FP method (0.58 for inputs, 1.66 for entities referenced and 0.26 for outputs): $FP = 0.58 W_i + 1.66 W_r + 0.26 W_o$. The 14 FP adjustment factors were extended by six other factors considering actual system aspects and leads to the possibility of effort estimation. It is standardized by ISO.</td>
<td></td>
</tr>
<tr>
<td>COSMIC full function point</td>
<td>The COSMIC Full Function Point (FFP) method was developed in the Common Software...</td>
<td></td>
</tr>
</tbody>
</table>
Measurement International Consortium (COSMIC) and is established as ISO/IEC 1976. A full function point only considers a data movement, which means that there are no (weighted) difference between inputs, outputs etc. The Cfsu (COSMIC functional size unit) is the FFP measurement unit. The basic formula for COSMIC FFP counting is:

$$\text{FFP} = \text{counting}(((\text{entry,exits}), (\text{reads, writes}))_{\text{architecture level}})$$. The COSMIC FFP measurement method is designed to be independent of the implementation decisions embedded in the operational artefacts of the software to be measured. To achieve this characteristic, measurement is applied to the FUR (functional user requirement) of the software to be measured expressed in the form of the COSMIC FFP generic software model. This form of the FUR is obtained by a mapping process from the FUR as supplied in or implied in the actual artefacts of the software. The architectural reasoning of boundaries is given through the software layers such as tiers, service structures or component deployments. The functional size of software is directly proportional to the number of its data transactions. All data movement subprocesses move data contained in exactly one data group. Entries move data from the users across the boundary to the inside of the functional process; exits move data from the inside of the functional process across the boundary to the users; reads and writes move data from and to persistent storage. Full function points are standardized by ISO.

### COCOMO and COCOMO II

The Constructive Cost Model (COCOMO) was defined by Boehm and is based on the formula:

$$\text{Personal effort} = \text{scale factors} \times \text{KDSI}^{\text{type of project}}$$

Where KDSI means Kilo Delivered Source Instruction that must be estimated at the beginning. The scale factors define the cost drivers Boehm classify three types of projects: organic, semidetached, and embedded.

The COCOMO II approach extends the set of cost drivers and considers the different/new aspects of software systems like code adaptation, reuse and maintenance. Furthermore, it is possible to execute/estimate the development time (TDEV) as:

$$\text{TDEV} = \text{scale factors} \times \text{PM}^{\text{calibration}}$$

A special kind of COCOMO is called as early design model equation and was executed by:

$$\text{Effort} = \text{KLOC} \times \text{adjustment factor}$$

### SLIM

Raleigh curve for the software development area in the following manner:

$$\text{Current effort} = (\text{Total effort/duration}) \times \text{t} \times e^{(-\text{t}^2/2 \times \text{duration})}$$

Where duration stands for the square of total duration of the development and t means the time point of evaluation. The current effort was measured in personal years.

Another kind of estimation based on the Raleigh formula is known as software equation:

$$\text{System size} = \text{technology constant} \times (\text{Total effort}^{1/3} \times \text{duration}^{2/3})$$

Where the technology constant depends on the development methodology.
<table>
<thead>
<tr>
<th>Customer cost</th>
<th>Customer Cost = Defect_density × Kilo_LINES_of_Code × Cost_per_defect × Defects_found_by_customer</th>
</tr>
</thead>
</table>
| Return on investment (ROI) | Based on the cost calculation given above, we can calculate: ROI_1 = (Cost saved – Investment) / Investment  
ROI_2 = (Cost saved – Investment) / Original cost  
New cost= Original cost × (1- ROI_2) |
<p>| Effort in person months of years |                                                                                                    |
| Percentage of tested requirements |                                                                                                    |
| Design progress | A percentage of estimated effort. All these can employ a number of the above metrics in order to measure effort |
| Code progress | A percentage of estimated size |
| Test progress | Percentage of test cases executed |
| Inspection Efficiency | LOC/hr |
| Effort per defect in peer reviews | Person hour/defect |
| Effort per defect in module test | Person hour/defect |
| Defects detected before integration | A percentage |
| Number of defects in design |                                                                                                    |
| Number of defects in peer reviews |                                                                                                    |
| Number of defects in module test |                                                                                                    |
| Number of defects in test |                                                                                                    |
| Number of defects in the field |                                                                                                    |
| Programmer productivity | LOC produced/ person months of effort |
| Module defect density | Number of defects/module size |
| Failure condition coverage | A metric highlighting the applied safety analysis (FTA, FMEA etc) for each failure condition. |
| Number of single cause failures |                                                                                                    |</p>
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of common cause failures</td>
<td>Validation is the activity of demonstration, by test and analysis, that the product meets in all respects its specified requirements.</td>
</tr>
<tr>
<td>Number of validation actions</td>
<td>Validation is the activity of demonstration, by test and analysis, that the product meets in all respects its specified requirements.</td>
</tr>
<tr>
<td>Number of wrong side failures</td>
<td>A failure condition in a piece of railway signalling equipment that results in an unsafe state</td>
</tr>
<tr>
<td>Percentage branch coverage</td>
<td>For measuring work progress, planned versus actual.</td>
</tr>
<tr>
<td>SLOC (Source Lines of Code)</td>
<td>Requests for changes to the software. They can be of different types, where type 0 is for critical defects, type 1 is for normal defects, type 2 is for improvements, and type 3 is for new features.</td>
</tr>
<tr>
<td>Number of software change orders (SCO)</td>
<td>Requests for changes to the software. They can be of different types, where type 0 is for critical defects, type 1 is for normal defects, type 2 is for improvements, and type 3 is for new features.</td>
</tr>
<tr>
<td>Number of SLOC per SCO type</td>
<td>This is the number of source code lined that changed based on the type 0,1, 2 and 3 SCO's</td>
</tr>
<tr>
<td>Cost per month per staff</td>
<td></td>
</tr>
<tr>
<td>People per month added versus people per month leaving a project</td>
<td></td>
</tr>
<tr>
<td>Number of SCO's closed per component or subsystem</td>
<td></td>
</tr>
<tr>
<td>Number of SCO's opened per component and subsystem</td>
<td></td>
</tr>
<tr>
<td>Average hours per change per component and subsystem</td>
<td></td>
</tr>
</tbody>
</table>