CRitical S**YST**em Engineering **A**cce**L**eration

System Family Engineering Framework – V1

D610.031

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	System Family Engineering Framework – V1
Deliverable No.	D610.031
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-01-30
Contact	Adeline Silva Schäfer
Organization	Fraunhofer IESE
Phone	+49-631-6800-2216
E-Mail	adeline.schaefer@iese.fraunhofer.de

AUTHORS TABLE

Name	Company	E-Mail
Adeline Silva Schäfer	Fraunhofer IESE	adeline.schaefer@iese.fraunhofer.de
Martin Becker	Fraunhofer IESE	martin.becker@iese.fraunhofer.de
Tiago Amorim	Fraunhofer IESE	Tiago.amorim@iese.fraunhofer.de
Peter Graubmann	Siemens AG	peter.graubmann@siemens.com
Anjelika Votintseva	Siemens AG	anjelika.votintseva@siemens.com
Andrea Leitner	VIF	Andrea.leitner@v2c2.at
Jason Mansell	Tecnalia	Jason.Mansell@tecnalia.com
Ralf Bogusch	Cassidian	ralf.bogusch@cassidian.com
Christian Reuter	Daimler	christian.c.reuter@daimler.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.01	05.12.2013	Initial structure	ALL
0.02	10.12.2012	Review of initial structure	ALL
0.03	16.12.2013	UC needs – use case 3.4b	8-12; 15
0.04	20.12.2013	Safety	
0.05	02.01.2014	Added Section on ISO 26550 Standard	
0.06	08.01.2014	Added section on Varies and PLUM	
0.07	10.01.2014	Removed section on PLUM – it didn't fit the main section	
0.08	13.01.2014	Added UC 2.3; Removed some safety sections	
0.09	16.01.2014	Added overview of SFEF, removed "template section"	
0.10	17.01.2014	Overall improvements	ALL
0.11	22.01.2014	Addressed some reviewer's comments & added content to section 3.1.2	ALL
0.12	23.01.2014	Changed sections 2.3,3.1 and 3.3 and minor adjustments in the whole document	ALL
0.13	23.01.2014	Changed section 3.2	35
1.0	29.01.2014	Addressed Reviewer's comments and minor changes to section 2.3	ALL

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of deliverable

This deliverable describes the System Family Engineering Framework (SFEF) brick. SFEF provides a customizable method, respective techniques, guidelines and best practices to engineer families of high-integrity (including safety critical) systems, which can be followed in different application domains. Specific features of the framework comprise the planning (esp. scoping) and specification of high integrity families, modularization approaches, and orthogonal variability management across several system levels.

1.2 Relationship to other CRYSTAL Documents

This document is the first in a series of three reports:

- D610.031 – System Family Engineering Framework V1 (this document)
- D610.032– System Family Engineering Framework V2 – with the description of the SFEF
- D610.033 – System Family Engineering Framework V3 – with the assessment of SFEF usage by the demonstrators

This document contains information provided in the following deliverables:

- D610.011 – Crystal Variability Management – V1
- D203.011 – MSE Report – V1 – Use Case 2.3
- D304.011 – Milestone Report – V1 – Use Case 3.4

1.3 Structure of this document

The deliverable is structured in two main sections: Section 2 presents an overview of existing variability management frameworks and methods and an overview of existing approaches for safety engineering for product lines. Section 3 presents the use cases and their requirements coverage for the SFEF brick.

2 State Of The Art in System Family Engineering

This section provides an overview of the state of the art with regard to engineering methods, techniques and tools of system families.

This section is structured as follows: Section 2.1 presents an overview of existing product line engineering frameworks, section 2.2 provides an overview of variant management related projects, and in section 2.3 we outline and discuss a possible approach for integrating product line and safety engineering.

2.1 Product Line Engineering

In this section, we provide an overview of the most prominent product line approaches. These approaches have been successfully applied in *software* product lines, however, when it comes to *software intensive systems*, new aspects have to be considered, which requires additional steps in the development lifecycle, not covered by the existing approach. In particular, for high integrity systems, safety aspects need to be considered in nearly every stage of the development lifecycle.

2.1.1 International Standard ISO/IEC 26550:201 J(E)

The international standard ISO/IEC 26550:201 J(E) is the entry point of the whole suite of international standards for software and systems product line engineering and management. It aims to define terms specific to Software and Systems Product Line (SSPL) engineering and management, establish a reference model for the overall structure and processes SSPL, and describe how the components of the reference model fit together, along with interrelationships amongst them. This standard does not describe any methods and tools associated with SSPL. The consecutive standards (ISO/IEC 26551 through 26556) will concentrate on them. This standard, was produced stating from References [Pohl, 05], [Northrop, 00] and [Kakbla, 06] and structures from ISO/IEC 12207:2008, ISO/IEC 15288:2008, ISO/IEC 15940:2006 and ISO/IEC 14102:2008.

SSPL engineering and management creates, exploits, and manages a common platform to develop a product line (e.g., software products, systems architectures) at lower cost, reduced time to market, and better quality. It provides a reference model abstracting the key processes of SPPL engineering and management and the relationships between the processes. It can be used in subsequent standardization efforts to create different levels of abstraction standards.

The reference model consists of domain engineering and application engineering life cycles and organizational management and technical management process groups (Figure 2-1:). The domain and application engineering life cycles need to be synchronized and applicable to different life cycle models, as needed to meet different quality criteria and to achieve different organizational objectives.

Domain engineering identifies the key commonality and variability, establishes a product line platform implementing this commonality and variability, and manages the platform throughout the life cycle of the product line. It is assisted by product line scoping, Domain requirements engineering and domain design.

Application engineering develops application assets and individual systems on top of the platform. Ideally, much or even most of the product line engineering effort and complexity have been allocated to domain engineering, reducing application complexity and shortening application development times. Application engineering involves customers and thus needs to deal with evolving market needs. It is supported by application requirements engineering, application design, application realization, and application verification and validation.

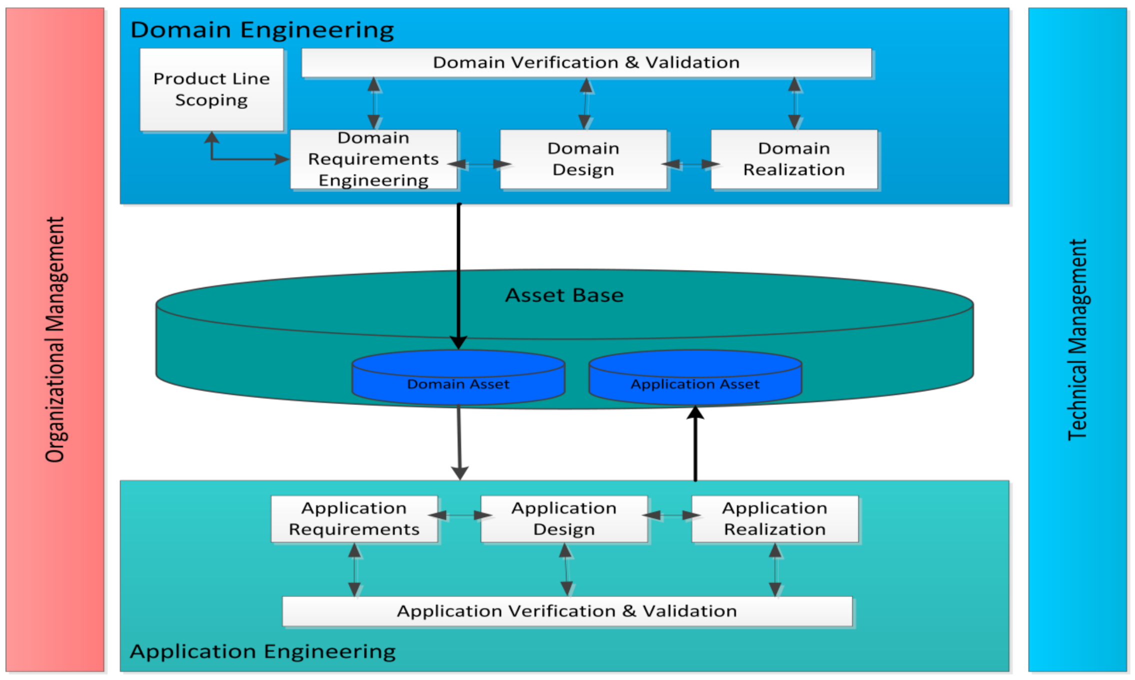

Figure 2-1: Reference model for SSPL engineering and management

Organizational management processes orchestrate the product line organization. Ongoing preparation, planning, execution, and improvement efforts needed for the introduction and institutionalization of the product line strategy is provided by these management processes. It is assisted by organizational level product line planning and organizational product line enabling management.

Technical management processes manage the creation and evolution of product line processes, domain assets, and products. It consists of process management, variability management, asset management, and support management.

Organizational and technical management process groups are necessary to help organizations to establish and improve capabilities for nurturing their product lines from conception to retirement and for establishing and managing relationships with customers, providers and other key stakeholders.

2.1.2 SEI Framework for Software Product Line Practice

The Framework for Software Product Line Practice (FSLP), version 5.0 [Northrop, 00], published by the Software Engineering Institute (SEI), describes the 3 essential activities and 29 practice areas necessary for creating and managing a software product line. The three essential activities are depicted in Figure 2-2. Each rotating circle represents one of the essential activities. All three are linked together and in continuous motion, showing that they are all essential, linked, iterative, and can occur in any order. The rotating arrows indicate not only that core assets are used to develop products, but also that revisions of existing core assets or even new core assets evolve out of the product development.

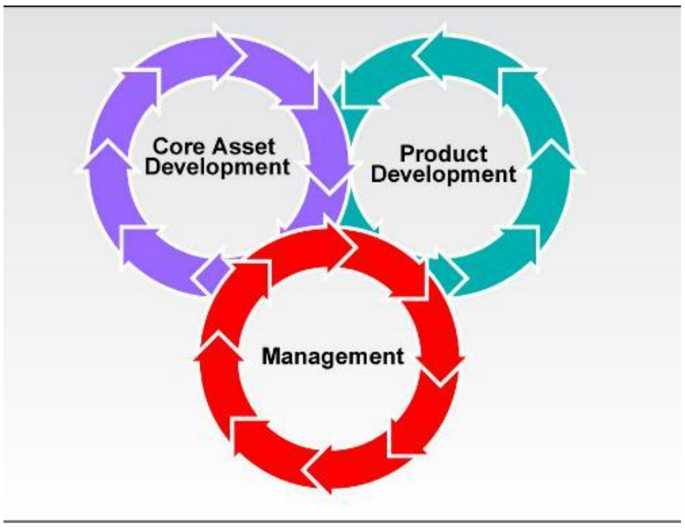

Figure 2-2: The Three Essential Activities for Software Product Lines

There is a strong feedback loop between the core assets and the products. Core assets are refreshed as new products are developed. There is a constant need for strong, visionary management to invest resources in the development and sustainment of the core assets.

Core Asset Development
The goal of the core asset development activity is to establish a production capability for products. The inputs of this activity are: Product constraints, production constraints, production strategy and inventory of preexisting assets. The outputs of the core asset development activity are required for a production capability to develop products:

- **Product Line Scope:** A description of the products that will constitute the product line or that the product line is capable of including.
- **Core Asset Base:** Includes all the core assets, which are the basis for the production of products in the product line. Each core asset should have an associated attached process that specifies how it will be used in the development of actual products
- **Production Plan:** A production plan prescribes how the products are produced from the core.

Product Development
The product development activity depends on the outputs of core asset development plus the requirements for each individual product. The creation of products may have a strong feedback effect on the product line scope, the core assets, the production plan, and even the requirements for specific products.

Management
Management is divided between technical and organizational levels and must be strongly committed to the software product line effort. Organizational management identifies production constraints and ultimately determines the production strategy.

Practice areas
Product line practice areas describe what an organization must do to perform those three previously described broad essential activities. The practice areas are loosely divided in three categories:

- **Software engineering practice areas** are those necessary for applying the appropriate technology to create and evolve both core assets and products. They are carried out in the core asset development and product development.
- **Technical management practice areas** are those necessary for managing the creation and evolution of the core assets and the products.
- **Organizational management practice areas** are those necessary for orchestrating the entire software product line effort.

2.1.3 Fraunhofer PuLSE

Fraunhofer PuLSE (Product Line Software Engineering) [Bayer et al., 00], [Muthig et al., 04] is a method for enabling the conception and deployment of software product lines within a large variety of enterprise contexts developed at Fraunhofer IESE. This is achieved via a product centric focus throughout its phases, customizability of its components, an incremental introduction capability, a maturity scale for structured evolution, and adaptations to a few main product development situations.

This approach focuses on a set of concrete products, not on a theoretically defined domain. These products can be current ones, planned ones, and anticipated ones that are likely to be needed in the future or that would be of strategic value for the enterprise.

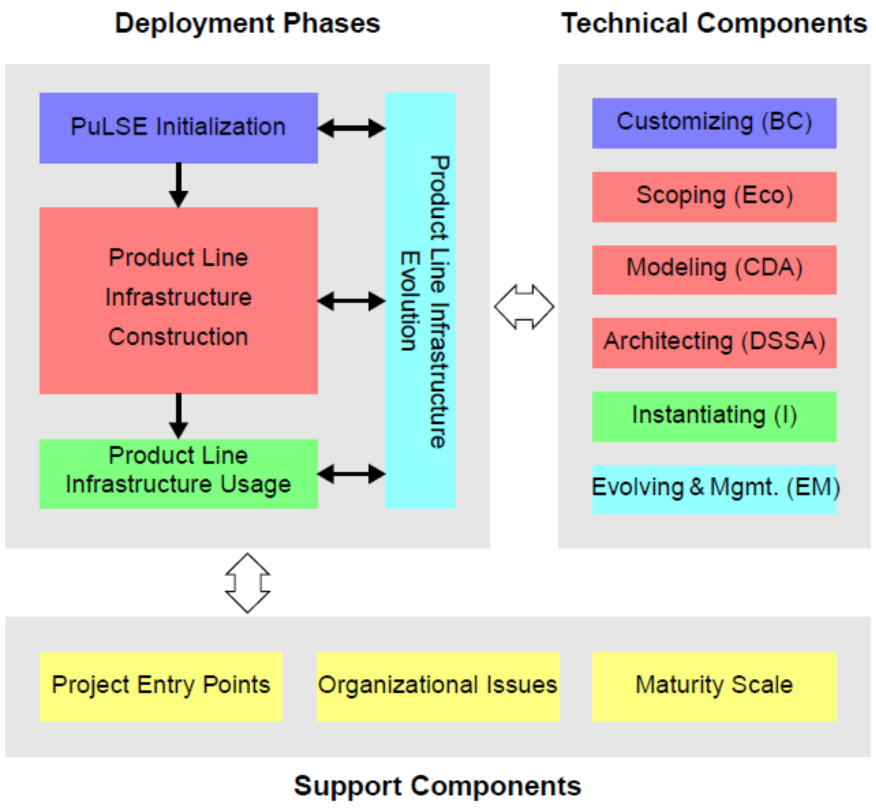

Figure 2-3: PuLSE Components Overview

PuLSE provides a complete framework that covers the whole software product line development life cycle, including infrastructure construction, usage, and evolution. It is centered on three main elements: The deployment phases, the technical components, and the support components:

- **Deployment Phases:** Logical stages of the product line life cycle. They describe activities performed to set up, use, and evolve product lines. The four phases are:
    - *Initialization*: An instance of PuLSE is produced, which is tailored to the enterprise context in which it will be applied.
    - *Product Line Infrastructure Construction*: To scope, model, and architect the product line infrastructure.
    - *Product Line Infrastructure Usage*: Aims at specifying, instantiating, and validating single product line members. This phase also contains the implementation of product line and instance specific assets in case they are not available.
    - *Product Line Infrastructure Evolution*: Manage the evolution of the product line infrastructure over time.
- **Technical Components:** Provide the technical know-how needed to operationalize the product line development. They are used throughout the deployment phases. A different facet of each component is often used in more than one of the phases – though some components directly correspond to phases. Technical components are the following:
    - *PuLSE-Customizing (BC)* is used to baseline the enterprise and to customize PuLSE accordingly.
    - *PuLSE-Scoping (Eco)* aims at identifying, describing, and bounding a product line by determining its members and their characteristics.
    - *PuLSE-Modelling (CDA)* is used to elicit the requirements for the domain through creating a product line model.
    - *PuLSE-Architecting (DSSA)* is concerned with the development of a reference architecture
    - *PuLSE-Instantiating (I)* aims at specifying, constructing, and validating single members of the product line.
    - *PuLSE-Evolving & Management (EM)* guides and supports the application of PuLSE throughout the product line life cycle. Since there are strong connections to the evolution and product line management, configuration management is described as part of this component.
- **Support Components:** Packages of information, or guidelines that enable a better adaptation, deployment, and evolution of PuLSE. They are used by the technical components. Currently there are the following support components:
    - *Project Entry Points*: Provides customizations of PuLSE to major project categories
    - *Organization Issues*: Provides guidelines to set up and maintain the right organizational structure for developing and managing product lines
    - *Maturity Scale*: Provides a scale for evaluating the quality of PuLSE process applications in an enterprise

2.1.4 BigLever 2nd Generation PLE Approach

The first generation of software product line methods emphasized a clear separation between the domain engineering and the application engineering. The first is producing assets for reuse and the latter is developing with reuse. The second generation approach advocates that this dichotomy between domain and application engineering creates problems continuously, as the product line needs to evolve. The changes in the product specific assets lead to an overhead in management effort, especially regarding the configuration management. Instead of having two separated processes, 2nd generation PL are based on so-called software mass customization.

Taking into account this paradigm, BigLever Software created the **PLE Lifecycle Framework** and **3-Tiered PLE Methodology**. This industry standard PLE framework and pragmatic methodology provide 2nd Generation PLE (2GPL) capabilities needed to shift from product centric approaches to create an automated, optimally efficient means of production for systems and software product lines [Krueger,14]. This PLE solution allows organizations to make a systematic and incremental transition to an operational PLE production line. By focusing both business and engineering teams on the operation of the production line, the organization can plan, develop, deploy and evolve the product line portfolio, from business case and analysis, to requirements, design, implementation, testing, delivery, maintenance and evolution.

PLE Lifecycle Framework

The PLE Lifecycle Framework provides a set of common PLE concepts and constructs that augment tools, assets and processes across the entire lifecycle:

- A **feature model** that expresses the feature diversity among the products in your product line.
- A **uniform variation point mechanism** to manage feature based variations in all stages of the engineering lifecycle.
- A **product configurator** to automatically assemble and configure assets and their variation points to produce all of the assets and products in product line.

A key capability of the PLE Lifecycle Framework is the integration of PLE concepts into tools, assets and processes across the systems and software development lifecycle.

3-Tiered PLE Methodology

The 3-Tiered Methodology is a practical tiered approach that allows a non-disruptive transition and successfully operating a PLE production line. Each tier builds upon and is enabled by the previous tier:

- **Base Tier (1): Feature-based Variation Management and Automated Production** Tools, integrations and infrastructure for engineering product line features, product feature profiles, product line hierarchy, feature based variation points in assets, and automated feature based configuration of product line assets into products and deliverables.
- **Middle Tier (2): Feature based Asset Engineering** Processes and organizational structures for engineering the full lifecycle of product line assets – from requirements to architecture, design, implementation and test – on multiple delivery streams in a production line.
- **Top Tier (3): Feature based Portfolio Management** Business wide management of a product line portfolio by the features offered and the profile of features allocated to each product.

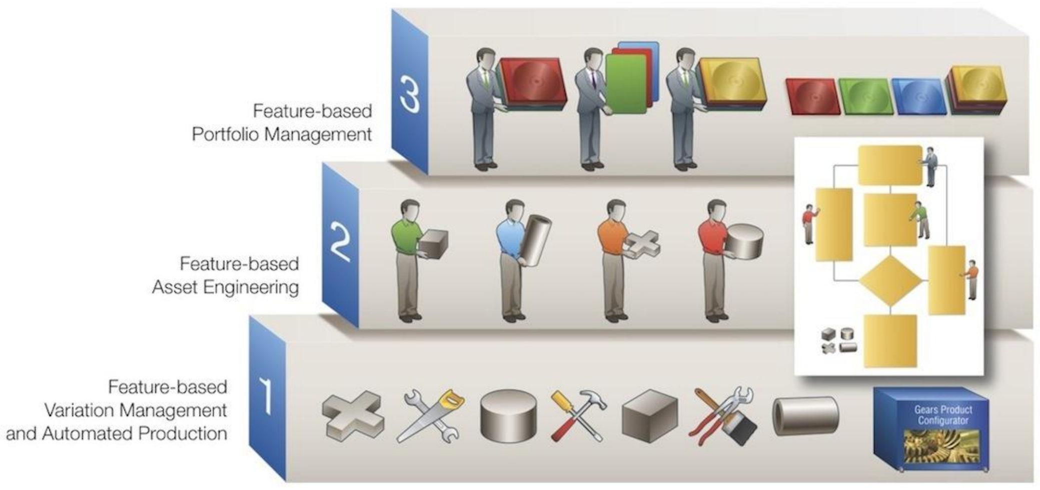

Figure 2-4: 3-Tiered Methodology

2.2 Variability Management Related Projects

2.2.1 VARIES Project

VARIES [1]is an industry driven research project in the ARTEMIS program on the topic of variability in safety critical embedded systems. Embedded Systems (ES) are rarely entirely conceived from scratch. Companies developing ES constantly face decisions about whether to use and adapt existing products or product assets or whether to develop new assets. Determining the long term risk and benefits of such decisions is very challenging. VARIES addresses the following question: How can the benefits offered by introducing variability into ES outweigh the increased product complexity caused by variability. Its objectives are

- to enable companies to make informed decisions on variability use in safety critical ES;
- to provide effective variability architectures and approaches for safety critical ES; and
- to offer consistent, integrated and continuous variability management over the entire product life cycle.

In addition, VARIES will create **a Center of Innovation Excellence** (CoIE) for managing variability in ES. VARIES started in May 2012 with duration of 3 years, and involves 23 partners from 7 countries, with the following objectives:

Objective 1: Enable companies to make an informed decision on variability use in safety critical ES

The VARIES project will enable a company to make variability decisions in their ES based on hard, tangible evidence instead of subjective “good feeling”, and will allow the same companies to assess the impact of their variability decisions across the entire product life cycle. Moreover, VARIES will allow companies to better define the product variants that are of most value to the market segments they serve.

[1] http://www.varies.eu/

Serving different market segments may introduce constraints on the product variants, e.g. safety regulations in automotive sub sectors. The VARIES project will provide companies support to better assess these boundary conditions and their impact on product variability

Objective 2: Providing effective variability architectures and approaches for safety critical embedded systems

Managing variability aspects of safety critical ES without negatively impacting the safety aspects is a huge challenge today. Managing the safety characteristics of an embedded system at (reusable) component level will introduce yet another level of complexity that will have to be properly managed.

By definition, safety is a property pertaining to the entire system; determining the system level and its justification based on the specific characteristics of the individual elements composing the system is far from straightforward today. Providing assurance concerning the adequate management of hazards and risks is a regulatory requirement and a prerequisite for the deployment of a safety critical system. A safety case, through its arguments and evidence, which provides this justification of the system, is acceptably safe to operate within its intended environment. From the perspective of a product line, the safety case should be systematically developed and maintained, ensuring the management of the impact of variation on safety assurance supported by the traceability between the product line artifacts and the safety case concepts.

VARIES will provide an integration of safety assessment into the product line processes, and concrete guidance with regard to reference architectures and approaches to realize variability without compromising the safety characteristics of the ES.

Special attention will be given to the composition of reliable and unreliable components and on the impact on safety of this composition. Moreover, VARIES will also enable a company to trade off safety and variability, i.e., determining the optimal balance between both aspects.

Objective 3: Consistent, integrated and continuous variability management over the entire product life cycle

VARIES will enable consistent, continuous and integrated variability management over the whole product life cycle for a single company as well as between collaborating companies. Special attention will be given to the multidisciplinary aspects in complex embedded systems engineering, including safety and certification aspects.

The main results of VARIES will be:

- Innovative embedded systems (ES) shall help to develop a reference framework for lean innovation, where the framework is made of new models, tools, and design methods.
- The VARIES Platform delivers a complete, cross-domain, multi-concern, state-of-the-art reference platform for managing variability in safety critical ES. Special attention will be given to aspects specific to safety critical ES, in particular the impact of reuse and composition on certification. In this platform different tools can be instantiated and chained to support the process flow of development for embedded systems product variants, over the whole product lifecycle, tailored to the specific context to a given company.
- A Center of Innovation Excellence on the topic of how to come to an optimized product variability that can be used for a variability of markets will be established.

2.3 Variability Management of High Integrity Systems

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

— C.A.R. Hoare (cited after [Bowen et al. 99])

High Integrity Systems are complex, software controlled systems with essential impact on life and health of humans or animals and the functionality of the human society, the environment and/or the existence of organizations.

Engineering of high integrity systems focuses on the development, analysis and justification of safety critical and security critical systems which primarily implies the consideration of safety analysis and failure modeling techniques. But further focus is also on

- the appropriate adaptation of methods for an integrated system and software architecture development,
- the consideration and adaptation of (a) requirements engineering methods, (b) formal development techniques, (c) analysis and development techniques for real time systems, and (d) testing approaches, as well as
- the elaboration of concepts for the eventually necessary assurance that the developed and realized systems conform to the expectations with respect to safety and security.

Here, in the context of variability management, the focus is on the reusability aspects of the artifacts that are developed for high integrity systems.. The intention is to provide an integrated process for the management of safety critical product lines together with guidelines and best practices for its utilization.

This process has two key aspects:

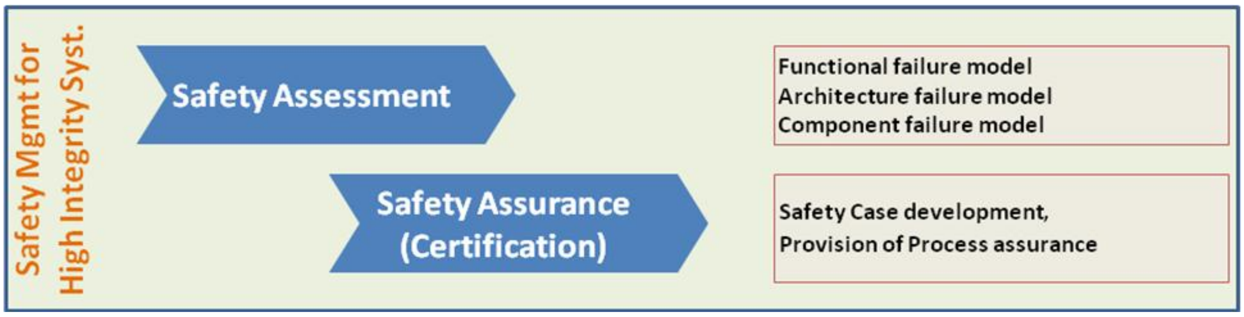
Safety Mgmt for High Integrity Syst.
Safety Assessment
Safety Assurance (Certification)
Functional failure model
Architecture failure model
Component failure model
Safety Case development,
Provision of Process assurance

Figure 2-5: Safety management process for high integrity systems

- **Safety assessment** is the part of the process where safety is provided. It comprises the scoping and identification of safety critical functions, the definition of the context model (i.e., the description of the expected environment), the various analysis steps, the identification of the safety requirements and the development of the mitigation strategies – all that under the assumption that there are variations in the system part as well as in the context/environment part.
- **Safety assurance** is the part of the process wherein the proofs or corroborations are provided that should convince the relevant authorities (certification authorities) that the required safety is achieved by the final product. Safety assurance comprises two activities: the development of the *safety cases* (which are the particular proofs of the compliance with the safety requirements) and the provision of *process assurance* (which has to demonstrate

the quality of the applied development processes, in order to corroborate the statements made in the safety cases)

The intended process is assumed to be – as far as possible – ignorant to the underlying safety analysis techniques.

2.3.1 Safety Requirements for PLE

It is often stated that "safety is intrinsically a property of the system as a whole" since the real hazards and risks become observable only at the physical system boundaries i.e. analyzing the system parts will not show the real risks. Thus safety is essentially not a "compositional property" and safety of the parts does not allow inferring the safety of the entire system. This is due to the fact that safety depends on the system environment and the observation that the interference of system properties may cause new hazards and risks.

In the context of product lines this seems to indicate that safety arguments have to be elaborated completely new for every single derived product, because the environment and/or components in the composition may have changed. This seems to be today's state of the practice in industry. For a product line approach, however, one would expect that safety arguments associated with reusable components are core assets from which the system safety argument can be derived jointly with the product composition.

System safety is given in a qualified way (e.g. freedom from unacceptable risk) or quantified (e.g. something highly undesirable will happen only with a probability lower than a maximally acceptable threshold). This kind of qualifying safety shows that variability is inherent to its definition

- What is acceptable and what not?
- How is the threshold defined?

One example of products with different safety properties derived from the same product line is, for instance, two cars, a family coach with child-proof locks and a sports car without. The environment into which the product will be placed determines whether a product will be considered safe or not.

Safety engineering has to start at the very beginning of system engineering. Typically, a safety engineering process will have the following tasks (See Figure 2-6):

- **Item Definition**: Identify what belongs to the system under consideration and what are the safety critical functions of the relevant system parts. This step includes also the definition of the environment in which the system is expected to operate.
- **Hazard Identification**: Hazards, which are potential source of harm, may be detected throughout the systems lifetime. Sequences of events that could lead to an undesirable state (e.g. an accident, damage, or injury) are identified as well.
- **Risk assessment:** Estimate the risk that is posed by a certain hazard. A risk is defined as the combination of the probability of occurrence of harm and the severity of that harm.
- **Safety Concept:** Created from collected hazards together with their risk classification. Also may be called safety requirements definition.

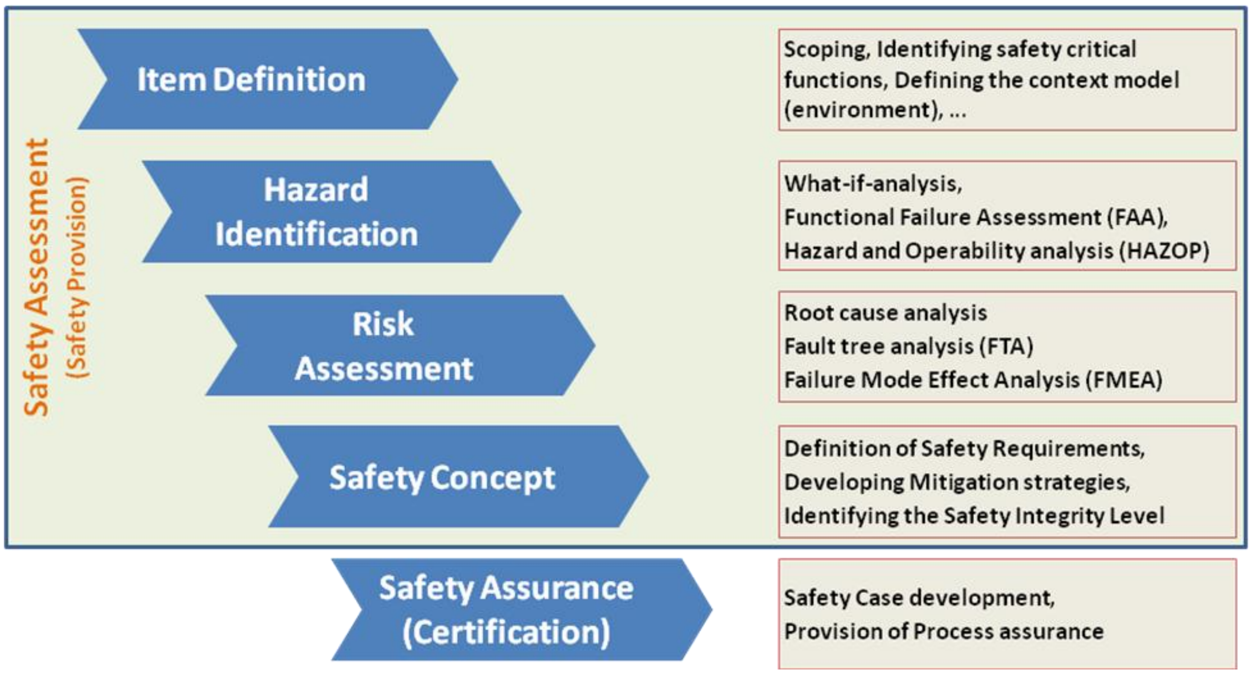
Safety Assessment
(Safety Provision)
Item Definition
Scoping, Identifying safety critical functions, Defining the context model (environment), ...
Hazard Identification
What-if-analysis,
Functional Failure Assessment (FAA),
Hazard and Operability analysis (HAZOP)
Risk Assessment
Root cause analysis
Fault tree analysis (FTA)
Failure Mode Effect Analysis (FMEA)
Safety Concept
Definition of Safety Requirements,
Developing Mitigation strategies,
Identifying the Safety Integrity Level
Safety Assurance (Certification)
Safety Case development,
Provision of Process assurance

Figure 2-6: The activities in the safety assessment process

The safety requirements specify the necessary risk reduction associated with each identified hazard and they are usually allocated to the system as a whole. They are transformed into the **derived safety requirements** when it comes to the realization of the safety requirements during the development phases.

There are also the **safety integrity requirements** which specify the reliability that has to be accomplished for safety and derived safety requirements. They are categorized into Safety Integrity Levels (SILs). Safety and derived safety requirements are associated with the system components within the overall system architecture. If a system component consists of hardware and software components, then the safety/derived safety and the respective safety integrity requirements have to be refined and broken down to these components.

Safety engineering is also concerned with the provision of safety assurance i.e. the support for certification of the safety critical system. This certification typically means to prove its compliance with safety standards in the respective domain and industry. In most cases the proof has to be presented as **safety case** concerned with both, the product-based and the process-based safety assurance. A safety case consists of a well-structured and well-reasoned **safety argument** and the corroborating **safety evidence**.

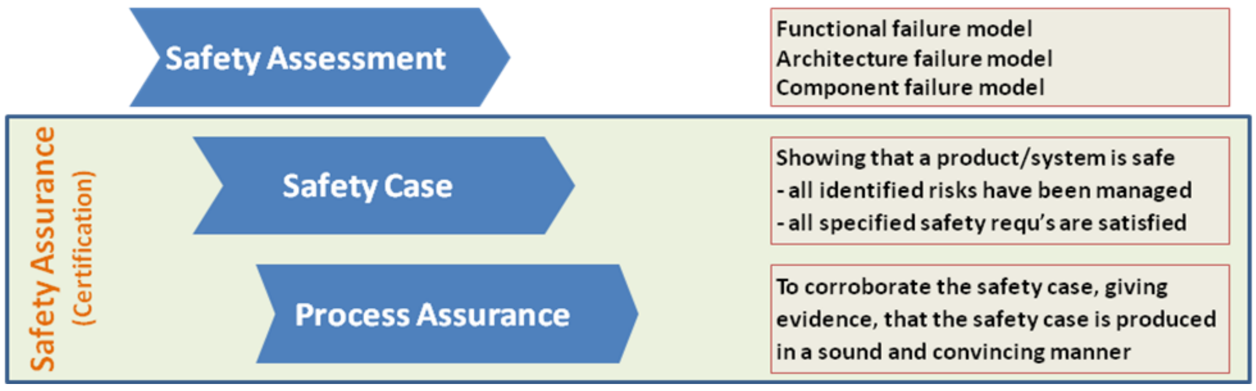
Safety Assessment
Functional failure model
Architecture failure model
Component failure model
Safety Assurance
(Certification)
Safety Case
Showing that a product/system is safe
- all identified risks have been managed
- all specified safety requ's are satisfied
Process Assurance
To corroborate the safety case, giving evidence, that the safety case is produced in a sound and convincing manner

Figure 2-7: The activities in the safety assurance process

The product-based part of the safety case demonstrates the acceptable safe behavior of the system (i.e., the product), showing the identified hazards, arguing the tolerability of the risks and claiming the satisfaction of all safety requirements. This claim is corroborated by direct evidence obtained from (formal) analysis, testing and simulation.

The development of safety-critical systems in a product line context raises the question if and to what extent the artifacts generated by the above described safety-related activities can be reused; it further poses the question: "*How product line engineering and safety engineering can be integrated?"*

In [Becker et al., 10], strategies for the integration of safety into PLE are discussed. The ideal conception would be to get the entire product line certified by a certification authority with the result that the derived products are considered certified as well. Due to the context dependency and the probably hazards produced by feature interference and also there is the evolutionary aspect of product lines, this goal is, in practice, infeasible. The safety argument for new features (added during application engineering) is not covered by the already done certification of the product line. The only way out would be to solely allow such changes in the newly derived products that are definitely not safety-relevant.

Another approach is to have a product line without any variability management for the safety-related artifacts i.e. the products are derived from the product line in the "classical way" without any safety considerations and the necessary safety artifacts are built anew for each derived product. This disregards a considerable reuse potential. However, [Becker et al.,10] warns that the reuse potential within safety engineering might be less than expected since minor changes in the architecture may cause considerable changes in the safety artifacts.

The converse approach would be to have a product line for safety engineering – the products themselves should be developed in a classical, non-PLE fashion, or in a separate ("safety-free") product line for the product derivation. Such an approach, however, seems to be also problematic since keeping safety engineering issues and the different products and their related artifacts might be a difficult task.

The only feasible way to integrated safety and product line engineering is through a holistic approach where both are set up together in one product line and safety is engineered into the product line. The artifacts produced during the safety engineering activities should become part of the core asset base where they are managed, like all others, in a prescribed well-defined process.

The relation of the safety assets to the production plan should be investigated, because it is the place where information about the safety engineering process has to be detailed.

2.3.2 A Holistic Approach to the Engineering of Safety-critical Product Lines

In the following, we outline a holistic, model-based, approach to an integrated safety-critical product line based on the work of Habli [Habli, 09].
This approach is a starting point for the identification and development of the necessary building blocks and best practices for our System Family Engineering Framework. It is worth noting that our approach will not be restricted to the procedures presented in the following sections.

There are in particular two "best practices" which very often are advocated in industrial settings whose appropriateness and pertinence are quite obvious:

- Encapsulate the safety relevant issues in a partition of the system as small and concise as possible and keep feature interaction with the rest of the product line features to a minimum
- Keep the number of variations in the safety relevant parts of the system as small as ever possible

These two rules are in particular relevant for product lines since effort and complexity increases with the number of safety relevant features/components and the variations therein. In particular, the two rules imply that safety and its variations have to be very carefully planned in the scoping phase of the product line (see Figure 2-8).

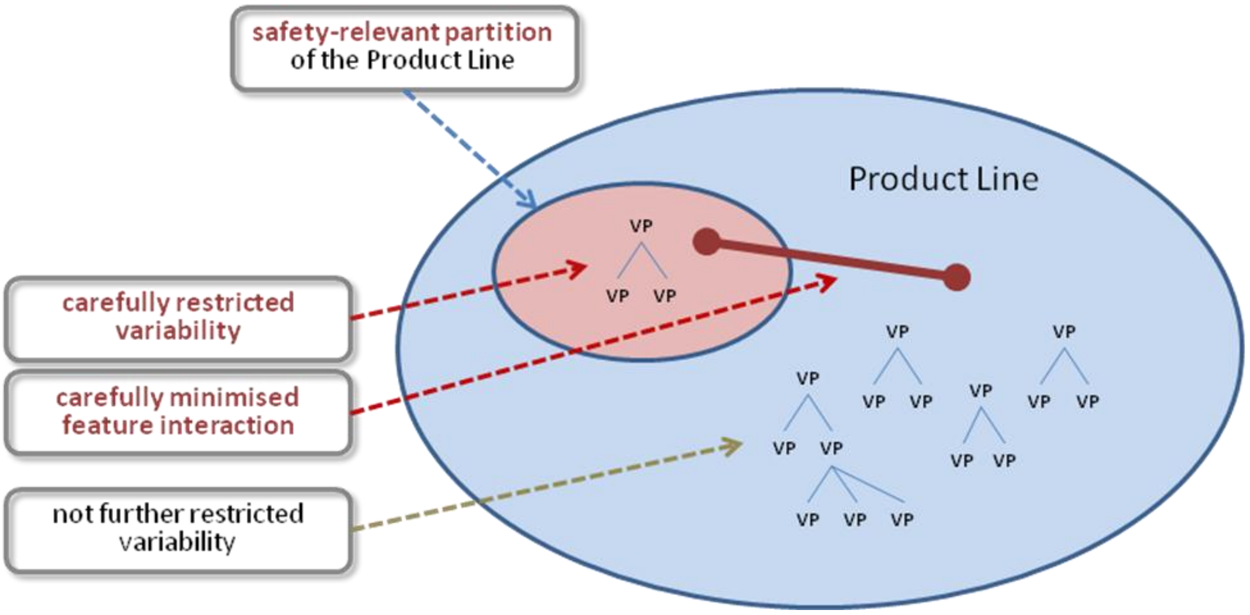
safety-relevant partition of the Product Line
Product Line
carefully restricted variability
carefully minimised feature interaction
not further restricted variability
VP
VP VP

Figure 2-8: Two basic "best practices"

Features and the specifically required contextual assets provide the entities to start the safety assessment process with. The contextual assets are outcomes of the contextual model which is a particular requirement for safety engineering since environment conditions and system context properties have an essential influence on system safety.

The basic concepts in the product line safety assessment process are safety conditions, causal relationships and safety requirements (see Figure 2-9). **Safety conditions** are an abstraction for safety condition types which are “capturing some ‘state-of-affair’ - event or state in the system or the environment” [Habli, 09]. In dependency of the domain or safety engineering method, safety conditions may be considered as hazards, failure conditions and faults, etc., just as the terminology requires.

Safety conditions are related to other conditions by causal relationships as “causes” or “effects” or “contexts” and they are associated to product line assets and the safety requirements they generate. This model is sufficient to cross-link the hazards, failures, etc., identified in the safety assessment phase, to each other, the safety requirements that are derived from them and the artifacts where they are described and “occur”. The elaborated linking of the safety conditions, in particular also to context and system assets, is essential in a safety-critical context since environment and deployment conditions have a potentially dangerous influence on the safety-relevant behavior of a system. There are many examples that a small change in the deployment method may have a far-reaching impact. Therefore, contextual and system assets (with may be either mandatory or subject to variation) are explicitly mentioned in Figure 2-9.

The proposed product line safety assessment process is split into two parts: The process for safety assessment, tightly integrated into the domain engineering process (see Figure 2-10), and the course of actions related to safety assessment during product derivation in the application engineering part of PLE. The two processes are sketched in the following sections.

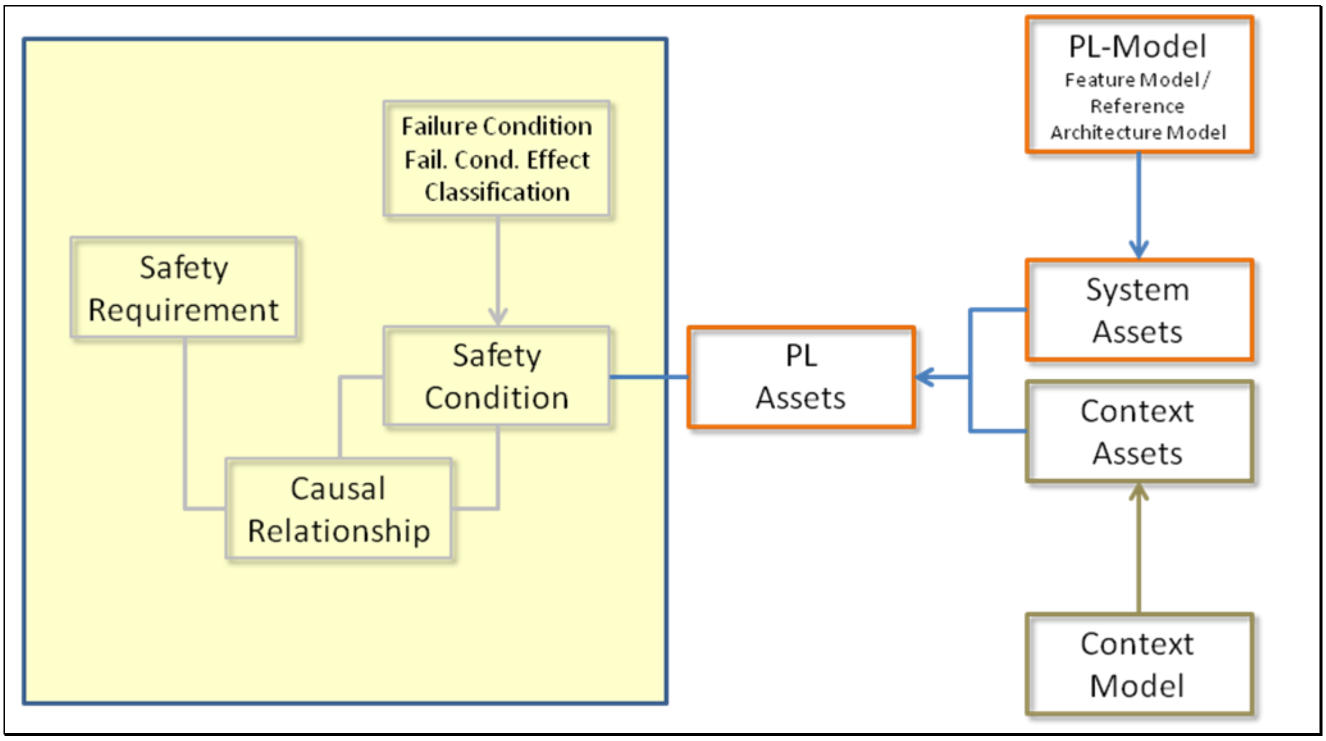

Figure 2-9: Product Line Safety Assessment and its relation to the product line assets

2.3.2.1 Safety Assessment in Domain and Application Engineering

In the domain engineering process, the failure behavior of the reusable functions and design assets have to be captured and analyzed. This is done by deriving three product line failure models which comprehend the safety conditions and how they are associated with the product line assets in the context, feature and reference architecture models. These models are:

- **Functional Failure Assessment in Domain Engineering:** Based upon the functional features and their dependencies with the context model (i.e. the product line environment). They become subject of the "product line" Functional Failure Analysis (FFA) or similar analysis methods wherewith the functional failure model, that is, the safety conditions and their traces to the context and feature models, are produced. Thereby, safety conditions subsume failure conditions, their effects, the respective classification of the risks with respect to type and severity, and the safety requirements.
- **Architectural Failure Assessment in Domain Engineering:** After having identified the failure conditions in the first step, now the failure modes have to be identified and analyzed. They are rooted in the design of the system and the expected behavior of the system environment. Failure mode chains, the causal derivation of the reason leading to a functional failure condition, have to be thus elicited (a) from the product line architecture model (the "reference architecture") where the reusable mandatory and optional design assets are defined and (b) from the respective context model. These artifacts again become subject to a safety analysis technique, as, for instance, Fault Tree Analysis (FTA) or a cognate analysis techniques (cf. root cause analysis) wherewith the failure mode causal chains and the derived safety requirements are produced.
- **Component Failure Assessment in Domain Engineering:** It is based upon the consideration of the components provided as core assets in the product line. This activity

defines the product line component failure model which relates the component failure modes with their effects. The analysis technique proposed is Failure Mode Effect Analysis (FMEA).

These three failure models “capture the *impact* of the defined *variation* in the product-line … on the product-line safety assessment data *explicitly*” [Habli, 09]. Safety assessment artifacts are produced and associated with the respective product line functions and design artifacts in a way that they can be reused during product derivation.

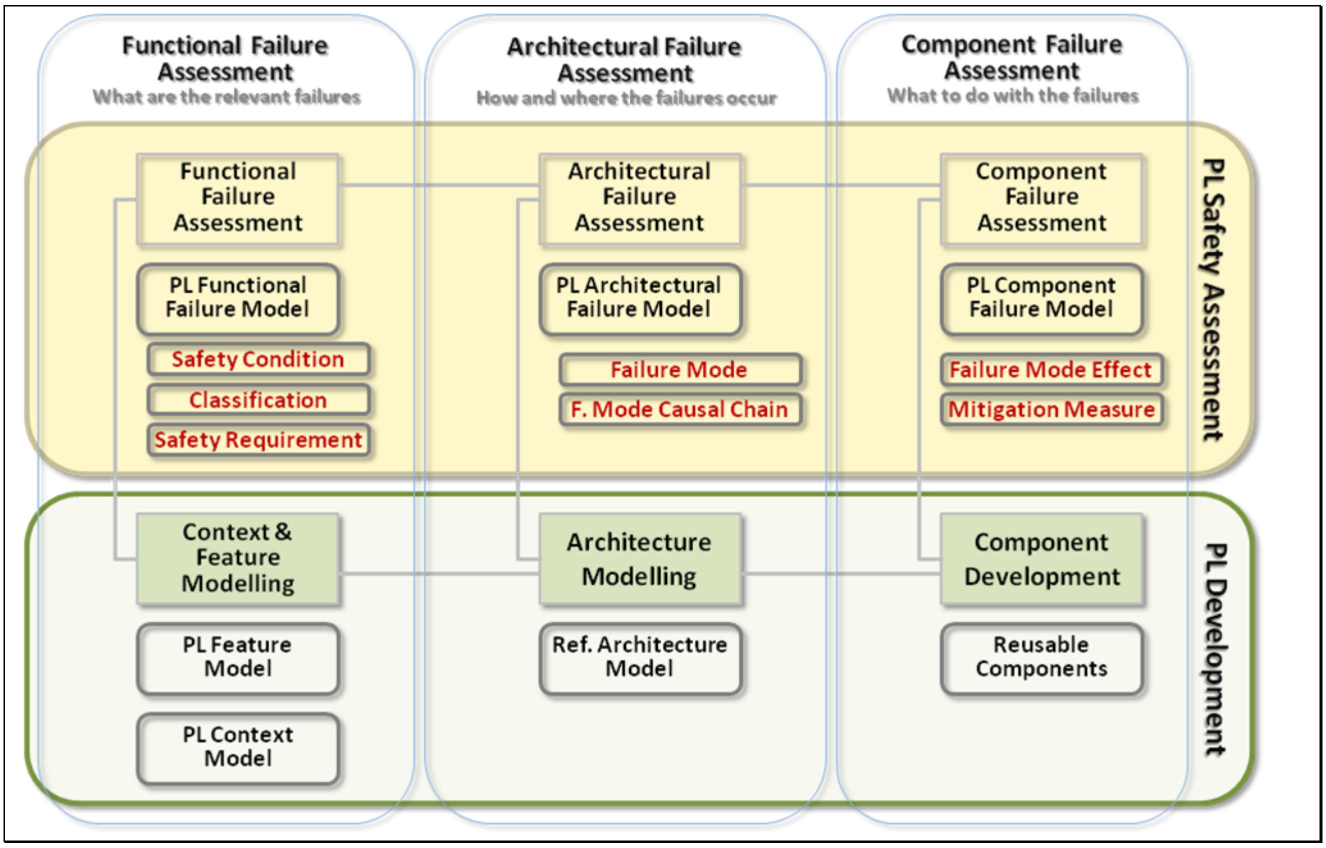
Functional Failure Assessment
What are the relevant failures
Architectural Failure Assessment
How and where the failures occur
Component Failure Assessment
What to do with the failures
Functional Failure Assessment
PL Functional Failure Model
Safety Condition
Classification
Safety Requirement
Architectural Failure Assessment
PL Architectural Failure Model
Failure Mode
F. Mode Causal Chain
Component Failure Assessment
PL Component Failure Model
Failure Mode Effect
Mitigation Measure
PL Safety Assessment
Context & Feature Modelling
PL Feature Model
PL Context Model
Architecture Modelling
Ref. Architecture Model
Component Development
Reusable Components
PL Development

Figure 2-10: Product Line Safety Assessment Process (Overview with regard to Domain Engineering)

Product Line Safety Assessment in the Application Engineering Process
Having derived the three product line failure models in the domain engineering phases, the obvious idea is to re-use them for the failure assessment during product derivation. This can be done to some extent, but needs considerable manual analysis. The product derivation safety assessment process has the same structure as its counterpart in domain engineering.

- The **product functional failure assessment** process starts, as soon as the configuration for the product is decided. Then, the features (system functions) are selected and the expected context is defined. Whenever it is necessary to add new features to the derived product, this has to be done now. With all these product features, a Functional Failure Analysis (FFA) has to be started, yet not from the scratch but using the already available product line FFA.
- The **product architectural failure assessment** is responsible for identifying the product’s failure mode chains in the context of the product architecture elements and to find out how they contribute to the products functional failure conditions. Unfortunately, slight alterations

of the product line reference architecture and mismatches with the product architecture can in principle invalidate the previously defined product line architectural failure model.

- The **product component failure mode assessment** procedure is very similar to the one devised for the product functional failure assessment: identifying the reused components and their associated failure modes; for all failure modes, determining the failure mode effects which are in context of a safety condition which in turn is associated to a reused component or contextual asset and collecting the associated mitigation measures.

Functional failure analysis helps to exclude unsafe configurations already in the domain engineering phase and its results can be taken over at least as far as the reused parts are concerned. Architectural failure mode analysis shows in domain engineering if solutions are unfeasible or unrealistic (e.g. too high mitigation costs); in application engineering, some effort has to be spent for a thorough manual analysis, even if some effort is gained by taking over the results from domain engineering. For the component failure analysis, there are no particular benefits on the domain engineering side, but the results can be taken over completely for the product derivation during application engineering.

The presented approach needs an alignment with the safety assessment and analysis techniques that are preferred in the use cases provided by the CRYSTAL applications. Guidelines and best practices have to be elaborated.

2.3.2.2 Safety Assurance (Certification and Process Assurance)

Safety Assurance is to document that all safety requirements are duly fulfilled. For this aspect, [Habli, 09] provides an approach to use the safety assessment models to develop a modular “product line safety case”, that is, a means to compose the safety cases of derived products from the product line safety assets. In order to corroborate the safety case(s), process assurance, that is, the evidence that a safety case is produced in a sound and convincing manner, is needed (and required by the certification authorities in most cases). [Habli, 09] also provides an approach to process assurance. In the following, a very brief summary of these two approaches will be given, in order to complete the picture for product line safety.

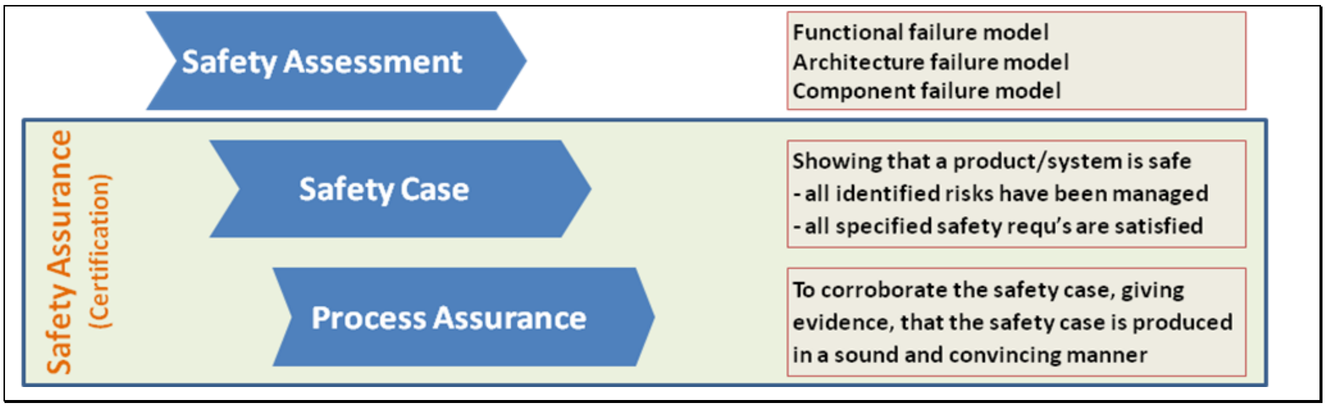
Safety Assessment
Functional failure model
Architecture failure model
Component failure model
Safety Assurance
(Certification)
Safety Case
Showing that a product/system is safe
- all identified risks have been managed
- all specified safety requ's are satisfied
Process Assurance
To corroborate the safety case, giving evidence, that the safety case is produced in a sound and convincing manner

Figure 2-11: Safety assurance

A safety case is a natural way to argue that a product or system is safe. It has to show that all identified risks have been managed, and all specified safety requirements are satisfied. The strength of such a “product-based” safety argument is corroborated by the argument about the quality of the engineering process that was followed in manufacturing the product. Safety case arguments for product lines should follow the structure of the product line reference architecture and be clearly traceable to its parts. There are five important “refers to”-relations:

- Argument Element — Product Line Asset: an argument element (e.g. a goal) refers to a product line asset thus expressing a claim concerning the reliability of the reusable asset.
- Argument Element — Product Line Process: an argument element refers to the product line process element that is responsible for its production (e.g., an analysis step, a test).
- Evidence Item — Product Line Asset: an evidence item (e.g. safety analysis data) refers to the product line asset it results from.
- Evidence Item — Product Line Process: an evidence item is associated to the process by which the respective product line asset is produced, ensuring that it can be identified wherefrom certain evidence with respect to a hazard comes. This also corroborates the process qualities.
- Product Line Asset — Product Line Process: Product line assets and the producing processes can be identified.

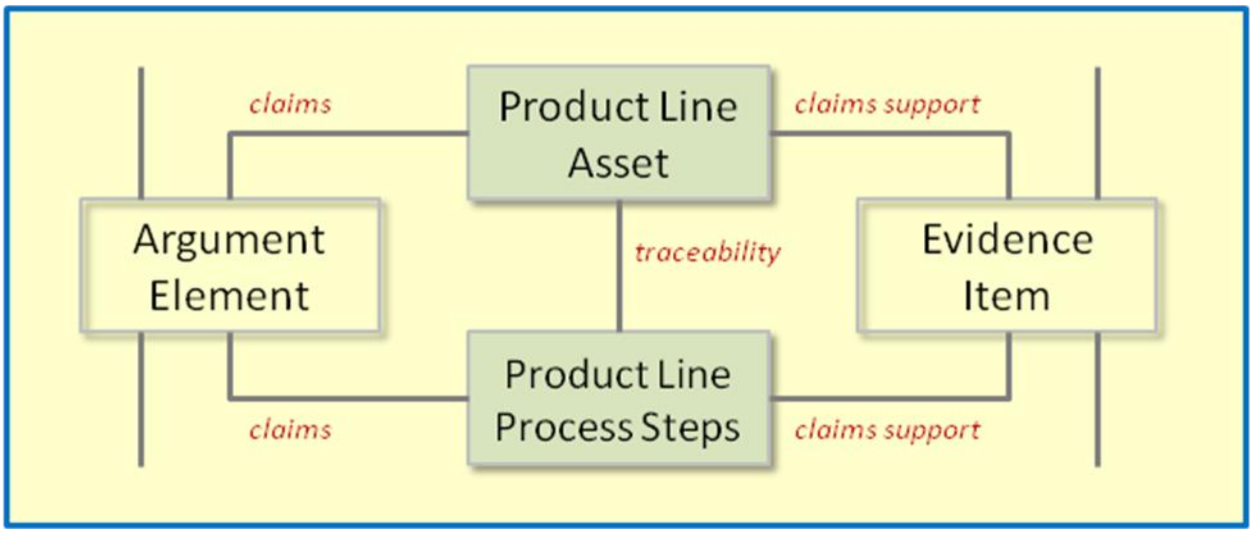

Figure 2-12: The core relations in a safety case

These five types of references describe the central relationship between the safety case and the product line assets and processes and they ensure that the impact of all the product line variability can be traced explicitly into the product line safety case. This allows the composition of the respective argument modules with their goals, solutions, assumptions, etc. according to the selected variability, but also the reference to the provenance of the assets, ensuring that it is traceable and justifiable.

[Habli, 09] concludes its approach with a framework for process assurance which is intended to provide a “process-based argument” for product safety. It is a way to ensure certification authorities that they can trust the evidence given in the safety case. The process assurance framework has four main steps:

- **Process Criteria:** Verifiable goals have to be defined which a product line process should satisfy in order to show a certain level of assurance.

- **Process Model:** Should be described thoroughly in order to allow a detailed and convincing analysis.
- **Process Analysis:** Aims at identifying safety flaws and vulnerabilities and eventually at the demonstration that the process is appropriate.
- **Process Justification:** Justifies the adequacy of the process to generate trustworthy evidence and the cost-effectiveness of a process, given safety-critical project constrains.

The process assurance framework needs to be aligned with the processes used in the development of the use cases. Best practices and pre-formulated descriptions for the entities produced by the process assurance framework steps have to be developed.

3 Application in CRYSTAL

This section provides an overview of the requirements of the different CRYSTAL Use Cases to the System Family Engineering Framework.

3.1 Requirements from the Use Cases

We have identified three Use Cases that require support for family engineering. This section provides an overview on these applications and their needs with regard to variability management. The three Use Cases are:

- UC 2.3 Mission Support Equipment, from EADS Cassidian
- UC 3.2 Autonomous Driving in Specific Environment, from Daimler
- UC 3.4 AVLAb, from AVL

3.1.1 UC 2.3 – Mission Support Equipment (Cassidian)

One of today's main challenges in the aerospace domain is to enhance the pilot's situational awareness. This means, that the pilot shall be able to identify, process and comprehend mission/flight critical elements of information under all circumstances e.g.:

- Take-off
- Landing
- Low level flight
- Planning of cargo load
- Mission data preparation

Possible root causes for loosing situational awareness can be for example:

- Confusion caused by multiple simultaneous visual and/or acoustic alerts
- Missing information about potential hazards
- Failure to meet planned objectives
- Ambiguous information

The risk of losing situational awareness might have catastrophic consequences with regards to mission or even human life.

The key success factor will be to support the pilot with information in a simple, precise way, clearly understandable, and focused on the relevant information. Therefore it is very important to adapt the information level depending on the current mission/flight scenario.

The Mission Support Equipment (MSE) shall therefore support different mission/flight scenarios. This shall be achieved by several functions to support the pilot on ground and/or in flight, providing adequate visual or acoustic information using databases, sensor data and/or a data fusion of both sources for a certain phase of flight/mission.

Complexity and System Features
During studies on airplane and helicopter mission/flight scenarios and workshops with pilots following functions have been identified, having potential positive impact on situational awareness:

- MAP display
- Tactical symbology
- On-ground mission preparation (map data, overlays)
- Real-time update of mission relevant information (e.g. weather or traffic)
- Obstacle detection and warning (sensor data and database)
- Landing aid based on sensor data and/or database
- Sensor data and database supported vision enhancement under bad environment condition (e.g. rain, snow, fog, dust, night)

3.1.1.1 Problem Description

For demonstration and evaluation purposes the Landing Symbology feature will be taken in this use case. The Landing Symbology supports helicopter pilots during the final landing approach in degraded visual environments which can be caused by e.g. rain, fog, dust and snow (see Figure 3-1). Many accidents can be directly attributed to such degraded visual environments where pilots often loose spatial and environmental orientation.

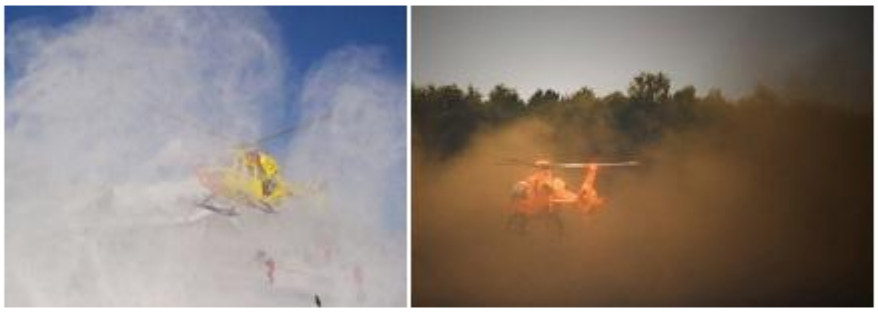
Figure 3-1: Degraded Visual Environments

The Landing Symbology feature allows to mark the landing point on ground using a head-tracked helmet mounted display. During the final landing approach it enhances the spatial awareness of flying crews by displaying 3D conformal visual cues on a helmet-mounted display. In addition it employs a surface grid conformal to the measured terrain for the landing area.

The Landing Symbology feature provides the following functionality:

- Display 3D conformal visual cues on a helmet mounted display visualizing the helicopter attitude and position relative to the intended landing point

- Determine and visualize the condition of the anticipated landing zone with respect to roughness and slope based on real time 3D data

- Display obstacles on a helmet mounted display relevant for the start and landing phase. The obstacles are taken from the real-time obstacle fusion, thus considering obstacles from the obstacle data bases and from real-time sensor obstacle classification.

Figure 3-2 shows an example of the landing symbology displayed during the final landing approach.

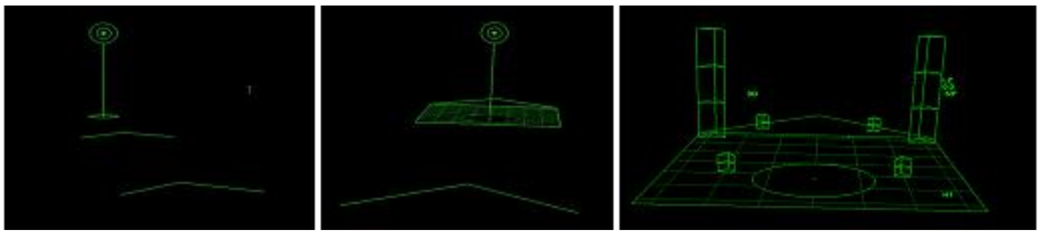

Figure 3-2: Display of the Landing Symbology

The Landing Symbology feature is part of a situational awareness system. It uses available information coming from databases and sensors.

3.1.1.2 Planned Extensions

In the context of CRYSTAL, several user stories have been defined for the MSE. In this section we describe only Variability Management User Story.

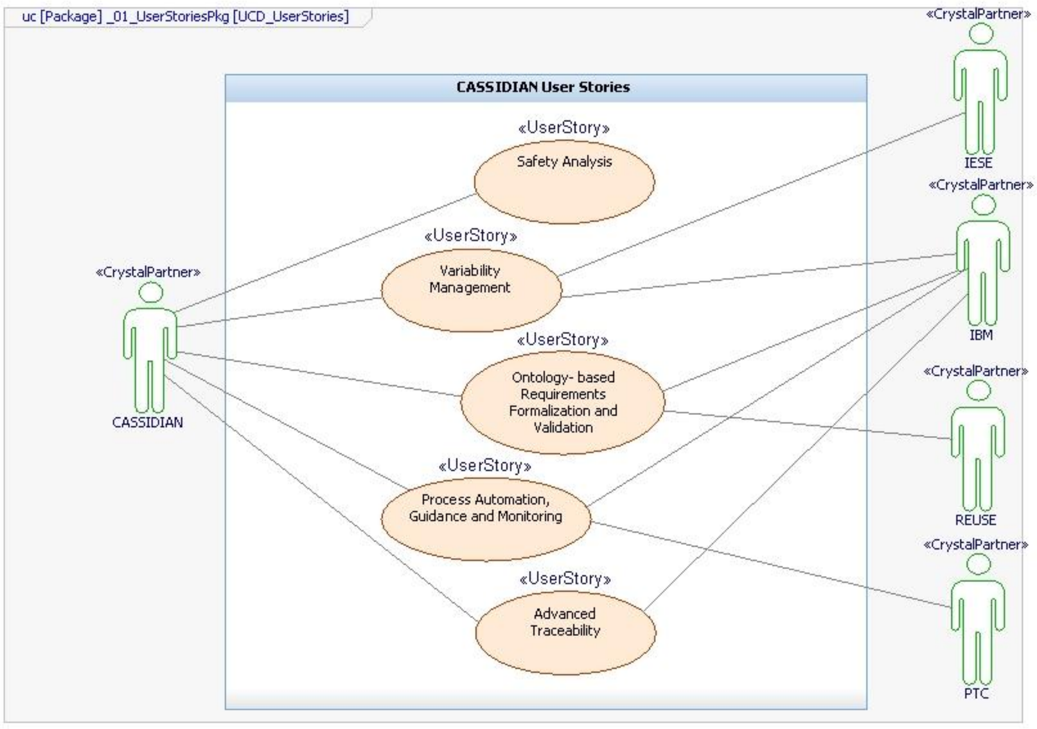

Figure 3-3: User Stories of UC2.3

This user story consists of a process to manage variability. It should support identification, specification and tracing of variability points to development artifacts. To achieve this, we need an approach which allows the support the following:

1. Variability modelling using feature models
2. Create a variability model covering external and internal variability.
3. Possibility to link the variability model with:

- Requirements.
- System / Safety Architecture and Design Model, including SW (architecture and source code)
- Test cases and Procedures

4. Product configuration
5. Create and manage component and system model libraries
6. Combine component and system model libraries together.
7. Modelling guidelines

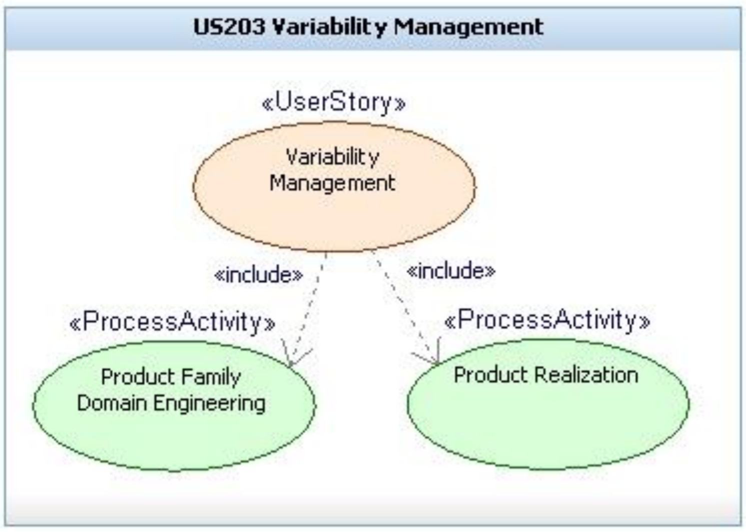

Figure 3-4: User Story Variability Management

3.1.2 UC 3.2 – Autonomous Driving in Specific Environment (Daimler)

UC3.2 faces the development of a host computer (next generation) for automated driving. The host computer is embedded in a big system called ADSE (autonomous driving in specific environment) and controls the vehicle to support testing activities of driver assistant systems with aspects of reproducible driving of trajectories, operation together with a partner car and operation with or without driver in the car. The host computer is not part of a series vehicle.

3.1.2.1 Problem Description

The project has to manage some common challenges of automotive OEMs:

- Transfer of knowledge of the engineering experts (implicit traces) into „explicit traces“ and methods in a tool based development system
- Support efficient impact analysis based on a given change request on requirement, system or component level to another abstraction level (vertical) or to the right side of the V-model (horizontal)
- Introduction of variant handling to manage the upcoming use cases of the host computer and to optimize the verification and validation activities
- Enable the consistency of different models on system, functional and component level

- Almost automatic safety case documentation during the development phase

3.1.2.2 Planned Extension

The project wants to gain improved procedures and their integration into the tool environment. Different tools and manual activities occur in this development process. Thus, the target is to raise the interoperability for a continuous, traceable and efficient development. The solutions have to be applicable so that the mentioned development project will be able to apply the processes, methods and tools after finalization of the CRYSTAL project, too. Furthermore, for other Daimler projects it should be possible to easily adapt the solutions for an efficient development process.

The rising number of artifact instances and the need of linking lead to the necessity for developing a continuous variant handling. So the different artifacts along the development process must have common variant attributes to support a general feature model and to use them for automated steps or for suggestions in manual steps.

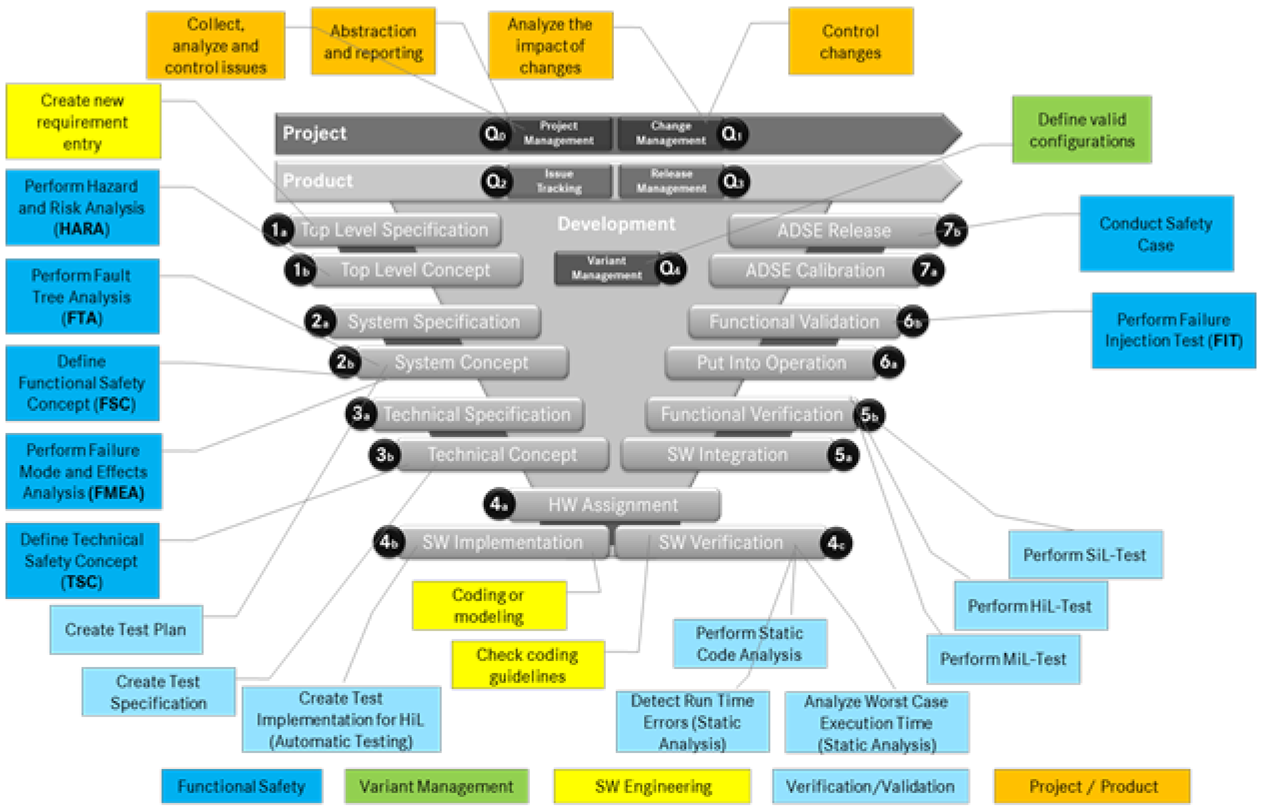
Collect, analyze and control issues
Abstraction and reporting
Analyze the impact of changes
Control changes
Create new requirement entry
Define valid configurations
Project
Q0
Project Management
Change Management
Q1
Product
Q2
Issue Tracking
Release Management
Q3
Perform Hazard and Risk Analysis (HARA)
1a Top Level Specification
Development
ADSE Release 7b
Conduct Safety Case
1b Top Level Concept
Variant Management
Q4
ADSE Calibration 7a
Perform Fault Tree Analysis (FTA)
2a System Specification
Functional Validation 6b
Perform Failure Injection Test (FIT)
Define Functional Safety Concept (FSC)
2b System Concept
Put Into Operation 6a
3a Technical Specification
Functional Verification 5b
Perform Failure Mode and Effects Analysis (FMEA)
3b Technical Concept
SW Integration 5a
4a HW Assignment
Define Technical Safety Concept (TSC)
4b SW Implementation
SW Verification 4c
Perform SiL-Test
Coding or modeling
Perform HiL-Test
Create Test Plan
Perform Static Code Analysis
Check coding guidelines
Perform MiL-Test
Create Test Specification
Create Test Implementation for HiL (Automatic Testing)
Detect Run Time Errors (Static Analysis)
Analyze Worst Case Execution Time (Static Analysis)
Functional Safety
Variant Management
SW Engineering
Verification/Validation
Project / Product

Figure 3-5: Engineering Methods assigned to different Steps in the V-Model

The single tool along the process chain should include the feature model for an easier use during the product development. The underlying data will be live exchanged via OSLC. This would be an enormous improvement in comparison to an import/export process nowadays.

3.1.3 UC 3.4 – AVLab (AVL)

The main need in this use case is the development of new and the integration of existing variability management methods, techniques, and tools in order to improve reusability in the automotive software development process. Systematic reuse is often not supported by existing approaches, as for example *Clone & own.* Nevertheless, many companies simply “reuse” existing solutions by copying them and adapting them for the new context. This approach can of course increase the efficiency, but has the disadvantage of various slightly different code bases. In case of identified bugs this could lead to problems, because it has to be identified which variants are affected and if they are affected the error has to be corrected in all variants.

Figure 3-6 illustrates the development steps which should be covered in this brick including MIL, SIL, and probably also HIL testing. Variability occurs at each development step and has to be represented, managed, and also bound. For example, for requirements this could mean that there is a set of requirements, which specifies the entire product space. Not all of these requirements are required for the instantiation of a concrete product. This means that requirements may be disabled for a specific product instance. The same applies for architecture, implementation, and testing. The reason why there is no repository for the architecture phase in this figure is that the architecture tool provides some kind of variability handling functionality. One task in this project will be the evaluation of these variability management techniques and their applicability in this concrete scenario.

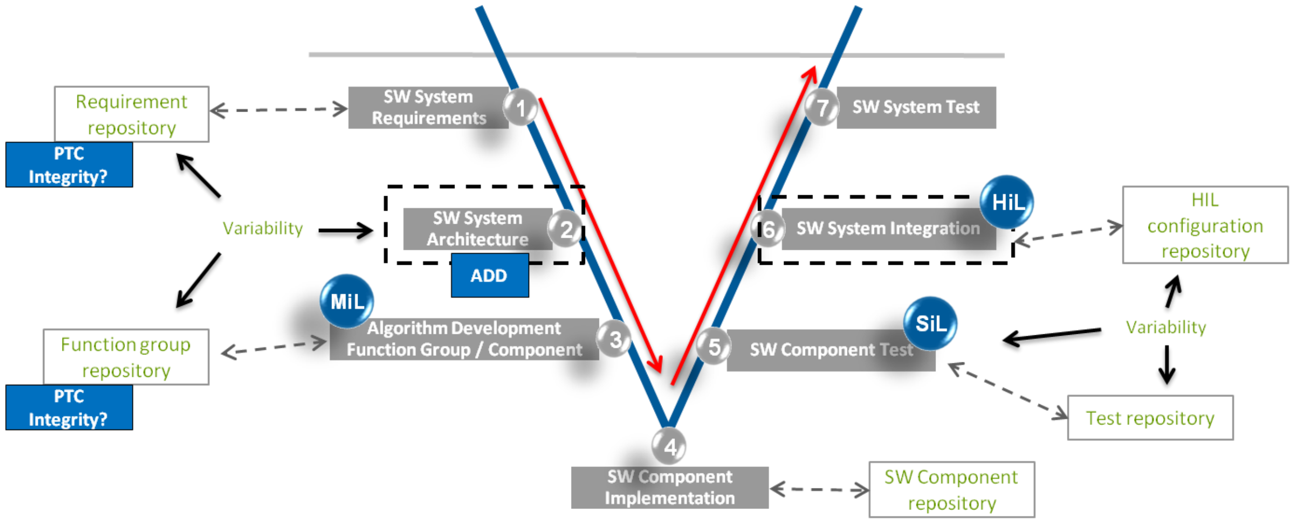

Figure 3-6: Integrated variability management approach

AVL-R applies a slightly improved variant handling approach compared to clone & own based on PTC Integrity RM and PTC Integrity Source. They have introduced a structure and method called *Powertrain Software Framework (PSF).* The basic building parts of the PSF are *function groups* and *components,* which are stored in *PTC* [2]*Integrity Source* and linked to Requirements in *PTC Integrity RM*. The concept of PTC Integrity enables basic variant handling based on versions and branches. The main problem with the current solution is the lacking distinction between version and variant.

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

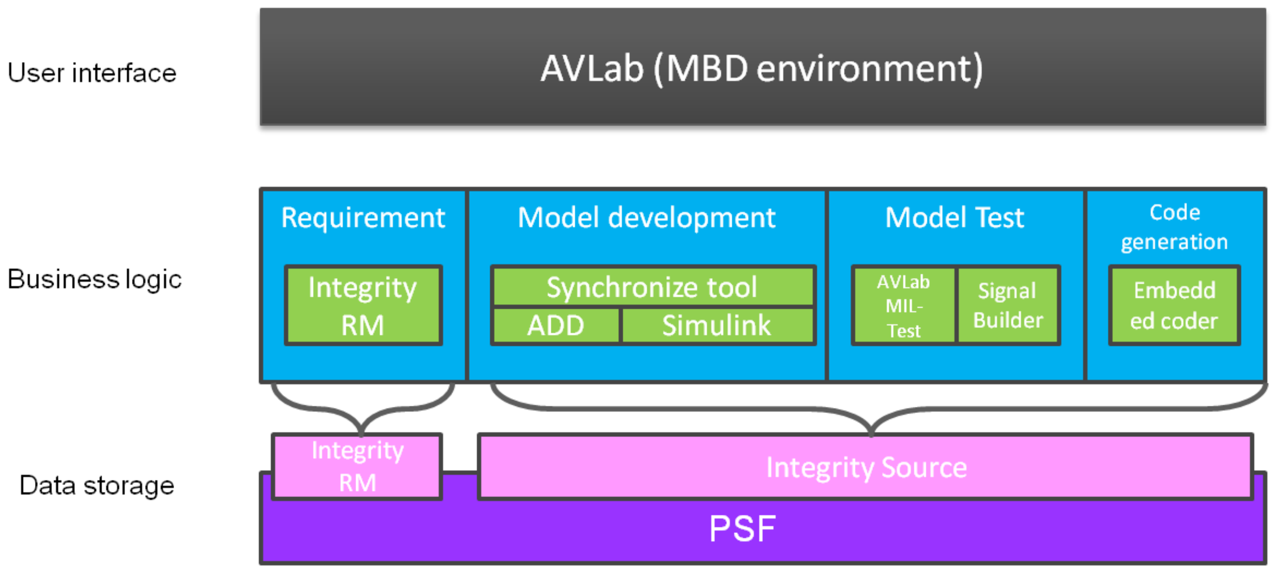

Figure 3-7: Current tool landscape

Figure 3-7 gives an overview of the current development environment. Integrity RM and Source are used for data management. This means that requirements, models, and test are stored in these tools, which also supports version control and traceability. The PSF can also be seen part of this layer. It use used to support reusability, but of course there are always project specific parts, which are simply stored in Integrity. The middle layer, here called business logic layer, covers all tools which are used for the different steps of the development process. The user actually works with AVLab, which is the common model-based development environment. The tools in the business logic layer are usually accessed through this common user interface. The improvement of interoperability will be part of another work package, but will be a main requirement for systematic variant management (see UC3.4b Description).

3.1.3.1 Problem description

The main problem which should be solved in CRYSTAL is the current lack of systematic reuse. As mentioned before, a reuse approach based on the PSF has been defined in the past. Nevertheless, reuse is still not as systematic as it could be. The main challenge in this specific use case is the existence of a broad product portfolio. This means that various domains (automotive, racing, shipping, etc.) can be supplied, but the actual range of products is rather small (control software). As a result, there is a high potential for reuse, especially in the domains, but sometimes even cross-domain.

The main challenge is the fact that AVL-R as a supplier cannot simply build a product line for their product portfolio, because they are highly dependent on customer requirements. Since the format of the requirements they get from customers can be quiet different, a solution based on requirements will probably be hard to achieve. Therefore, requirements may are not the appropriate way to identify applicable variants. One possible solution is the description of variability and commonality of the problem domain in so called feature models. Feature models have already been introduced in the nineties [Kang, 90] and are a good approach to describe variability.

One need of this use case is the definition of a concept how feature models can be used to describe variability at AVL-R. This also includes the question at which level of abstraction feature models should be applied to achieve the best performance. Most likely they should be applied on function group level, because there is little correspondence between different function groups. If a first investigation shows that there are dependencies, a second layer (an additional feature model), which describe the dependencies between the function groups could be introduced.

The description of variability is only one aspect in this use case. Furthermore, variability has to be reflected throughout the development process. This means that the development process and all involved tools have to be investigated in order to identify needs as well as possibilities for variability.

3.1.3.2 Planned extension

Figure 3-8 illustrates the potential extension of the existing development environment based on a first investigation. AVLab is a model-based development environment, which is (more or less) used as a single point of contact for developers. Therefore, the description of variability and selection of variants could also be integrated in this environment. For the storage of variants, there could also be a need to include strategies for variant handling at the lowest layer. Basically, this could be understood as a separation of problem and solution space [Czarnecki, 00]. The problem space specifies the problem from a more abstract point of view and the solution space describes various technical realizations.

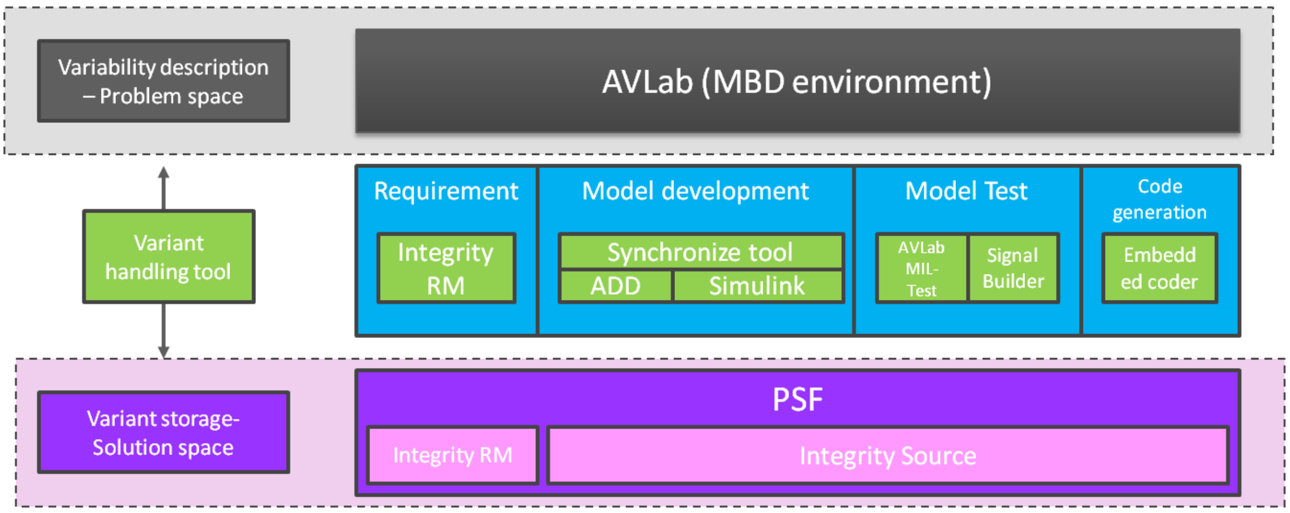

Figure 3-8: Planned extension of the current development environment

This requires as a first step the investigation of the current process and tool landscape in order to get an overview of the main challenges and needs. A second step will be the definition of a variability description concept. As mentioned before, this concept will most likely be based on feature models. Most probably there will be a common feature model structure, but single feature models for the different function groups. Additionally, the integration in the PTC tool environment has to be evaluated and maybe extended.

3.2 What will be implemented/provided in the CRYSTAL project

Based on the current Use Case descriptions, the following feature demands can be derived for the SFEF:

	UC2.3	UC3.2	UC3.4
Variability management for safety-critical systems	X	X	X
Multi-level Variability management for modular systems	X	X	X
PL scoping approach and tool	X	X	X
Overview on practical variability management approaches	X	X	X
Multi-view variability modelling (e.g.) external vs. internal features	X	X	X
Traceability to requirements artifacts	X	X	X
Traceability to design models	X	X	X
Traceability to test artifacts	X	X	X
Variability modelling guidelines	X	X	X
Incremental Variability Management adoption / improvements			X
Extractive core asset development, including analysis of existing variants for commonality and variability			X

Table 3-1: Requirements from Use Cases

These demands will be refined as part of the next project activities together with the Use Case partners. It is foreseen, that further Use Cases will raise demands regarding variability management, which are currently not considered. To this end, other Use Case specifications will be analyzed for further demands with regard to variability management.

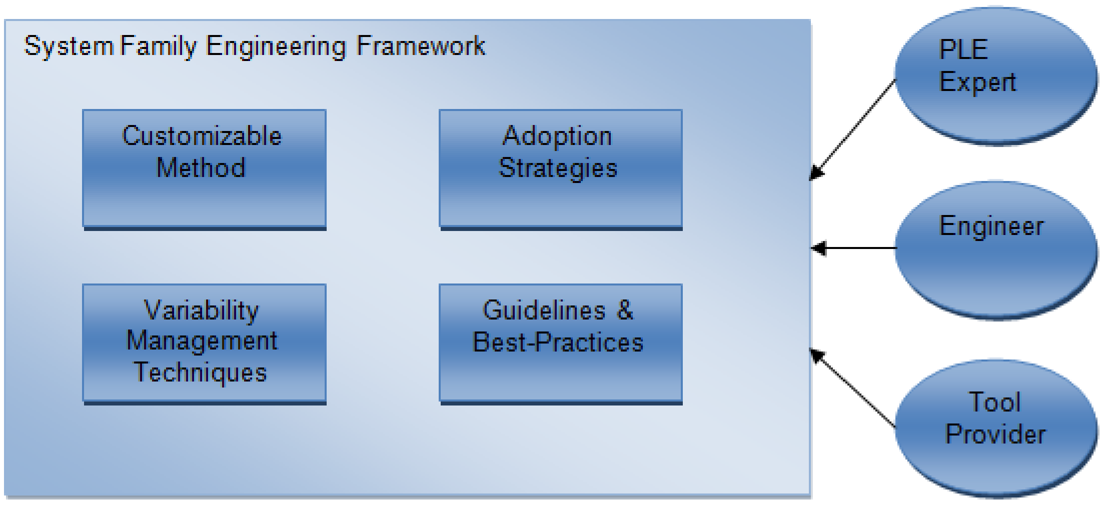
System Family Engineering Framework
Customizable Method
Adoption Strategies
Variability Management Techniques
Guidelines & Best-Practices
PLE Expert
Engineer
Tool Provider

Figure 3-9: SFEF components

Driven by the identified feature demands, the SFEF will provide the following features to the Use Cases:

1. **Customizable System Family Engineering method**
   Provide a method framework for the engineering of high-integrity system and software families, which identifies relevant practice areas and proposes concrete processes within the practice areas. Additionally, templates will be provided for the respective process activities, where helpful.
   The methods will be tool-independent, but tool-based, i.e. the brick will not depend on a specific variability management tool, but it will show how to conduct the practices with industrial-strength tool support.

2. **Family Engineering Adoption strategies**
   Provide reasonable strategies to improve family engineering in different settings, e.g. incremental transition from single system to family engineering, or improvement of existing family engineering practices.
3. **Variability Management Techniques**
   Provide an overview on practical variability management techniques and the respective tool support.
   Identify typical tool chain settings and interoperability demands for an integrated, orthogonal, multi-level, multi-view variability management
4. **Variability Management Guidelines**
   Provide guidelines, best practices, and examples for efficient and effective variability management in the Use Case settings. Where necessary, identify context factors, which have an influence on the guidelines, etc.

3.3 How will this brick be integrated in the Use Cases

The SFEF will be a customizable method, intended to be used in any of the domains approached by the CRYSTAL project. However, in order to make it more efficient and prove its applicability, we will create so-called profiles for the Use cases described in section 3.1.

These profiles are versions of the SFEF tailored for the enterprise needs. In order to do that, we will gather information about existing tool chain and process and this will be used to adapt our pre-defined methods and guidelines, as depicted in Figure 3-10.

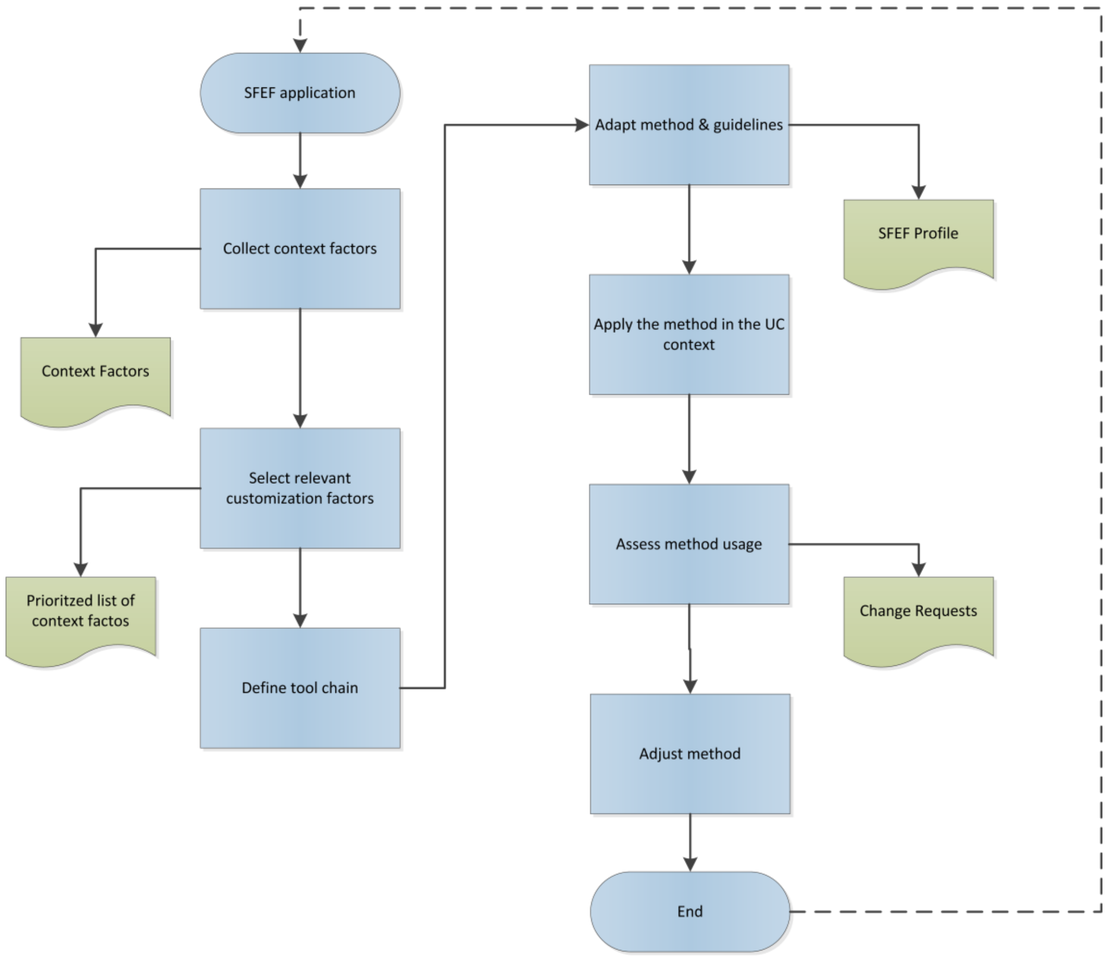

Figure 3-10: SFEF Instantiation

We plan to produce one profile for each of the Use Cases described in Section 3.1. Ideally, the profiles will be created in consecutive iterations, allowing the framework to be improved before the next usage (See the dashed line in Figure 3-10).

4 Terms, Abbreviations and Definitions

4.1 CRYSTAL-specific managerial abbreviations:

CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
R	Report
P	Prototype
D	Demonstrator
O	Other
PU	Public
PP	Restricted to other program participants (including the JU).
RE	Restricted to a group specified by the consortium (including the JU).
CO	Confidential, only for members of the consortium (including the JU).
WP	Work Package
SP	Subproject

Table 4-1: CRYSTAL-specific managerial abbreviations

4.2 ACRONYMS, used in this deliverable:

ADSE	Autonomous Driving in Specific Environment
FFA	Failure Functional Analysis
FMEA	Failure Mode Effect Analysis
FTA	Fault-Tree Analysis
MIL	Model in the loop
MSE	Mission Support Equipment
OEM	Original Equipment Manufacturer
OSLC	Open Services for Lifecycle Collaboration
PLE	Product Line Engineering
PSF	Powertrain Software Framework
PuLSE	Product Line Software Engineering
SEI	Software Engineering Institute
SFEF	Software Family Engineering Framework
SIL	Safety Integrity Levels
UC	Use Case
VM	Variability Management

Table 4-2: Acronyms

5 References

[Author, Year]	Authors; *Title*; Publication data (document reference)
[Kang, 90]	Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson: **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. Technical report, Carnegie-Mellon University Software Engineering Institute (1990)
[Bayer et al., 00]	Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, **PuLSE – Product Line Software Engineering,** Technical Report, 2000
[Czarnecki, 00]	K. Czarnecki and U. Eisenecker, **Generative Programming: Methods, Tools, and Applications**.Boston, MA: Addison-Wesley, 2000
[Bowen et al., 99]	Jonathan P. Bowen, Michael G. Hinchey: **High-integrity System Specification and Design. Formal Approaches to Computing and Information Technology FACIT**, Springer Verlag, 1999
[Northrop, 00]	Linda M. Northrop, Paul C. Clements. ***A Framework for Software Product Line Practice,*** **Version 5.0**. Software Engineering Institute, Carnegie Mellon University, July 2000
[Muthig et al., 04]	Dirk Muthig, Isabel John, Michalis Anastasopulos, Thomas Forster, Jörg Dörr, Klaus Schmid, **GoPhone - A Software Product Line in the Mobile Phone Domain**. IESE-Report No. 025.04/E, Version 1.0, March 5, 2004
[Pohl, 05]	Klaus Pohl, Gunter Beckie, Frank J. van der Linden. **Software Product Line Engineering: Foundations, Principles and Techniques**. Springer 2005
[Kakbla, 06]	Timo Kakbla and Juan Carlos Duenas, ***Software Product Lines – Research Issues in Engineering and Management***, Springer, 2006
[Habli, 09]	Ibrahim Mustafa Habli: **Model-Based Assurance of Safety-Critical Product Lines**. PhD Thesis, University of York, September 2009
[Becker et al., 10]	Martin Becker, Soeren Kemmann, Kodambali C. Shashidhar: **Integrating Software Safety and Product Line Engineering using Formal Methods: Challenges and Opportunities**. 1st International Workshop on Formal Methods in Software Product Line Engineering FMSPL 2010, co-located with the SPLC'10, Proc. of the 14th SPLC, Volume 2 – Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools, Jeju Island, South Korea, 13-17 September, 2010
[Krueger,14]	Charles Krueger , **Second Generation Product Line Engineering for Systems and Software (Part 3)** http://www.biglever.com/newsletters/2G_SPL_Part3.html (Accessed Jan 2014)