08AE-53

Software Product Lines in Automotive Systems Engineering

Steffen Thiel[1], Liam O'Brien[2], Muhammad Ali Babar[1], Goetz Botterweck[1]
1) Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland
2) NICTA – National ICT Australia, Canberra, Australia

Copyright © 2007 SAE International

ABSTRACT

Product line approaches are well-known in many manufacturing industries, such as consumer electronics, medical systems and automotive [1]. In recent years, approaches with a similar background have rapidly emerged within Software Engineering, so called Software Product Line (SPL) approaches [2], [3].

As automotive manufacturers and suppliers design and implement complex applications, such as driver assistance [4], they strive for mechanisms that allow them to implement such functionality on integrated platforms. This offers the opportunity to build a variety of similar systems with a minimum of technical diversity and thus allows for strategic reuse of components. This has resulted in a growing interest in SPL approaches both in the software engineering and the automotive systems domain.

This paper discusses the increasing importance that SPL approaches could play within the context of Automotive Systems Engineering. To accomplish this, we first provide an overview of the major challenges faced by Automotive Systems Engineering [5]. We then present a selection of SPL approaches, which could provide solutions for the described challenges. To complement this we make the case for empirical evaluation as a basis for well-founded decisions and selection of techniques.

Finally, we present and in-depth discussion of how the approaches and techniques outlined can be used to address the identified challenges. The paper concludes with an overview of open research questions and expected benefits for the development of automotive systems.

INTRODUCTION

Product line approaches are well-known in many manufacturing industries, such as consumer electronics, medical systems and automotive [1]. In recent years, approaches based on similar paradigms have rapidly emerged within Software Engineering, so called Software Product Line (SPL) approaches [2], [3].

As automotive systems get more and more complex and software-intensive manufacturers and suppliers face numerous challenges, such as the handling of complexity and variability, the need for integrated platforms and architectures and the inherent conflict between limited time and resources and high-quality products. In this paper we explore and discuss the contributions SPL approaches could make to provide solutions for these challenges.

The remainder of the paper is structured as follows: In the next section we give an overview of challenges faced by automotive engineering. Next, we first explain (on a general level) which role SPL Engineering could play within automotive systems engineering. We then discuss particular approaches which could offer solutions to the challenges discussed in the preceding section. We finish the paper by outlining related work and drawing conclusions.

SPL IN AUTOMOTIVE ENGINEERING

For more than 30 years reuse has been the long-standing notion to solve the cost, quality, and time-to-market issues associated with the development of software-based systems [6]. A major addition to existing reuse approaches since the 1990s are software product lines (e.g., [3], [7], [8], [9], [10], [11]). A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [7]. The basic reuse philosophy of software product lines is intra-organisational reuse through the explicitly planned exploitation of similarities between related products. This philosophy has been adopted by a wide variety of organisations and has proved to substantially decrease costs and time-to-mar-

ket and increase the quality of their software products (e.g., [7], [8], [9], [10], [11]).

Software product lines are particularly important to the automotive domain as car suppliers tend to integrate and combine more and more software functionality such as driver assistance, car dynamics, and airbag control systems on powerful platforms on different makes and models to meet different requirements across multiple markets. Many suppliers and manufacturers already use a product line approach so as to be able to build different variants of their products for use within a variety of automotive systems (e.g., [7], [12], [13], [14]).

In the software product line context, software products are developed in a two-stage process, i.e., a domain engineering and a concurrent application engineering process, as depicted in Figure 1 (e.g., [3], [8], [15]). The *domain engineering* process is responsible for establishing the reusable platform [3]. The platform consists of artefacts such as the requirements specification, architecture documentation, design specification, implementation, and test cases. Domain engineering defines the commonality and variability between product line members. It involves, among other activities, the implementation of an adequate product line architecture and a set of reusable software components such that the commonalities can be exploited economically while retaining the ability to vary the products.

The *application engineering* process is responsible for deriving product line applications or products from the platform established in domain engineering. It exploits the variability of the product line and ensures the correct binding of the variability to the specific needs of the customer. In many industrial organisations, these artefacts consist of both hardware and software components (e.g., [16]). Domain and application engineering is also known as core asset and product development (e.g., [7]).

Domain and application engineering are typically supported and coordinated by the technical and organisational management. The technical management supports co-operation between domain and application engineering. It includes tasks such as project planning, configuration management and support by software tools. The organisational management must create a basis that supports the technical activities within the existing organisational structure. Besides, it must ensure that the business goals of the product line are specified and communicated. In addition, the organizational management is responsible for the commitment of sufficient resources for the development of the reusable artefacts such that the advancement and effective use of those artefacts is ensured.

The idea behind a product line approach to product engineering is that the investments required to develop the reusable artefacts during domain engineering are outweighed by the benefits in deriving the individual products during application engineering (e.g., [7], [11]). A fundamental reason for researching and investing in approaches and technologies for product lines is to obtain the maximum benefit out of this upfront investment – in other words, to minimise the application engineering costs of automotive systems.

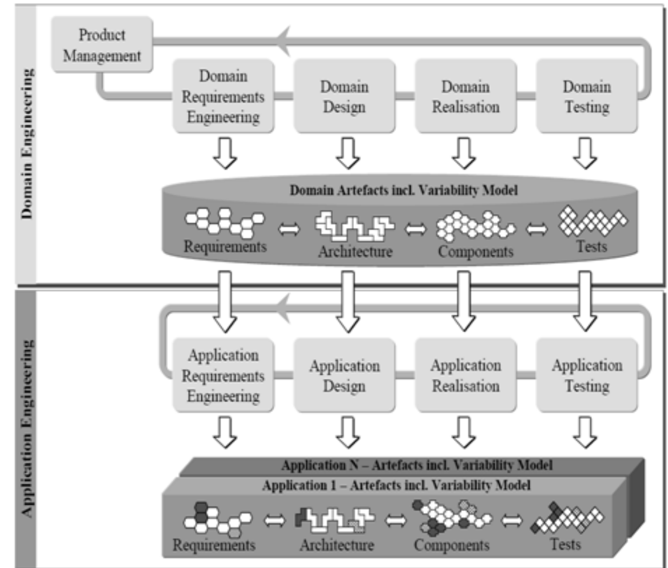

Figure 1 – Software Product Line Engineering [3]

Clements et al. [7] provide a product line practices framework that describes essential practice areas as well as the relationship of these practice areas within system and software engineering, technical management, and organizational management. The framework identifies 29 practice areas that need to be considered when moving from single system development to product line development (see Figure 2).

Technical Management	Systems and Software Engineering	Organizational Management
Scoping	Understanding Relevant Domains	Market Analysis
Technical Planning	Requirements Engineering	Technology Forecasting
Data Collection, Metrics, and Tracking	Architecture Definition	Building a Business Case
Technical Risk Management	Architecture Evaluation	Funding
Make/Buy/Mine/ Commission Analysis	Component Development (Make)	Structuring the Organization
Configuration Management	COTS utilization (Buy)	Operations
Tool Support	Mining Existing Assets (Mine)	Launching and Institutionalizing
Process Definition	Developing Acquisition Strategy (Commission)	Customer Interface Mgmt
	Software System Integration	Organizational Planning
	Testing	Organizational Risk Mgmt
		Training

Figure 2 – Practice Areas in Product Line Development (adapted from [7])

For example, the practice area "Scoping" refers to the specification of products and features that should be in scope for the product line under consideration. Scoping requires inputs from market analysis, business case and technology forecasting. The result provides a clear guideline of "what to build" during the development. It also provides support for the development of the re-

quired core assets. Other practice areas refer to similarly important topics of product line development and help to address and avoid common pitfalls.

CHALLENGES IN AUTOMOTIVE ENGINEERING

We will now present some challenges that automotive engineering faces today. These have been collected from a review of the relevant literature [4, 5] as well as from workshops with industry partners.

REDUCTION OF COMPLEXITY

Many automotive suppliers and manufacturers such as Cummins [7], Bosch [12] [17], DaimlerChrysler [14], and Volkswagen [18] use a product line approach so as to be able to build different variants of their products for use within a variety of automotive systems. The size of the product lines is usually large since only in this case significant economies of scale can be achieved. Therefore, automotive software platforms are typically developed in such a way that they can be customized and used in hundreds of products (e.g., [17]). These platforms can easily incorporate thousands of variation points and configuration parameters.

Managing this amount of variability is extremely complex and requires sophisticated modelling techniques. In particular, there is a strong need for appropriate approaches that support the different stakeholders in carrying out their development tasks in complex software product line efforts with a large number of variants [19].

IMPROVED ARCHITECTURAL DESIGN PRACTICES

Software architectures for automotive systems need to be comprehensive enough to capture and describe the multi-functionality and all related issues. A comprehensive architecture should provide a sophisticated structural view that covers all the aspects that are relevant to different kinds of stakeholders such as drivers, passengers, maintainers, and production staff. A comprehensive car architecture should be described at different levels such as functionality level, design level, cluster level, hardware level, and deployment level [5].

Architecture and interface specification is another big challenge in software engineering for automotive systems. There is general lack of suitable and reliable methods to accurately and sufficiently provide interface specifications. This is particularly important in the automotive industry because of its distributed mode of software development.

From software to systems engineering

Traditionally the design, manufacturing and assembly of cars relied on a modular approach, which required that each component was as independent as possible from others. However, in recent years we see more and more complex functions (such as occupant protection systems or driver assistance systems) which (1) require complex control processes (which lead to software-intensive implementations) and (2) require the integration with various subsystems in the car (which results in clusters of hardware and software components that are highly integrated and interdependent).

This change in implementation structures has to be reflected in the engineering processes used to create these structures. Hence, automotive engineering has to adapt methods and techniques from software engineering. But that's not enough. What really is required are approaches that incorporate both hardware and software components and their interplay. Hence, we have to move on from software engineering to system engineering.

IMPROVED EVIDENCE IN MODELING AND EVALUATING QUALITY ATTRIBUTES

A quality attribute is a non-functional requirement of a software intensive system, e.g., reliability, modifiability, performance, usability and so forth. According to IEEE Standard 1061 [20], software quality is the degree to which the software possesses a desired combination of attributes. Quality is a major issue for automotive systems engineering. Like any other domain, software quality is fundamental to any automotive system's success. However, quality as a concept is very challenging to define, describe and understand. This aspect of any system has very strong subjective interpretation. That is why it is vital to systematically elicit and precisely define quality aspects of a software intensive automotive system. There are a number of classifications of quality attributes. McCall listed a number of classifications of quality attributes developed by software engineering researchers including himself [21]. A later classification of software quality is provided in [22].

An examination of quality definitions, meanings and views in [23] describes quality as hard to define and measure but easy to recognize. However, quality experts including some with a software background have proposed models, e.g., [24], [25], [26], not to measure quality itself but to measure surrogate attributes which, when combined, can provide a notion of the quality of the product. While, there are several quality attributes required by different automotive systems, there are two main quality attributes that must be satisfied at a level required by the law of a country, where an automobile is going to be sold. These two quality attributes are reliability and safety.

Legislation at national levels, European Union and global levels along with market place demand is directing the automotive industry to comply with more stringent safety levels, increasing reliability levels, higher fuel efficiency and low emission levels [27]. The AUTOSAR standard is a move to establish an open standard for automotive embedded electronic architecture. AUTOSAR tries to achieve modularity, scalability transferability and reusability of functions. However there is

no formal verification of reliability and safety concerns or formal (mathematically based) testing in AUTOSAR.

DESIGN AND EVALUATING ARCHITECTURES FOR FAMILY OF SYSTEMS

Software architecture provides the key framework for the earliest design decisions taken to achieve functional and quality requirements [28]. Architecture for a family of systems also helps identify the commonality among different systems and explicitly document variability. As we have mentioned architecting automotive systems is a complex and challenging design activity, and architecting a product family is even more challenging. It involves making decisions about a number of interdependent design choices that relate to a range of design concerns. Each decision requires selecting among a number of design options; each of which impacts differently on various quality attributes. Additionally, there are usually a number of stakeholders participating in the decision-making process with different, often conflicting, quality goals, technical and project constraints, such as existing platforms, cost and schedule [29].

Apart from the challenge of devising optimal architectural solutions, specifying the architecture and interfaces of components of an automotive system is a difficult task, which poses several kinds of challenges. Usually OEMs (Original Equipments Manufacturers) have to provide an overall architecture of the automotive systems in its cars and distribute these to potential suppliers of systems and components who do the implementation. The AUTOSAR standard is a move to establish an open standard for automotive embedded electronic architecture. AUTOSAR tries to achieve modularity, scalability transferability and reusability of functions. However, even if the architecture and components are specified using AUTOSAR, there is still no checking of conformance or conformance validation. Hence, there is a need for specific methods and tools to validate that those implementations actually conform to the specifications and that the combination of the various implementations conforms to the OEMs' specifications.

ARCHITECTURE KNOWLEDGE MANAGEMENT

We have mentioned that designing and evaluating software architectures of a family of systems in automotive domain involves complex and knowledge intensive tasks. The complexity lies in the fact that tradeoffs need to be made to satisfy current and future requirements of a potentially large set of stakeholders, who may have competing vested interests in architectural decisions. The knowledge required to make suitable architectural choices is broad, complex, and evolving, and can be beyond the capabilities of any single architect [30]. Due to the recognition of the importance and far reaching influence of the architectural decisions, several approaches have been developed to support architecting processes. Examples are the Generic Model for architecture design [31], Attribute-driven design [32], Architecture Tradeoff Analysis Method (ATAM) [33], 4+1 views [34], Rationale Unified Process (RUP) [35] and architecture-based development [36]) While these approaches help to manage complexity by using systematic approaches to reason about various design decisions, they provide very little guidance or support to capture and maintain the details on which design decisions are based, along with explanations of the use of certain types of design constructs (such as patterns, styles, or tactics). Such information represents architecture knowledge, which can be valuable throughout the software development lifecycle [37].

Lack of a systematic approach to capture and use architecture knowledge may preclude organizations from growing their architecture capability and reusing architectural assets. Moreover, the knowledge concerning the domain analysis, architectural patterns used, design alternatives evaluated and design decisions made is implicitly embedded in the architecture and/or becomes tacit knowledge of the architect [38].

INCREASED PROCESS EFFICIENCY

Reduction of costs-per-product, shorter time to market

One of the greatest challenges for engineering approaches are the conflicting goals of creating products with a sufficient quality level and reducing the costs-per-product. On the one hand, components have to fulfil the required quality criteria, especially for safety-relevant functions. On the other hand, in automotive engineering we have to deal with target costs which are fixed in advance and hence dictate the resources which are available for component design and implementation. This conflict has to be dealt with by approaches and process models that allow for efficient creation of components with the required quality level.

Seamless model-driven development

The intelligent use of models promises to be one of the major foundations for efficient processes in automotive systems engineering. Models can be used to describe the different forms of knowledge which are captured and transformed during the engineering process, such as requirements, high-level or detailed design, implementation or test cases. This is the foundation for tooling and automation, which in turn promotes efficient engineering processes [5].

However, the current usage of models in automotive systems engineering is insufficient and is far from realising its full potentials. For instance, models are used in isolated areas, without an integrated flow of information. Often models are used only in a semiformal way as a form of communication on the whiteboard or as illustrations in a textual specification.

This lack of clearly and precisely defined semantics undermines the use of real model-*driven* approaches, where models are expressive enough to be used in pow-

erful interactive tools and the automatic derivation of further artefacts including the implementation.

Balancing agile approaches with plan-driven development

Traditionally automotive systems engineering, especially when performed in an industrial scale, is dominated by systematic and plan-driven approaches. Similar systematic and plan-driven approaches have been promoted in software engineering – focussing on process structures, comprehensive documentation, contracts and plans. These fixed and rigid approaches have been criticized by the, so called, Agile Software Development community which values "individuals and interactions, working software, customer collaboration and responding to change" instead [39].

Under certain conditions it might be beneficial to introduce similar agile approaches to automotive systems engineering. So far, however, it remains unclear under which conditions this is appropriate and how agile approaches can be balanced and integrated with plan-driven processes.

Although SPL and Agile approaches both aim to increase the efficiency of the software development process and shorten the time to market, they address this challenge from different perspectives (flexibility vs. controlled process) which – at least at first sight – seem to be contradictory.

So it remains an interesting research challenge to identify conditions under which a combination of agile and plan-driven approaches is beneficial – and how the special constraints of automotive systems engineering, such as the rigid quality and safety requirements, influence such scenarios. A related challenge is the question how agile approaches can be scaled up [40] to meet the requirements of large, industrial SPL projects as they are found in the automotive industry.

APPROACHES TO ADDRESSING CHALLENGES

We are now going to discuss approaches that address the challenges we identified earlier.

VISUALLY INFORMED VARIABILITY MANAGEMENT

As mentioned in the previous section, industrial size automotive product lines can easily incorporate thousands of variation points and configuration parameters for product customization. A promising approach to address the complexity problem in automotive systems engineering is to improve the efficiency of variability management and product derivation.

Variability management is the process by which the variability of the product line development artefacts (e.g., architectural models, software components, and hardware components) is planned, documented, and managed throughout the development lifecycle. Variability management supports critical product line engineering tasks such as product derivation.

Variation points identify locations in product line artefacts at which variation will occur [41]. The binding time refers to the moment in a product's lifecycle at which a particular variant for a variation point is bound to the system, e.g., pre- or post-deployment. A realisation mechanism refers to the technique that is used to implement the variation point. A variant is an artefact that has been realised (or configured) for a particular product. It can also be used to describe a derived product.

As mentioned before (see Section "SPL IN AUTOMOTIVE ENGINEERING") a large part of the application engineering activities in an established and optimized product line approach consist of reusing the platform artefacts from domain engineering and binding the variability as required for the different applications/products. In this process, the variability realisation mechanisms defined in the domain design and implementation artefacts are exercised and fixed to the particular customer product. Potentially pre-developed customer-specific application components are integrated into the product stub without violating the design decisions embedded in the overall software and hardware architecture. If domain and application engineering are concerted in such a way then product production becomes more a configuration and composition than a development activity. In this way, a mass-customization of products can be achieved and the upfront investment in domain engineering can be compensated by orders of magnitude.

Systematic variability management and product derivation can be supported by visualisation techniques and tools that support the understanding, management, and effective use of product line development artefacts, their built-in variability, and the dependencies among them.

Visualisation has proven useful to amplify cognition in a number of ways, for example, by increasing the "memory" and "amount of processing" available to users, by supporting the search for information, and by encoding information in a manipulable medium [42]. Visualisation takes abstract data, and gives it a form suitable for visual presentation. Such data can be explicitly collected from software or can be codified by software engineers from their own implicit knowledge. With suitable techniques such visualisations can also amplify the cognition about large and complex data sets created and used in industrial software product line engineering. The exploration of the potential of visual representations, such as trees and graphs, combined with the effective use of human interaction techniques, such as dynamic queries and direct manipulation, are interesting and novel research aspects in the software product line context.

Particularly, there are three areas where visualisation and interaction techniques could be effectively applied (see Figure 3):

1. Domain and application requirements artefacts
2. Domain and application realisation artefacts
3. Dependencies among requirements and realisation artefacts

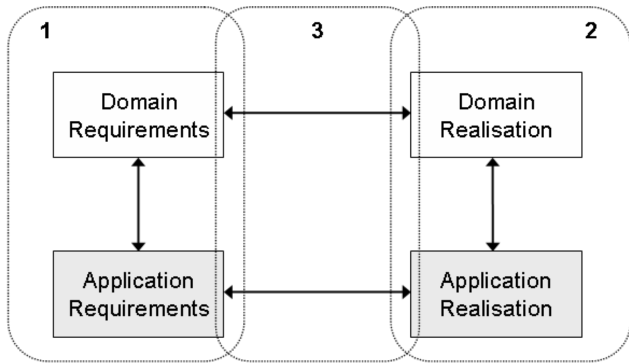

Figure 3 – Modelled areas which can be visualized

For an example of a tool for visually informed variability management see the screenshot of the VISIT-FC prototype [43] shown in Figure 4.

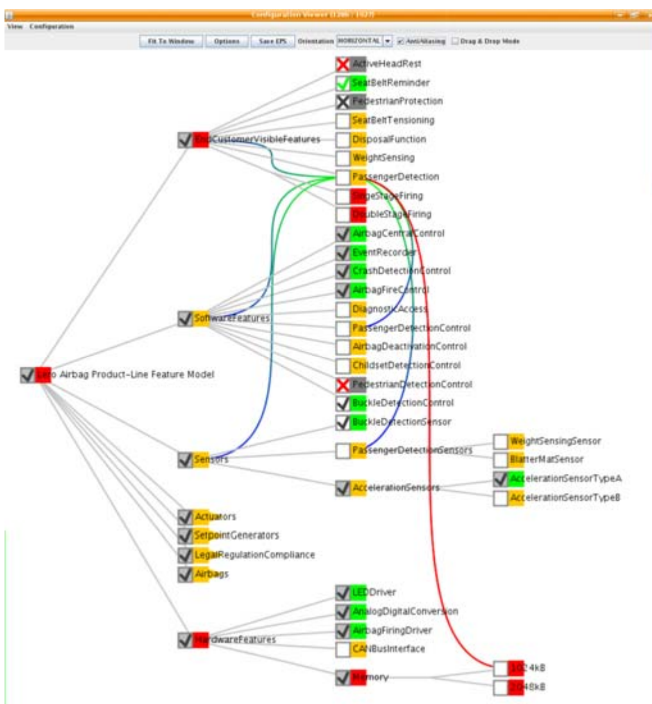

Figure 4 – Visual and interactive presentation of a feature configuration [43]

Potential outcomes of visually informed variability management approaches that address COMPLEXITY are:

- *Reduced conceptual complexity.* Reduced conceptual complexity in understanding common and variable requirements and realisation artefacts and the dependencies among them in product lines with a high number of variation points. Performance improvement measures include the time for determining the number of common and variable requirements in the product line, the time for determining the number of dependencies among requirements and realisation artefacts.
- *Increased derivation efficiency for requirements artefacts.* Increased efficiency of work tasks of product line stakeholders dealing with the derivation of requirements artefacts during product derivation. Measures include the time for deriving the requirements specification for a particular customer product.
- *Increased derivation efficiency for realisation artefacts.* Increased efficiency of work tasks of product line stakeholders dealing with the derivation of realisation artefacts during product derivation. Measures include the time for deriving a product-specific realisation based on a given product-specific requirements specification.
- *Improved quality of derivation.* Improved quality of the derivation process by reduction of the probability of derivation errors caused by stakeholders dealing with requirements and realisation artefacts. Measures include the probability of errors for deriving the requirements specification and realisation artefacts for a particular customer product.

ARCHITECTURE-BASED DEVELOPMENT

We have discussed that software architecture embodies some of earliest design decisions, which are hard and expensive to change if found flawed during downstream development activities. The role of software architecture in a family of system becomes much more vital as architectures of individual products are derived from the core architecture. Hence, any flaw in the core architecture usually has ramifications for the achievement of required quality attributes for individual products in a family.

A systematic and integrated approach is required to address architectural issues throughout the software development lifecycle. Hofmeister et al. have proposed a general model of software architecture design [31]. This model has three activities: architectural analysis, architectural synthesis, and architectural evaluation. However, it does not consider the post-architecting activities, which are equally vital to ensure that intent behind architecture design remains correct during implementation and maintenance of software architecture. Such an approach is called architecture-based development [36].

One of the main characteristics of architecture-based development is the role of quality attributes and architecture styles and patterns, which provide the basis for the design and evaluation of architectural decisions in this development approach.

Figure 5 shows a high level process model of architecture-based development that consists of six steps, each having several activities and tasks.

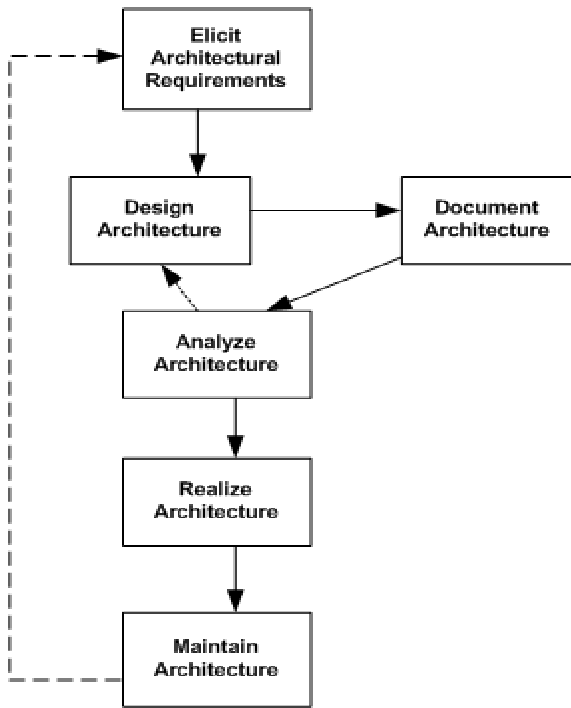

Figure 5 – Architecture-Based Development Process model [36]

Architecture and interface specification

To address the challenges in architecture and interface specification we are developing methods and techniques for architecture conformance validation with an initial focus on structural behaviour. The method is based on an extension of the Reflexion modelling technique and is being tailored for use within a software product line context. This method can be used as part of the development or maintenance processes within the development organisation.

The main outcome of introducing such a method is to ensure that the architecture of the implemented systems actually is in step with what is designed. This conformance can be validated at various stages in the development or maintenance cycle. This ensures that costly rework or re-architecting of the implementation is avoided.

Quality attribute modelling

To address the elicitation and precise specification of quality attributes, we have been working on refining and evaluating existing scenario-based techniques as well as developing new ones. Scenarios have been used for a long time in several areas (military and business strategy and decision making). The software engineering community started using scenarios in user-interface engineering, requirements elicitation, performance modelling and more recently in software architecture discipline. Scenarios are considered quite effective for characterizing quality attributes (e.g., performance, usability) for eliciting and specifying software architectures because they are very flexible. Scenarios can be used for precisely specifying most quality attributes. For example, we can use scenarios that represent failure to specify availability and reliability, or scenarios that represent change requests to describe modifiability. Moreover, scenarios are normally concrete, enabling the user to understand their detailed effect. Scenarios used by the software architecture community can be classified into various categories, such as direct scenarios, indirect scenarios, complex scenarios, use case scenarios, growth scenarios, or exploratory scenarios.

Researchers have developed various frameworks for eliciting, structuring and classifying scenarios. For example, Lassing et. al. proposed a two dimensional framework to elicit change scenarios [44], Kazman et. al. proposed a generic three-dimensional matrix to elicit and document scenarios [45]. Bass et. al. provided a six elements framework to structure scenarios [46]. We have empirically evaluated different techniques for eliciting and structuring scenarios. Based on the results of our empirical research, we have devised various techniques to elicit and structure architecturally significant requirements. Table 1 presents a four element framework to elicit and structure scenarios. This framework is based on the six element framework suggested by Bass et al.

Elements	Brief Description
Stimulus	A condition that needs to be considered when it arrives at a system
Response	A measurable activity undertaken after the arrival of the stimulus
Environment	A system's condition when a stimulus occurs, e.g., overloaded, running etc.
Stimulated Artefact	Some artefact that is stimulated; may be the whole system or a part of it.

Table 1 – Four elements scenario generation framework (Adapted from [46])

Having precisely specified quality attributes, an architect needs to sufficiently address the required quality attributes. We have mentioned that reliability and safety are considered the most vital quality attributes required by law in automotive industry. Our research has been focused on developing methodologies to improve safety, reliability and facilitate certification conformation to the emerging AUTOSAR standard. One methodology being developed incorporates formal verification using such tools as timed automata model checkers to satisfy safety and reliability properties in the context of software product lines.

A second methodology being developed incorporates formal (mathematically based) testing which includes test sufficiency analysis aiming to reduce testing of product lines to a scientifically measured sufficient minimum while maintaining the same level of reliability. These methodologies will be incorporated in the AUTOSAR specification and development processes.

The main outcome of introducing such methodologies especially in a product line context may be to reduce the

amount of effort in testing and reliability analysis that is needed at the final product build stage. If the testing and reliability of the domain/core assets can be guaranteed to a certain level, then when these are integrated to form a product there may be only a need to do the additional testing and reliability analysis rather than start from scratch.

Software architecture knowledge management

Software architecture researchers and practitioners have recently proposed different ways to capture contextual knowledge underpinning design decisions. An essential requirement of all these approaches is to describe architecture in terms of design decisions and the design rationale surrounding them. However, design decisions and their rationale are often not rigorously documented. One of the main reasons for this is the lack of suitable methodological and tool support. We have developed a framework for managing architecture knowledge (technical and contextual) [37]. This framework consists of techniques for capturing design decisions and contextual information, an approach to distil and document architectural knowledge from patterns, and a data model to characterize architectural constructs, their attributes and relationships. In order to support this framework, we have developed a web-based tool called PAKME (Process-centric Architecture Knowledge Management Environment) [30]. PAKME is also designed to act as a knowledge source for those who need rapid access to experience-based design decisions to assist in making new decisions or discovering the rationale for past decisions. Thus, PAKME serves as a repository of an organisation's architecture knowledge analogous to an engineer's handbooks, which consolidate knowledge about best practices in a certain domain. We have been modifying PAKME to support different activities of software architecture process (i.e., design, documentation, and evaluation) in automotive industry. Based on our successful trial of using PAKME to support architecture evaluation process for Avionics systems [47], we believe that PAKME is capable of addressing the knowledge management challenges in software architecture processes in the automotive industry.

Verification and validation of architecture decisions

Since software architecture plays a significant role in the life of a system, it is important to evaluate a system's architecture as early as possible [46]. Architecture evaluation is considered one of the most important and effective techniques of addressing quality related issues at the software architecture level and mitigating architectural risks [33]. Moreover, architecture evaluation sessions are effective means of sharing and capturing architecture design rationale, the reasoning behind architecture design decisions [37]. Hence, before implementing a proposed architecture design, each architecture design decision needs to be sufficiently verified and validated not only for the quality requirements and project constraints, but also evaluated for interdependencies among different design decisions and their potential impact on quality requirements. Moreover, architecture design decisions also need to be optimized given the set of quality requirements and technical and non-technical constraints.

SEAMLESS MODEL-DRIVEN DEVELOPMENT

For seamless model-driven development with a flexible combination of interactive tools and automation we need semantically rich models. As a foundation for this, we require well-defined languages that allow for describing the knowledge used in the particular domain.

Consequently, if we want to apply model-driven approaches in an SPL context we require languages to describe typical SPL aspects (such as, variation points, variants, features, configurations, realization of variability). These modelling languages also have to support the different conceptual levels of domain and application engineering (compare the two levels in Figure 1).

Similarly, if we want to apply model-driven SPL approaches in an automotive engineering context we require languages to describe domain-specific knowledge, for instance the dynamic behaviour of electronic components, the interplay between software and hardware or the communication primitives exchanged on a CAN-Bus network [4].

Hence, we require such languages to describe the structure information processed in model-driven approaches. Besides modelling languages for general software engineering we require domain-specific modelling languages for the SPL/Automotive context. Models described in the modelling languages can be used as a foundation for interactive tools or automation. For instance, we have defined a meta-model for feature models [43]. Such models can be used as foundations for tools which allow one to (a) visualize the complex dependencies between the numerous features and (b) interact with the related feature configuration.

At the same time, such feature models can be used as input for model-driven product derivation. For instance, we developed a model-driven approach with model-transformations described in ATL [48] that derives the architecture for a particular application from the domain architecture for the overall product line. The decision which components are included in the application-specific architecture are based on feature configurations based on the aforementioned meta-model.

AGILE SOFTWARE DEVELOPMENT

In order to allow the integration of agile approaches with plan-driven development we are currently developing a Product Derivation Framework. This framework structures the derivation process and, hence, can be used to describe the integration of agile activities into a systematic and well-controlled process.

As a main outcome of this research we aim to combine the advantages of both the agile and plan-driven worlds – on the one hand gaining the flexibility of agile approaches, while maintaining the predictability of a well-controlled methodology.

RELATED WORK

SPL AND AUTOMOTIVE SOFTWARE ENGINEERING

There are numerous general texts on SPL which provide an introduction and overview of the research area, e.g., Clements et al. [2] and Pohl et al. [3]. An overview of automotive software engineering is provided by Schäuffele et al. [4]. Current challenges for automotive software engineering are presented and discussed in [5].

VISUALLY INFORMED VARIABILITY MANAGEMENT

Several approaches to variability modelling have been proposed (e.g., [3], [16], [49], [50], [51], [52], [53]). These approaches use UML-like notations to model variability and variation points in requirements and design artefacts. Recently, the formalization of variability emerged as an important research area for automation in software product lines. Existing approaches deal with the formalization of domain requirements in feature models (e.g., [54], [55], [56], [57]), analysis of dependencies among features (e.g., [58], [59], [60]), and (prototypical) tool support (e.g., [61], [62]).

Several approaches and tools that support or partly automate product derivation activities in software product line development have been proposed (e.g., [15], [63], [64], [65], [66], [67], [68], [69]). Deelstra et al. [15] present a product derivation framework developed based on two industrial case studies. McGregor [70] introduces the production plan to facilitate product derivation in a product line organization. Krebs et al. [64] propose a derivation methodology based on a configuration model that represents functionality and variability in the product line.

Asikainen et al. [65] provide a product configuration modelling language (PCML) and configuration tool (WeCoTin). PCML supports the creation of feature models for a software product line. WeCoTin is used to derive valid feature models for particular products of the product line. A similar tool for feature-based configuration is discussed in [62].

Gomaa and Shin [66] focus on consistency in derivation and present the PLUSEE tool (Product Line UML Based Software Engineering Environment). PLUSEE can be used for consistency checking among multiple product line models (use case model, feature model, class model, collaboration model, and statechart model, as defined in [50]) and for deriving the product-specific models based on a pre-selection of features. Similar approaches based on variation point propagation are discussed in [68] and [69].

Recently, the need for research in visually informed variability modelling has been described by Nestor et al. [19]. A research tool to support a visually informed and interactive approach to feature configuration is presented in [43].

IMPROVED ARCHITECTURAL DESIGN PRACTICES

The software architecture community has been paying significant attention to support the process of verifying and validating architecture design decisions with respect to desired functional and non-functional requirements. There are numerous methods (such as ATAM [71], QADA [72], and ArchDesigner [29]) available to validate and optimize architecture design decisions and well-known approaches have been described in published books, e.g., Clement, at al., and Bosch [73].

SEAMLESS MODEL-DRIVEN DEVELOPMENT

There are numerous model-driven approaches suggested in the literature. Introductions and overviews are provided for instance by [74] and [75]. One of the most prominent approaches is the Model-Driven Architecture (MDA) suggested by the OMG [76].

Model-driven approaches require meta-modelling mechanisms which allow the definition of new languages for a particular purpose, e.g., OMG MOF [77], KM3 from the ATLAS project [78] or Ecore from the Eclipse Modelling Framework [79].

The processing of models can be described using model transformation languages, such as ATL [80] or the languages suggested by OMG QVT [81]. These technologies can be combined to platforms for model-driven software development, which integrate meta-modelling, different forms of transformations, template mechanisms and allow to configure the sequence of transformation steps. An example for such a platform is openArchitectureWare (oAW) [82].

AGILE SOFTWARE DEVELOPMENT

Authors who focus on the integration of SPL and Agile approaches [83, 84] acknowledge the apparent conflict between plan-driven processes and the flexibility promoted by agile approaches. However, they claim that these different worlds can be integrated.

The risk-based approach presented in [85] claims to help organizations to decide whether they should use a plan-driven approach, agile approaches or a mix of both. To compare and distinguish agile and plan-driven environments the author suggests discriminators such as size, criticality and dynamism (see Table 2). When mixing both types of approaches the authors suggest to first structure the overall application with an architecture so as to be able to encapsulate agile parts.

	Agile Discriminators	Plan-driven Discriminators
Size	Small products and teams; Reliance on tacit knowledge limits scalability	Methods for large products and teams; Hard to tailor down to small projects
Criticality	Untested on safety-critical products; Potential difficulties with simple design and lack of documentation	Methods for highly critical products; Hard to tailor down efficiently to low criticality products
Dynamism	Good for highly dynamic environments, Source for expensive rework for stable environments	Good for highly stable environments, Source for expensive rework for highly dynamic environments
Personnel	Continuous presence of a critical mass of scarce high-level experts required	Need a critical mass of scarce high-level experts during project definition, but can work with fewer later
Culture	Many degrees of freedom, thrive on chaos	Roles defined by clear policies, thrive on order

Table 2 – Discriminating factors for agile and plan-driven software development environments [85]

CONCLUSIONS

Automotive systems have become software intensive and the implementations required to provide the desired features get more and more complex. As a consequence it is necessary to use and extend practices from general software engineering and adapt them for the specific constraints of the automotive domain.

As in many other engineering disciplines, automotive software engineers strive to build high quality systems which fulfil all requirements. However, at the same time they are faced with the need to stay cost effective and build the system within time and resource constraints. Software Product Line approaches seem to be a promising solution to deal with these challenges and the special conditions of software-intensive products in the automotive domain, since they offer the opportunity to build a variety of similar systems with a minimum of technical diversity and thus allows for strategic reuse of components.

In this paper we have outlined some of the challenges facing automotive software engineering and approaches that address these challenges, which we feel can make a significant difference in the development of automotive software systems. Our approaches tackle issues in complexity, architecture and process efficiency which as we continue to develop our approaches and apply them in the automotive systems domain have the potential to make significant improvements on current practices.

ACKNOWLEDGMENTS

This work is partially supported by Science Foundation Ireland under grant no. 03/CE2/I303_1.

REFERENCES

[1] F. v. d. Linden, "Software Product Families in Europe: The ESAPS & CAFE Projects," *IEEE Software,* vol. 19, pp. 41-49, 2002.

[2] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*: Addison-Wesley, 2001.

[3] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering : Foundations, Principles, and Techniques*, 1st ed. New York, NY: Springer, 2005.

[4] J. Schäuffele and T. Zurawka, *Automotive software engineering : principles, processes, methods, and tools*. Warrendale, Pa: SAE International, 2005.

[5] M. Broy, "Challenges in automotive software engineering," in *Proceeding of the 28th international conference on Software engineering (ICSE 2006)* Shanghai, China, 2006, pp. 33-42.

[6] M. D. McIlroy, "Mass Produced Software Components," in *Software Engineering: Concepts and Techniques*, J. M. Buxton, P. Naur, and B. Randell, Eds. New York: Petrocelli/Charter Publishers Inc., 1969, pp. 88-98.

[7] P. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Boston: Addison-Wesley, 2002.

[8] F. van der Linden, "Software Product Families in Europe: The Esaps and Café Projects," *IEEE Software,* vol. 19, pp. 41--49, July/August 2002.

[9] R. Macala, L. Stuckey, and D. Gross, "Managing Domain-Specific, Product-Line Development," *IEEE Software,* pp. 57--67, May 1996.

[10] M. H. Meyer and A. P. Lehnerd, *The Power of Product Platforms*. New York, NY: Free Press, 1997.

[11] D. M. Weiss and C. T. R. Lai, *Software Product-Line Engineering: A Family-Based Software Development Process*: Addison-Wesley, 1999.

[12] A. Hein, T. Fischer, and S. Thiel, "Produktlinienentwicklung für Fahrerassistenzsysteme," in *Software-Produktlinien: Methoden, Einführung und Praxis*, G. Böckle, P. Knauber, K. Pohl, and K. Schmid, Eds.: dpunkt-Verlag, 2004, pp. 193-205.

[13] M. Steger, C. Tischer, W. Stolz, and S. Ferber, "Introducing Product Line Approach at Bosch Gasoline Systems: Experiences and Practices," in *Proceedings of the Third Software Product Line Conference*, 2004, pp. 34--50.

[14] M.-O. Reiser and M. Weber, "Using Product Sets to Define Complex Product Decisions " in *9th International Software Product Line Conference (SPLC 2005)*, Rennes, France, 2005.

[15] S. Deelstra, M. Sinnema, and J. Bosch, "Product derivation in software product families: a case study," *The Journal of Systems and Software,* vol. 74, pp. 173-194, 2005. http://www.msinnema.nl/journalExperiencesInSPF.pdf

[16] S. Thiel and A. Hein, "Modeling and Using Product Line Variability in Automotive Systems," *IEEE Software,* vol. 19, pp. 66--72, July/August 2002.

[17] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices," in *Software Product Line Conference (SPLC-2004)*, Boston, MA, USA, 2004, pp. 34-50.
[18] K.-T. Neumann, H. Kopetz, P. Malaterre, and W. Specks, "Architectural Leadership in the Automotive Industry," in *SAE 2000 World Congress*, Detroit, USA, 2000.
[19] D. Nestor, L. O'Malley, A. Quigley, E. Sikora, and S. Thiel, "Visualisation of Variability in Software Product Line Engineering," in *1st International Workshop on Variability Modelling of Software Intensive Systems (VaMoS-2007)*, Limerick, Ireland, 2007.
[20] IEEE, *IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology*. New York: Institute of Electrical and Electronic Engineers, 1992.
[21] J. A. McCall, "Quality Factors," in *Encyclopedia of Software Engineering*. vol. 2, J. J. Marciniak, Ed. New York, U.S.A.: John Wiley, 1994, pp. 958-971.
[22] ISO/IEC, *Information technology - Software product quality: Quality model*, ISO/IEC FDIS 9126-1:2000(E).
[23] B. Kitchenham and J. Walker, "A quantitative approach to monitoring software development," *Software Engineering Journal,* pp. 1-13, 1989.
[24] D. Budgen, *Software design*, 2nd ed. Harlow, England ; New York: Addison-Wesley, 2003.
[25] P. Crosby, *Quality is Free*. Maidenhead: McGraw Hill, 1978.
[26] D. Galin, *Software Quality Assurance, From theory to implementation*: Pearson education, Ltd., 2004.
[27] U. Kiencke and L. Nielsen, *Automotive Controls Systems: For Engine, Driveline and Vehicle*. Berlin: Springer-Verlag, 2005.
[28] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2003.
[29] T. Al-Naeem, I. Gorton, M. Ali-Babar, F. Rabhi, and B. Benatallah, "A Quality-Driven Systematic Approach for Architecting Distributed Software Applications," in *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, USA, 2005.
[30] M. Ali-Babar and I. Gorton, "A Tool for Managing Software Architecture Knowledge," in *Proceedings of the 2nd Workshop on SHAring and Reusing architectural knowledge - Architecture, rationale, and Design Intent (SHARK/ADI 2007), Collocated with ICSE 2007.*, Minneaspolis.
[31] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America, "A General Model of Software Architecture Design Derived from Five Industrial Approaches," in *the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 05), Pittsburgh, PA, USA*, 2005.
[32] L. Bass, M. Klein, and F. Bachmann, "Quality Attribute Design Primitives and the Attribute Driven Design Method," in *Proceedings of the 4th International Workshop on Product Family Engineering*, Bilbao, Spain, 2001. http://www.cse.unsw.edu.au/~malibaba/research-papers/softwareArchitectures/bilbao_paper.pdf
[33] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*: Addison-Wesley, 2002.
[34] P. B. Kruchten, "The 4+1 View Model of architecture," *Software, IEEE,* vol. 12, pp. 42-50, 1995.
[35] P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed.: Addison-Wesley, 2000.
[36] L. Bass and R. Kazman, "Architecture-Based Development," Software Engineering Institute (SEI), Carnegie Mellon University, Pittsburgh, USA, CMU/SEI-99-TR-007, 1999.
[37] M. Ali-Babar, I. Gorton, and B. Kitchenham, "A Framework for Supporting Architecture Knowledge and Rationale Management," in *Rationale Management in Software Engineering*, A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, Eds.: Springer, 2006, pp. 237-254.
[38] M. Ali-Babar, I. Gorton, and R. Jeffery, "Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development," in *Proceedings of the 5th International Conference on Quality Software*, Melbourne, Australia, 2005.
[39] K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," 2001. http://agilemanifesto.org/
[40] D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling Agile Methods," *IEEE Software,* vol. July/August 2003, pp. 12-14, 2003.
[41] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse. Architecture, Process and Organization for Business Success*: Addison-Wesley, 1997.
[42] S. K. Card, J. D. MacKinlay, and B. Shneiderman, *Readings in Information Visualization - Using Vision to Think*, 1st ed. San Francisco: Morgan Kaufmann Publishers, 1999.
[43] G. Botterweck, D. Nestor, A. Preussner, C. Cawley, and S. Thiel, "Towards Supporting Feature Configuration by Interactive Visualisation," in *Proceedings of 1st International Workshop on Visualisation in Software Product Line Engineering (ViSPLE 2007), collocated with the 11th International Software Product Line Conference (SPLC 2007)* Kyoto, Japan, 2007.
[44] N. Lassing, P. Bengtsson, J. Bosch, and H. V. Vliet, "Experience with ALMA: Architecture-Level Modifiability Analysis," *Journal of Systems and Software,* vol. 61, pp. 47-57, 2002.
[45] R. Kazman, S. J. Carriere, and S. G. Woods, "Toward a Discipline of Scenario-based Architectural Engineering," *Annals of Software Engineering, Kluwer Academic Publishers,* vol. 9, pp. 5-33, 2000. http://www.cse.unsw.edu.au/~malibaba/research-papers/softwareArchitectures/annals-scenario.pdf
[46] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2 ed.: Addison-Wesley, 2003.
[47] M. Ali-Babar, A. Northway, I. Gorton, P. Heuer, and T. Nguyen., "Introducing Tool Support for Knowl-

edge Management in Software Architecture Evaluation Process," National ICT, Australia 2007.
[48] J. Bézivin, E. Breton, G. Dupé, and P. Valduriez, "The ATL Transformation-based Model Management Framework," 2003. http://www.sciences.univ-nantes.fr/info/recherche/Vie/RR/RR-IRIN2003-08.pdf
[49] J. Coplien, D. Hoffmann, and D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software,* pp. 37--45, November/December 1998.
[50] H. Gomaa and M. E. Shin, "Multiple-View Meta-Modeling of Software Product Lines," in *8th International Conference on Engineering of Complex Computer Systems (ICECCS 2002)*, 2002, pp. 238-246.
[51] T. Asikainen, T. Soininen, and T. Männistö, "A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families," in *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5)*, 2003, pp. 225-249.
[52] R. van Ommering, "Software Reuse in Product Populations," *IEEE Transactions on Software Engineering,* vol. 31, pp. 537-550, July 2005.
[53] M. Moon, K. Yeom, and H. S. Chae, "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in a Product Line," *IEEE Transactions on Software Engineering,* vol. 31, pp. 551-569, July 2005.
[54] A. G. J. Jansen, R. Smedinga, J. van Gurp, and J. Bosch, "First Class Feature Abstractions for Product Derivation," *IEEE Proceedings Software,* vol. 151, pp. 187-197, August 2004.
[55] T. von der Maßen and H. Lichter, "Deficiencies in Feature Models," in *Workshop on Software Variability Management for Product Derivation*, Boston, MA, 2004.
[56] A. Metzger, S. Bühne, K. Lauenroth, and K. Pohl, "Considering Feature Interactions in Product Lines: Towards the Automatic Derivation of Dependencies between Product Variants," in *Feature Interactions in Telecommunications and Software Systems (FIW 2005)*, Leicester, UK, 2005, pp. 198-216.
[57] D. Batory, "Feature Models, Grammars, and Propositional Formulas," in *9th International Conference on Software Product Lines (SPLC 2005)*, Rennes, France, 2005, pp. 7-20.
[58] L. Hotz, T. Krebs, and K. Wolter, "Dependency Analysis and its Use for Evolution Tasks " in *27th German Conference on Artificial Intelligence, Workshop on Planning, Scheduling, and Configuration (PuK 2004)*, Ulm, Germany, 2004.
[59] M. Ryan and P. Y. Schobbens, "FireWorks: A Formal Transformation-Based Model-Driven Approach to Features in Product Lines," in *3rd Software Product Line Conference (SPLC 2004), Workshop on Software Variability Management for Product Derivation*, Boston, MA, 2004.
[60] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Modeling Dependencies in Product Families with COVAMOF," in *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006)*, Potsdam, Germany, 2006.
[61] M. Sinnema, O. de Graaf, and J. Bosch, "Tool Support for COVAMOF," in *Workshop on Software Variability Management for Product Derivation*, Boston, MA, 2004.
[62] D. Beuche, "Variants and Variability Management with pure::variants," in *3rd Software Product Line Conference (SPLC 2004), Workshop on Software Variability Management for Product Derivation*, Boston, MA, 2004.
[63] G. Chastek and J. D. McGregor, "Guidelines for Developing a Product Line Production Plan," Software Engineering Institute, Carnegie Mellon University CMU/SEI-2002-TR-006, June 2002.
[64] T. Krebs, K. Wolter, and L. Hotz, "Model-based Configuration Support for Product Derivation in Software Product Families," in *28th German Conference on Artificial Intelligence, Workshop on Planning, Scheduling, and Configuration (PuK 2005)*, Koblenz, Germany, 2005.
[65] T. Asikainen, T. Männistö, and T. Soininen, "Using a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models," in *Workshop on Software Variability Management for Product Derivation, Workshop at SPLC2004*, Boston, MA, 2004.
[66] H. Gomaa and M. E. Shin, "Tool Support for Software Variability Management and Product Derivation in Software Product Lines," in *Workshop on Software Variability Management for Product Derivation*, Boston, MA, 2004.
[67] J.-P. Tolvanen and S. Kelly, "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences," in *9th International Conference on Software Product Lines (SPLC 2005)*, Rennes, France, 2005, pp. 198-209.
[68] O. Díaz, S. Trujillo, and F. I. Anfurrutia, "Supporting Production Strategies as Refinements of the Production Process," in *9th International Conference on Software Product Lines (SPLC 2005)*, Rennes, France, 2005, pp. 210-221.
[69] P. Tessier, S. Gérard, F. Terrier, and J.-M. Geib, "Using Variation Propagation for Model-Driven Management of a System Family," in *9th International Conference on Software Product Lines (SPLC 2005)*, Rennes, France, 2005, pp. 222-233.
[70] J. D. McGregor, "Guidelines for Developing a Product Line Production Plan, Technical Report CMU/SEI-2002-TR-006," Software Engineering Institute June 2002 2002.
[71] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method," in *Proceedings of the International Conference on Engineering of Complex Computer Systems*, 1998.
[72] M. Matinlassi, E. Niemela, and L. Dobrica, "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," VTT Technical Research Centre of Finland, Espoo 456, 2002.

[73] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product-line approach*: Addison-Wesley, 2000.
[74] D. C. Schmidt, "Model-Driven Engineering," *IEEE Computer,* vol. 39, 2006. http://www.cs.wustl.edu/~schmidt/PDF/GEI.pdf
[75] T. Stahl and M. Völter, *Model-driven software development : technology, engineering, management*. Chichester, England ; Hoboken, NJ: John Wiley, 2006.
[76] OMG, "OMG Model Driven Architecture," 2007. http://www.omg.org/mda/
[77] OMG, "MOF 2.0 / XMI Mapping Specification v2.1," Object Management Group, 2006. http://www.omg.org/cgi-bin/doc?formal/2005-09-01
[78] ATLAS Group, "KM3: Kernel MetaMetaModel Manual (Version 0.6)," 2005. http://www.eclipse.org/gmt/atl/doc/KernelMetaMetaModel%5Bv00.06%5D.pdf
[79] Eclipse Foundation, "EMF - Eclipse Modelling Framework." http://www.eclipse.org/modeling/emf/
[80] "ATL Home page." http://www.eclipse.org/m2m/atl/
[81] OMG, "Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification," Object Management Group, 2005. http://www.omg.org/cgi-bin/apps/doc?ptc/05-11-01.pdf
[82] "openArchitectureWare." http://www.openarchitectureware.org/
[83] G. K. Hanssen, "Agile software product line engineering, DT8100 Essay," 2007. http://www.idi.ntnu.no/emner/dt8100/Essay2007/hanssen-essay07.pdf
[84] R. Kurmann, "Agile SPL–SCM -- Agile Software Product Line Configuration and Release Management," in *APLE 2006 Workshop colocated with SPLC2006* Baltimore, MD, USA.
[85] B. Boehm and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *IEEE Software,* vol. June 2003, 2003.

CONTACT

Goetz Botterweck

Lero – The Irish Software Engineering Research Centre, University of Limerick, Limerick, Ireland

Email: goetz.botterweck@lero.ie

Telephone: +353 61 234309