Titre: Title:	Use of Model-Based Software Product Line Engineering for Certifiable Avionics Software Development
Auteur: Author:	Neset Sozen
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Sozen, N. (2016). Use of Model-Based Software Product Line Engineering for Certifiable Avionics Software Development [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2492/

UNIVERSITÉ DE MONTRÉAL

USE OF MODEL-BASED SOFTWARE PRODUCT LINE ENGINEERING FOR CERTIFIABLE AVIONICS SOFTWARE DEVELOPMENT

NESET SOZEN
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

USE OF MODEL-BASED SOFTWARE PRODUCT LINE ENGINEERING FOR CERTIFIABLE AVIONICS SOFTWARE DEVELOPMENT

présentée par: SOZEN Neset
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M. KHOMH Foutse, Ph. D., président
M. MERLO Ettore, Ph. D., membre et directeur de recherche
M. DAGENAIS Michel, Ph. D., membre
Mme DSSOULI Rachida, Ph. D., membre externe

DEDICATION

I would like to dedicate this work to my wife and my children in recognition of all their sacrifices over the last few years.

. . .

ACKNOWLEDGEMENTS

I would like to thank my family, my director and specially Esterline CMC Electronics and my colleagues for their moral and technical support in this project.

RÉSUMÉ

Tous les systèmes logiciels avioniques sont soumis aux contraintes de certification imposées par les normes DO-178. Les fabricants d'équipements avioniques civils sont très conservateurs dans leur processus de développement de logiciels et la plupart utilisent encore des outils et des méthodes d'ingénierie logicielle éprouvés en raison des contraintes de certification strictes. Les contraintes de certification, avec la taille et la complexité du logiciel des systèmes avioniques modernes qui augmentent continuellement, ont un impact considérable sur le coût du développement de logiciel avionique certifiable. Pour réduire le coût de développement, les fabricants d'équipements avioniques doivent utiliser des méthodes de développement logiciel modernes, ce qui est possible avec la publication de la norme DO-178C.

Dans le cadre de ma thèse, nous explorons l'utilisation de l'ingénierie de ligne de produit basée sur des modèles pour le développement de logiciels avioniques certifiables et proposons des solutions au niveau industriel pour utiliser un processus de ligne de produit utilisant des outils commerciaux.

Dans le cadre de ma thèse, nous explorons également l'applicabilité de notre processus de development logiciel basé sur le concept de ligne de produit au développement de logiciels avioniques certifiables contrôlés. Nous identifions les contraintes qui limitent la réutilisation des composants logiciels dans les logiciels avioniques sous contrôle d'exportation et proposons des solutions techniques qui facilitent l'application de ligne de produit logiciel basée sur des modèles au développement de logiciels avioniques certifiés et sous contrôle d'exportation. Nous validons nos solutions proposées par des études de cas industriels.

ABSTRACT

All avionics software systems are subjected to certification constraints imposed by DO-178 standards. Civil avionics equipment manufacturers are quite conservative in their software development processes: most still use time-tested software engineering tools and methods, due to strict certification constraints. These certification constraints, along with the increasing size and complexity of modern avionics software-intensive systems, are having a huge impact on the cost of certifiable avionics software development. To cope with this increasing complexity, avionics equipment manufacturers need to use modern software development methodologies. This is possible with the release of DO-178C standard.

In my thesis, I have explored the use of model-based software product line engineering for certifiable avionics software development, and have proposed industrial-level solutions for using a model-based software product line process based on commercially available tools.

In this thesis, I have also explored the applicability of our model-based software product line process to export-controlled, certifiable avionics software development, identifying constraints that limit the reuse of software components among export-controlled avionics software and proposing technical solutions that facilitate the application of a model-based software product line to export-controlled, certifiable avionics software development. The proposed solutions are validated using industrial case studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

FAA	Federal Aviation Administration
NextGen	Next Generation Air Transportation System
SESAR	Single European Sky ATM Research
ATM	Air Traffic Management
FMS	Flight Management System
RTCA	Radio Technical Commission for Aeronautics
MDA	Model-Driven Architecture
PIM	Platform Independent Model
PSM	Platform Specific Model
RSC	Reusable Software Components
SPL	Software Product Line
SPLE	Software Product Line Engineering
MDD	Model Driven Development
UML	Unified Modelling Language
FOSD	Feature-Oriented Software Design
FODA	Feature-Oriented Domain Analysis
FORM	Feature-Oriented Reuse Method
DER	Designated Engineering Representative
OCL	Object Constraint Language

CHAPTER 1 INTRODUCTION

The Next Generation Air Transportation System (NextGen), a Federal Aviation Administration (FAA) program, and the Single European Sky ATM Research (SESAR) program, its European equivalent, are ongoing international collaborative projects, created in order to reduce air traffic congestion and improve air safety and airspace system efficiency [2]. The need for a more efficient airspace system will have repercussions on the size and complexity of avionics software systems such as Flight Management System (FMS). On the other hand, aerospace companies who manufacture civil avionics equipment are quite conservative in their software development processes. Most still use time-tested software engineering tools and methods. The source of this cautious and restrained approach to software development is strict certification constraints (e .g. *DO-178B, Software Considerations in Airborne Systems and Equipment Certification Standard*). This said, the recent publication of DO-178C facilitates and enforces the use of model-based methods, object-oriented programming, and code generation tools.

1.1 Research Area

My thesis was carried out within the framework of Research Project CRIAQ 5.5, in collaboration with École Polytechnique de Montréal, Esterline CMC Electronics, and other industrial and academic partners. Research Project CRIAQ 5.5 had as mission to explore various new software development technologies, such as model-based software development, formal methods, and model-based testing, with the sole objective of reducing the cost of avionics software development and certification. My thesis focuses on the use of model-based software development technologies to reduce the cost of complex avionics software development. More specifically, I have investigated the variability and separation of features in the CMA-9000 FMS legacy certified software, and proposed more cost effective ways to handle variability and separation of features in complex certifiable avionics software.

Esterline CMC Electronics manufactures various FMS products that have different variations, along with numerous complex optional features and navigational capabilities [3]. FMS equipment can be installed on both civil and military aircraft. In order to prevent infractions to governmental export regulations, currently there is no software reuse between civil certified FMS and export-controlled military FMS equipment. Within the context of the *Software FMS* research project, at Esterline CMC Electronics, Software Product Line Engineering (SPLE) is being explored as the next generation in FMS's software development

process. In my thesis, I have investigated the feasibility of software reuse between civil and military FMS products.

1.1.1 Motivations and Scope of Work

My thesis has been driven mainly by challenges posed by certifiable and export-controlled software development using model-based methods and off-the-shelf commercial Software Product Line (SPL) variability management tools.

DO-178 certification is mandatory for deploying avionics equipment on civil aircraft. Aircraft software certification consists in meeting a set of objectives that are satisfactory at various levels, from requirements to code and test development [4]. The most notable new challenges in the certification of aircraft software are the use of model-based design and object-oriented programming [5]. Model-based methods tend to blur the distinction between requirements and software design [5].

Improving and providing methods for the reuse of software components among various FMS equipment was another impetus for my thesis. Within the context of reusability, the separation of export-controlled software components from certifiable civil aircraft software components is also an important aspect in the avionics software development that must be addressed. Export regulations require that access to export-controlled software components is limited to authorized personnel.

Software Product Line Engineering is an engineering discipline emphasizing the reuse of software artifacts and variability management. Software development using model-based Software Product Line Engineering is a threefold method, consisting of: (1) domain engineering with feature models; (2) architectural modeling with UML; and (3) code generation with C/C++ (or Ada). During the last decade, Software Product Line Engineering has progressed considerably, and several commercial SPLE tools have matured enough to be used for software development in other safety-critical domains [6]. However, several challenges remain if we wish to use these SPLE tools for software development in export-controlled and certifiable avionics systems.

The first challenge is that Software Product Line Engineering requires a practical toolchain leading from a feature model to an auto-generation of the source code of the variant avionics equipment. The product derivation process must be automated, which is already the case for most commercial variability management tools, such as *pure::variants*[1] or *Gears*[2]. However, these tools perform automated product derivation at the implementation level only; the

[1] www.pure-systems.com

[2] www.biglever.com/solution/product.html

architecture modeling and design phases are not addressed [7].

Secondly, those commercial tools advocate negative variability, where each variant product software's UML design and architecture are derived by stripping design elements that are not part of the variant product from a reference UML model containing implementation of all features. With this approach, every one has access to design and implementation of all features. This may raise issues within the context of export-controlled software development, because access to export-controlled software components must be limited to authorized persons.

Another limitation was discovered during UML design modeling. The variability management tools were able to add a variation point to any UML element (e.g. classes, operations, attributes). For example, when there is a variation point on an operation, the operation will have multiple versions, and during the product derivation process, only one version will be selected for the variant product. This level of granularity, in UML models, leads to a great deal of code duplication, because during our experiment, most of the variations were inside the operations: in some cases they consisted of no more than a few lines of code.

All these limitations with commercial tools will be addressed in the second paper *"Software Design Pattern Providing Reusability Among Certifiable, Export-Controlled Avionics Software"*, presented in Chapter 5.

1.2 Statement of the Problem

Certifiable and export-controlled software-intensive avionics system development is very expensive. The certification process increases the cost of avionics software development by 75% to 150% [8]. Export regulations further increase the cost of software development by imposing additional constraints on software development, such as limiting access to software on export-controlled avionics equipment, and limiting the reuse of avionics features implementation among military (export-controlled) and civil (non-controlled) variants of avionics equipment.

Avionics system manufacturers use qualified software development tools; this means that these tools are subjected to similar certification processes. In order to cope with the cost of certifiable software development, manufacturers tend to be conservative in their software development process and they will use time-tested software development tools and methods.

Modern civil avionics systems are central to achieving a safer, more efficient airspace, as required by ongoing international projects such as NextGen [9]. An efficient airspace system requires extensive and complex avionics features. Over the last few decades, the size and complexity of modern avionics system software has increased exponentially, and traditional

avionics software development methods are nearing the limits of their capacity to cope with the increasing cost of software development [10].

In my thesis, I have tried to address the high cost of certifiable and export-controlled avionics software development by proposing a model-based software product-line engineering method that is adapted to the constraints emerging from both the DO-178C certification standard and export regulations. The main advantage of the solution we present is its reuse of software components among export-controlled and non-controlled variants of avionics equipment. In order to prevent any infringement of export regulations, most avionics equipment manufacturers separate the implementation of the export-controlled variants of their avionics equipment from the non-controlled variants, with no reuse possible, even though the controlled variants contain implementation that is up to 70% similar to that of the non-controlled variants. Providing reusability among the controlled and non-controlled variants of avionics equipment may result in considerable cost reduction. The second advantage of my solution is the use of mainstream commercial modeling and SPLE variability management tools, which will greatly reduce the effort required to develop a complete toolchain.

1.3 Research Objectives

The main goal of my thesis has been to answer the following question: *"How can the development cost of certified avionics software be reduced, using model-based software development and software product line engineering?"*

The subsections below describe the general and more specific research objectives that may reduce the cost of certifiable avionics software development using off-the-shelf, model-based software development and SPL variability management tools.

1.3.1 General Objectives

The high cost of certified avionics software is driven by multiple factors such as certification constraints and export regulations that have a negative impact on the reuse of software artifacts among avionics software projects, and the inadequacy of traditional software development methods to handle the exponentially increasing size and complexity of the software used in modern avionics systems. Another requirement driving this research is the use of off-the-shelf model-based software development and SPL variability management tools.

My general objectives for this thesis have been to provide a software development methodology based upon model-based software product line engineering, in order to produce and maintain high-integrity avionics software that meets certification criteria using off-the-shelf

tools, and to provide the means to allow reusability among export-controlled and certifiable avionics software.

1.3.2 Specific Objectives

My thesis has three specific objectives:

Objective 1 Proposal of SPL-based software development process and design models that suit avionics software and that agree with certification criteria.

Objective 2 Proposal of SPL-based software development process and design models that provide reuse of software components among export-controlled avionics software systems and that are in agreement with export regulations.

Objective 3 Validation of the proposed process and models by implementing prototype avionics software within an industrial project, using off-the-shelf tools.

1.4 Thesis Contributions

My thesis contributions are listed below:

- In addition to the papers described in Chapter 4 and Chapter 5, I have published another paper [11] in *Workshop PLEASE, 2012*. In this paper, we introduced the use of software product line engineering for complex certifiable avionics software development, and described certification challenges for SPLE adoption.

- Multiple tools for software analysis were developed during my thesis. The first tool that I developed was used to generate UML diagrams from the CMA-9000 FMS source code (in C). Since there is no concept of "object" in C language, the packages and classes in the UML model were generated using the naming convention standard section of the CMA-9000 FMS coding standards. The naming convention was based on CMA-9000 FMS software architecture. This tool was used internally at Esterline CMC Electronics in order to understand the deviation of the current implementation from the intended architecture. The second tool, described in [12] and [13], was used to automatically locate a feature implementation in the source code of CMA-9000 FMS. In CMA-9000 FMS, features are dynamically activated using configuration variables. We used this tool to understand features interactions and features mapping to source code. This

tool was developed in collaboration; my main task was to implement the code parser and the feature data from the configuration variables.

- Another contribution is the case study, which analyzes variability management in CMA-9000 FMS certified software quantitatively and qualitatively and compares it to our new, model-based variability management using software product lines. The case study was conducted from three perspectives: the certification process, software evolution, and software maintenance.

- The proposal for a model-based software product line engineering process for certifiable avionics software development is another contribution of my thesis. My proposal handles variability in complex avionics software and reduces the cost of complex avionics software development by improving the reuse of certified software components.

- For the first time, the applicability of a software product line to the software development of certifiable and export-controlled avionics systems is analyzed, and the constraints preventing the use of commercial SPL management tools for the software development of certifiable and export-controlled avionics systems are identified.

- In my thesis, I have proposed a new design pattern that can enable the use of SPLE for certifiable and export-controlled avionics software development, and that can facilitate software reuse among controlled and non-controlled variants of certifiable and export-controlled avionics software systems.

1.5 Thesis Organization

The remainders of this text is structured as follows: The basic concepts of my thesis are described in the following section. In Chapter 2, my research is compared to the current literature, and in Chapter 3, research methodology is described. Chapter 4 presents the first paper, entitled *"Implementing Software Product Line Engineering for Certifiable Avionics Software,"* and Chapter 5 presents the second paper, entitled *"Software Design Pattern providing Reusability among Certifiable, Export-Controlled Avionics Software."* These two papers constitute the foundation for my thesis. The general discussion is described in Chapter 6, and the thesis ends with the Conclusion, found in Chapter 7.

1.6 Background

This section describes the concepts and definitions that underlie my research. There are three axes to my thesis: avionics software certification, model-based software development

and software product line engineering. This section also describes the challenges involved in avionics software certification, model-based software development and software product line engineering.

1.6.1 Avionics Software Certification

In order to use software on commercial aircraft, certification following DO-178C guidance is required [4]. This guidance consists of a set of objectives that must be met for the software life cycle process, and activities that will help achieve these objectives. In sum, certification is about providing evidence that the full software life cycle process has been followed.

DO-178C focuses on safety solely from a software process perspective. The level of effort exerted in order to comply with the objectives of DO-178C will depend on the safety level of the software. The system safety assessment process establishes the software level of a software component, based on the potential system failure conditions that can be provoked by an error in this software component. Table 1.1 lists five software levels established by DO-178C, along with their corresponding failure conditions and the number of objectives required in order to satisfy each level.

Some of those objectives must be met with *independence*. To achieve an objective with independence, verification activities are performed by a person (or persons) other than the author(s) of the item that is being verified.

The DO-178C is based on a defined software life cycle, which is divided into a set of processes, as follows:

- Planning Process
- Development Process
- Requirements Process
- Design Process
- Coding and Integration Process
- Testing and Verification Process
- Configuration Management Process
- Quality Assurance Process

Table 1.1 DO-178C, Software Safety Levels

Failure Condition	**Software Level**	**No. of Objectives**	**Objectives with Independence**
Catastrophic	Level A	71	33
Hazardous	Level B	69	21
Major	Level C	62	8
Minor	Level D	26	5
None	Level E	0	0

There is a set of objectives for each process listed above. Developers can use any software development model, provided that they comply with the DO-178C objectives. Compliance is achieved by producing a list of software life cycle data (e.g. software development and verification plans, software requirements, design description, source code, test procedures/test results). The precise list of DO-178C deliverables (i.e. software life cycle data) can be found in [4]. The DO-178C deliverables are reviewed by the certification authorities and/or their representatives. For example, a Designated Engineering Representative (DER) or an inspector from regulatory authorities, such as Federal Aviation Administration, should be able to trace the pathway from a system requirement down to a line of code or a test case, as well as, in the opposite direction, from a line of code or a test case up to a system requirement.

Reusable Software Components (AC20-148)

AC 20-148 is a non-traditional standard that provides guidance for DO-178C compliance of Reusable Software Components (RSC) [14]. A DO-178C RSC is a software collection that is recognized as meeting the objectives of DO-178C. A DO-178C RSC may be used on more than one project without having to regenerate certification artifacts. Certification authorities grant RSC acceptance as part of the normal certification process, when an applicant complies also with AC 20-148. An FAA acceptance of RSC allows software deployment on future projects without the added cost and risk of recertification.

To receive RSC certification, compliance should be done twice, at the component level and at the system level (i.e. target platform). If future users of the RSC change the target platform, the certification will be invalidated. In this case, future users will have to apply for new RSC compliance. Therefore, the DO-178C RSC initial cost is higher due to this certification at two levels. To be cost effective, the target platform should be reuseable on future systems where the RSC is intended for use.

Challenges involved in Avionics Software Certification

Until now, no aircraft crash caused by software failure has been reported. Modern avionics software systems are exceptionally safe, because software implementation defects are eliminated by the certification process, but this high level of safety comes at great cost. The cost is the main challenge to be overcome. There are several aspects of software certification, which contribute to the high cost of certifiable avionics software development:

- Avionics software certification compliance requires that the avionics system and its software are certified together as a complete aircraft system. The certification compliance must be reassessed for a new or modified aircraft type.
- In DO-178B/C, there are typically two categories of software development tools: *development tools* that may introduce errors such as a code generator or a compiler and *verification tools* that may fail to detect errors. If a tool is qualified, there is no need for a verification to ensure that no error was introduced or missed by the tool. Tool qualification reduces certifiable software development activities; however, qualification of a tool is very expensive.

Another challenge of modern avionics software certification is what is known as *"high integration"*. In a traditional aircraft, avionics functions are provided by separate sets of avionics equipment (e.g. autopilot, flight management system), which communicate with each other to form a *"federate"* system. Modern aircraft are built using Integrated Modular Avionics (IMA), where avionics functions are highly integrated and share the same computer system, communicating with each other through networks. This "high integration" may have several types of impact on the certification process: an error in one function may spread to another function on the same computer; the overall complexity of the software will be increased, and obtaining compliance for overall avionics systems will be more difficult.

One more challenge consists of software errors caused by faulty requirements, and defects introduced into requirements during transitions between system requirements developed under ARP 4754A and software requirements under DO-178C [5].

Over the past twenty years, various software development methods have matured and are now part of the mainstream software engineering (e.g. model-based design, object-oriented programming, formal methods). The main difficulty in using alternative means of compliance has been to obtain *"certification credits."* DO-178C provides guidance for obtaining compliance with alternative methods of software development. At the outset of my research, the DO-178C was not yet published. All avionics manufacturers, including Esterline CMC

Electronics, were still using DO-178B to obtain certification compliance. In my thesis, I explore the use of model-based methods and software product line engineering to tackle the challenges listed above.

1.6.2 Model-Based Software Development

Model-Driven Architecture (MDA) [15] bases the software development process on formal models and automated transformation of those models from the high abstract level to the system source code level. Therefore, with MDA, the focus shifts from code to product independent models. With the MDA approach, the models are the basis for software development. There are two types of models (or levels of abstraction) defined by MDA: the Platform Independent Model (PIM) and the Platform Specific Model (PSM).

Figure 1.1 Model-based Software Development using MDA

The PIM describes the system with the highest level of abstraction, and it is independent of any implementation technology. The PSM is the result of PIM transformation. The PSM contains information specific to the implementation technology used. Therefore, for each specific platform, a different PSM is generated from PIM. Lastly, each PSM is transformed to a lower-level abstraction artifact (e.g. source code, test, document).

All models, both PSM and PIM, should be consistent and precise, and should contain as much information as possible about the system. For example, UML diagrams can be augmented with rules and constraints using Object Constraint Language (OCL), because UML diagrams alone do not typically provide enough information [16]. OCL is a formal specification language extension used for supplementing UML with formal expressions. The most important elements of the MDA are the high-level models (PIMs) and low-level models (PSMs), which describe the system with and without the knowledge of the final implementation platform, respectively. Both PIMs and PSMs are written using a formal language (or meta-language); otherwise, the automated transformation of those models would not be possible. The transformation definition describes how a source model can be transformed into a target model. A more detailed MDA framework is illustrated in Figure 1.2.

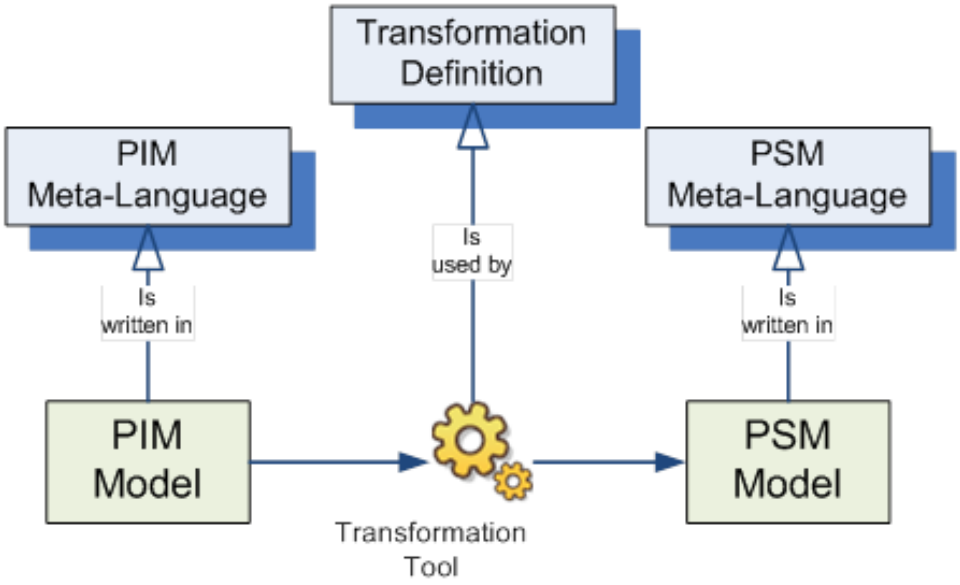

Figure 1.2 MDA Framework

1.6.3 Software Product Line Engineering

The fundamental difference between single system development and software product line engineering is the shift of focus from a single project with software reuse to multiple projects with development for software reuse. With SPL, it is all about modeling the commonalities and differences between system variants that are under development for various projects [17]. Therefore, SPL has two levels of engineering: domain engineering and application engineering. Domain engineering is responsible for establishing the reusable platform and for defining the commonalities and variabilities of the product line. Application engineering is responsible for deriving product line applications (i.e. the specific products) from the platform established during domain engineering. SPL engineering has two other logical separations: the *"problem space"* and the *"solution space."* The *"problem space"* describes the commonalities and variabilities while the *"solution space"* describes the constituent assets of the Product Line. The Software Product Line Engineering is illustrated in Figure 1.3.

SPLE and Model-Driven Development (MDD) are two software development approaches which, if used together, could be much more efficient. For example, in the case of a true product line, in which functionality is largely the same from client to client, the richness of the transformation tools improves over time to the point that setting top-level parameters (instead of low-level framework modifications) achieves greater customization. Therefore, model-driven software development can reduce the cost of application development through domain-specific models or generative techniques. We can see in [18], [19] and [20], the SPLE

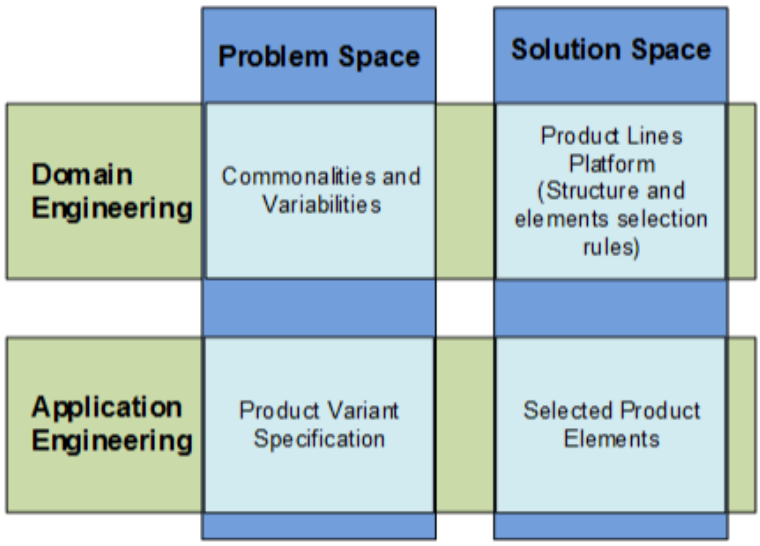

Figure 1.3 Software Product Line Engineering [1]

together with MDD. The combination of SPLE and MDD approaches is promising. There are various ways in which these two approaches can be combined. One is to use feature modeling to define variants of models created using domain-specific languages.

Definition of Variability and Commonality

Considering a product line based system where single software intensive products are derived from a platform, a *commonality* is a mandatory feature present in all of its derived products (also called variants). From a certifiable avionics software perspective, this will be all the software artifacts (e.g. requirements, source code, test procedure) that are required to in order to realize the mandatory feature. On the other hand, a *variability* is an optional feature that is only present in some of the variants.

A feature model is an abstract concept for describing a system's commonalities and variabilities. A feature is a *"prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system"* [21]. Feature models have a tree structure, with features forming the nodes of the tree. Feature variability is represented by arcs and groupings of features. There are four different types of feature groups: "mandatory," "optional," "alternative," and "or." A feature model, implemented using pure-variants (a variant management tool), is illustrated in Figure 1.4.

In Figure 1.4, the "mandatory" features are represented by "!"; "optional" features are repre-

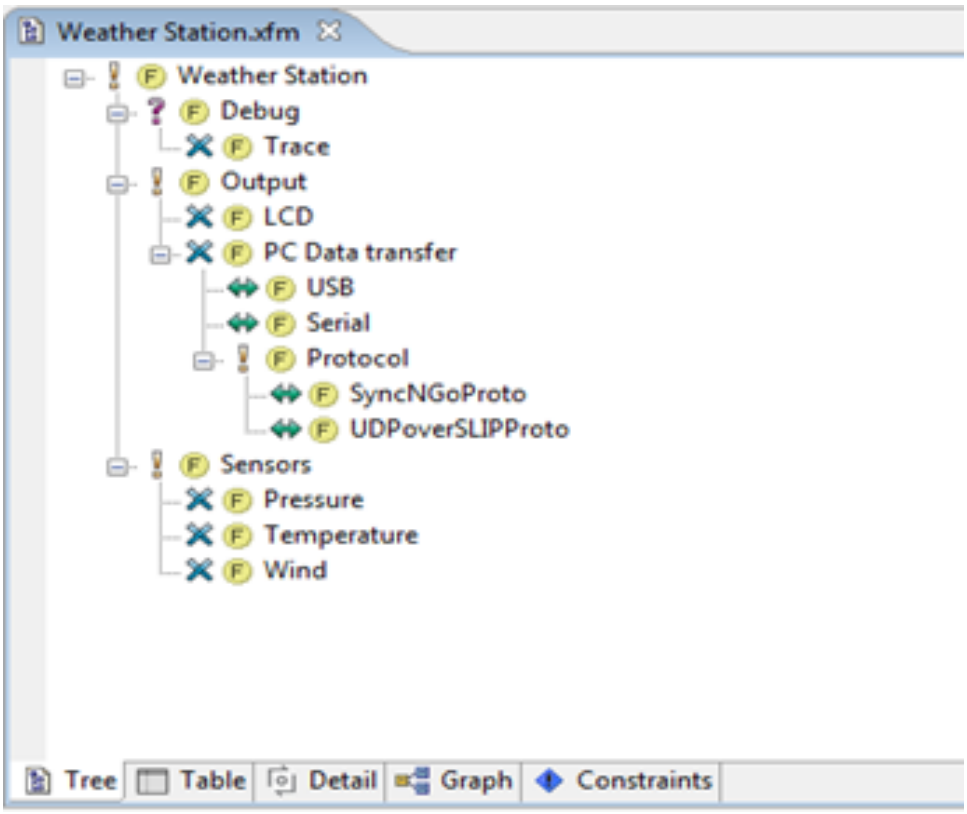

Figure 1.4 Feature Model using pure-variants

sented by "?"; "alternative" features are represented by "X" and "or" features are represented by "<->". There are examples of a few feature-oriented developments in [22] and [23]. Based on the SPL definition above, SPL implementation with models is illustrated in Figure 1.5.

The Variant Description Model (VDM) describes the selection of features from the *Feature Model* and the selection of software assets from the *Family Model*.

1.6.4 Challenges involved in Applying Model Based Software Product Line Engineering to Certifiable Avionics Software Development

Considering the CMC Electronics FMS products, with different software versions, that are currently used in various civil and military aircraft, FMS products are good candidates for exploring the use of MDD combined with Software Product Line Engineering as an alternative software development method to reduce the cost of certifiable avionics software development by improving the reusability of all software artifacts.

The main challenge in implementing SPLE is that an organizational adaptation is required: the focus is shifted from a single project to multiple projects, where domain engineering needs

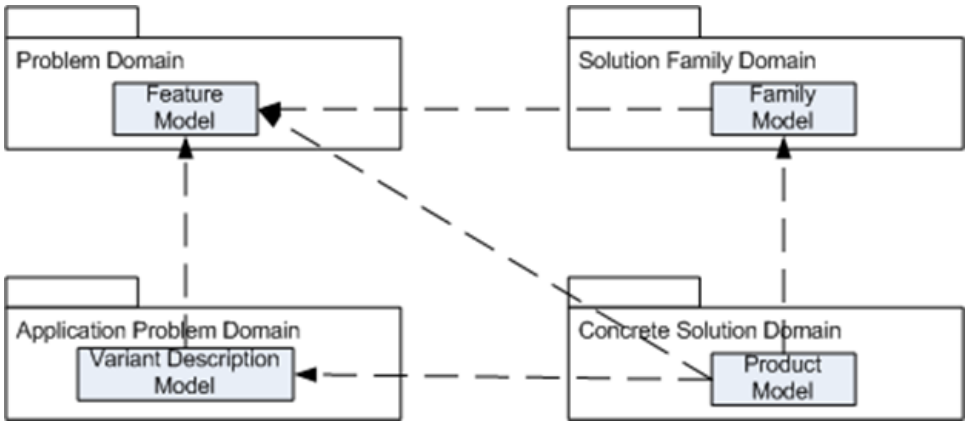

Figure 1.5 Software Product Line Model [1]

to be collectively shared among those projects. The efficiency of SPLE will be greatly reduced or may even have a negative impact on software development cost if a matrix organization is not put in place, because SPLE adds an overhead cost to the engineering process; in order to have a return on investment, there should be at least 3 products in the product line [17].

Software development using model-based Software Product Line Engineering is threefold method: (1) domain engineering with feature models; (2) architectural modeling with UML; and (3) code generation.

Another challenge is the fact that SPLE is mostly used as a tool for domain analysis or for code generation. The architectural modeling step is left mostly unaddresssed. SPLE requires a practical toolchain that covers the full software life cycle, where each step is clearly defined and handled by the toolchain.

Moreover, the product derivation process must be automated. Commercial variability management tools provide automated product derivation mechanisms, but within the source code only [7]. There are no variability management tools with well-defined stages from feature models to design models and design models to code, with an automated derivation mechanism at each stage.

Certified avionics equipment manufacturers tend to use qualified software development tools, which means that these tools are subjected to a certification process similar to the one used for software installed on the aircraft. Otherwise, the tool's output must be verified. For model-based Software Product Line Engineering with automated derivation mechanisms, the software artifacts generated must be reviewed if the tool is not qualified. The SPLE qualification may not be trivial, either considering the complexity of the derivation mechanism; it may even be impossible if the SPLE method is based on commercial tools where access

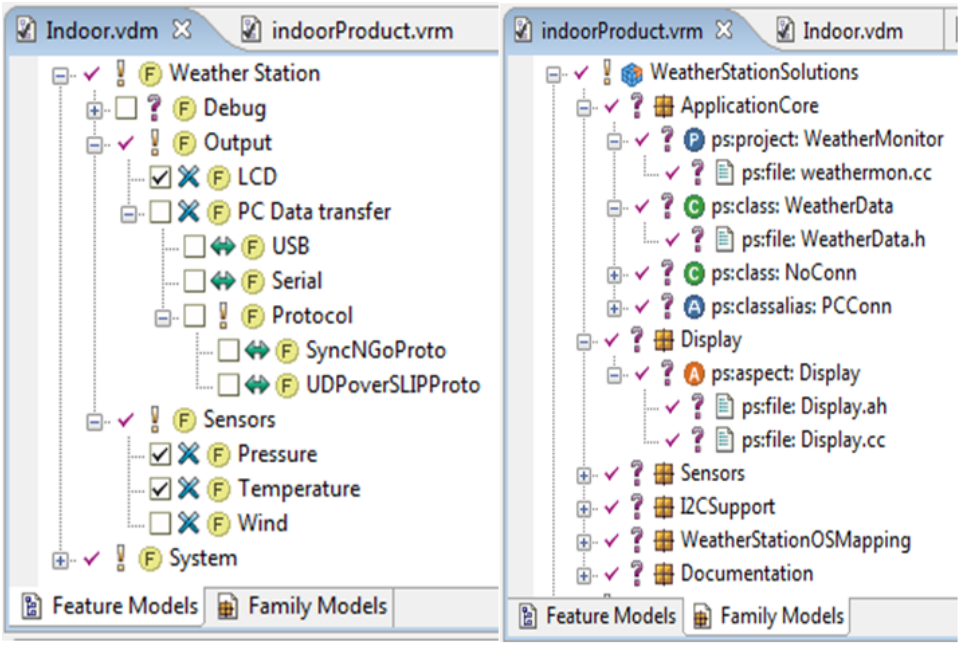

Figure 1.6 Variant Description Model using pure-variants

cannot be granted to tools' proprietary information.

CHAPTER 2 LITERATURE REVIEW

2.1 Model-Driven Development within the context of Certifiable Avionics Software

With Model-Driven Development, models are the basis for software development. Model-Driven Architecture bases the software development process on formal models and the automated transformation of these models from a high abstract level (e.g. UML model) to a lower abstract level (e.g. C++ code) [24], and MDA the covers full software development cycle [15]. With the MDA process, the focus of software developers shifts from code to product-independent models. Within the context of embedded systems, having abstract models as main software development artifacts will facilitate the reusability of the software among different hardware platforms. On the other hand, the cost and effort are shifted to an early stage of the software development cycle. Moreover, extra effort will be required to validate the transformation rules and to implement tools supporting model transformation. In cases where tools are qualified, costs may be even higher.

Formal models and model transformation following the MDA process have evolved to a high level of maturity. Formal models and automated code generation are widely used in the aerospace industry [25]. For example, tool vendors such as Esterel Technologies[1] provide qualifiable tool kits, where code generated from models developed using SCADE language do not require verification. Mathworks[2] is another tool vendor who also provides a DO-178 qualification kit, which contains all documentation and artifacts required to qualify their tools. Therefore, certifiable code can be generated from Simulink models.

There is also UML modeling language, which is a general-purpose modeling language based on an object-oriented programming paradigm from which source code can be generated. For example, IBM's Rational Rhapsody[3] complies with DO-178 and provides the ability to generate code from UML models, but their code generator is not qualified. Therefore, source code generated with Rhapsody do not eliminate the code verification activities required by the certification process.

When these three modeling languages are compared, based on their accuracy, correctness and reliability, SCADE is the most deterministic: it is accurate because it has a formal semantics

[1] http://www.esterel-technologies.com/products/scade-suite/automatic-code-generation/scade-suite-kcg-ada-code-generator/

[2] https://www.mathworks.com/solutions/aerospace-defense/standards/do-178.html

[3] http://www-03.ibm.com/software/products/en/ratirhapfami

language[26]. That is the reason why SCADE is the only modeling language which has a commercially available, qualified code generator. On the other hand, the Simulink language uses a subset to generate code instead of relying on formal semantics, because formal semantics that can be used in industrial contexts require too much efforts [26]. For Simulink language, a different approach is used in order to comply with certification constraints. Instead of qualifying the code generator, code review activities are automated.

Compared to SCADE and Simulink, UML language is less deterministic because it is a general purpose modeling language, which deal with the full spectrum of software engineering. On the other hand, SCADE and Simulink scope is limited to control engineering.

Even though UML modeling language is less accurate than SCADE or Simulink, UML language is still more suited for designing a FMS because only a small portion of a FMS functions can be represented by control theory. Design of complex avionics functions of a modern FMS requires a general-purpose, object-oriented modeling language. Another aspect that favors UML over Simulink and SCADE is that UML is an open modeling language, and Simulink and SCADE are proprietary modeling languages. Considering avionics equipment's very long life span, UML language provides more flexibility. Over time, we have possibility to switch another modeling tool.

In [27], an approach is proposed to use UML for safety-critical software modeling. What they propose is an approach to combine Unified Modelling Language (UML) and SCADE for safety-critical software system development, where UML is used to specify high-level requirements and SCADE specifies software behavior. They also propose a wrapper code generator to integrate SCADE generated code into UML run-time framework code. Therefore, code-level integration between UML and SCADE will allow to co-simulate hybrid applications and to identify earlier problems within the safety-critical software systems. This solution is very interesting but it is not applicable to the context of FMS equipment because very small portion of FMS functions can be designed with control engineering.

In [28], a tool suite for model-driven development of safety-critical, real-time system is proposed. This tool is now replaced with another tool, called MechatronicsUML [29]. For software modeling, the tool relies on their UML-based custom modeling language, *MechatronicUML Modeling Language*. It also provides integration with Matlab Simulink/Stateflow, thus it offers the possibility to do model-in-the-loop simulation and formal verification. The tool provides integration of software and control engineering. In our case, this tool can not be used because it relies on domain specific language more suited to control engineering domain. The use of software product line engineering with this tool will be limited because the tool relies on some custom modelling language.

Actually, both works [27] [28] try to resolve the main limitation of SCADE and Simulink: the narrow context of the software engineering and lack of semantic for designing complex architectural solution.

2.2 Software Product Lines within the context of Certifiable Avionics Software

In [30, 20], authors explain how Software Product Line Engineering can be modeled with UML, through static and dynamic diagrams. They suggest UML extensions using stereotypes for class diagrams (static) and sequence diagrams (dynamic / behavioural). An algorithm to derive a product from a product line model is also proposed. An algebraic approach to derive product-specific statecharts from product line sequence diagrams is also proposed. Use of UML diagrams with stereotypes is easy to comprehend for programmers. Specific product models can be generated from the product line model. Statecharts are generated from sequence diagrams. Code could be generated from those statecharts. On the other hand, there is no way to represent differentiation at a lower level than class level (for example, at function level). Statecharts generation requires detailed product line sequence diagrams. Their proposed UML model is very similar to the Gomaa model [18], which is more widespread. In conclusion, The use of stereotypes to represent product line elements in UML diagrams is a good idea. However, the stereotypes proposed by Gomaa should probably be used, since they are better known. Generation of code from statecharts is an option to strongly consider. Class diagrams may be too high level for our product line models. Another solution would be preferable. Extracting sequence diagrams detailed enough to represent the product line behaviour of the avionics systems is improbable because it would take a significant amount of work and may even not be possible in a reasonable timeframe.

In [18], Gomaa demonstrates how UML can be used to represent software product lines. He goes through multiple topics such as Use Case, Features, Finite State Machines and Statecharts modelling. Architectural patterns are also presented. The book includes multiple case studies which serve as complete examples. The book is a reference in the domain of SPL. The stereotypes proposed by Gomaa are now well known and often used. Most UML diagrams are presented with the necessary modifications for SPL. The methodology is clearly defined, with examples. The complete methodology proposed by Gomaa should probably not be adopted to represent SPL architecture, since it necessitates too many artefacts. Specific UML diagrams adapted to SPL by Gomaa can however be selected and used in our SPL representation. The concept of feature modelling is recurrent in the literature and may be used to easily represent multiple features of a system.

In [31], the feature modelling is introduced. The paper explains what to do and the pitfalls to

avoid when creating a feature diagram. The paper provides a very comprehensive guideline for feature modelling. On the other hand, feature diagrams are not linked to any software implementation artefacts. It may only be used as a domain analysis tool. The feature diagrams may be useful, but only if the feature model is associated with the variations inside the software artefacts.

In [32, 21, 22], the authors have extended the Feature-Oriented Domain Analysis (FODA) into the Feature-Oriented Reuse Method (FORM). FORM supports architecture design and object-oriented development, but also includes a marketing perspective and explores analysis and design issues from that perspective. The methodology includes a detailed feature model, a control behaviour model and an architecture design. The suggested feature model is detailed and includes the notions of optional and alternative features. It also takes into account dependencies between features. The architecture design that they suggest is very complex for a relatively simple system. It would most likely be too complex to use for a system as complex as an avionics system. Their proposed approach is aimed at the management level. It is not necessarily something pertinent to what we intend to do. The feature model they use may be useful though, at least as an example of a complete feature model.

In [33], the author use feature models to represent commonalities and variabilities in a concise way. Those features are mapped to other models (activity diagrams, data specifications, etc.) to give them semantics. The paper proposes an approach to map feature models to other models, and they give examples with activity and class diagrams. The use of feature diagrams allows for a clear notion of feature-based variations in the system. Linking features from feature models to design models may bridge the gap between features and UML models. The examples and approach, provided in the paper, are incomplete: no example is given for a real, complete system. They do not explain how the models are mapped to the feature models and will all fit together. Their mapping is not very well defined: they do not precisely explain why certain elements from the feature diagrams are mapped to models, while others are not. In conclusion, mapping feature models to other models is a great idea. This could be a good starting point, since feature models can easily show the variabilities and commonalities of a system. Defining precisely which design models are associated with a feature is very important for software product line engineering.

In [34], authors design a SPL architecture based on legacy systems. The SPL architecture is not automatically extracted from multiple products. Instead, they extract the original architectural view from legacy code, and do feature modelling based on those extracted models. Feature models are then used to identify core assets and design a common architecture for SPL. The use of feature diagrams illustrates the features that must be present in the SPL

and the identification of core assets allows reuse of legacy systems. This approach is similar to our context because our research is conducted on the legacy FMS system and afterwards a new FMS system is developed using model-based software product line engineering. On the other hand, their process is not automated. They only extract the initial model and rebuild a new SPL architecture almost from scratch. Also, the article presents results and does not really propose a detailed methodology. It is an industrial report.

In [35, 36], authors propose a method to detect changes to product features during evolution. They use a model differencing algorithm to identify changes between different versions of a model. Their approach scales to large systems and can lead to consolidation of product variants in a SPL. Their technology does a "diff" between multiple models. It could be used for pretty much any simple diagram. They currently have the results of their differencing algorithm, but they do not apply them to anything. It is only mentioned that the results could be used in the case of SPL systems. In conclusion, finding the differences between different versions of a model can definitely be useful for SPL, especially once the initial SPL model is extracted. This paper is especially pertinent to manage future modifications made to the SPL models.

Multiple aerospace equipment manufacturers are already employing SPLE for software development [37][38][39][40][41]. Configuration management and certification challenges in adopting SPLE are discussed in [38]. In [41] model-based and product-line technologies are used to reduce certification cost for avionics software of NH90 military helicopter, at Airbus. Cost reduction of certified software development, using model-based software product line, is also the main topic of my thesis. The main difference is that I am using off-the-shelf commercial software modeling and SPL management tools. On the other hand, Wölfl and others, at Airbus, use open source and in-house software development tools. Using open source tools provides possibility to qualify the tools, and consequently eliminate the verification activities of the artifacts generated by those tools.

CHAPTER 3 THE RESEARCH PROCESS

My research project is conducted in two steps, as illustrated in Figure 3.1:

- At first step, I proposed my SPL-based software development methodology for certifiable avionics software development based on model-based software product line engineering. Then the proposed methodology is evaluated by implementing a model-based FMS software component in a real world industrial project and comparing this new methodology to existing certifiable software development method in place. The comparison data is extracted from CMA-9000 certification archive of previously completed projects and the extracted data is validated through workshops involving domain expert from the industry.
- Second step starts with a study of the export-controlled, certifiable avionics software systems, with objective of characterizing the constraints that limit the applicability of the software product line engineering. The study is conducted with semi-structured interviews and the findings are validated through workshops with domain experts. Based on the findings, I propose a new design pattern that enables the reuse of software components among controlled and non-controlled variants of export-controlled, certifiable avionics software systems.

My first article *"Implementing Software Product Line Engineering for Certifiable Avionics Software"*[1], found in chapter 4, reports the results from the first step of my research. My specific research objective 1 and objective 3 are addressed during this step.

The second article *"Software Design Pattern Providing Reusability Among Certifiable, Export-Controlled Avionics Software"*[2], found in chapter 5, communicates the results of the second step of our research. My specific research objective 2 is addressed during this second step.

[1]Paper "Implementing Software Product Line Engineering for Certifiable Avionics Software", in chapter 4, is submitted to journal "IEEE Aerospace & Electronics Systems Magazine", with manuscript # SYSAES-201600256.

[2]Paper "Software Design Pattern Providing Reusability Among Certifiable, Export-Controlled Avionics Software", in chapter 5, is submitted to journal "Innovation in Systems and Software Engineering", with manuscript # ISSE-D-16-00048.

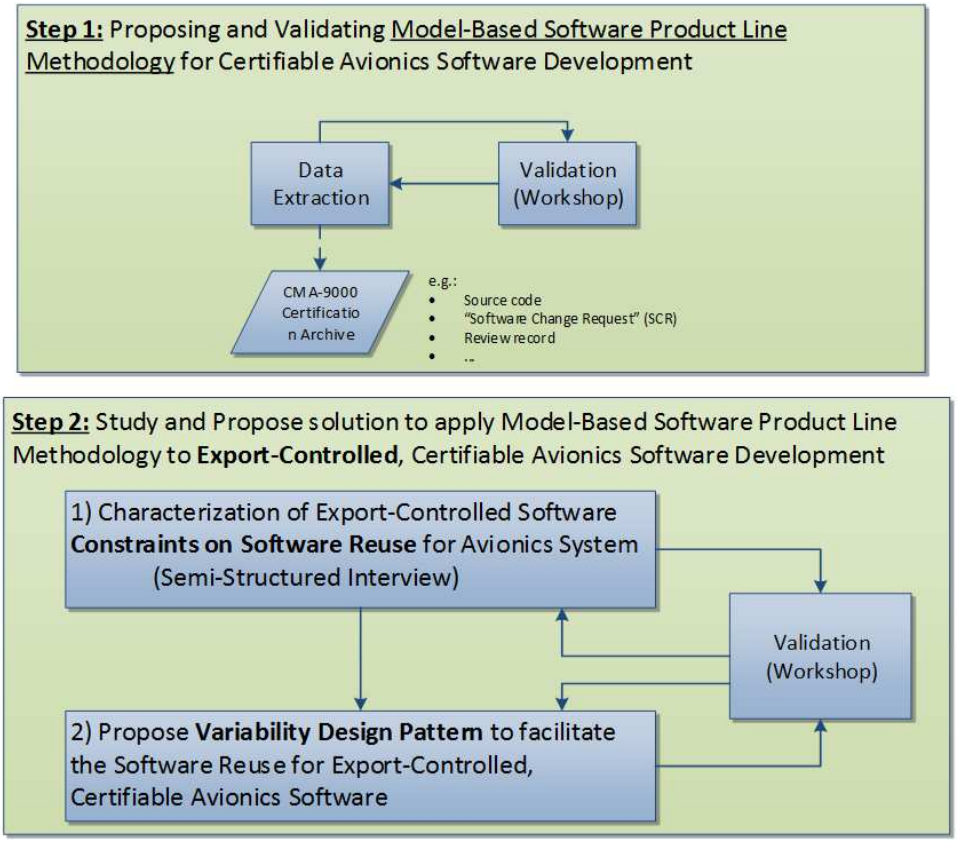

Figure 3.1 Research Process

CHAPTER 4 ARTICLE 1: IMPLEMENTING SOFTWARE PRODUCT LINE ENGINEERING FOR CERTIFIABLE AVIONICS SOFTWARE

Neset Sozen, and Ettore Merlo,
Department of Computer and Software Engineering, Polytechnique de Montreal, Canada

Journal "IEEE Aerospace & Electronics Systems Magazine", manuscript # SYSAES-201600256

Abstract

In this paper, we propose a novel SPL-based software development process for handling variability in complex certifiable avionics software, with the main objective of reducing the cost of complex avionics software development. We also undertake a quantitative and qualitative comparison of our SPL-based solution to existing approaches on handling variability.

We report the results of our case study on the variability and separation of features mechanism in the CMA-9000 Flight Management System (FMS) certified software as well as its impact on certifiable software evolution and maintenance. The insights provided by this case study were used to better tailor and validate our implementation of the software product line (SPL) process for certifiable avionics software development.

Keywords: Software Product Line Engineering, Certifiable Avionics Software, DO-178C, DO-331, Model-Based Design.

4.1 Introduction

Aerospace companies, who manufacture civil avionics equipment, are very conservative in their software development processes. Most still use time-tested software engineering tools and methods. Certifiable complex avionics software development can thus be tedious and costly. The source of this cautious and restrained approach to software development can be found in strict certification constraints such as *DO-178B, Software Considerations in Airborne Systems and Equipment Certification standard*. The recent publication of DO-178C, however, facilitates and enforces the use of model-based methods, object-oriented programming, and code generation tools.

Certification following DO-178C guidance is mandatory for all software on commercial aircraft [4]. This guidance is process oriented and certification is achieved by providing evidence (i.e.

certification artifacts) that the full software life cycle process has been respected. The safety level of avionics equipment determines the quantity of the certification artifacts that are required by regulation authorities. Typically, the certification process increases the cost of software development by 75% to 150% [8].

Esterline CMC Electronics has a family of Flight Management System (FMS) products with different variations and with numerous, complex optional features and navigation capabilities [3]. For various aircraft platforms, customers can select specific configuration options and a subset of features available with the FMS. When manufacturers request certification for avionics equipment, they do so based on the set of features that their specific equipment supports. These features are defined by Technical Standard Orders (TSO), which is a design approval that authorizes the applicant to manufacture the equipment. Similar to product lines in other industries, avionics software requires feature-based variability and handling of feature interactions in software implementation.

4.1.1 Objectives of the paper

In this paper, we will report on the experience we gained as part of Research Project CRIAQ 5.5, in collaboration with École Polytechnique of Montréal, Esterline CMC Electronics Inc., and other industrial and academic partners. The main objective of this project was to explore the use of model-based software development technologies to reduce the cost of complex avionics software development.

The main focus of this paper is to investigate the *"variability and separation of features"* in the legacy CMA-9000 FMS certified software archives, and to propose more cost-effective ways of handling "variability and separation of features" in complex certifiable avionics software. As a secondary objective, we grappled with challenges in certifiable complex avionics software development using model-based methods and commercial off-the-shelf SPL tools.

Aircraft software certification consists in meeting a set of objectives to satisfy at various stages of software development process [4]. The most notable new challenges in the certification of aircraft software include the use of model-based designs and object-oriented programming [5]. Model-based methods tend to blur the distinction between requirements and design [5].

The Software Product Line (SPL) domain has made a great deal of progress over the last decade, and there are several commercial SPL tools that are already in use for software development in other safety-critical domains [6]. But, those tools have some limitations when it comes to certifiable software development. These limitations will be discussed in Section 4.7.

4.1.2 What we accomplished?

Avionics equipment has very long life span, from 20 to 30 years, and certifiable avionics equipment manufacturers are obliged to archive their software products, including all related certification artifacts (e.g. test results, plans, formal reviews, problem reports) throughout the service life of the equipment. Therefore, we had rich databases in which to conduct our exploratory research and case study.

First, we investigated the CMA-9000 FMS family of products in order to understand how variations are handled within the software.

Then, we conducted an exploratory case study to understand and quantify the effort required to implement and certify the features in the legacy FMS CMA-9000 software. Another goal of this exploratory research was to identify the impacts of the current variability mechanism within FMS CMA-9000 implementation on the software maintenance and software evolution.

Based on the findings of this exploratory analysis, we proposed a SPL-based variability mechanism that addresses the challenges of "variability and separation of features" within the context of certifiable complex avionics software development. Then we compared our new variability mechanism to current variability mechanisms in terms of software evolution, maintenance and certification.

In this paper, we also report on our experience using commercial off-the-shelf software product line tools for the development of certifiable complex avionics software.

4.1.3 Contributions

The contributions of this paper are the following:

- Analysis of the variability management mechanism in legacy certifiable avionics software and their impact on software evolution and maintenance;
- Proposal of new ways of handling variability in complex avionics software;
- Proposal of solutions for reducing cost of complex avionics software development;
- Quantitative and qualitative comparison of our solution to current variability management mechanisms used in the legacy system.

The novelty of our solution, as compared to other model-based SPLE applications for certifiable avionics software development, rests on the fact that our solution covers the entire software life cycle process, starting with domain engineering using features models, followed

by architectural modeling with UML, and ending with automatic code generation. Most other solutions tend to skip variability management at the architectural modeling stage, or use domain-specific language which makes their solution impossible to transfer to other domains.

4.2 Certifiable Complex Avionics Software

This section describes the target avionics software system that we used for this study, and the avionic software development and certification environments, with an emphasis on software evolution, maintenance and variability. FAA Advisory Circular (AC) 20-148 *"Reusable Software Components"* [14] is also described in this section. This standard provides guidance for certifying a software component as a reusable software collection that can be used on several projects without having to regenerate certification artifacts. AC 20-148 can facilitate the application of Software Product Line Engineering to certifiable avionics software development.

4.2.1 The FMS Family of Products

Esterline CMC Electronics manufactures various complex avionics systems, ranging from Global Positioning Systems (GPS) and Flight Management Systems (FMS) to fully integrated cockpit systems. This paper will focus on the FMS product family only because in our project, we had direct access to the FMS products and to engineers with over ten years of experience with FMS products.

FMS computes several predictions such as aircraft position, altitude, fuel burn, and Estimated Time of Arrival (ETA), etc. It supports certain complex functionalities such as performance-based vertical navigation, and military functionalities such as definition and navigation of Search and Rescue (SAR) patterns. In short, it helps flight crew reduce in-flight workload. FMS is a software-intensive system that is a central part of aircraft avionics.

Esterline CMC Electronics has several FMS products used on commercial air transport aircraft such as the B747, DC-10 and MD-80; military transports and helicopters such as P-3 patrol aircraft, C-130 air transport, and UH-60, HH-60M and HH-60L Black Hawk helicopter, and numerous trainer aircraft including the Hawker Beechcraft T-6B, Alenia Aermacchi M-311 jet demonstrator, the Korean Aerospace Industries KT-1C and the upgraded Patria Hawk Mk51. Esterline CMC Electronics is also in the retrofit aircraft market, in which interfacing with legacy avionics equipment is required [3].

FMS Software Evolution and Variability

There are over ten FMS software variants that support various aircraft platforms [3] (e.g. helicopter, air transport aircraft). Currently, each FMS equipment software for a specific platform evolves on its own branch. The software for FMSs manufactured by Esterline CMC Electronics is easily configurable. In some configurations, it runs on proprietary hardware CMA-9000 FMS, while in other configurations, it runs on an integrated modular avionics (IMA) platform (i.e. modular hardware using a real-time partitioning operating system for avionics). Also, a feature can be activated or deactivated dynamically at installation, depending on type of aircraft. FMS software is developed using a procedural programming language (i.e. C) and is certified at Design Assurance Level (DAL) C.

With this architecture comes an infinite number of possible variations. This also implies that all supported functions are available in the software. The advantage of this architecture is that the software can be used on multiple aircraft with no modification, simply by activating features specific to an aircraft platform, where the executable object code of the software does not change. On the other hand, if the customer requires new features not present in the software, effort toward certification will increase, because deactivated functions must also be certified. Every line of code in a software component installed on an airborne system requires certification.

There is also NextGen, an FAA program and the SESAR program its European equivalent; these were developed to reduce air traffic congestion, and improve air safety and airspace system efficiency [2]. These programs require that the modern FMSs support upcoming complex avionics functions.

Software FMS project

The *Software FMS* research project was started in response to all these new avionics functions, and the mounting costs of certification. *Software FMS* project goal is to develop the next generation FMS software that will be used on various Esterline CMC Electronics FMS products in the future.

To start with, the following business and software requirements were imposed on the *Software FMS* project:

- FMS software must be open architecture: the software must support the development of independent or customer-designed software applications. This is a very important criterion for military customers, who develop their own mission-critical functions.

- FMS software must be platform independent: the software must support different compilers (e.g. GNU compiler gcc, Microsoft compiler Visual C++), different processor architectures (e.g. Intel x86, PowerPC), ARINC 653 safety-critical avionics real-time operating systems, and existing proprietary FMS hardware platforms.

- Off-the-shelf software development tools must be used: this is to ensure that the focus stays on designing software avionics systems and not on developing a set of software development tools.

- Object-oriented Programming must be used: with the release of DO-332 *"Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A"* [42], the guidelines for designing more sophisticated software architecture were provided.

- Model-based software development with code generation must be used: with the release of DO-331 *"Model-Based Development and Verification Supplement to DO-178C and DO-278A"* [43], the guidelines for using model-based designs with code generation were provided. Therefore, using simulations, we were able to test and validate our software without writing extensive software code. Unified Modeling Language (UML) models using IBM Rhapsody and Mathworks Simulink models were both used for designing and generating code in C++.

A stand-alone piece of software can not be certified. It must be installed into an airborne system. In order to validate our model-based avionics software development process with SPL, the *Software FMS* project was executed within an existing project, with actual customers. In the project where the Software FMS project was executed, the software was component-based. To reduce risk to the schedule, only a single component was developed using model-based. Other software components were either in C or ADA.

4.2.2 Certified Avionics Software Development

DO-178C focuses on safety solely from a software process perspective. The level of effort required to comply with the objectives of DO-178C depends on the safety level of the software. The system safety assessment process establishes the software level of a software component based on potential system failure conditions that can be provoked by an error in this software component.

Some of these objectives must be achieved with *independence*. To achieve an objective with independence, verification activities are performed by a person(or persons) other than the

author(s) of the item that is being verified.

Compliance is achieved by producing a list of certification artifacts (e.g. software development and verification plans, software requirements, design description, source code, test procedures/test results). The precise list of DO-178C deliverables (i.e. software life cycle data) can be found in [4]. Compliance with DO-178C also requires that all tools generating or modifying a certification artifact must be qualified; otherwise, any modifications made to a certification artifact made by the tool must be verified. The objective here is to ensure that the tool has not injected an error into a certification artifact.

Certifiable Avionics Software Development in Practice

Since DO-178C is process oriented, at CMC Electronics every software development activity, from planning to assurance quality, is clearly defined, with all preconditions and inputs to an activity and the outputs from that activity. Most certification artifacts are generated manually, except for the results of automated tests: these are generated using a set of qualified testing tools.

Each modification to the software is captured through *Software Change Request*. The modification may be a bug fix or a very complex feature. A software change request will trace to all certification artifacts related to the feature that it implements. Other information is also recorded in a software change request, such as time spent on implementing the feature; issues recorded (e.g. bugs caused by this software change request).

In next Section, we summarize the certification constraints and coding/design constraints arising from current practice in certifiable avionics software development.

Certification Constraints for Avionics Software

Avionics software certification constraints can be summarized as follow:

- Certification artifacts must be produced and archived for every line of code throughout the service life of the avionics software system, that will be installed an aircraft.
- Traceability from requirements to design, implementation and test artifacts. Traceability is bidirectional (i.e. every requirement must be linked to a line of code and a test procedure, and vice-versa). The goal is to ensure that there is no undocumented function and that all requirements are implemented and tested.
- Validation (i.e. execution of test procedures) must be done on the target. If the target

is changed with no modification to the requirements, design and code, compliance will be lost and the whole certification process must be restarted.

- Exhaustive code coverage is required depending on the design assurance level(DAL). As a minimum, full structural coverage (i.e. every line of code is executed) is required and MC/DC (Modified Condition/Decision Coverage) coverage will be required for software with a higher level of DAL. For CMA-9000 FMS, MC/DC coverage is not required, because it is level C.

Coding and Design Constraints for Avionics Software

In order to meet high standards with respect to software quality and safety, certifiable avionics software must be verifiable and predictable in every aspect(e.g. execution time and memory size) and it must be able to handle hardware failures, errors in hardware/software interfaces, incorrect software response or abnormal conditions. We even need to consider the possibility of a bit shift in the executable code loaded on target avionics systems that may be caused by cosmic radiations. Therefore, several constraints are included in the coding standards and design standards. The most representative constraints are listed below:

- Static memory allocation at system initialization and no dynamic objects;
- No unbounded recursive algorithms; and
- No unreachable code (i.e. dead code).

4.3 Case Study

In this section, the case study design is described. The case study was conducted by inspecting more than a decade's worth of archived certification artifacts of CMA-9000 FMS. The goal was to collect information for a better understanding of the current variability and separation of feature implementation and its implications for software evolution, maintenance and ultimately software development cost.

In the next subsection, research objectives are described, followed by ae description of the research questions and research methodology. The case study was conducted following the guidelines proposed in [44] and [45].

4.3.1 Research Objectives

The subject of this study was to understand the variability and separation of features mechanism in CMA-9000 FMS certified software, and its impact on certifiable software evolution and maintenance. Our main objective was to obtain insights in order to better tailor our implementation of the software product line process for certifiable avionics software development.

4.3.2 Research Questions

The research questions for this study were:

RQ1 How were variability and separation of features handled in the legacy CMA-9000 FMS software?

RQ2 How much effort was required to implement and qualify the features in the legacy CMA-9000 FMS software?

RQ3 What impact do the current variability and separation of features have on the maintenance and evolution of the software?

4.3.3 Methodology

The case study was conducted using the following two techniques iteratively: data extraction and workshops with a domain expert. The information collected from the CMA-9000 FMS certification archives during the data extraction step was guided by our research questions. Then, the information was presented and approved in workshops with the domain expert. Also, during these workshops, it was decided whether enough information had been collected, to answer the research questions.

Data Collection

Data was obtained by manual inspection of the CMA-9000 certification archives. The archives contain all the certification artifacts required for obtainning DO-178 compliance of CMA-9000 FMS equipment for all projects over the last decade. The archives are rich and extensive covering every aspect of the software development process.

To narrow and focus our data collection on the research objectives, we followed various strategies: (1) Variation of the source code (Bottom-up); (2) Impact of a "Software Change Request" (Top-down); (3) Code inspection

Variation of the source code

The objective here was to collect information on software evolution. Starting from a baseline of the code that was certified, we counted the number of version increments for each source file grouped by software component. The baseline was selected with the approval of the domain expert.

Impact of a "Software Change Request"

The objective here was to collect information on the impact of a feature implementation on software maintenance and software certification. Then we selected Software Change Requests (SCR) and analyzed their impact on the software, test and requirements. There were three major types of software change requests: improvements, problem fixes (e.g. software bugs) and requirement changes. As a starting point, we focused on requirements changes because we wanted to see the impact of a new feature implementation and to quantify the cost of development. The software change requests were identified with the domain expert.

4.4 Results

For reasons of confidentiality, coarse outlines of the results are shown and generic names are used when presenting data.

We began our study from a baseline dating back to 2010 that contained most of the major avionics features certified. Each baseline corresponds to at least one project with a specific customer and an aircraft platform. The study was conducted on the third release following the reference baseline.

By evaluating the number of version increments for each source code file in the system, we discovered that most of the core CMA-9000 FMS software components were stable. As shown in Figure 4.1, 80% of the core software components did not change at all or had less than fuive modifications since the reference baseline. The core software components represented 61% of the system.

On the other hand, some parts of the CMA-9000 FMS software were modified extensively. For example, we took one software component implementing a complex avionics software, representing 12% of the whole system. 25% of this component had been extensively modified and 41% of this component had considerable modification (see Figure 4.2).

From this analysis, we were able to identify the sections of the system that were stable and the other sections that were unstable. This information can be used to identify the common-

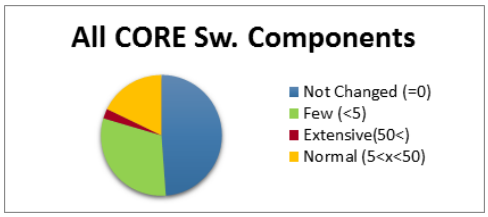

Figure 4.1 Changes in Core Software Components

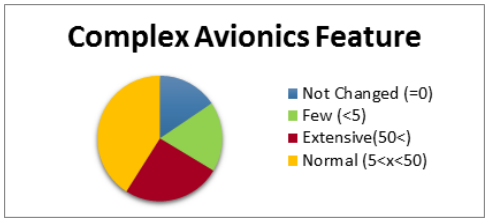

Figure 4.2 Changes in a Complex Avionics Feature Software Component

alities and variabilities in CMA-9000 FMS. For example, core software components are good candidates for commonalities in CMA-9000 FMS, because they had very few modifications between releases with different aircraft platform.

In order to collect information on software maintenance and quantify the effort to develop certification artifacts, we began our study by focusing on software change requests. We selected software change requests that were relatively large, with requirement changes. For CMA-9000 FMS, a software change request is considered large when its development is estimated at more than 20 man/days. We grouped all software development activities for a software change request into three categories: system, code, and test. As shown in Table 4.1, we were given the average time spend for tasks in each category.

During our analysis, we also discovered that large "software change requests" with requirement changes had systematically more source files modified during the code review than during the initial implementation. In some cases, the code review generated twice as much source file modifications. On the other hand, we did not observe this pattern for software testing activities. Therefore, we analyzed the code review reports and we discovered that, on average, 25 % more defects were found after the code formal peer review occurred. Here are some representative examples of the coding defects were discovered later:

- high-level requirement not implemented;
- implementation does not match high-level requirement;

Table 4.1 Effort required to generate certification artifact

Sw. Dev. Tasks Category	Average Effort(in %)
System	11 %
Design/Coding	28 %
Testing	61 %

- high-level requirement changes occurs after the formal code review;
- high-level requirement not implemented;
- software bug found during software testing activities;
- operational issues found during system level validation activities;
- implementation was broken for other variants (i.e. aircraft platform);

All these defects were generated by software testing activities and system-level verification activities. By analyzing those code review defects, we discovered two issues: certification artifacts were generated too early in the process and no variability analysis was done by system engineers.

For CMA-9000 FMS, software development was carried out using a waterfall approach, in which high-level requirements are completed before starting coding activities and coding activities must be completed prior to testing. Completing high-level requirements activities means that all certification artifacts are created before starting the next software development activities (i.e. coding and testing). Once all software development activities at various levels (system, code and testing) are completed, then the test pilot validates the new feature implementation. If the test pilot finds an operational issue, we restart the whole process of creating high-level requirements, coding and testing activities. For example, the operational issues found by the pilot are captured by defects found after formal reviews are performed. From the collected data, on average, 25 % more defects were found after formal peer review; this also means that the cost of software development is increased by 25 % because all software development activities must be redone for those operational issues.

The other issue that we discovered relates to the different variants (aircraft platforms) supported by the CMA-9000 FMS. Typically, every project has one aircraft platform and system engineers will do their validations with the current aircraft platform only, but the software validation team will run tests for all variants supported by CMA-9000 FMS. For example, all variant-related defects were discovered during software validation activities.

4.4.1 Key Insights and Findings

- Stable, unchanging, mature parts of the CMA-9000 FMS software were identified (e.g. 80% of these core functions had no impact or had undergone very few modifications during the last 3 major releases). This information can be used to identify commonalities and variabilities in CMA-9000 FMS software.
- Certification artifacts were created too early in the process for CMA-9000 FMS. This increases the cost of certifiable software development by 25% on average. This issue can be solved by using agile software development methodologies.
- Variability is not handled properly by the current CMA-9000 implementation. When we add a new feature to CMA-9000 FMS for one aircraft platform (i.e. variant), previous implementation for other aircraft platform brakes down.
- The system engineers limited their scope to the aircraft platform for the current project. There is no domain engineering or variability analysis encompassing all aircraft platforms that are supported by CMA-9000 FMS.

Based on these findings, we propose a software product line that is based on a certifiable avionics software development process for FMS equipment. Before introducing our solution, we will present an aviation industry standard "Advisory Circular (AC) 20-148 Reusable Software Components" [14] published by Federal Aviation Administration (FAA).

4.5 Reusable Software Components

AC 20-148 [14] provides guidance for DO-178C compliance of Reusable Software Components (RSC). A DO-178C RSC is a software collection that is recognized as meeting the objectives of RTCA/DO-178C; it may be used on more than one project without having to regenerate certification artifacts. Certification authorities grant RSC acceptance as part of a normal certification process, when the applicant complies also with AC 20-148. An FAA acceptance of RCS allows for the software's deployment on future projects without the added cost and risk of recertification.

In order to obtain RSC certification, compliance should be done twice: at the component level and at the system level (i.e. target platform). If the future users of the RSC change the target platform, this will invalidate its certification. In this case, future users must apply for new RSC compliance. The DO-178C RSC initial cost is higher due to the certification

at two levels (component level and system level). To be cost effective, the target platform should be reused on future systems where the RSC is intended to be used.

The AC 20-148 will be followed to obtain RSC certification for the software components that represent the commonalities of the CMA-9000 FMS discovered during our case study (these are the core software components that remain unchanged for several consecutive projects).

4.6 SPL-Based Certifiable Avionics Software Development

Initially, for the *Software FMS* project, there was no requirement to use software product lines (SPL). Redesigning our FMS with a SPL-based software architecture came up naturally. Avionics systems functionalities are feature-based. Avionics software certification is also feature-based, because when manufacturers apply for certification of their avionics equipment, they do this based on the set of features supported by specific equipment. Therefore, SPLE is the intrinsic approach for certifiable software-intensive avionics system development.

By reducing the scope instead of deactivating a feature that is not required for a specific FMS installation, there is no need to expend extra effort to certify deactivated features.

The Software FMS project (a CMC Electronics research project) was carried out within another project with a customer. In this project, the software life cycle data (i.e. Plan for Software Aspects of Certification, Software Development Plan, Software Verification Plan), required by the certification authorities, had already beed defined. These plans necessitated an update for model-based software development. But instead of modifying the existing plans, supplements were created for those plans that were specific to the model-based software development. For example, we had a standard Software Development Plan and a supplementary Model-Based Software Development Plan, see Figure 4.3.

4.6.1 PLE-Based Avionics Software Development Process

For domain engineering, we used theff-the-shelf variability management tool BigLever Software Gear. The variability was at the UML model level: the variation points were in the UML design models. There was no variation point in the code. The target UML models were generated from the source UML model, which contained the variation points. The source code was then generated using Rhapsody code generator from the target UML model. The Model-driven architecture (MDA) scheme was followed. The source model was the platform-independent model (PIM), the target model was the platform-specific model (PSM).

Software Product Line Engineering deals with multiple projects simultaneously. It suggests

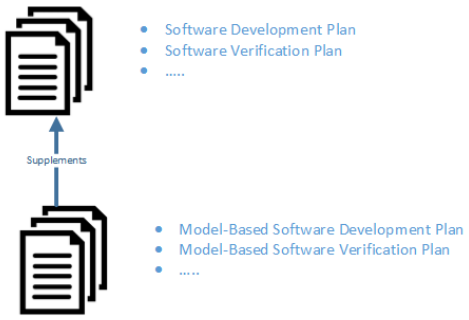

Figure 4.3 Model-Based Document Supplements

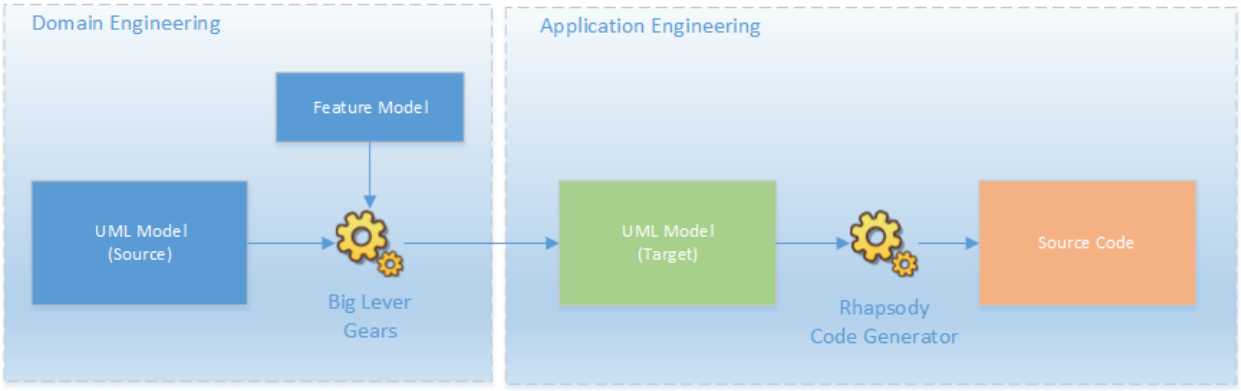

Figure 4.4 PLE-Based Software Development Process

an horizontal organizational model, whereas within most avionics equipment manufacturers, organizational structure is vertical, where what is missing is a domain engineering unit, extending over multiple products within a product family, performing the upfront tasks and laying the groundwork for future reusability.

In the *Software FMS* project, the certification infrastructure was created for any FMS project (i.e. plans, design models, source code, test procedures and test framework).

A generic FMS project was created with all documents, requirements, system models and design models on a separate branch within the configuration management system. This project was called *Core FMS* (see Figure 4.5). Core FMS is not specific to any project and it contains all the variation points in the software plans and, design models. Then, we had a project-specific FMS on another branch, where all the software artifacts were auto-generated.

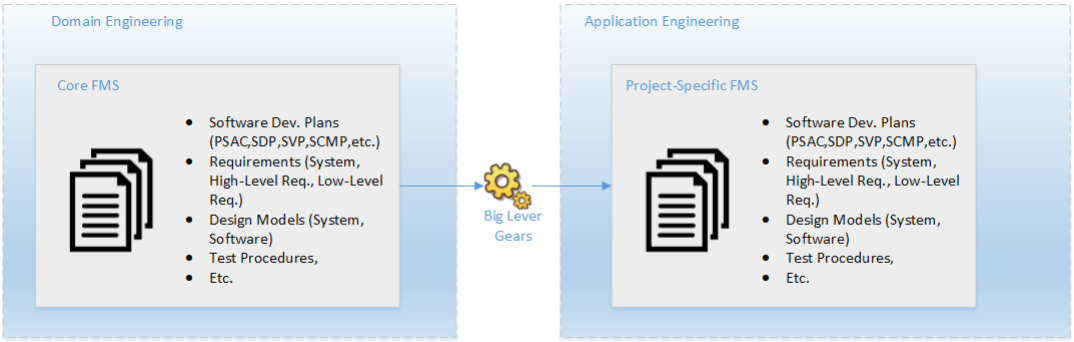

Figure 4.5 Core FMS vs Project-Specific FMS

4.6.2 Software FMS architecture

For FMS equipment, there are two types of features: industry-specific features (e.g. Aviation technical standards such as DO-236C) and customer-specific features. As shown in Figure 4.6, FMS software architecture was composed of three layers in order to reflect those different types of features: *Common Core*, *Core Extension*, and *Customer-Specific*. The Common Core represents the commonalities in the system. It is owned by the Software FMS team because Common Core contains most of the variation points.

The *Core Extension* layer is a container on another branch[1] for the implementation of various industry specific features. The variation points for these features are located in the *Common Core*.

The *Customer-Specific Assets* layer is a container on another branch for the implementation of different customer-specific features. The variation points for customer-specific features are located on the *Core Extension* layer because customer-specific features typically extend the industry specific features.

Each project member could have modifications in all layers and all three layers are at the SPL domain (i.e. all three layers contain source UML models with variation points, see Figure 4.4). The changes to the software components in *Common Core* will require the approval of the *Software FMS* team. They must ensure that there is no impact on another FMS project.

4.6.3 Encapsulating Variations

Each feature implementation is a self-contained Rhapsody project[2], as shown in Figure 4.7. Therefore, each FMS team can work on its specific features separately.

[1]In Rhapsody, it is possible to reference a package located in another project.
[2]In Rhapsody, it is possible to reference another project as a package.

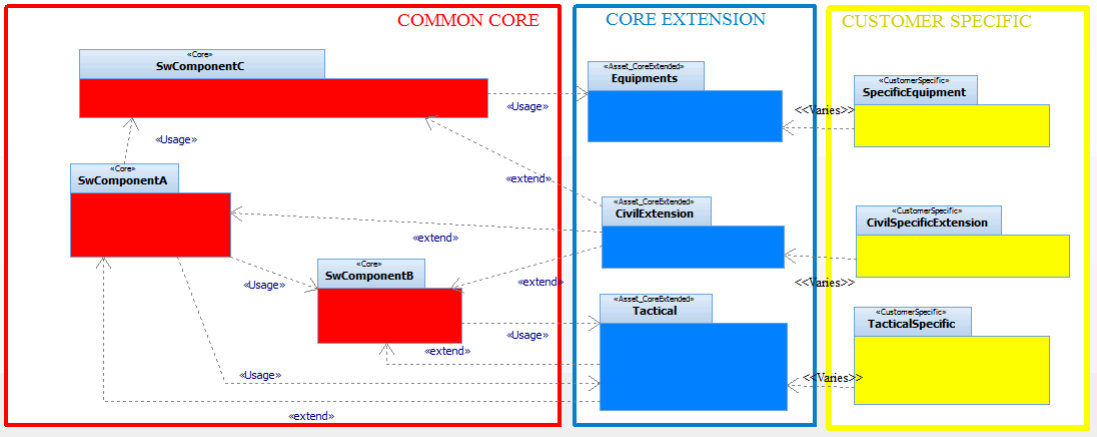

Figure 4.6 FMS Design Model Structure

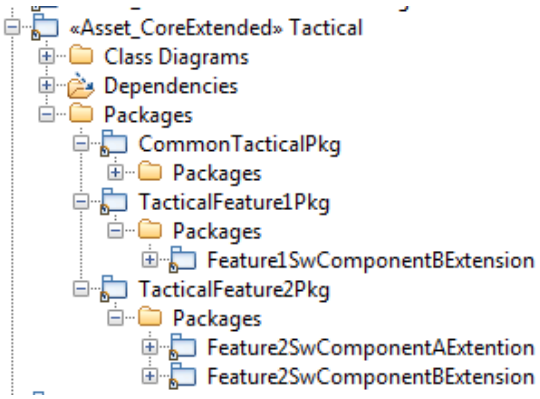

Figure 4.7 Feature Implementation Structure

The variability management tool used in our project allowed us to put variation points on state machines (e.g. on transitions and states), classes, class operations and class attributes. That level of granularity was not enough because most of the variation points, were inside the operations, where the algorithms were defined. There was no mixin operation similar to FeatureC++ [46].

The feature implementation is not located in a secluded section in the code. For example, a feature implementation could have an impact on several software components and inside each software component, there could be new classes introduced by the feature as well as several small modifications (one or two lines of code) to operations.

Structural variability (such as variation on a class, an attribute or any structural element) was achieved using variation points provided by Big Lever Gears. The tool was also very successful in handling behavioral variabilities inside statecharts and on an operation. It was

more difficult to handle behavioral variabilities inside an operation without duplicating the commonalities inside the operations.

4.7 Discussion

In this section, we will analyze our SPL-based certifiable avionics software development process, using the findings from the case study conducted on the CMA-9000 FMS certification artifacts.

The case study identified the most stable and mature parts of the CMA-9000 FMS. Those parts represent the commonalities of the FMS, and this information can be used to identify the software components in the core layer of the new SPL-based FMS architecture. This information can also be used to identify the candidate features in the extension layer that can be certified as a DO-178C reusable software component to reduce the cost of certification over the long term.

During our case study, we also discovered that the creation of certification artifacts upfront before validating implementation was increasing costs by 25% on average. The interesting part is that an agile SCRUM method was used in CMA-9000 FMS projects but the software development was done via a waterfall approach. Of course, using SPL-based software development will not resolve this issue. An analysis of the type of defects that were contributing to increases in software development costs, revealed that several of these defects were related to variability issues (i.e. implementation for one variant broke other variants implementation). For CMA-9000 FMS projects, no variability analysis was done by system engineers because the focus was on the variant of the current project. There was no domain engineering involving all variants of the FMS. Using SPL-based software development would eliminate these types of defects upfront and consequently reduce the additional cost related to these defects that are present with the traditional, single project software development approach.

The features we implemented were not certified even though we followed the DO-178C standard, because the Software FMS project was hosted by a military project, where the DO-178C certification process was not a prerequisite for this host project. The customer required only a demonstration that our process complied with DO-178C and they did not require a compliance certificate from a civil regulation authority.

First we tried to separate each feature implementation from the common core software, to have a self-contained feature that could be certified as a DO-178C reusable software component. Another reason for confining a feature implementation to one package was to reduce the variation points in the system. For example, during our evaluation of the SPL

management tool, even with a simple UML model, we were systematically making errors. Every time an error was fixed for one product variant, another product variant was broken, and this was not visible until all variants were generated.

In addition to the number of variation points in the system, another difficulty in implementing features was the level of granularity of the variations in the core software. The SPL management tool allowed us to put variation points on a class, an attribute or an operation. But, instead of adding fine-grained variation points on attributes and operations, we relied on inheritance and polymorphism and added variation points on the classes to manage the system's complexity. For example, in Figure 4.7, the tactical feature 2 has an impact on two software components in Common Core (i.e. SwComponentA and SwComponentB), but inside each software component, there will be new child classes, and the variation points will be on those classes instead of having several variation points on each attribute or operation with minor modifications (one or two lines of code) in the Common Core. Having coarse-grained variations in the core software allowed us to have all our variations in one secluded location.

For other types of certification artifacts, such as requirements in Doors, or any other text files (e.g. *rtf,*.doc) containing test procedures, test results or software development and verification plans were handled efficiently by the SPL management tool that we selected. Therefore, the main focus of this case study was on handling variation points in software design and code.

4.8 Threats to Validity

The generalizability and confirmability of our results are limited because the study was conducted at the Esterline CMC Electronics site, where access to certification archives was limited to authorized persons. Since the research project is an industrial case study, we had little control over its orientation and execution.

One threat to internal validity relates to the use of version increments to gather data on software evolution. If someone added and removed a modification it would be covered as two modifications. To mitigate this threat, we excluded all version increments related to branching in and out from the main branch. Also, the common practice was to keep the main branch stable by keeping all check-ins to a minimum, and verified by engineering testing (i.e. informal testing). Therefore, the number of version increments was minimal.

Another threat to internal validity is the metrics we used to measure software evolution: not changed (x=0), few changes (x<5), normal(5<x<50) and extensive (50<x). The rationale

behind those metrics is as follows: for a small software change request (e.g. a bug fix), there is a minimum of two version increments: one code check-in for the initial implementation, and another for fixing defects from the code review. And for a large software change request, there will be 2 to 3 software engineers modifying the code, which will result in at least 6 check-ins for the large software change request. 50 version increments to a source file meant that, on average, the file was affected by close to 10 large software change requests (SCR). We also wanted to single out cases where there were few changes, and normal modification, because a file affected by 1 or 2 SCRs means that it is relatively stable.

4.9 Related Work

Use of Software Product Line Engineering for Certifiable Avionics Software

This project is a continuation of Paper [11]. In [11], we explored challenges in applying software product line engineering to certifiable avionics software development and proposed recipes for a successful software product line framework for certifiable avionics software; there is no industrial application of a software product line in this paper, however.

Surveys [37] and Papers [38] [39] [40] [41] have revealed that several avionics and aerospace companies are already using some forms of Software Product Line Engineering (SPLE) for aerospace equipment software development. Some of these papers tackle certification challenges directly. For example, in [38], the authors present the results of applying SPLE to an Engine Monitoring Unit with a focus on configuration management and generation of certification artifacts.

In [47] SPLE was used for certifiable avionics software development. The authors proposed an infrastructure for the SPLE process, aiming at development of certified Unmanned Aerial Vehicles (UAV). They propose a metamodel to support a certifiable SPL development process. In sum, they first identify a list of six issues with certification using SPLE. Then, they start from ProLiCES, a model-driven process for development of SPLs, and propose a list of five modifications at the domain engineering level and five modifications at the application engineering level to ProLiCES, in order to adapt it for certifiable avionics software development. They also identify two non-issues with SPL-based software development process, which they qualify as major certification problems:

1. Standard DO-278B do not dictate a process.
2. Standard DO-278B provide no guidance to certification agencies on how to certify reusable assets.

From our perspective, however, their first issue is not a problem, but an advantage. It allows each organization to adapt its own development process to meet DO-278B objectives. For their second issue, in fact, FAA provides guidance on certifying reusable components through Advisory Circular *AC 20-148 "Reusable Software Components"* [14]. We must also recognize that in their case, they could not validate their proposal, because the guidance for certifying commercial UAVs was not provided by FAA at the time of their paper was published, and avionics software cannot be certified without a host avionics equipment installed on a commercial aircraft.

Another project very similar to ours was realized by Wölfl and others in [41], who assessed cost reduction on generating certification artifacts by using model-based and product-line technology for the avionics software of NH90 military helicopter at Airbus. In our paper, we also attempt to reduce the cost of certifiable software development for an FMS, using model-based software product line technologies. The main difference is that we are using off-the-shelf commercial software modeling and SPL management tools. Wölfl and others, at Airbus, used open source and in-house software development tools. The advantage of using open-source tools is that these tools ca be qualified,and there is no need to review the certification artifacts they generate. In our case, we needed to adapt our software development process to the commercial tools we were using. For example, we needed to review the code automatically generated by Rhapsody, in addition to the Rhapsody UML models, while Wölfl and others, at Airbus, reviewed just the source model, not the code generated. In our case, Esterline CMC Electronics made it clear that they did not want to allocate resources for the development, qualification and support of software development tools. Their decision is understandable, considering that the code reviews represent a small portion of all the effort related to certifiable software development. For example, in our case study we found that the average effort for design and code was 28% of total software development costs and that code review represents a small portion of this 28%, in order of 5% to 10% of total costs for certifiable avionics systems.

Feature-Oriented Software Design

Feature-oriented software design (FOSD) is a paradigm for software conception in SPL. FOSD is based on incremental development of the software in terms of the features that it provides. There are three pillars to FOSD: feature modeling, feature interaction, and feature implementation [48].

A longstanding problem in feature implementation is tracing features easily from the problem space to the solution space [49] [48]. Traceability is mandatory for achieving civil avionics

software certification.

Feature implementation can not be addressed without tackling the notion of feature interaction. Feature interaction is a concept introduced in telecommunication industry [50]; it, along with concepts of feature modeling and feature implementation, is one of the pillars in Feature-Oriented Software Development [48]. Feature interaction is the occurrence of an unexpected behavior, in the presence of two or more features, that does not occur when the features are used in isolation [48]. There are several examples [50] [48] in the literature that illustrate the feature interaction problem. The most common example is the phone with two basic features: *call waiting* and *call forwarding*. When both features are combined and the line is busy, it is not clear which feature will prevail.

Several solutions are proposed to resolve feature interaction [51] [46] [52] [53]. The most prominent solution is to make features and feature interactions explicit at the programming language level [51]. In [51], the author proposes separating the additional code, induced by a feature selection, from the base code. Additionally, the author also proposes extracting the extra code required to handle the interaction of two features into a separate module [51]. Feature C++ [46] is a more practical example of this approach.

The solutions proposed were not directly applicable to our project, because all solutions were at the programming language level and in our project, the FMS avionics software was developed using the UML models, and the source was then auto-generated. Nevertheless, some concepts underlying these solutions were useful for us. One example is the separation of each feature from feature interaction implementations. During our case study, the main focus was on feature implementation and the implementation of the *Common Core* layer of the FMS. Therefore, in our experiment, only *Common Core* features were implemented, and we did not experience any feature interaction issue. When we will start *Core Extension* and *Customer Specific* feature implementation, we will need to use a similar approach, in which we will need to isolate features implementation and feature interaction implementation, into separate classes.

Feature-Oriented Model-Driven Development (FOMDD) is proposed in [54]. The authors define the mathematical properties to validate the abstractions, tools and specifications of FOMDD that they are proposing. As the authors suggest, while their results can be beneficial in the development of tools, models, and specifications for other domains using FOMDD [54], their results could not be used in our project, because the *Software FMS* project is about applying the SPLE, not about developing new SPLE tools.

4.10 Conclusions

The Software FMS Project is a complex one, in which several new technologies are introduced simultaneously (Model-based software development, automatic code generation, object-oriented programming, and software product lines).

Nevertheless, using only mainstream, off-the-shelf tools, we have proposed a model-based SPLE methodology for certifiable avionics software development, which covers the entire software life cycle. We have also validated our methodology within an industrial context. One of the main advantages of our methodology is that it is not specific to FMS equipment. It can be replicated for any certifiable avionics software system because our solution is based on general-purpose languages (e.g. UML, C++) and mainstream off-the-shelf tools. Most other SPL-based proposals either use domain-specific language, which can not be used for other domains of application, or use custom-built tools that are not publicly available or that address only a small portion of the full software life cycle (e.g. only code generation, or only domain analysis with feature models).

In this paper, the main focus has been feature implementation and validation with our SPLE process. Consequently, some important aspects of the SPLE have not been adressed. A complete solution on feature implementation cannot be proposed without addressing feature interaction issues, because these pose a major obstacle to feature-oriented software development [50, 48, 53].

FeatureC++ is another way to accomplish feature-oriented software development. With the release of DO-178C and DO-332, which provides guidelines for object-oriented programming, the use of C++ for certifiable avionics software development will be more common in the aerospace industry. Therefore, in a future work, FeatureC++ should also be investigated in order to verify its usefulness within an industrial context.

4.11 Acknowledgements

The authors would like to thank CRIAQ and Esterline CMC Electronics for their support of this project.

4.12 References

R. DO, "178," *Software considerations in airborne systems and equipment certification. RTCA and EUROCAE*, 2011.

E. Thomas, "Certification cost estimates for future communication radio platforms," Rockwell Collins France, Tech. Rep., 2009.

C. E. Esterline, "http://www.esterline.com/avionicssystems/en-us/productsservices/navigationampfmsgps.aspx," 2016.

J. Rushby, "New challenges in certification for aircraft software," in *Proceedings of the ninth ACM international conference on Embedded software*. ACM, 2011, pp. 211–218.

R. Flores, C. Krueger, and P. Clements, "Mega-scale product line engineering at general motors," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 2012, pp. 259–268.

A. Advisory Circular, "Ac 20-148," *Reusable Software Components, US Department of Transportation, Federal Aviation Administration*, 2004.

P. Brooker, "Sesar and nextgen: investing in new paradigms," *Journal of navigation*, vol. 61, no. 02, pp. 195–208, 2008.

R. DO, "332 object-oriented technology and related techniques supplement to do-178c and do-278a," 2011.

——, "331:" model-based development and verification supplement to do-178c and do-278a"," 2011.

D. E. Perry, S. E. Sim, and S. M. Easterbrook, "Case studies for software engineers," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. IEEE, 2004, pp. 736–738.

S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311.

S. Apel, T. Leich, M. Rosenmüller, and G. Saake, *FeatureC++: Feature Oriented and Aspect Oriented Programming in C+*. Citeseer, 2005.

N. Sozen and E. Merlo, "Adapting software product lines for complex certifiable avionics software," in *Product Line Approaches in Software Engineering (PLEASE), 2012 3rd International Workshop on*. IEEE, 2012, pp. 21–24.

W. T. Force, "State-of-the-art survey for product lines," CESAR, Tech. Rep., 2009.

I. Habli and T. Kelly, "Challenges of establishing a software product line for an aerospace engine monitoring system," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*. IEEE, 2007, pp. 193–202.

F. Dordowsky and W. Hipp, "Adopting software product line principles to manage software variants in a complex avionics system," in *Proceedings of the 13th International Software Product Line Conference*. Carnegie Mellon University, 2009, pp. 265–274.

F. Dordowsky, R. Bridges, and H. Tschope, "Implementing a software product line for a complex avionics system," in *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, 2011, pp. 241–250.

A. Wölfl, N. Siegmund, S. Apel, H. Kosch, J. Krautlager, and G. Weber-Urbina, "Generating qualifiable avionics software: An experience report (e)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 726–736.

R. T. V. Braga, O. T. Junior, K. R. C. Branco, L. D. O. Neris, and J. Lee, "Adapting a software product line engineering process for certifying safety critical embedded systems," in *Computer Safety, Reliability, and Security*. Springer, 2012, pp. 352–363.

S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994, pp. 94–101.

T. F. Bowen, F. Dworack, C.-H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The feature interaction problem in telecommunications systems," in *Software Engineering for Telecommunication Switching Systems, 1989. SETSS 89., Seventh International Conference on*. IET, 1989, pp. 59–62.

C. Prehofer, "Feature-oriented programming: A fresh look at objects," in *ECOOP'97—Object-Oriented Programming*. Springer, 1997, pp. 419–443.

J. Liu, D. S. Batory, and S. Nedunuri, "Modeling interactions in feature oriented software designs." in *FIW*, 2005, pp. 178–197.

C. Kästner, S. Apel, M. Rosenmüller, D. Batory, G. Saake *et al.*, "On the impact of the optional feature problem: analysis and case studies," in *Proceedings of the 13th International Software Product Line Conference*. Carnegie Mellon University, 2009, pp. 181–190.

S. Trujillo, D. Batory, and O. Diaz, "Feature oriented model driven development: A case study for portlets," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007, pp. 44–53.

CHAPTER 5 ARTICLE 2: SOFTWARE DESIGN PATTERN PROVIDING REUSABILITY AMONG CERTIFIABLE, EXPORT-CONTROLLED AVIONICS SOFTWARE

Neset Sozen, and Ettore Merlo,
Department of Computer and Software Engineering, École Polytechnique de Montréal, Canada

Journal "Innovation in Systems and Software Engineering", manuscript # ISSE-D-16-00048

Abstract

There are two types of avionics systems: civil and military. The software on civil avionics system requires civil certification following DO-178C, and software on military avionics systems is subject to strict control by governmental export regulations, such as the Arms Export Control Act (ACEA) and the International Traffic in Arms Regulations (ITAR). This dichotomy leads to multiple challenges, when Software Product Line (SPL) Engineering is used for certifiable, export-controlled avionics software development. The main challenge is to give access to the civil(non-controlled) part of an avionics software system while limiting access to military (controlled) part of the software, which runs counter to the SPL paradigm. The second challenge is contamination of the civil software component by controlled software component: this occurs when controlled software is added to civil software, which then falls under ITAR regulations and can no longer be used for civil purposes without additional ITAR licenses, even after removal of its military software parts.

In this paper, we analyze the development of certifiable and export-controlled avionics systems regarding the applicability of software product line process. We identify challenges preventing the applicability of commercial SPL management tools to the development of certifiable and export-controlled avionics systems software. We identify the list of design patterns from the literature that are suitable for variability implementation. Lastly, we propose a new design pattern that can enable the use of SPL management tools for certifiable and export-controlled avionics software and facilitate software reuse among controlled and non-controlled variants of certifiable and export-controlled avionics software systems.

Keywords: Software Design Pattern, Software Product Line Engineering, Certifiable Avionics Software, Export-controlled Software, DO-178C, DO-331, Model-Based Design.

5.1 Introduction

All civil avionics software systems are subject to certification constraints imposed by DO-178 standards. Certified avionics software development can be tedious and costly, due to strict certification constraints. To cope with the cost of certifiable software development, manufacturers tend to be very conservative in their software development processes; they tend to use time-tested software development tools and methods. However, these time-tested software development tools and methodologies are insufficient in dealing with the increased size and complexity of modern avionics software systems.

With the publication of DO-178C, use of modern software development technologies is facilitated. To reduce the cost, most avionics manufacturers are switching to model-based software development or software product line engineering, which promote the software reuse.

On the other hand, most avionics equipment can be installed on military aircraft as well as civil aircraft. Like any military equipment, avionics equipment used on a military aircraft are subject to export regulations. As a consequence, access to software on export-controlled avionics equipment is limited to authorized persons. To prevent any infringement of export regulations, most avionics equipment manufacturers will have a separate branch for their export-controlled variants of their avionics equipment with no possible reuse of their software with non-controlled variants, even if major portion of the software implementation is identical on both variants. Providing reusability among controlled and non-controlled variants of an avionics equipment may have considerable cost reduction on software development.

5.1.1 The Research Problem

Most avionics equipment designed for civil aircraft can be deployed on military aircraft too. In this case, avionics equipment manufacturers complies with export regulations by duplicating the software of their avionics equipment. Then, the software of the export-controlled variant of their equipment will evolve on a separate branch and sharing any software artifacts with non-controlled variants will be averted.

This strategy for complying with export regulations further increases the costs of certifiable avionics software development. Providing reusability among controlled and non-controlled variants may yield considerable cost reduction for the development of certifiable and export-controlled avionics equipment.

Although software product line (SPL) engineering promotes reusability, its application to export-controlled and certifiable software development presents multiple challenges. The first challenge is limiting access to export-controlled features of the avionics equipment's

software in an SPL-based system. SPL engineering consists in creating a domain system, which contains all possible variations in the system, then deriving an application system (or a variant system) containing only features relevant to this application. For example, a civil variant of an avionics equipment will contain only civil avionics functions. Therefore, with an SPL-based system, everyone will gain access to the domain system, which also has export-controlled features.

The second challenge is contamination of non-controlled software by export-controlled software. Software contamination occurs when non-controlled software is used for military applications and afterwards used for a civil applications; this requires additional export licenses, which were not initially required.

5.1.2 Research Objectives

This case study was conducted within the context of the application of software product line engineering to certifiable avionics software system development, where variability is handled with feature models. This case study tackled challenges in using model-based software product line engineering methods for certifiable and export-controlled avionics software development using commercially available SPL tools. Software product line engineering provides a systematic software reuse.

The main goal of this case study is to provide reusability for software artifacts among civil (i.e. non-controlled) and military (i.e. controlled) avionics projects, using software product line engineering. More specifically, we wanted to understand the limitations and constraints imposed by export-controlled avionics systems on software product line engineering methods. Next, we propose software design solutions that address those constraints, because using commercially available SPL tools, we can not modify either the tools or the concepts and formal languages that define those tools. As a result, the following specific objective emerged: *"Design and evaluation of software design patterns for export-controlled, certifiable avionics software systems that will increase reusability among civil and military projects."*

5.1.3 Context and Limitations

This study was conducted at the Esterline CMC Electronics Flight Management System (FMS) group. Esterline CMC Electronics has FMS products with different variations and with numerous complex optional features and navigation capabilities [3]. For various aircraft platforms, customers will select specific configuration options and a subset of features available with the FMS.

Aircraft software certification consists in meeting a set of objectives at various levels from requirements to code and test development [4]. The most notable new challenges in the certification of aircraft software is the use of model-based design and object-oriented programming [5].

Software Product Line (SPL) domain has progressed appreciably over the past decade; there are several commercial SPL tools that are already in use for software development in other safety-critical domains [6]. But, those tools have some limitations when it comes to certifiable and export-controlled software development. This case study will focus on such limitations and propose solutions for alleviate them.

5.1.4 Contributions

This industrial case study makes the following contributions:

- It analyzes the development of certifiable and export-controlled avionics systems regarding the applicability of software product line engineering.
- It identifies challenges preventing the applicability of commercial SPL management tools to certifiable and export-controlled avionics systems software development.
- It identifies software design patterns from the literature that are suitable for variability implementation.
- It proposes a new design pattern that can enable the use of SPL management tools for certifiable and export-controlled avionics software and that can facilitate software reuse among controlled and non-controlled variants of certifiable and export-controlled avionics software systems.

5.1.5 Organization of the Paper

The rest of this paper is structured as follows: the next section sets out certifiable and export-controlled avionics software systems; we then present our case study design, followed by its results; the succeeding section discuss our results; threats to validity are presented in the following section; related work in this field is outlined next; the paper ends with conclusions and acknowledgements.

5.2 Certifiable, Export-controlled Avionics Software Systems

In order to use software on commercial aircraft, the certification following DO-178C guidance [4] is required. This guidance consists of a set of objectives that must be met so that the software life cycle process and activities will achieve these objectives. In sum, certification is about providing evidence that the full software life cycle process is respected.

DO-178C focuses on safety solely from a software process perspective. The level of effort required in order to comply with the objectives of DO-178C depends upon the safety level of the software. The system safety assessment process sets the software level of a software component based upon potential system failure conditions that can be provoked by an error in this software component.

Some of these objectives must be met with *independence*. To achieve an objective with independence the verification activities are performed by a person(persons) other than the author(s) of the item being verified.

Developers can use any software development model provided they comply with DO-178C objectives. Compliance is achieved by producing a list of software life cycle data (e.g. software development and verification plans, software requirements, design description, source code, test procedures/test results). The precise list of DO-178C deliverables (i.e. software life cycle data) can be found in [4]. The DO-178C deliverables are reviewed by the certification authorities and/or their representatives.

5.2.1 Aviation Technical Standards

Typically, the operational and performance features (i.e. requirements) of an avionics system are defined by standards. For example, the Radio Technical Commission for Aeronautics (RTCA) document *DO-236C "Minimum Aviation System Performance Standards: Required Navigation Performance for Area Navigation"* [55] sets out the requirements for area navigation such as performance based navigation and aircraft path definition.

When a manufacturer requests certification for avionics equipment, a Technical Standard Orders (TSO) authorization is issued to the manufacturer. A TSO is a design approval; it will authorise the applicant to manufacture the equipment. This means that the design of the avionics equipment will comform to the requirements in the technical standards.

DO-236C encompasses features and capabilities related to aircraft navigation; other technical standards define other aspects of avionics systems. Each TSO refers to a part of a technical standard. The avionics systems functionalities are feature-based. Hence, software product

line based software is well suited to avionics systems. For example, the DO-236 provides the vertical navigation (VNAV) requirements, an optional avionics feature. There are different types of VNAV implementation: barometric VNAV, performance VNAV or geometric VNAV. Therefore, it is easy to represent an avionics system's capabilities using feature diagrams, as shown in Figure 5.1.

5.2.2 Military vs Civil Avionics

While the military does not require civil certification, most will request it for higher quality in avionics systems; also, by having civil certification, they will be able to use their military aircraft on civil airbases as well. Without a certification of airworthiness, an aircraft, whether civil or military, is not allowed to operate on a civil airport.

Avionics equipment with military functionalities are controlled by the International Traffic in Arms Regulations (ITAR) in the United States. Similar regulations exist in nearly all countries, and these regulations control the export and import of defense-related articles.

Most companies have separate version of their avionics equipment, for military and for civil aircraft. Still, in 2014, several large aerospace manufacturers were fined, as much as $79 million, because of ITAR violations [56]. We have no way to tell if some of those violations are caused by civil avionics equipment containing military features. Within the context of software product line engineering (SPLE), it is cruicial to separate the ITAR regulated piece of software and its related software life cycle artifact from the civil part of avionics software.

5.3 The SPL-Based Avionics Software Development Process

Variability was done at the UML model level (i.e. the variation points were put on the UML design model elements). There is no variation point in the code. As shown in Figure 5.2, the target UML model was generated from the source UML model, which contained the variation points. Then, the source code was generated using the Rhapsody code generator from the target UML model. The Model-driven architecture (MDA) scheme was followed: The source model was the platform-independent model (PIM), the target model was the platform-specific model (PSM).

As shown in Figure 5.3 FMS software architecture was composed of three layers: *Common Core*, *Core Extension* and *Customer Specific*. The Common Core contains most of the variation points.

The *Core Extension* layer is a container on another branch for the implementation of different

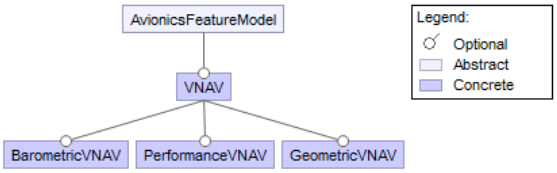

Figure 5.1 VNAV Feature Diagram

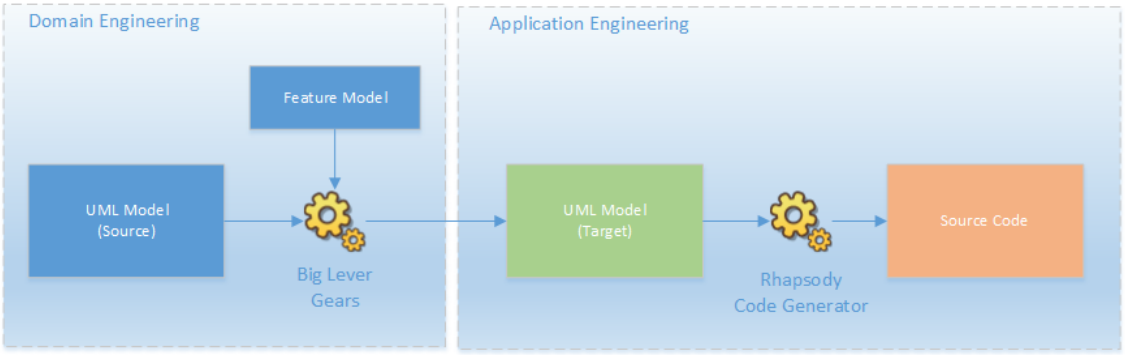

Figure 5.2 PLE-Based Software Development Process

features. The variation points for these features are located on the *Common Core*. For example, features originating from aviation technical standards such as DO-236C are located in this layer.

Customer-Specific Assets layer was created as a container on another branch for the implementation of different customer specific features. The variation points for these customer-specific features are located on the *Core Extension* layer.

Each project member could undergo modifications in all layers, and all three layers are in the SPL domain.

5.4 Case Study Design

This case study was conducted within the context of software product line engineering application to certifiable avionics software system development, where variability is handled using feature models. The main goal of this case study was to reduce the cost of software development for export-controlled, certifiable avionics systems by facilitating reusability among civil and military projects. More specifically, we wanted to understand the limitations and constraints imposed by this type of system on software variability and to propose software design solutions that address these constraints. Therefore, the following specific objective

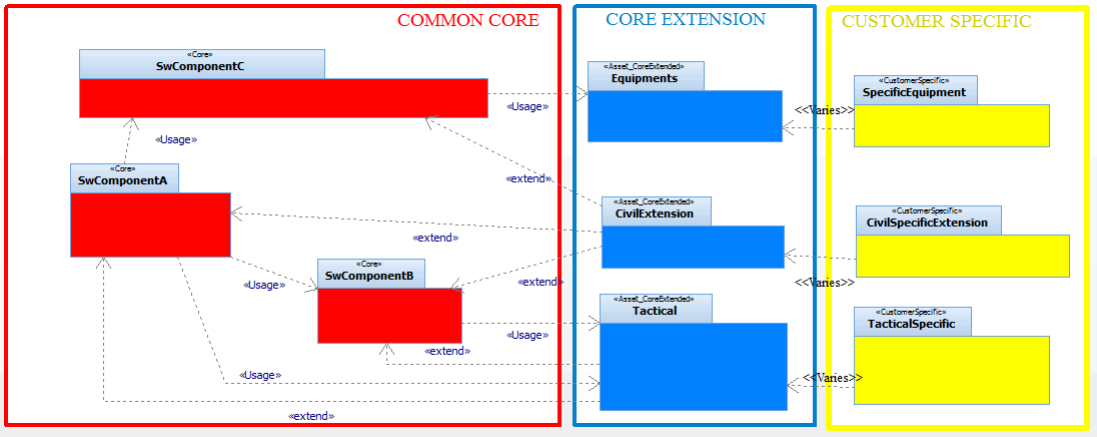

Figure 5.3 FMS Design Model Structure

emerged: "Design and evaluation of software design patterns for export-controlled, certifiable avionics software systems that will increase reusability among civil and military projects."

In this section, the case study design is described. First, the research questions are described, followed by the description of the proposals and units of analysis; this section ends with a description of our the research methodology. The case study was conducted following the guidelines proposed in [44] and [45].

5.4.1 Research Questions

From our specific objective, the following research questions arose:

RQ1 How can we reuse software components between civil and military software projects without infringing exportation rules or contaminating civil software?

RQ2 What are the limitations of commercial SPL tools for reusing software components between civil and military software projects, without infringing exportation rules or contaminating civil software?

RQ3 What impact do variability design patterns have on software evolution, maintenance and certification?

5.4.2 Cases and Subjects of the Study

Esterline CMC Electronics manufactures various Flight Management System (FMS) products for civil and military aircraft with different variations and with numerous complex optional features and navigation capabilities [3]. To prevent infractions to export regulations, there is

currently no reuse between civil certified FMS and export-controlled military FMS. Within the context of a research projects, at Esterline CMC Electronics, where software product line engineering is studied to develop the next generation of FMS software, we wanted to explore the feasibility of reuse between civil and military FMS products. Therefore we conducted a series of semi-structured interviews in order to understand the context and extract constraints preventing reuse between civil certified FMS and export-controlled FMS. The interviews were conducted with two project engineers and two senior software engineers who worked on civil certified FMS projects and export-controlled FMS projects. Participants were selected on their experience and their extended knowledge in the field of certifiable, export-controlled FMS equipment. All participants had no fewer than ten years of experience with certifiable FMS software development and, they worked on at least one export-controlled FMS equipment. They all had training with export-controlled software development process.

Based on these findings, we proposed a design solution that copes with these challenges and constraints that prevent software reuse between civil and military FMS products. Our design solution has been demonstrated by means of an example taken from existing FMS products.

5.4.3 Methodology

Our case study was conducted in two steps. First, we characterized the constraints limiting reusability for export-controlled, certifiable avionics software systems. Secondly, we proposed a new design pattern to address these constraints, and validated our design pattern by implementing a military avionics feature using our new design pattern, then evaluating that design pattern with domain experts.

Data Collection Procedure

Step 1: Export-controlled Software Constraints on Software Reuse for Avionics Software

We began this first step using with documents and literature reviews, in order to understand the field of avionics software development and software product line engineering. Semi-structured interviews were our main source of data for understanding the domain of certifiable, export-controlled avionics software and for identifying constraints that preventing software reuse. These interviews were conducted on the topic outlined in Table 5.1.

Table 5.1 Topics for the semi-structured interview

Research Question	**Topic**
RQ1	Certification process
RQ1	Military avionics domain
RQ1	Military and civil avionics standards
RQ2,RQ3	Used software development tools, methods and process
RQ2	Software product line engineering
RQ1	FMS civil and military functional and non-functional requirements
RQ1,RQ3	Certifiable avionics software development
RQ3	Evolution and maintenance of avionics software
RQ3	Evolution of functional requirements
RQ3	Evolution of non-functional requirements

Step 2: Design and Evaluation of the Variability Design Pattern for Export-controlled Certifiable Avionics Software

Based on findings from the Step 1, we conducted a survey of currently available SPL tools. We then selected software product line tools and used them to implement a military FMS avionics feature, which we had identified with the help of domain experts. The goal was to assess the applicability of these tools to certifiable, export-controlled avionics software. The limitations of the tools selected were identified and used to define our design pattern solution.

A literature review was conducted to extract state-of-the-art design patterns, with the goal of identifying potential design patterns that would facilitate software component reuse within certifiable and export-controlled avionics software systems.

Based on the data previously collected, we proposed our design pattern solution and implemented the military FMS avionics feature used during the software product line tools assessment. Then, the design patterns were evaluated through workshops with domain experts. These workshops had three purposes: first, to validate our findings at each steps; second, to complement our results with expert feedback; and, third to validate the applicability of our design pattern by demonstrating an implementation of military FMS avionics feature using our design pattern in a real world setting.

5.5 Case Study Results

For confidentiality reasons and for the sake of clarity, the coarse outlines of the results are presented and generic names are used when presenting software design solutions.

The results are structured into four sections: (1) constraints for software reuse between

controlled and non-controlled avionics software systems; (2) results from evaluation of commercial software product line tools for certifiable and export-controlled avionics software systems; (3) the results from the literature review, done to identify potential design patterns facilitating software reuse for certifiable and export-controlled avionics software; (4)lastly, presentation of our design pattern solution, which handles constraints for software reuse with certifiable and export-controlled avionics software systems. This section ends with a description of threats to the validity of our results.

5.5.1 Constraints for Software Reuse between Controlled and Non-controlled Avionics Systems

At Esterline CMC Electronics, separate versions of FMS products are available for military and civil aircraft, respectively. Export regulations require that access to an export-controlled FMS software artifacts (e.g. requirements, source code, test procedure, test results) is limited to authorized personnel. Also, access is project based: for example, a software engineer with access to an ITAR *Project A* does not have access to another ITAR *Project B*, even in case where the customer is the same for both projects. In order to limit access to an export-controlled project, separate servers must be used to store software artifacts; the test stations must be located in separate, limited-access laboratories, and engineers must not leave any sensitive documents on their desks. The security agents verifies the desk of every employees working on an export-controlled project to ensure that there are no sensitive documents left unsupervised.

Another issue with reusing software components between controlled and non-controlled software systems is the *software contamination* of non-controlled avionics software systems. Software contamination occurs when military-related software is added to a non-military avionics equipment, even if this piece of software has been deactivated, the equipment falls under ITAR regulations, which means that the software's initial intent was to implement only features required for civil certification, and so it may no longer be used for civil-purpose aircraft without additional ITAR licenses and approvals.

5.5.2 Evaluation of Commercial SPL Management Tools for Certifiable and Export-controlled Avionics Software Systems

When we presented the list of (over 40) SPL management tools currently available for SPLE [57], our industrial partner made it clear that the focus of this research project was not to evaluate the applicability of all these tools to our context but to select a tool that is commercially available, mature enough to be used in an industrial context, and for which

the tool vendor will provide post-purchase support. This narrowed our list to two tools: GEARS[58] and PureVariants [59]. In short, the selection of the SPL management tool was made by our industrial partner.

For the evaluation of the tool, one tactical military avionics feature was selected, with the help of domain experts. The whole project was in Rhapsody UML. During tactical feature implementation, selected SPL management tools allowed us to put variation points on state machines (e.g. on transitions and states), classes, class operations and class attributes. That level of granularity was not yet enough, because most of the variation points, introduced by our tactical feature, were inside the operations, where the algorithms were defined.

Following our experiment, we evaluated the tool based on the constraints on software reuse identified in the previous step: (1) restricted access to controlled software implementation, and (2) non-controlled software contamination. Both constraints were violated, because the source UML model contains all the variation points and the implementation of each variant for all projects (controlled or not). All projects need access to the source UML model in order to generate their target models and to generate their source codes from the target UML model. Therefore, unauthorized access is given for controlled software implementation and the non-controlled software will be contaminated, since the same source (the UML model) is used to generate the target model and source code of the non-controlled software.

5.5.3 Design Patterns for Variability Implementation

Gamma and others, in [60], have categorized design patterns into three categories: creational , structural and behavioral patterns. Our focus was on structural and behavioral patterns. Creational patterns were dismissed because they abstract the instantiation process and do not contain any details on system features implementation.

We reviewed the established design patterns for variability implementation guided by the constraints identified in the first step of the study. The following design patterns were identified:

- Bridge (structural);
- Strategy (behavioral); and
- Template method(behavioral);

The intent of all the identified design patterns is to decouple the abstraction from its implementation, which is an important criteria for limiting access to implementation of a variant.

The Bridge design pattern accomplishes this separation of the abstraction from its implementation at the class level. Strategy and template method patterns do this separation at the operation level (i.e. the algorithm level). The difference between strategy and template method patterns is that strategy variates the whole algorithm by defining a family of algorithms; with template method pattern the structure of the algorithm is fixed, and certain steps of the algorithms variate. For our case, the strategy pattern was redundant, since the SPL management tool did exactly the same thing by adding a variation point to an operation.

None of the selected design patterns could be used as suggested in the literature [60], because most of them rely on dynamic memory allocation. To predict the memory footprint of an avionics system at compile time, dynamic memory allocation is prohibited; the memory must be allocated statically at system initialization.

5.5.4 Export-controlled Feature Implementation

Previously identified constraints on variability implementation will be addressed in two steps. Firstly, each feature implementation must be separated as a self-contained Rhapsody project[1], on a separate branch as shown in Figure 5.4. Each FMS team can thus work on its specific features separately without having access to other features implementation.

Secondly, we will propose a new design pattern that will separate controlled features implementation from non-controlled features implementation in an avionics software system. We have called this design pattern "Template Method with Extension Class", as shown in Figure 5.5.

Template Method with Extension Class Design Pattern

In our case study, we encountered two types of variation:

- Fine-grained variations (e.g. certain steps inside an operation);
- Coarse-grained variations (e.g. new classes, new operations).

Only coarse-grained variations can be handled by the SPL management tool. The template method with an extension class is proposed for handling fine-grained variations. With this pattern, fine-grained variations are encapsulated in primitive operations, and primitive operations are called from common core operations. This way, the piece of code that implements the fine-grained variation is separated from common core operations.

[1] In Rhapsody, it is possible to reference another project as a package.

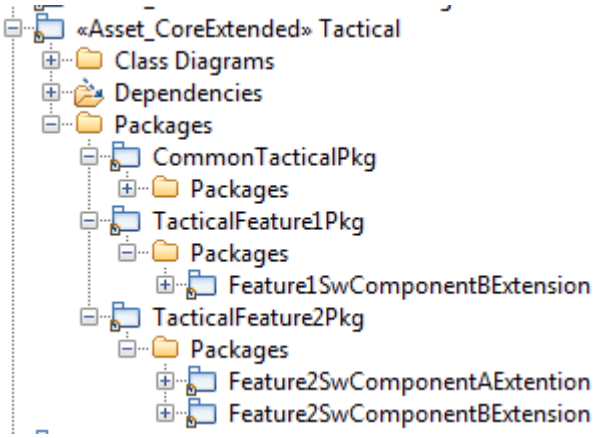

Figure 5.4 Feature Implementation Structure

As shown in Figure 5.5, the template method with an extension design pattern is created by mixing the two design patterns we had identified during the previous step: the Bridge and Template methods.

Each time a common core class operation is affected by a feature, an abstract extension class (e.g. *CAbstractExtensionClass*) is created in the core extension branch (e.g. *CivilExtensionPkg*), accessible to all features. The primitive operation is implemented by a concrete extension class (e.g. *CCivilExtensionClass*) located in the feature package on a separate branch (e.g. *Feature1Pkg*). Then, the primitive operation (e.g. *primitiveOperation_2()*) defined in the common core class calls the primitive operation (e.g. *extendedPrimitiveOperation_2()*) defined in the abstract extension class: see Figure 5.6.

Our template method pattern is slightly different from the classical template method pattern[60] proposed by Gamma. In the classical template method pattern, there are only two types of classes: an abstract class, which defines the skeleton of the algorithm in the *templateMethod()* operation, and concrete classes that implement variant versions of primitive operations. A concrete class is a subclass of the abstract class. In our pattern, the primitive operations implementation is separated from its definition similar to what obtains in the Bridge design pattern[60].

This design has several advantages. First, it hides the feature implementation from the common core, which prevents common core software contamination with export-controlled features. Second, the common core software will not change. In consequence, the executable object code of the common core software remains intact. Within the context of certifiable avionics software, this can be a cost-saving factor, because each time executable object code of certifiable avionics software changes, the entire certification process must be redone. Cer-

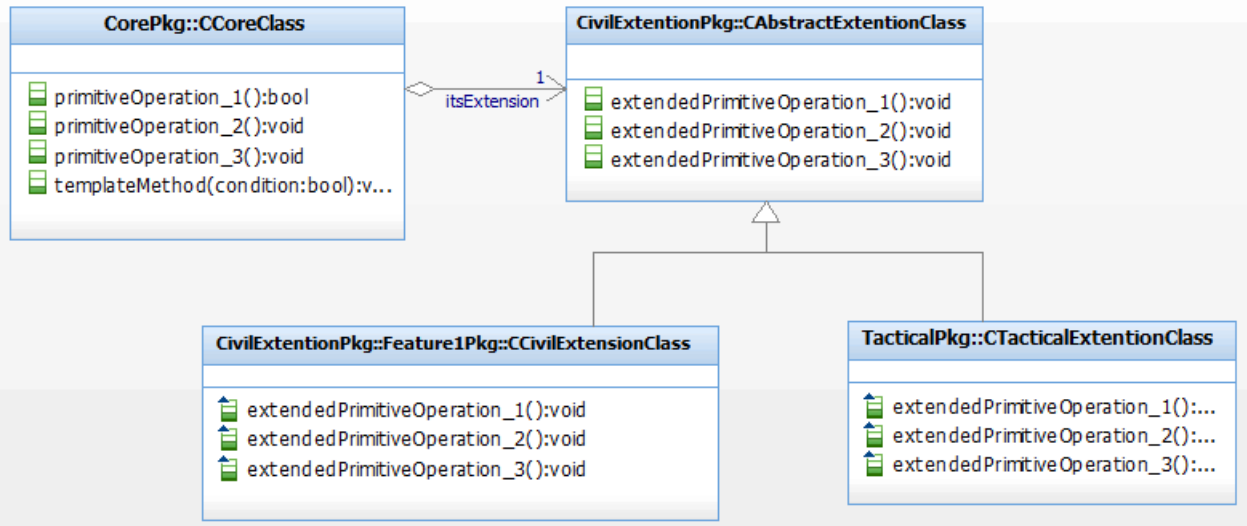

Figure 5.5 Template Method with Extension Class Design Pattern

```
void CCoreClass::primitiveOperation_2()
{
    itsExtension->extendedPrimitiveOperation_2();
}
```

Figure 5.6 Primitive Operations Implementation

tifying the common core of the FMS as a reusable software component with advisory circular AC 20-148 *"Reusable Software Components"* [14], can help reduce the cost of software development.

5.5.5 Impact on Software Evolution, Maintenance and Certification

The data collection on the impact of our design pattern solution on software evolution, maintenance and certification was accomplished by demonstrating our solution to participants, followed by a semi-structured interview where all participants were present.

The centralization of all changes that are related to a feature in one location (i.e. one UML package) will facilitate evolution, maintenance and certification. From a software evolution perspective, we can easily identify the impacted software components, because the feature implementation will have an extended version for each software component modified by feature implementation. For example, in Figure 5.4, we see that TacticalFeature2 has extended Software Component A and B. On the other hand, TacticalFeature1 has impacted only Software Component A. With this design, we can also encapsulate interactions between features,

as shown in Figure 5.7. Software maintenance will also be improved, because the modifications are localized and there is no risk of impacting any other software component. The certification process will be facilitated, because it will be easy to demonstrate the impact of the feature on the overall system to certification authorities.

5.6 Discussion

The certification process concerns the software's entire life cycle, not just its source code or design. Since the focus of this research project is to provide a technical solution to promote software reuse among certifiable, export-controlled avionics software systems, we will limit our discussion to software implementation.

In this section we will expand two points, related to the use of SPLE for certifiable and export-controlled avionics software development:

- The *Variability implementation* in commercial tools and its impact on certifiable avionics software development.
- The *handling of feature interaction*. Feature interaction is the occurrence of an unexpected behavior, in the presence of two or more features; this behaviour does not occur when features are used alone.

Variability Implementation

The first point of discussion is the way SPL tool vendors implement the variability. Typically, there are three types of variability [61]:

- *Negative variability*, where parts of a big and complex system are removed. The advantage of this approach is its simplicity, but on the other hand, the system may lack validity because it contains all possible variations in the system.
- *Positive variability*, where variant parts are added to a minimal core system. With this approach, the system itself will be minimalist and typically will contain only the common part; but on the other hand the variability management will be much more complex.
- *Parametrization* is about changing the behaviour of an algorithm by changing input variables.

Most commercial SPL tool vendors opt for negative variability. In our opinion, negative variability is selected because this approach seems to be easier to implement. For example, the selected SPL management tool will transform a "source model" containings all the variation points to a "target model", which generates a specific product.

When we mentioned our interest in positive variability to one of the SPL tool vendors, the vendor argued that there is no fundamental difference between the two approaches, as long as the derived product is the same. And, they were right. Within the context of export-controlled complex avionics software development, positive variability might have been useful for separating controlled feature implementation from non-controlled software. Still, we have managed to exclude export-controlled features from the civil common core software components with our design pattern without having to rely on positive variability. In our case, limiting access to controlled features was an important requirement, because access to an export-controlled feature is granted on a project basis only. For example, two persons working on two separate export-controlled projects, do not have access to each others work.

Handling Feature Interaction

A software-intensive system is constructed with operations, classes, and software components. A feature implementation will not necessarily correspond to a specific class or software component. Two different features implementations may be located either in one class or in one operation. Feature interaction is the occurrence of an unexpected behavior, in the presence of two or more features, that does not occur when the features are used in isolation [48], [50]. The unexpected behavior is eliminated by adding extra code to the derived software and this additional code, called a *lifter*[51], is present only when both features are active.

In our case study, only one export-controlled feature was implemented. Therefore, there was no feature interaction in our case study; nevertheless, feature interaction needs to be addressed in order to achieve full-scale use of SPL for certifiable avionics software development.

As set out in [51], theoretically, the number of lifters can grow exponentially along with the number of different feature combinations. Nonetheless, in reality, based on decades of experience in certifiable complex avionics software development, the possible feature combinations requiring lifters are much fewer than the theoretical numbers and in general, a set of no more than three features will require lifters.

The template method pattern with extension class can be used to handle feature interaction as well. As shown in Figure 5.7, an extension class (e.g. *CFeature1andFeature2ExtensionClass*)

is created for every possible feature combination that requires lifters. The additional code for each lifter is placed in a separate, primitive operation. The extension class implementing the feature interaction should be located in a separate package because it will be present only when all the features from which it originated are active, and the feature interaction extension class is not involved with feature implementation when this feature is unaccompanied by other features.

5.7 Related Work

5.7.1 Software Product Line Engineering for Certifiable Avionics Software

In [47] an infrastructure for SPLE process for development of certified Unmanned Aerial Vehicle (UAV) was proposed. Similar to our solution, they also suggested a self-contained feature with all their certification life cycle data. At the time of the publication of this paper, the guidance for certifying commercial UAVs were not provided yet by FAA. Therefore, there is no experimental validation of

Software product line engineering is already being used by aerospace companies for aerospace equipment software development as shown in surveys [37] and papers[38] [39] [40] [41].

Some of those papers specifically focus on certification challenges in applying of software product line engineering, in an industrial context. In [38], SPLE is used for software development of an Engine Monitoring Unit. The authors mainly focused on the configuration management and generation of certification artifacts. For configuration management with a product line, they reported on challenges of having two levels of engineering (i.e. product line and project) and they proposed a configuration management process to address conflicts in managing artifacts versions over time and over projects. For certification process with a product line, they considered different approaches: non-traditional approaches (e.g. Reusable Components, AC 20-148), or software reuse as *Previously Developed Software (PDS)* as specified in the "Additional Consideration" section of DO-178. Then they concluded that product line should be considered as PDS because non-traditional approaches prolong the certification process with extra certification artifacts that must be provided. To reduce the certification effort, they proposed to attach the certification life cycle data to each reusable artifacts. We also come up with the same conclusion that a reusable software artifact should contain all its certification lifecycle data. On the other hand, the paper stayed on the process level. There is no description of how the software product line is implemented, if the products are generated automatically or manually. While our research focus on the implementation of the software product line and challenges related to automatic software generation from the

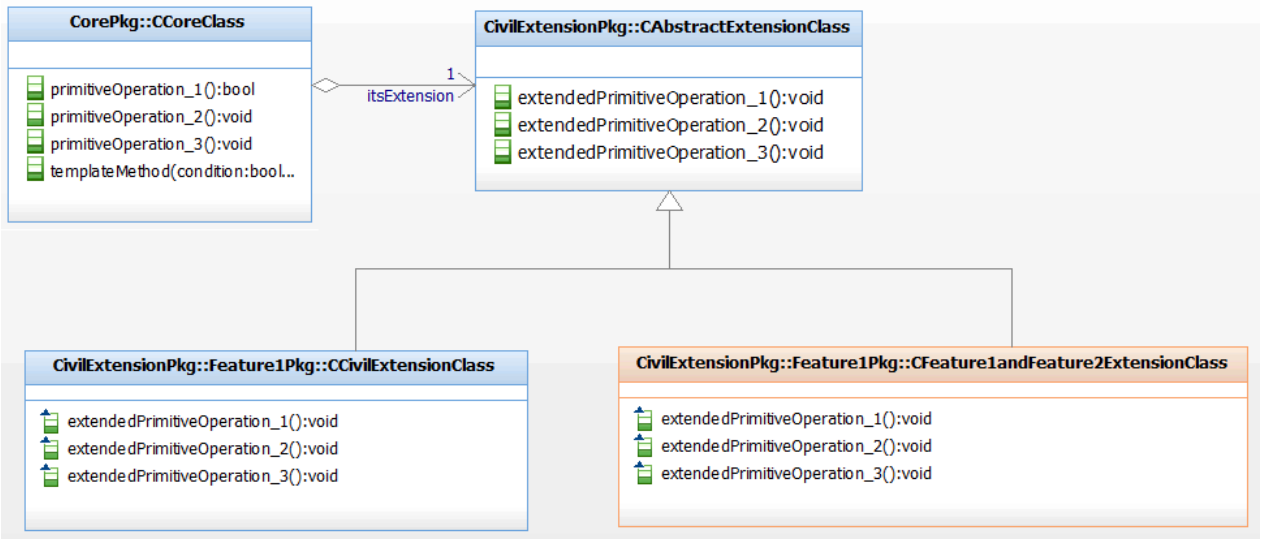

Figure 5.7 Template Method Design Pattern with Extension Class for Feature Interaction

product line.

Wölfl and others in [41] assessed the cost reduction of certifiable software development by automating certification artifacts generation using model-based and product-line technologies for avionics software of NH90 military helicopter at Airbus. Their work was very closed to our research project but in our paper, we try to resolve a different problem in the context of applying SPLE to certifiable avionics software development: the limitation on applying SPLE to certifiable avionics equipment which may have controlled and non-controlled variants.

5.7.2 Controlled Software Development

No paper was found discussing about controlled software development but a SPLE based auditing method for protecting restricted content in derived products were found in [62]. They propose an auditing method that will ensure that the content restrictions in the derived products have been met. What they propose is complementary to our work. In our research, the main focus is at export-controlled, certifiable avionics software development with the model-based design while their auditing method focus mostly at feature modeling level and writing assertion to ensure that no content restrictions can be violated.

5.7.3 Feature-Oriented Software Design

Feature-oriented software design (FOSD) is a paradigm for software conception in SPL. FOSD is based on incremental development of the software in terms of features that it provides. There are three pillars to FOSD: feature modeling, feature interaction, and feature implementation [48]. The focus of our case study is feature implementation and feature interaction.

The necessity to separate export-controlled features from non-controlled features brings another dimension to existing problems in feature implementation. A longstanding problem in feature implementation is tracing features easily from the problem space to solution space [49] [48]. Traceability is mandatory to achieve civil avionics software certification but it is not enough to have a complete separation of the export-controlled features implementation. It important that there is no visibility of the export-controlled features implementation to those who has access to civil features only. In other word, a person working on a civil FMS project should have no access to export-controlled features.

Feature implementation can not be addressed without tackling the notion of feature interaction. Feature interaction is a concept introduced in telecommunication industry [50] and it is one of the pillar, with concepts of feature modeling and feature implementation, in Feature-Oriented Software Development [48]. Feature interaction is occurrence of an unexpected behavior, in presence of two or more features, that does not occur when features are used in isolation [48]. There are several examples [50], [48] in the literature to illustrate the feature interaction problem. The most common example is the phone with two basic features: *call waiting* and *call forwarding*. When both features are combined and the line is busy, it is not clear which feature will prevail.

Several solutions are proposed to resolve feature interaction [51] [46] [52] [53]. The most prominent solution is to make features and feature interactions explicit at the programming language level [51]. In [51], the author proposed to separate the additional code, induced by a feature selection, from the base code. Additionally, the author also proposed to extract into a separate module the extra code required to handle the interaction of two features [51]. The Feature C++ [46] is a more practical example of this approach.

The proposed solutions were not directly applicable in our project because all solutions were at programming language level and in our project the FMS avionics software was development using UML model then the source was auto generated. Nevertheless, some concepts behind those solutions were useful for us. For example, the separation of each feature and feature interaction implementations. In our solution, the features implementation and feature

interaction implementation are isolated in separate extension classes.

5.8 Threats to Validity

The generalizability and confirmability of our results are limited because the study was conducted at Esterline CMC Electronics where access to data is limited to only authorized people. To mitigate this threat we used several participants validate our results. Another threat to generalizability is the subject of the study was very specific and focused because the study was conducted to resolve a specific industrial need. But our solution could still be used in other domain where we want to limit access to a feature implementation.

A threat to internal validity was the fact that the main researcher was from the domain of research. To reduce the bias of the main researcher, the case study was conducted iteratively where other participants from the industry were included in the study to validate and to approve the findings through workshop at every step of the study.

Another threat to internal validity was that only one export-controlled feature was used for experimentation. The main reason for that was the context of the study: the access to other industrial participants and their direct involvement on the way the study was conducted. For example, the SPL management tool used in the study was imposed by the industrial partners. This threat was mitigated by involving expert knowledge from the participants.

5.9 Limitations

In our project, an important aspect of the feature-oriented SPLE were not studied in detail: the feature interactions. A complete solution on feature implementation can not be proposed without addressing the feature interaction issues because it poses a major obstacle to feature-oriented software development [50, 48, 53]. It is imperative to redo this case study with more features.

Another limitation was to validate our design pattern solution to facilitate reusability among controlled and non-controlled variants of an avionics software system, with the export regulation authorities. During, this research project we did not have access to export regulation authorities, instead we validated our solution with several participants previously involved in the process of getting authorization from export regulation authorities.

5.10 Conclusions

Most avionics equipment certified for civil aircraft can also be installed on military aircraft. Also there are multiple studies [38] [39] [40] [41] [47] on the use of software product line engineering for avionics software development and certification.

For the first time, the applicability of software product line engineering for export-controlled and certifiable avionics software development is studied in this paper.

Software product line engineering is about systematic reuse of software assets where every project has full access to whole product line, which is in contradiction with export regulations. We also identified the constraints that limit the application of software product line engineering to export-controlled and certifiable avionics software development: (1) limiting access to control software artifacts of the product line and (2) preventing non-controlled software contamination by controlled software artifacts.

And, we proposed a new design pattern that enables the application of software product line engineering to export-controlled and certifiable avionics software development and facilitate the software reuse among controlled and non-controlled variants of certifiable and export-controlled avionics software systems.

5.11 Acknowledgements

The authors would like to thank Esterline CMC Electronics for their support in this project.

5.12 References

C.E. Esterline. http://www.esterline.com/avionicssystems/en-us/productsservices/navigationampfmsgps.aspx (2016)

R. DO, Software considerations in airborne systems and equipment certification. RTCA and EUROCAE (2011)

J. Rushby, in *Proceedings of the ninth ACM international conference on Embedded software* (ACM, 2011), pp. 211–218

R. Flores, C. Krueger, P. Clements, in *Proceedings of the 16th International Software Product Line Conference-Volume 1* (ACM, 2012), pp. 259–268

C. SC-227, *DO-236C Minimum aviation system performance standards: Required navigation performance for area navigation* (RTCA, 2013)

U.D. of State. https://findit.state.gov/search?utf8=(2014)

D.E. Perry, S.E. Sim, S.M. Easterbrook, in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on* (IEEE, 2004), pp. 736–738

S. Easterbrook, J. Singer, M.A. Storey, D. Damian, in *Guide to advanced empirical software engineering* (Springer, 2008), pp. 285–311

J.A. Pereira, K. Constantino, E. Figueiredo, in *International Conference on Software Reuse* (Springer, 2015), pp. 73–89

C.W. Krueger, in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion* (ACM, 2007), pp. 844–845

O. Spinczyk, D. Beuche, in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (ACM, 2004), pp. 18–19

E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns: elements of reusable object-oriented software*, vol. 206 (Addison-wesley Reading, MA, 1995)

A. Advisory Circular, Reusable Software Components, US Department of Transportation, Federal Aviation Administration (2004)

M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L.C. Kats, E. Visser, G. Wachsmuth, *DSL engineering: Designing, implementing and using domain-specific languages* (dslbook. org, 2013)

S. Apel, C. Kästner, Journal of Object Technology **8**(5), 49 (2009)

T.F. Bowen, F. Dworack, C.H. Chow, N. Griffeth, G.E. Herman, Y.J. Lin, in *Software Engineering for Telecommunication Switching Systems, 1989. SETSS 89., Seventh International Conference on* (IET, 1989), pp. 59–62

C. Prehofer, in *ECOOP'97—Object-Oriented Programming* (Springer, 1997), pp. 419–443

N. Sozen, E. Merlo, in *Product Line Approaches in Software Engineering (PLEASE), 2012 3rd International Workshop on* (IEEE, 2012), pp. 21–24

W.T. Force, State-of-the-art survey for product lines. Tech. rep., CESAR (2009)

I. Habli, T. Kelly, in *Software Product Line Conference, 2007. SPLC 2007. 11th International* (IEEE, 2007), pp. 193–202

F. Dordowsky, W. Hipp, in *Proceedings of the 13th International Software Product Line Conference* (Carnegie Mellon University, 2009), pp. 265–274

F. Dordowsky, R. Bridges, H. Tschope, in *Software Product Line Conference (SPLC), 2011 15th International* (IEEE, 2011), pp. 241–250

A. Wölfl, N. Siegmund, S. Apel, H. Kosch, J. Krautlager, G. Weber-Urbina, in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (IEEE, 2015), pp. 726–736

R.T.V. Braga, O.T. Junior, K.R.C. Branco, L.D.O. Neris, J. Lee, in *Computer Safety, Reliability, and Security* (Springer, 2012), pp. 352–363

P. Clements, C. Krueger, J. Shepherd, A. Winkler, in *Proceedings of the 17th International Software Product Line Conference* (ACM, 2013), pp. 218–226

O.C. Gotel, A.C. Finkelstein, in *Requirements Engineering, 1994., Proceedings of the First International Conference on* (IEEE, 1994), pp. 94–101

S. Apel, T. Leich, M. Rosenmüller, G. Saake, *FeatureC++: Feature Oriented and Aspect Oriented Programming in C+* (Citeseer, 2005)

J. Liu, D.S. Batory, S. Nedunuri, in *FIW* (2005), pp. 178–197

C. Kästner, S. Apel, M. Rosenmüller, D. Batory, G. Saake, et al., in *Proceedings of the 13th International Software Product Line Conference* (Carnegie Mellon University, 2009), pp. 181–190

S. Trujillo, D. Batory, O. Diaz, in *Proceedings of the 29th international conference on Software Engineering* (IEEE Computer Society, 2007), pp. 44–53

CHAPTER 6 GENERAL DISCUSSION

This research project was conducted in an industrial context. As a consequence, the research was oriented by our industrial partner needs and they were directly involved in defining the research objectives as well as methodological aspects of the research.

Our critical literature review was regrouped under two themes: Model driven development for certifiable avionics software and software product line engineering for certifiable avionics software.

Within the context of certifiable avionics software, recent research projects [27, 28, 29], on model driven development, focus mainly on the control engineering or system engineering where the requirements are validated early in the software life cycle, or the source code is generated using qualified tools (e.g. SCADE Suite). The modeling is realized using with domain specific languages (e.g. SCADE, Simulink).

As established with our industrial partner, the focus in our research was to do model driven development using off-the-shelf commercial software modeling tools (e.g. IBM Rhapsody) with general-purpose software modelling languages (e.g. UML). We list some of the reasons behind this decision: availability of the software engineers, no formation is required to use the modelling language, no need to allocate resources for implementing and maintaining the software modelling tools. Also, the whole FMS software can not be designed just with control oriented modelling language. As a comparison, other research project, conducted in the similar context of certifiable avionics software domain for Airbus Helicopter [41], was realized using open source SPL management tools. Their main reason of opting for in-house tools was the possibility of qualifying the tools and by using qualified tools, they do not need to systematically review the output of the tools. In our case, in addition to the lack of qualification of the tools (unless the tool is qualified by the vendor), we had to deal with the commercial SPL management tools limitations in their applicability to export-controlled and certifiable avionics software development. The second paper, presented in chapter 5, addressed those limitations with commercial SPL management tools and their applicability to export-controlled, certifiable avionics software development. Therefore, my thesis complements current research in the field of model-driven development for certifiable avionics software development.

The state-of-the-art research [37, 38, 39, 40, 41] on application of software product line engineering to certifiable avionics software development is very similar to our research project, resulting with similar outcomes. Similarly, the main focus of most of the recent research

activities, in this field, is the cost reduction on generating certification artefacts. However, two aspects that differentiate our research from the current state-of-the-art researches are (1)proposition of solutions to allow the reusability among export-controlled certifiable avionics software systems and (2)the use of widely available, off-the-shelf, commercial tools for applying model-based, software product line engineering to certifiable avionics software development and resolving the limitation of those tools in their applicability to the certifiable avionics software domain through software design patterns and other means instead of developing tools in-house from scratch.

CHAPTER 7 CONCLUSION AND RECOMMENDATIONS

At the time this research work began, the DO-178C certification standard had not yet been released. Compared to DO-178B, which sets limits on current avionics manufacturers using modern software development methodologies, the DO-178C standard provides guidelines for using modern model-based, object-oriented methods and technologies. The initial goal was to allow our industrial partner to be ready when DO-178C is released, in order to be be more competitive against foreign avionics manufacturers. From this perspective, we conclude that by providing technical solutions for applying model-driven methodologies to certifiable avionics software development, we have achieved our goal.

During my research, I have published papers other than those describd in Chapter 4 and Chapter 5, which constitute the backbone of my thesis. For example, challenges in adopting software product line engineering for complex certifiable avionics software development were introduced in [11]. I was also involved in [12] and [13], which describe a software comprehension tool that provides automated features mapping to source code of legacy CMA-9000 FMS software. In CMA-9000 FMS software, the features of the FMS system are dynamically activated (or deactivated) using configuration variables.

My thesis also provides a case study with an analysis of the CMA-9000 FMS certified software artifacts, in order to understand the variability management in a legacy FMS system, and compares it to our new model-based variability management system using software product lines.

Lastly, my thesis presents the topic of software development for export-controlled avionics systems to the research community for the first time. I have also identified the constraints preventing the applicability of the commercial SPL management tools to certifiable and export-controlled avionics systems software development, and have proposed technical design solutions that alleviate those constraints

In the following sections, I describe the limitations encountered during my research in more details, and go on to suggest areas for future research.

7.1 Limitations

Most of the limitations in our research arose from the nature of the research project itself: it was an industrial research project. Our industrial partner had direct control over the way in which the research was conducted, and their main objective was to find a quick technical

solution to their problems with no emphasis on an empirical validation of the proposed solutions. For example, when we were selecting the SPL management tool in order to conduct the experiment, we proposed more than 40 different tools; however, the list shrank to two tools following our industrial partner's intervention, with no quantitative evaluation of more than 30 other tools. This was because they wanted to focus on commercially available tools.

The generalizability and confirmability of our results are also limited, because the study was conducted at Esterline CMC Electronics on FMS equipment only and access to source artifacts (e.g. CMA-9000 FMS certification archives) is limited to authorized people only.

Another limitation was the lack of validation with export regulation authorities of the use of design pattern solutions to facilitate reusability among controlled and non-controlled variants of an avionics software system. During this research project, we did not have access to export regulation authorities: therefore we validated our solution with several participants who had previously been involved in the process of obtaining authorization from the export regulation authorities.

7.2 Future Work

DO-178C proposes a set of supplemental guidelines in order to use formal methods (DO-333), model-based development (DO-331) and object-oriented technologies (DO-332). The focus of my thesis was on model-based development and object-oriented paradigms. It would be interesting to continue this research with application of formal methods for certified FMS software implementation.

In my thesis, I proposed a model-based SPLE process, using a feature-oriented software design paradigm for certified and export-controlled avionics software development; the main focus was on feature implementation and validation. As a next step, feature interactions in the CMA-9000 FMS legacy source code should be studied in more detail, using the feature mapping tool that we developed during my thesis. This feature mapping tool is described in [12] and [13]. The findings from this future study may then be used to refine or improve my model-based SPLE process.

As described in [50, 48, 53], the concept of feature interactions is a major challenge for feature-oriented software development, and a complete, feature-based SPLE solution requires that feature interaction issues be handled beforehand.

BIBLIOGRAPHY

[1] D. Beuche and M. Dalgarno, "Software product line engineering with feature models," *Overload Journal*, vol. 78, pp. 5–8, 2007.

[2] P. Brooker, "Sesar and nextgen: investing in new paradigms," *Journal of navigation*, vol. 61, no. 02, pp. 195–208, 2008.

[3] C. E. Esterline, "http://www.esterline.com/avionicssystems/en-us/productsservices/navigationampfmsgps.aspx," 2016.

[4] R. DO, "178," *Software considerations in airborne systems and equipment certification. RTCA and EUROCAE*, 2011.

[5] J. Rushby, "New challenges in certification for aircraft software," in *Proceedings of the ninth ACM international conference on Embedded software*. ACM, 2011, pp. 211–218.

[6] R. Flores, C. Krueger, and P. Clements, "Mega-scale product line engineering at general motors," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*. ACM, 2012, pp. 259–268.

[7] A. Rashid, J.-C. Royer, and A. Rummler, *Aspect-Oriented, Model-Driven Software Product Lines: The AMPLE Way*. Cambridge University Press, 2011.

[8] E. Thomas, "Certification cost estimates for future communication radio platforms," Rockwell Collins France, Tech. Rep., 2009.

[9] F. A. Administration, "http://www.faa.gov/nextgen/," 2016.

[10] J. Delange, J. J. Hudak, W. R. Nichols, J. McHale, and M.-Y. Nam, "Evaluating and mitigating the impact of complexity in software models," 2015.

[11] N. Sozen and E. Merlo, "Adapting software product lines for complex certifiable avionics software," in *Product Line Approaches in Software Engineering (PLEASE), 2012 3rd International Workshop on*. IEEE, 2012, pp. 21–24.

[12] M. Ouellet, E. Merlo, N. Sozen, and M. Gagnon, "Locating features in dynamically configured avionics software," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 1453–1454.

[13] M. Ouellet, F. Gauthier, E. Merlo, N. Sozen, and M. Gagnon, "Mapping features to source code in dynamically configured avionics software," 2012.

[14] A. Advisory Circular, "Ac 20-148," *Reusable Software Components, US Department of Transportation, Federal Aviation Administration*, 2004.

[15] F. Truyen, "The fast guide to model driven architecture the basics of model driven architecture," *Cephas Consulting Corp*, 2006.

[16] M. Richters and M. Gogolla, "Validating uml models and ocl constraints," in *International Conference on the Unified Modeling Language*. Springer, 2000, pp. 265–277.

[17] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[18] H. Gomaa, *Designing software product lines with UML: from use cases to pattern-based software architectures*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 2004.

[19] H. Gomaa and M. Shin, "Automated software product line engineering and product derivation," 2007.

[20] T. Ziadi and J. Jézéquel, "Software product line engineering with the UML: Deriving products," *Software Product Lines*, pp. 557–588, 2006.

[21] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, and C.-M. U. P. P. S. E. INST., *Feature-oriented domain analysis (FODA) feasibility study*. Citeseer, 1990.

[22] K. Kang, M. Kim, J. Lee, and B. Kim, "Feature-oriented re-engineering of legacy systems into product line assets–a case study," *Software Product Lines*, pp. 45–56, 2005.

[23] V. Vraniæ and J. Šnirc, "Integrating feature modeling into uml," in *Proc. of NODe 2006*, ser. LNI P-88, R. Hirschfeld *et al.*, Eds. Erfurt, Germany: GI, Sep. 2006, pp. 3–15.

[24] A. Kleppe, J. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[25] V. Wiels, R. Delmas, D. Doose, P.-L. Garoche, J. Cazin, and G. Durrieu, "Formal verification of critical aerospace software," *AerospaceLab*, no. 4, pp. p–1, 2012.

[26] J.-L. Dufour, B. Corruble, and B. Tavernier, "The Unified Model-Based Design: how not to choose between Scade and Simulink," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[27] A. Le Guennec and B. Dion, "Bridging UML and safety-critical software development environments," in *Int. Conf. on Embedded and Real-Time Software, ERTS*, 2006.

[28] S. Burmester, H. Giese, M. Hirsch, D. Schilling, and M. Tichy, "The fujaba real-time tool suite: model-driven development of safety-critical, real-time systems," in *Proceedings of the 27th international conference on Software engineering*. ACM, 2005, pp. 670–671.

[29] H. N. Institute, "http://www.mechatronicuml.org," 2016.

[30] T. Ziadi, L. Hélou
"et, and J. Jézéquel, "Towards a UML profile for software product lines," *Software Product-Family Engineering*, pp. 129–139, 2004.

[31] K. Lee, K. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," *Software Reuse: Methods, Techniques, and Tools*, pp. 62–77, 2002.

[32] K. Kang, J. Lee, and P. Donohoe, "Feature-oriented product line engineering," *Software, IEEE*, vol. 19, no. 4, pp. 58–65, 2002.

[33] K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants," in *Generative Programming and Component Engineering*. Springer, 2005, pp. 422–437.

[34] K. Kim, H. Kim, and W. Kim, "Building Software Product Line from the Legacy Systems Experience in the Digital Audio and Video Domain," in *Proceedings of the 11th International Software Product Line Conference*. IEEE Computer Society, 2007, pp. 171–180.

[35] Y. Xue, Z. Xing, and S. Jarzabek, "Understanding feature evolution in a family of product variants." WCRE, 2010.

[36] Z. Xing, "GenericDiff: A general framework for model comparison," Technical Report, NUS, 2009, Tech. Rep.

[37] W. T. Force, "State-of-the-art survey for product lines," CESAR, Tech. Rep., 2009.

[38] I. Habli and T. Kelly, "Challenges of establishing a software product line for an aerospace engine monitoring system," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*. IEEE, 2007, pp. 193–202.

[39] F. Dordowsky and W. Hipp, "Adopting software product line principles to manage software variants in a complex avionics system," in *Proceedings of the 13th International Software Product Line Conference*. Carnegie Mellon University, 2009, pp. 265–274.

[40] F. Dordowsky, R. Bridges, and H. Tschope, "Implementing a software product line for a complex avionics system," in *Software Product Line Conference (SPLC), 2011 15th International*. IEEE, 2011, pp. 241–250.

[41] A. Wölfl, N. Siegmund, S. Apel, H. Kosch, J. Krautlager, and G. Weber-Urbina, "Generating qualifiable avionics software: An experience report (e)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 726–736.

[42] R. DO, "332 object-oriented technology and related techniques supplement to do-178c and do-278a," 2011.

[43] ——, "331:" model-based development and verification supplement to do-178c and do-278a"," 2011.

[44] D. E. Perry, S. E. Sim, and S. M. Easterbrook, "Case studies for software engineers," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. IEEE, 2004, pp. 736–738.

[45] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311.

[46] S. Apel, T. Leich, M. Rosenmüller, and G. Saake, *FeatureC++: Feature Oriented and Aspect Oriented Programming in C+*. Citeseer, 2005.

[47] R. T. V. Braga, O. T. Junior, K. R. C. Branco, L. D. O. Neris, and J. Lee, "Adapting a software product line engineering process for certifying safety critical embedded systems," in *Computer Safety, Reliability, and Security*. Springer, 2012, pp. 352–363.

[48] S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[49] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994, pp. 94–101.

[50] T. F. Bowen, F. Dworack, C.-H. Chow, N. Griffeth, G. E. Herman, and Y.-J. Lin, "The feature interaction problem in telecommunications systems," in *Software Engineering for Telecommunication Switching Systems, 1989. SETSS 89., Seventh International Conference on*. IET, 1989, pp. 59–62.

[51] C. Prehofer, "Feature-oriented programming: A fresh look at objects," in *ECOOP'97—Object-Oriented Programming*. Springer, 1997, pp. 419–443.

[52] J. Liu, D. S. Batory, and S. Nedunuri, "Modeling interactions in feature oriented software designs." in *FIW*, 2005, pp. 178–197.

[53] C. Kästner, S. Apel, M. Rosenmüller, D. Batory, G. Saake *et al.*, "On the impact of the optional feature problem: analysis and case studies," in *Proceedings of the 13th International Software Product Line Conference*. Carnegie Mellon University, 2009, pp. 181–190.

[54] S. Trujillo, D. Batory, and O. Diaz, "Feature oriented model driven development: A case study for portlets," in *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 2007, pp. 44–53.

[55] C. SC-227, *DO-236C Minimum aviation system performance standards: Required navigation performance for area navigation*. RTCA, 2013.

[56] U. D. of State, "https://findit.state.gov/search?utf8=2014.

[57] J. A. Pereira, K. Constantino, and E. Figueiredo, "A systematic literature review of software product line management tools," in *International Conference on Software Reuse*. Springer, 2015, pp. 73–89.

[58] C. W. Krueger, "Biglever software gears and the 3-tiered spl methodology," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 2007, pp. 844–845.

[59] O. Spinczyk and D. Beuche, "Modeling and building software product lines with eclipse," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM, 2004, pp. 18–19.

[60] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995, vol. 206.

[61] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. Kats, E. Visser, and G. Wachsmuth, *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.

[62] P. Clements, C. Krueger, J. Shepherd, and A. Winkler, "A ple-based auditing method for protecting restricted content in derived products," in *Proceedings of the 17th International Software Product Line Conference*. ACM, 2013, pp. 218–226.