# Implementing Multi-variant Avionic Systems with Software Product Lines

Frank Dordowsky
ESG Elektroniksystem- und Logistik GmbH
frank.dordowsky@esg.de

Walter Hipp
Eurocopter Deutschland GmbH
walter.hipp@eurocopter.com

**Abstract**

Eurocopter is member of the international NHIndustries (NHI) consortium that develops and produces the NH90, a medium weight multi-role helicopter. The growing number of customers and their specific mission requirements for the NH90 has led to an increasing number of functionally different helicopter variants. These variants have a large impact on the avionics system and therefore also on the software of the main computers that control it. Although the software is different for every system variant, they all share a large amount of common functions and code making it reasonable to reuse software across variants. In order to cope with the high number of software variants, the NH90 software team adopted a Software Product Line strategy. This paper describes the fundamental concepts and achievements of this strategy and discusses the potential benefits of extending the product line principles to the whole system development process. The software product line approach is also an ideal complement to the concept of Integrated Modular Avionics (IMA). On the other hand, adopting the concept of incremental certification that has been developed for IMA systems could reduce the qualification effort of product line systems when compared to traditional system development.

## 1 Introduction

### *1.1 NH90 Avionics System Architecture*

The NH90 Avionics System is decomposed into two major subsystems: the CORE System and the MISSION System [5, 6]. Each major subsystem is controlled and monitored by a redundant set of main computers, the CORE computer (*CMC*) and the MISSION computer (*MTC*) respectively. Major end items of the CORE system are the Display and Keyboard Units (DKUs) and the Multifunctional Displays (MFDs) that together form the main human-machine interface (HMI) of the NH90 helicopter. The equipment of each subsystem is connected to its main computers either via a dual redundant MIL-STD 1553 bus, via ARINC 429 lines, serial RS-485 lines, or via Ethernet lines. The main computers act as bus controllers for their respective data buses.

### *1.2 NH90 Avionics Software Architecture*

The software architecture of the CORE and MISSION computers is completely determined by the real-time framework *NSS* that is developed as a separate product [5, 6, 9]. NSS stands for *NH90 Embedded System Software* and frees the application programmer from intricate real-time programming tasks such as real time scheduling, device and I/O handling, data conversion between Ada data structures and raw I/O data, error and exception handling, redundancy management etc. The NSS is implemented on top of the *equipment software* that provides hardware related services and is furnished by the computer manufacturer (see figure 1). The *isolation layer* separates most of the software from the specialities of the hardware and equipment software.

The application components provide the mission functions and control the avionics equipment. These components are called *Operational Processing Functions* (OPFs) and vary between CORE and MISSION computer as well as between helicopter variants.

The NSS must be adapted to the specific needs of the CMC and MTC software. This adaptation is performed as specific modification of certain NSS source files, which is performed either manually or by code generation. The majority of the source code is generated (see 1.3), especially the control and data interfaces between the NSS and the OPFs.

The DKU operational software controls the page and display logic and performs format checking on

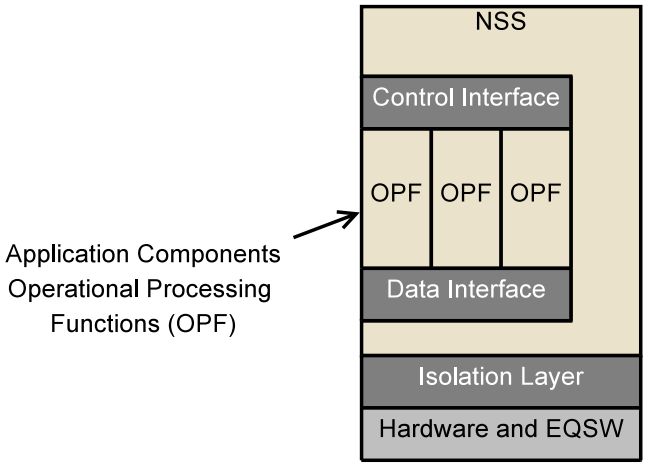


Figure 1: Software Architecture of the NH90 Main Computers

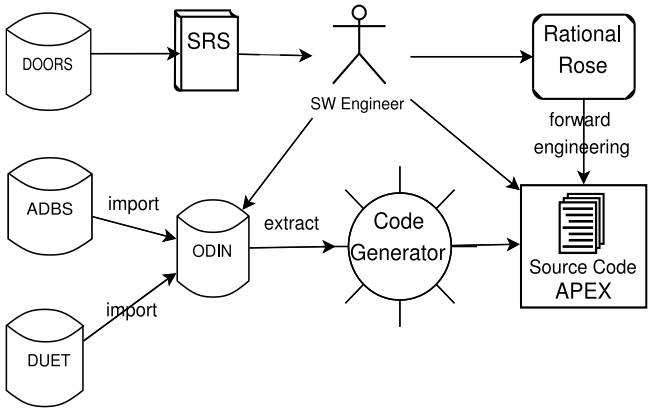


Figure 2: NH90 Development Tools (Subset)

manual data entry. It requests data to be displayed (called parameters) from either CORE or MISSION computer and sends back crew commands and their associated parameters.

The DKU software architecture is completely different to that of the main computers: all page and display logic in the DKU is table driven. These tables had been factored out into a binary format so that they can be separately loaded to the target system. The DKU tables are completely generated.

### 1.3 NH90 Software Development Process and Tools

The NH90 software development process follows the *V-model* as mandated by DOD-STD-2167A [1]. It starts with software requirements analysis, followed by preliminary and detailed design, coding, host integration and testing, and finally target integration and testing. The development of a software version terminates with successful pass of the formal qualification tests that also mark the acceptance of the software by customers and certification authorities.

The software requirements are prepared by the systems engineering department, and the qualified software is delivered to the NH90 integration rigs.

NH90 system and software development is performed with a large set of heavily customised commercial tools as well as a high number of custom developed tools [6, 9]. All these tools form a closely coupled tool chain. Figure 2 shows a subset of these tools that are relevant to this article.

The left hand side of figure 2 depicts the system engineering tools that provide the input to software development:

- The commercial tool *DOORS* is used to capture and manage system and software requirements. DOORS has no direct interface to the software engineering tools. Instead, the *Software Requirements Specification* (*SRS*) set of documents is generated out of the content of the DOORS repository.
- Interface definitions (e. g. message and signal definitions) on systems level are defined in a proprietary database called *Avionic Data Base System* (*ADBS*).
- The *DKU Interface Definition Tool* (*DUET*) contains all data necessary to describe the page logic and content of the DKU, as well as associated crew commands and parameters. This tool is also used to generate the DKU tables (not shown in figure 2).

The majority of the data kept in ADBS and DUET is imported into the *Software Generation Database Tool* (*ODIN* - OFRS Data and Interfaces of NH90). The software engineers augment this data with software related information. The content of ODIN is extracted and fed into a set of generators that produce more than 60 percent of the application source code of the main computers. Especially the virtual subsystems and the button database are completely generated out of ODIN data.

The UML modelling tool Rose from IBM/Rational was until recently used to prepare the preliminary design. The Ada package specs of the application components are forward engineered from the Rose model. Rose is currently being replaced by Topcased.

Detailed design and coding is performed in the software development environment APEX of IBM/Rational.

The *Software Design Description* (*SDD*) and the *Interface Design Description* (*IDD*) are the principal documents produced by the NH90 software team for every computer and every software version. The IDD is generated out of the ODIN data augmented by some manually written text. The SDD is a very

large, much more complex document, consisting of more than 1,000 pages. It is composed of manually written sections and of sections extracted from the Rose model and the source code.

### 1.4 NH90 Variants

The NH90 was originally planned for 4 nations (France, Germany, Italy, Netherlands) for all armed forces (sea, land, and air) so there were already 8 variants from the beginning [13]. Every new customer then increased the total number of variants. Moreover, for every contractual variant there may be more than one technical variant.

The first dramatic increase occurred in 2002 when the number of variants rose from under 10 to nearly 25. The second large raise was in 2005 (from 25 to over 40).

The helicopter variants manifest themselves in variations of the NH90 avionics system. These variations are characterised by the fact that most avionics systems functions today are related to avionics equipment and their features. Therefore, helicopter variants are determined by their equipment configurations.

It is also very common that equipment that is already integrated into one customer's variant is replaced by a functionally equivalent device which is manufactured by another customer's industry.

Complex equipment provides optional features; different customers may select different combinations of equipment features.

There is also equipment that is integrated into the CORE system for one set of helicopter variants, and into the MISSION system for another.

The most important areas that are affected by the variations listed above are:

- *Data Bus Framing*: The message traffic on the databuses of the NH90 avionics system is strictly deterministic. The message schedule is determined at design time and encoded in tables called *frames*. Helicopter variants have different equipment that leads to different messages and hence to different data bus frames for every variant. The operational software must contain the data bus frames of all variants it supports and must be able to select and activate the frame that corresponds to the actual variant.

- *MFD and DKU formats*: MFD as well as DKU page content, the page navigation logic and the associated crew commands depend on the equipment that is fitted to the actual variant.

- *Varying set of application functions*: The majority of application functions control and monitor the avionics equipment. Varying sets of equipment lead to differing sets of application functions that are related to that equipment.

- *Dependencies between application functions*: Some application functions rely on data or services provided by other application functions so that they are affected by presence or absence of those functions.

- *Different operating environments*: The original hardware architecture consists of two processor boards with Motorola 68040 processors and shared RAM, with Ada83 as the programming language together with the corresponding Ada run-time kernel. More recent operating environments are based on single processor boards and Ada95. Future operating environments may use ARINC-653 [4] conformant operating systems [5]. The operating environment has a large impact on the software, for example on the allocation of software components to processors, or on the real-time scheduling.

### 1.5 Initial Solution Attempts

The initial attempt to re-use already developed software was to copy all source code and documentation and to continue the development with independent teams in separate branches. This method, usually referred to as 'clone and own' approach, has some severe disadvantages:

- The adaptation of software of one variant to another is difficult if the software is not designed for re-use. Designing for re-use is a costly extra effort that is normally not done unless required by the customer. The adaptation usually takes longer than planned and it introduces new errors so that the clones are rather unstable.

- The software versions for the different variants must be developed and maintained in parallel, i.e. error corrections and functional enhancements must be performed in every copy which requires many developers that do the same activity at the same time. Since software development costs are driven by labour costs, the clone and own approach becomes very expensive.

## 2 The NH90 Software Product Line

In order to overcome these problems, the NH90 software needed a more sophisticated re-use strategy.

Such a sophisticated re-use approach is the software product line (SPL) strategy that has been developed in the recent years by the software development community. This strategy is especially driven by the Software Engineering Institute (SEI) of the Carnegie Mellon University [8] and it has already been transferred to avionics systems and software (e.g. the CAAS product line of Rockwell Collins [7, 11]). The NH90 software team decided to also adopt the SPL strategy and created the concept of the NH90 SPL.

### 2.1 Strategic Goals of the NH90 SPL

The original NH90 SPL concept recognised the following strategic goals that the SPL must achieve in order to be successful and accepted as future development model:

- *Functional extensions*: It must be possible to incorporate future functional features with no or minimal effort.
- *Future variants*: It must be possible to produce all current and all future helicopter variants out of a single asset base with no or minimum adaptation to application code.
- *Non-disclosure principle & customer relevance principle*: Every customer shall only receive exactly the source code and documentation that covers the features of the customer's helicopter variants.
- *Investment protection*: It shall be possible to port all existing functionality to new technology (Ada95, Ada05 and beyond, ARINC-653, etc.) with no or minimum impact on existing application code.
- *Effort and cost reduction*: The NH90 SPL shall enable the software team to support and maintain existing variants and technologies together with future ones out of a single asset base without parallel development and maintenance.
- *Migration to standards*: The NH90 SPL shall facilitate the adoption of existing and emerging standards such as ARINC-653 in order to satisfy future customer expectations with respect to these standards.

### 2.2 Learning from Industrial Experience

A software product line is basically the transfer of industrial production techniques to software development in order to plan and design for a much higher degree of re-use of already developed software. Basic concepts borrowed from industrial production techniques are:

- Software components correspond to prefabricated components. These components are used as designed and will not be modified. This is especially the case for airborne equipment that, once qualified, will not be changed. The software components shall be standardised to the highest degree possible.
- The software production plan corresponds to industrial assembly lines. There is a standardised, variant independent integration process which allows the assembly of the software components to final products. This approach makes code production predictable and thus ensures that the software team can then consistently meet the deadlines.
- Industrial production uses a high degree of automation relieving staff from tedious repetitive activities and increasing efficiency and accuracy. The equivalent of automation for software production is code generation (also called auto coding). Within the NH90 software, about 60 percent of the code is auto coded.

The NH90 SPL strategy is to take on these industrial concepts. At completion of the SPL, the manual development techniques will have been replaced by the assembly of standardised components, together with a high degree of automation via auto coding.

### 2.3 Feature based Product Assembly

The NH90 components are represented by so called *assets*. Assets are software engineering artifacts such as software requirements, source code, test scripts, test data and test results, or the data in the software engineering databases ADBS, DUET and ODIN that are used for code generation.

All assets are kept in a central *asset base* that is common to all variants and software versions. Error correction and development of additional functions take place in that asset base. From there, the individual software versions and variants are assembled into full products.

The mission requirements of every helicopter variant are grouped into so called *features* that form the basic building blocks of the avionics system. Every asset shall be linked to exactly one feature.

Features can be seen as high level functions or groups of functions, or as high level system properties that are visible on system contract level. As such, features are often related to complete equipment or equipment capabilities. The final helicopter

product can be seen as the basic helicopter itself and a set of customer chosen features which together allow the customer to fulfil the intended missions. This is similar to buying a car where the customer can also select a basic model that can then be configured on the basis of an accessories catalogue.

The software engineering products such as software requirements documentation, software qualification documentation, source code and executables shall no longer be engineered on a case by case and variant by variant base. Instead, they shall be *assembled* from the assets of the features that are selected for a certain helicopter configuration, as shown in figure 3. The intended assembly approach is as follows:

1. During contract negotiations, features are identified and assigned to the helicopter variant or variants that are going to be developed. These contract negotiations could be supported by a configuration tool, a tool similar to todays car configurators.

2. Several helicopter variants may be combined into a single software version that can be loaded to all supported variants. For example, a customer may order in a single contract several helicopter variants for different armed forces or mission profiles. The software version that will be produced on this contract can support all these variants.

3. For the software product, all assets that are needed to implement the supported features are selected.

4. The selected assets are assembled into the software engineering products for this software version, including full software documentation.

However, the implementation of this concept is far from trivial, especially in view of the fact that it was introduced rather late into the project.

### 2.4 Implementation of the NH90 SPL

The conception phase for the NH90 SPL concluded with the approval of the NH90 SPL master concept. According to that master plan, the NH90 software team had to modify its development processes and tools and modularise the existing assets:

- The DKU software was extended so that the DKU became 'variant aware', i.e. the DKU is now able to adapt its variant specific display logic to the detected current helicopter variant.

- The NSS was modified so that it can activate and de-activate operational functions depending on the detected current helicopter variant.

- Application software components were modularised so that its components correspond to functional features.

- The database tools and the generators were modified so that they include variant specific data, and, in the case of ODIN, allow variant specific data extraction from a single data volume.

These changes had to be introduced into the development cycle with the smallest increments possible in order to not endanger the ongoing delivery of NH90 software releases. This was not always easy to achieve since some modifications were fundamental and even rather large. A more detailed account on the concepts and their implementation can be found in [9].

The assets were not designed from scratch since that would have been far too expensive and time consuming. Instead, the software team reengineered the existing products to fit to the overall concept.

The implementation of the NH90 SPL is not completed yet. There are still some application software components that need to be modularised to get truly standardised components, and the tool improvement process is also still ongoing. The major activities in the next future are to extend the software concept to also include system and software requirements as well as qualification and test procedures. At the end, the SPL shall also support proposal and quotation preparation as well as technical documentation.

## 3 Benefits of a Product Line

According to experience reports in the literature, the benefits of a SPL begin to materialise between the second and third product instance that is derived from the SPL. This is similar to the NH90 SPL where the first export variants have recently been assembled from the asset base. The major benefits that are now being realised are:

- Increased planning reliability since assembly is less risky and can be completed within a fixed timeframe, as compared to single product development.

- Decreased development budget because only one person or a small group is responsible for a component instead of several persons that independently develop the same component for

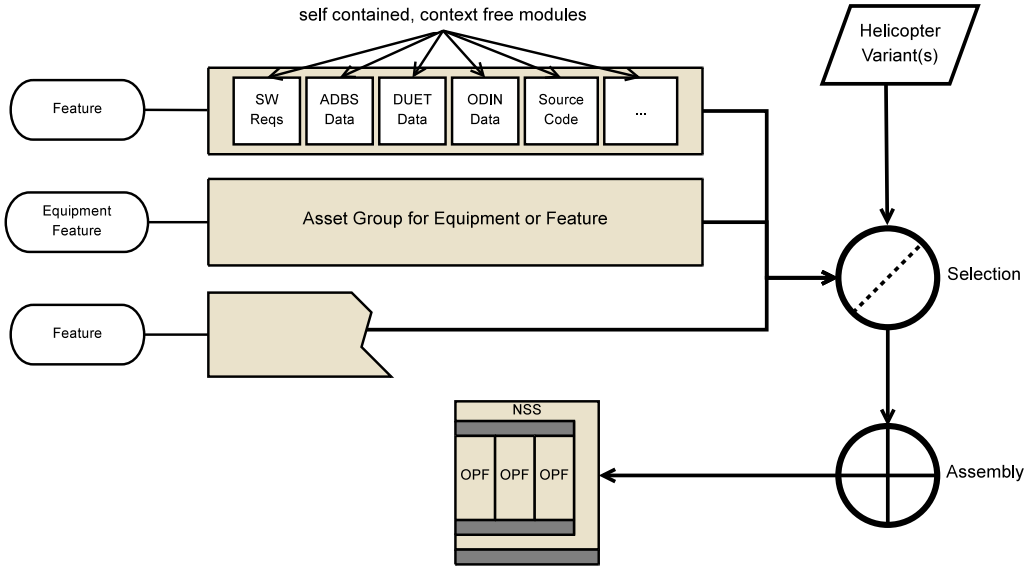


Figure 3: Feature based Selection and Assembly

several product variants. For example, errors are fixed in a component once for all product variants instead of several times for every product variant individually.

- Components are used in several product variants and become therefore more stable in shorter time. This will lead to earlier stabilisation of the software and to a much higher degree of quality which in turn will increase customer satisfaction.

- Features can be added or removed in much shorter time during development which allows reacting in short time to customer change requests.

- It will be easier to isolate a single customer variant for maintenance.

These benefits materialise in the reduction of effort for creation and maintenance of product variants: the effort for updating a product instance is negligible by now, and the effort for the creation of a new product instance has been reduced to a quarter of the original effort without the SPL.

# 4 Extending Software Product Line Principles to System Development

The NH90 SPL concept is currently mainly implemented in the area of detailed design and coding. In order to maximise the profit from the product line, it would be meaningful to extend the scope to the complete system development life cycle (V-model),

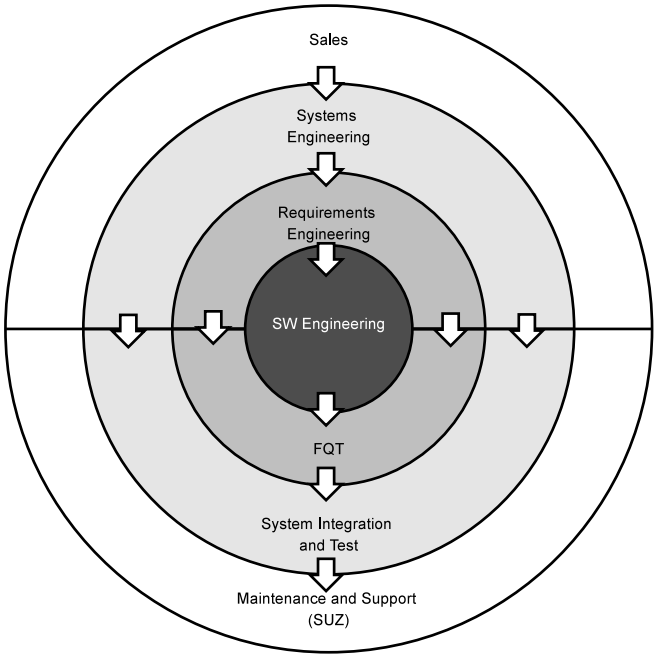


Figure 4: Extending the SPL to Systems Engineering

i.e. including system and software requirements engineering as well as formal qualification and rig tests.

The overall long-term vision is shown in figure 4. The development chain starts with the sales department who defines the mission features and agrees them with the customer. System Engineering takes over and defines the system architecture, followed by requirements engineering that develops the input to software engineering at the centre of the diagram.

Variant information is needed on all levels of the V-model. Today, this type of information is maintained individually in every tool or document and thus a potential source of inconsistencies. There will be very likely the need to introduce a configu-

ration tool to manage all features, variants, kits and options centrally. There is already a proposal for such a tool with the concept name *SCORE* (*Shared NH90 COnfiguration REpository*). SCORE would also maintain the relation between features and helicopter variants in a configuration matrix.

In a first step, SCORE could be set up with the feature information currently defined in the code generation database ODIN, extended by information about feature characteristics and dependencies. Then, all processes and tools along the V-model could gradually be transferred to use SCORE as a single source for variant information. During this process, inconsistencies will come up that can be clarified and that will also lead to a refinement of the centrally defined variant information.

For the MISSION computer, some redesign activities are planned that will include a reorganisation of the NH90 software requirements. Currently, several ways of variant handling are used in parallel. For example, it is allowed to clone requirement sets for certain variants and then to modify single requirements as applicable. Such an approach limits the consistent modularisation of the system because cloned and modified software requirements can no longer be implemented as an invariant building block. This approach should therefore not be accepted in the future, and requirements should instead be grouped into modules that are invariant against helicopter configurations.

Furthermore, some activities have been started to improve the change management process with respect to helicopter variants. Currently changes are tracked against individual variants. However, a change usually affects many variants at once and the implementation of the change is also valid for many variants. It would be much more efficient to track changes against features and related assets. The new change management process will therefore contain the possibility to assign problem reports and change requests to features. The configuration matrix that associates features with the helicopter variants then automatically provides the information on which variants are affected by the problem report or change request.

# 5 Product Lines and Integrated Modular Avionics

Integrated modular avionics (IMA) systems are being installed on a number of new aircraft. The goal of IMA systems is to reduce the number of LRUs on an aircraft in order to decrease space, weight and power, to reduce spares holding and to cut down maintenance and operating costs.

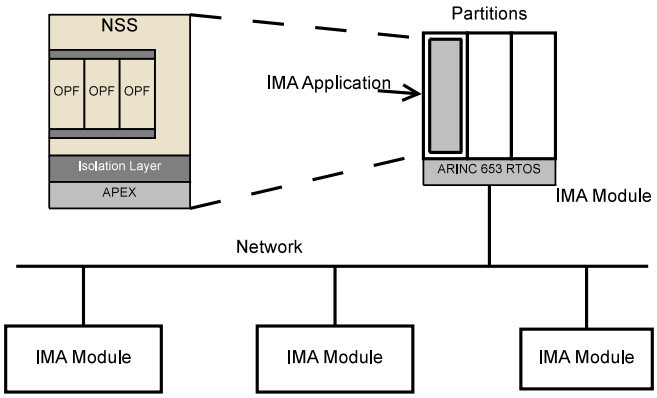


Figure 5: IMA System with NH90 SPL

IMA systems are built from an *IMA platform* and a number of *applications* that execute on the platform. The platform consists of a number of hardware nodes called *modules*, a network that connects the modules, and a real-time operating system (RTOS) that provides communication, computing and memory resources to the applications. The applications access the platform services via a standardised application programming interface that is named APEX and is defined in the ARINC 653 standard [4]. One of the main characteristics required by ARINC 653 is the provision of strict memory and time partitioning to support functional isolation.

Product lines and especially the NH90 SPL form an ideal complement to IMA systems because (1) they allow a flexible assembly and re-assembly of the applications based on features, and (2) they facilitate the migration of todays federated systems towards IMA systems.

Decomposing a large avionics system into such isolated applications reduces complexity and simplifies development, maintenance and qualification activities. For IMA systems it is crucial to decide which parts of the system shall be realised by which application. This decision is based on standard architectural considerations like cohesion and coupling but also on aspects like expected change rate, planned functional extensions and replacements or project management decisions including supplier selection. Applications shall ideally be designed in a way that they can be replaced by alternative implementations of the same functionality without modifications to the rest of the system.

Software product lines basically decompose the system into individual components that implement a certain feature. Together with a framework such as the NSS with its standardised interfaces, the components do not make any assumption on the context within which they will be used and thus can be assembled freely. IMA applications can conceptually be regarded as a realisation of a set of features im-

plemented with the corresponding set of SPL components (see figure 5). The ideal combination of features into applications with respect to design criteria such as coupling and cohesion can be determined from the feature model if this maintains information about dependencies between the features. Other criteria such as implementation or qualification status of the features could also be derived from a suitable feature model.

The migration of the NH90 main computer software to an ARINC 653 compatible RTOS is facilitated by the fact that the NSS isolates the application functions from the underlying hardware and equipment software. Only the isolation layer of the NSS and the code generators for the control and data interface must be adapted to the new RTOS.

One of the major tasks of an IMA systems integrator is to negotiate the resource requirements between application developer and platform provider. The feature model of an SPL, when extended by the resource requirements of each feature, can then be used to determine the resource requirements of the applications that implement these features.

Building a complete IMA system imposes some additional requirements to the product line because it will be necessary to model how partitions work together: a classic feature model assumes that the components are assembled to a single application but for IMA systems one must distinguish between intra-application and inter-application dependencies. This can be regarded as an extension of the feature model. Automatic consistency based on the extended feature model can check both the integrity of each application with respect to included features and the complete IMA system with respect to deployed applications.

It may happen that a set of applications hosted on a module exceeds that modules resources. Often it is not possible to simply add additional modules due to space and weight limitations. In that case it may be necessary to reallocate the features to a different set of applications. With the generation technology that has been developed for the NH90 SPL it would be possible to simply restructure the allocation of features to applications and to re-assemble them into a new set of applications that better fits to the available platform.

## 6 Incremental Certification

Simply reusing source code will not realise the desired cost benefits of a product line since coding represents only a minor fraction in the avionics software development effort (According to Capers Jones [10], coding represents only 16 percent of the total software development effort in military projects). One of the major cost drivers for avionics projects is the certification of the system. Therefore, most savings could be realised if it would be possible to obtain certification credit for single components that can then be used in subsequent certification efforts. Unfortunately, software reuse has acquainted a bad reputation in safety critical systems. The spectacular failure of the ARIANE 5 first flight for example has been attributed to software reuse malpractice [12].

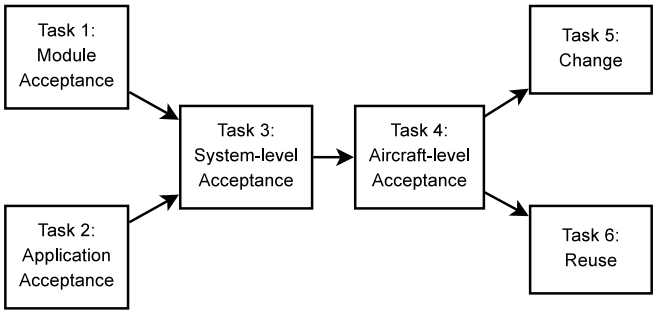


Figure 6: DO-297 Certification Tasks

Because of these concerns, the FAA has developed the advisory circular AC 20-148 [2] that provides guidance for obtaining certification credit for reusable software components (RSC). However, this advisory circular is not exactly applicable to the NH90 SPL since it aims more towards closed commercial RSCs such as operating systems, provided by different vendors to a wide range of systems that are not necessarily restricted to the avionics domain. A certification strategy that is more suited to software product lines is an approach known as *incremental certification* that has been developed for IMA systems.

The RTCA DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations [3] provide guidance for applicants and authorities for the incremental certification of IMA systems. It defines six certification tasks: (1) module or platform acceptance, (2) application acceptance, (3) system-level acceptance, (4) aircraft-level acceptance, (5) change and (6) reuse. Figure 6, taken from [3], illustrates these tasks.

A comparison between NH90 SPL artefacts and IMA elements shows that it may be possible to adopt a similar strategy for the NH90 software product line:

- The NSS plays the role of the IMA platform (the NSS is separately qualified, so this is already a first step towards incremental certification).
- The feature related components compare to the IMA applications.

- The IMA system integration is equivalent to product assembly and integration including HW/SW integration.
- Aircraft integration is represented by system integration, starting from avionics rig integration of the main computers with the equipment and ending with integration into the helicopter.

This approach has not yet been developed into a detail that would allow it to be introduced into the project, and it has not yet been discussed with certification authorities. But even if certification credit cannot be obtained for single components in a way similar to that outlined above, it will still be possible to reduce certification and qualification costs by reusing the life cycle data that are necessary to obtain acceptance. This includes requirements, design descriptions and models, source code, test cases, test procedures and test results, quality and configuration management records, and all other sort of validation and verification data. This of course requires that the functionality of every single feature remains unchanged across variants or at least is adapted only via well established variation points, based on parameterisation or configuration for example. The reviews and analysis that still needs to be performed can then concentrate on the variation points and will therefore be much less intensive and time consuming as the original review effort. This of course only works if the variation points of the product line have initially been verified.

However, two critical areas need further investigation:

- Features are not unrelated to each other, some features depend on the availability of another features, or features may even be exclusive.
- The NH90 components compete for common resources such as processing time, memory space and data bus cycle time.

The first issue can be addressed by maintaining these feature relations in the configuration tool SCORE that has been introduced in section 4 above. The latter issue must still be verified during integration of the product instance since the NSS does not provide strict partitioning of the components.

The authors believe that with the certification guidance that is already available for reusable software and IMA systems, product lines have a larger potential for cost reduction than any other development technique for large avionic systems. However, it requires that also system engineering data such as the system and software requirements are structured and organised in accordance with product line principles. This means that in order to realise the full benefit of the product line approach, it is necessary to extend the NH90 SPL principles to systems engineering as described in section 4.

## 7 Summary

Although the benefits of product lines are well known in the field, only few real-world examples exist in the domain of avionics [11]. The only other example of a real-world airborne avionics product line known to the authors is the U.S. Army's Common Avionics Architecture System (CAAS) Product Line [7]. It has been reported that its application to the Bell ARH-70A program allowed to complete software design, development and test within seven months after the contract agreement was signed [11].

One of Eurocopter's key competitive advantages is the ability to produce highly customer specific helicopters that in return enable the customers to fulfil their specific mission needs. The Software Product Line strongly supports this company strategy as it enables flexible combination of common modules in a short production time and at reasonable cost. Moreover, it also provides a suitable migration path to a modern IMA system. The NH90 software team members are convinced that the SPL strategy is a model for future software development approaches and is not restricted to NH90 avionics – the existing product line could be extended to future helicopter avionics, especially when ported to an IMA platform. A consistently high degree of software re-use would increase Eurocopter's competitiveness in the future avionics system software market.

## References


[1] DOD-STD-2167A: Military Standard Defense System Software Development, 1988.

[2] Advisory Circular AC 20-148 Reusable Software Components, December 2004.

[3] RTCA DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations, November 2005.

[4] Arinc 653 Avionics Application Software Standard Interface. PART 1 – Required Services, March 2006.

[5] Richard Bridges. NH90 helicopter avionics systems from the 1990s to 2010 and beyond. In *Workshop Software-Architekturen für Onboardsysteme in der Luft- und Raumfahrt*. Fachausschuss T6.4 Software Engineering, Deutsche Gesellschaft fuer Luft- und Raumfahrt, Oct 2007.

[6] Gerd Budich. Generation of Ada code from specifications for NH90 computers. In *Proceedings of the 26th European Rotorcraft Forum ERF*, 26–29 September 2000.

[7] Paul Clements and John Bergey. The U.S. Army's Common Avionics Architecture System (CAAS) Product Line: A Case Study. Technical Report CMU/SEI-2005-TR-019, ESC-TR-2005-019, Software Engineering Institute (SEI), September 2005.

[8] Paul Clements and Linda Northrop. *Software Product Lines - Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2002.

[9] Frank Dordowsky and Walter Hipp. Adopting software product line principles to manage software variants in a complex avionics system. In John D. McGregor and Dirk Muthig, editors, *Proceedings of the 13th International Software Product Line Conference, San Francisco, California, USA 2009*, volume 1. Software Engineering Institute, August 2009.

[10] Capers Jones. Software cost estimation in 2002. *CrossTalk: The Journal of Defense Software Engineering*, pages 4–8, June 2002.

[11] Ronald J. Leach. Missed opportunities in software cost reduction. In *Proceedings of the 64th Forum of the Aeronautic Helicopter Society AHS*, 2008.

[12] Jacques-Louis Lions. Ariane 5 fligth 501 failure. report by the inquiry board. Technical report, Ariane 501 Inquirey Board, July 1996.

[13] Stefan Steyer. NH90 Beschaffungsdrama. Die Entwicklung des NH90. *Rotorblatt*, 16(3):52–55, 2009.