Abstract
The introduction of a software product line may pose a great organizational challenge in the domain of high-integrity systems. Project and technical managers within an organization need to be assured that the reusable assets of a product line are reliable and trustworthy, particularly when project teams do not have full control over the development of these assets. In this paper we report on our experience with the establishment of a software product line for an aerospace Engine Monitoring Unit (EMU). Specifically, we report on challenges encountered with the configuration management and certification of EMU products derived from the product line. These two areas are still to be addressed adequately by the product line community as they are central for the management of product line assets across different projects within an organization.

1. Introduction
The achievement of an acceptable level of system integrity is traditionally associated with the application of rigorous and expensive methods and techniques. McDermaid estimates that a verified line of code (LoC) in high-integrity system development costs approximately $150 to $250 [1]. On average, the complete development of a high-integrity software of 100 kLoC costs $25M. The most expensive aspect is the realisation and verification of dependability qualities such as reliability, availability, assurance and performance. Such qualities require specialised techniques, processes, and personnel [2].

Software reuse is a sensible approach to reducing the development cost of high-integrity software. Instead of analysing, designing, implementing, and verifying a piece of software from scratch every time, it would be more cost-effective to develop the software once and reuse it in multiple systems. Specifically, reuse could be more cost-effective if it is managed according to predefined contextual and architectural constraints such as in product line development. Product line development is an approach to large-scale and holistic reuse. A product line can offer the reuse of complete lifecycle artefacts such as process plans, requirements, design rationale, analysis models and test cases. The reuse of these artefacts is constrained by a predefined set of variation rules.

In this paper, we consider whether it is feasible to reduce the costs of high-integrity software development using a product line approach without compromising dependability. Specifically, we focus on the challenges of configuration management and certification of software systems developed as part of a product line. These areas are addressed in the context of our experience with the establishment of a software product line for an aerospace engine monitoring Unit (EMU).

The rest of this paper is structured as follows. Section 2 presents some concerns of the system dependability community with regard to software reuse. Section 3 provides a brief introduction about the domain of the product line discussed in this paper, namely the EMU of gas turbine engines. Section 4
summarises the approach we have adopted for the establishment of the EMU product line. Sections 5, 6, 7 focus on three product derivation sub-processes: product derivation planning, configuration management, and certification. Section 8 discusses the advantages and limitations of the EMU product line and future research work.

2. Software Reuse and Dependability

Software reuse has a bad reputation in the high-integrity systems domain. According to Leveson [3], several billions of dollars and many critical missions have been lost because of poor spacecraft software reuse at the National Aeronautics and Space Administration (NASA), the European Space Agency (ESA), and the US Air Force. One of the most cited examples of a software reuse error is the failure of the ARIANE 5 launcher [4]. The launcher performed reliably in the first 37 seconds after countdown, after which “it suddenly veered off its flight path, broke up, and exploded” [4].

The software had been reused from ARIANE 4 and contained features not needed for ARIANE 5. It was assumed that, since these features had worked well in ARIANE 4, they would not pose any risk in ARIANE 5 [4]:

“This realignment function, which does not serve any purpose on ARIANE 5, was nevertheless retained for commonality reasons and allowed, as in ARIANE 4, to operate for approx. 40 seconds after lift-off.”

However, ARIANE 5 had a higher initial acceleration and a different trajectory than ARIANE 4 did which produced an excessive value that caused the “computers to cease operation” and consequently “the complete loss of guidance and attitude information”.

The main argument against software reuse in the dependability domain is based on the observation that dependability is a context-specific and a complete system attribute. Redmill stresses the danger of context and design mismatches by noting that even trivial variation in design between the new and old applications may have great implication on the dependability and trustworthiness of the entire system [5].

Leveson in [6] elaborates on the above statement by claiming that reuse may even decrease system integrity as the potential failures of the new system might have not been considered in the original analysis of the reused software.

Despite the firm position taken by Redmill and Leveson with regard to the impact of software reuse, they both acknowledge the importance of getting “the benefits of reuse without the drawbacks” [3] through “an increasing need for care (and guidance)” [5]. We believe that reuse can be more systematic if it is planned from early development stages. Establishing a traceable link between the requirements, design, and the reusable code can enable the deployment of the reusable software in an effective and analysable way. Therefore, reuse should be addressed holistically, rather than being limited to the code, where most reuse attempts have failed [3].

Being a holistic approach to systematic and large-scale reuse, software product line development can address some of the concerns raised by the dependability community in regard to reuse. A product line comprises a flexible and reconfigurable architecture and a set of reusable core assets [7]. New software products can be derived from the product line’s architecture and assets based on a predefined process that manages and controls permitted variation. Principally, variation in a product line is not an indication that the development is unstable. Instead, it indicates that differences between products are predictable and controlled. Product line development is an integrated approach in the sense that any development artefacts, such as requirements, design rationale, analysis models or test cases, can be reused as long as they adhere to the context, architectural constraints and variation rules defined in the product line [11]. In summary, dependability is a complete system attribute and so is reuse in product lines. Product line reuse targets the software system in its entirety rather than as a collection of pieces of software and hence has the potential to meet some of the expectations of the dependability community.

3. Aerospace Engine Monitoring Units

This section provides brief contextual and domain information about the software product line discussed in this paper, namely the Engine Monitoring Unit of aircraft gas turbine engines.

Civil aircraft are mostly powered by gas turbine engines. In simple terms, a gas turbine engine works as follows (Figure 1). It absorbs a sheer volume of air through a fan and a compressor. The air is compressed up to 40 times atmospheric pressure after which it is mixed with fuel and heated in a combustion chamber at a very high temperature (up to 2000°C). Finally, thrust is provided by forcing the expanded gas through the turbine.
Modern aircraft gas turbine engines are controlled by a Full Authority Digital Electronics Control system (FADEC) which is a high-integrity computer system that controls the operation of the engine. The FADEC relieves the pilots of the burden of monitoring and controlling all performance parameters of the engine. Some of these parameters monitor engine parts such as rotators, blades and shafts [8]. These parameters provide warning to the pilot and the maintenance team so that actions can be taken to protect the engine against costly maintenance.

Specifically, health parameters are monitored by an Engine Monitoring Unit (EMU). An EMU is an engine mounted unit comprising two processing modules (Figure 2):

- Signal Processing Module (SPM)
- Main Processing Module (MPM)

The SPM collects data from various monitoring sensors. It then transmits such data to the MPM for further specialised processing and analysis. The MPM comprises two software components: Operating Software and Application Software. The Application Software performs three main functions: (1) Reading and calculation, (2) analysis and (3) reporting.

The incentive for an EMU product line arose during the development of two different EMU software systems for two different engines. We discovered a large number of common engine monitoring functions and attributes between the two engines. After a preliminary commonality and variability analysis, it was decided to develop a software product line to support the development of the two EMUs in addition to other on-going and future EMU software developments.

However, in order to justify funding of the development it is necessary to demonstrate cost effectiveness. This was achieved by ensuring that the reusable artefacts of the product line cover a relatively large amount of work. In other words, any new EMU product that is developed as part of the product line should be composed of large-scale reusable artefacts, i.e. not fine-grained, in order to reduce integration and testing effort. This is particularly important as the ‘embedding’ control system is not developed as part of a product line. We believe that a detailed commonality and variability analysis of fine grained artefacts may deter system and software engineers from reusing the product line artefacts as it is likely to be seen as a labour intensive activity.

Therefore, to be successful, the new EMU product line should integrate well with the development process and terminology of the control system as currently implemented in the company Control Systems Engineering Division. Ideally, a top down approach to product line development would be desirable, i.e. starting from the control system level. However, as later discussed in Section 8, the cost and integrity of a high-integrity system such as the engine controller enforce different and more rigorous constraints than those expected for EMU software.

4. Development Approach of the EMU Product Line

The EMU software has around three hundred requirements. A typical EMU requirement is a mixture of natural language and complex mathematical equations. Our preliminary commonality and variability analysis has revealed that the mathematics behind the monitoring requirements provided by the EMU software rarely changes between different
engines. When a variation occurs, it is noticed that the variation is a result of the differences in the number and fine-tuning of the EMU functions, rather than in the implementation of these functions.

Figure 3. EMU Software Application Elements

Figure 3 shows a high-level conceptual model of EMU software. Two EMU software applications can only differ in the number of analysis functions they provide. However, if these two EMU software products provide the same analysis function, the implementation of this function is the same in terms of the underlying mathematics.

Given this neat separation between the EMU functions (i.e. calculation, analysis and reporting), it was decided to document the EMU requirements in a modular way, where each cohesive set of requirements is encapsulated in a separate requirements specification document. This has paved the way for the satisfaction of the aforementioned concern about the need for large-scale product line artefacts.

The product line repository includes the following reusable requirements sets:
- Four documents detailing requirements for calculation functions
- Nine documents detailing requirements for analysis functions
- One document detailing requirements for reporting functions
- Two data models

The product line repository also includes the design, implementation and verification artefacts, including:
- Software architecture
- Low-level design
- Test cases
- Source code
- Requirement-based test scripts
- Requirement-based test results
- Review records
- Proof review files
- Software configuration index
- Certification case

Figure 4. Product Line Repository

The traceability between the product line requirements and the rest of the reusable artefacts of the product line is well established. It is straightforward to identify the implementation and verification artefacts that realise each set of requirements. As shown in Figure 4, an integrator can extract a vertical core or optional ‘slice’ from the product line repository. The derivation of a new EMU product is carried out by composing these slices which are constrained by the product line data models and software architecture. Software functions can be added to or removed from an EMU software product with little implementation and verification effort as long as they adhere to the architectural decompositions and interfaces and use the data structures provided by product line’s data models. It is important to note that the software architecture maintain a clear separation between the EMU software subsystems and the underlying platform. Every interaction between these subsystems and the underlying platform is carried out
through an abstraction layer and hence the real identity and installation of FADEC sensors, for example, is hidden behind a generic interface. This allows more engine projects to develop their EMU software products as part of the product line.

We will not elaborate further in this paper on role of the software architecture in pulling together the reusable artefacts of the product line. Instead, we will explore the key role of the product line plans, especially the derivation plan (or what we call ‘The User Guide’), in managing and maintaining the product line assets and products.

5. Product Line User Guide

Software reuse often exists between different projects. However, it does not always take the form of a managed product line development. Hence, with the EMU product line, we are careful with the use of terminology. We call the plan for deriving new EMU products from the product line repository ‘The User Guide’. The user guide is a relatively short document which provides an overview of the product line. It also offers a set of guidelines for deriving, integrating, verifying and maintaining new EMU products developed from the product line reusable assets. The user guide provides an entry point for technical managers and architects. It is kept concise and focused through referencing other detailed software plans which are available in the product line repository, namely:

- Software Development Plan
- Software Verification Plan
- Software Configuration Management Plan
- Software Quality Assurance Plan
- Software Requirements Standards
- Software Design Standards
- Software Code Standards

The fact that complete software lifecycle artefacts are stored for reuse in the product line repository does not aim to imply that 100% reuse is achieved. The derivation, integration, validation, verification, and maintenance and the overall management of the product line products and assets are overhead activities. These activities could lead to undesirable end-results if not accounted for during initial planning.

Real-time performance is a major attribute of the EMU software. The user guide maintains information about the processor and memory usage of each reusable software subsystem. The overall latency and memory usage of any new EMU software product can be calculated using a mathematical formula and a set of spreadsheets which are provided in the product line repository. Consequently, performance engineers are relieved of the burden of estimating from scratch the processing and memory usage of new EMU software products.

The user guide is a live document as it also maintains a log of managerial and technical risks and assumptions. For example, one key assumption of the product line sets a limit on the number of hours after which the EMU software should be restarted. This is not an issue for flight software but would need to be addressed if the software was to be re-used, for example, in an industrial or marine application. The reliability of the EMU software can be compromised if such an assumption is not communicated explicitly to different projects reusing the product line assets. For example, communicating this assumption is critical if the EMU product line scope is widened in the future to include the EMU of power station engines which operate continuously over very long periods without the need to be restarted. Any change to these assumptions, triggered by changes to the product line artefacts, must be controlled and carefully managed by the product line configuration management. This has raised some major concerns: Who owns the deployed product line artefacts? Who should carry out a change to a deployed product line artefact? Is it the product line team or the project teams?

6. Product Lines and Configuration Management Challenges

The purpose of the configuration management activities is to establish and maintain the integrity of the EMU product line artefacts. This includes both the artefacts stored in the product line repository and those reused by individual projects. In particular, the aerospace software standard DO-178B [9] stresses the need for planning and implementing configuration management activities such as configuration identification, control, status accounting, and audits.

The configuration identification and versioning of individual EMU products, developed as part of the product line, are stored in the configuration management tool of the product line. It is possible to retrieve the configuration identification and version of the product artefacts incorporated into any specific project. Each individual EMU software product is traced to a schematic folder containing a set of product line artefacts that belong to a specific product line baseline (Figure 6). The configuration management of the product line has therefore two dimensions: Managing the artefacts ‘across time’ (version update) and ‘across different projects’.
The defect or enhancement is detected:
(1) during the integration, deployment or in service of the reusable artefacts of the Product Line in a specific project
(2) during the maintenance of the Product Line

Before confirming that a change is related to the Product Line artefacts, the change is reviewed by representatives of the project and the Product Line.

The problem report should have a Product Line-specific configuration identification. It should indicate the release number of the defected software artefact. If the problem is originally raised in a project, the Product Line problem report should reference the project problem report.

The validity of the Product Line change request is reviewed.

The change can be implemented by either project team or Product Line team.

Before updating the Product Line baseline, the implementation of the change is formally reviewed.

All Product Line documentation affected by the change are also updated.

These steps are specific to the project that has originally raised the problem.

Figure 5. Configuration Management Process
The EMU product line programme is owned and managed by a product line manager. The product line manager negotiates the reuse and change of the product line artefacts. However, this does not imply that a particular project, reusing the EMU product line artefacts, cannot make a change to these artefacts. Rather, the project can modify the product line artefacts as long as such modifications are reviewed and approved by the product line manager. Once a change is implemented, the product line baseline is updated. All projects reusing the modified artefacts are subsequently notified about the change. The configuration management system of the EMU product line has two different processes, namely:

- Implementation of a change to product line artefacts
- Propagation of a change into individual projects

Figure 5 shows the process for implementing a change to the product line artefacts. A problem can be detected by the product line team or by a particular project implementing the EMU software as part of the product line. If the problem is detected by a project, the project team should create a project problem report (not a product line problem report). After confirming with the product line manager that the problem is related to the product line artefacts, a product line problem report is created that references the project problem report. In many cases, the project team may insist on implementing the change themselves (if they cannot wait for the product line to implement the change, for example). In such cases, the product line manager can subcontract the implementation of the change to the project team. Once the change is implemented and verified, the associated problem report and change request are closed. The baseline of the product line is then updated and all concerned projects affected by the change are notified (of course including the project that implemented the change). These projects can either accept to update their EMU software product or postpone updating their software and stay synchronised with a previous product line baseline.

In summary, despite the need for a high degree of rigour in the management and control of the product line artefacts, the configuration management plan of the EMU product line provides a high degree of flexibility. Individual projects can carry out a change themselves as long as they coordinate it with the product line manager. Additionally, other projects can, in principle, reject updating their EMU software and stay linked to a previous baseline of the product line repository. A systematic and rigorous configuration management plan is necessary as it assures the certification authorities of the configuration integrity and consistency of an EMU software product derived from the product line.

7. Product Lines and Certification Challenges

Many of the current software certification standards are highly prescriptive (e.g. the aerospace guidance document DO-178B [9] and the generic international standard IEC 61508 [10]). They mandate or recommend a set of activities, techniques and methods that should be incorporated into the development and assessment lifecycle. There is an implicit assumption in many of the software standards that the software is developed from scratch. When software reuse is considered by a standard, it is addressed as an exception to the norm (e.g. in a ‘Special Consideration’ Section in DO-178B). The high level of prescription in software certification standards has deterred many in the dependability domain from exploiting reuse approaches such as component-based development and product line development.

To reap the full benefits of the EMU product line, the product line repository provides most of the lifecycle data required by DO-178B. This is to avoid the creation of the certification evidence from scratch for each EMU software product developed as part of the product line. However, software and system engineers tend generally to avoid the use of non-traditional approaches, e.g. reusable components or formal methods, and stick to what is prescribed by the standards in order not to prolong the certification process. Our concern for the EMU product line was whether we needed to consider new software developed as part of the product line as previously developed software, given that the product line can provide complete lifecycle plans and data.
DO-178B addresses the incorporation of Previously Developed Software (PDS) in the “Additional Consideration” Section. However, it does not provide a clear and detailed definition of PDS. Fortunately, DO-248B clarifies the PDS term as follows [4]:

“PDS is defined as software already developed for use. This encompasses a wide range of software, including commercial off-the-shelf (COTS) software through software developed to previous or current software guidance.”

Based on the abovementioned clarification, we have concluded that any EMU software product developed as part of the product line should be considered PDS regardless of availability of the entire DO-178B lifecycle data for reuse. In other words, providing the lifecycle data and documentation does not alter the fact that the software has been developed previously and has not been developed for a specific project.

Having said that, the intention to develop an EMU software product as part of the product line should be stated in the Plan for Software Aspects of Certification (PSAC). Additionally, the reusable software of the product line should satisfy the same certification objectives as the software/system of which it is part. Therefore, it is the responsibility of each individual project, developing their software as part of the product line, to certify the software (Certification authorities do not certify software components. They only certify completed systems with well-defined contexts).

Table 1 summarises the responsibilities regarding the generation of certification lifecycle data. Most of such data are provided by the product line team, including planning, development and verification (available in full for low-level verification and partially for software/hardware integration and verification). There are a number of software/hardware integration tests that need to be performed by individual projects (Figure 7). Although guidance on the execution of these tests is provided by the product line team, the actual work should be carried out in the context of a specific platform environment. Once the certification package is completed for any EMU software product, it is the responsibility of the project team to liaise with the certification authorities.

<table>
<thead>
<tr>
<th>Table 1. Producing Lifecycle Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DO178B Lifecycle Data</strong></td>
</tr>
<tr>
<td>Plan for Software Aspects for Certification</td>
</tr>
<tr>
<td>Software Accomplishment Summary (SAS)</td>
</tr>
<tr>
<td>Software Development Plan</td>
</tr>
<tr>
<td>Software Verification Plan</td>
</tr>
<tr>
<td>Software Configuration Management Plan</td>
</tr>
<tr>
<td>Software Quality Assurance Plan</td>
</tr>
<tr>
<td>Software Requirements Standards</td>
</tr>
<tr>
<td>Software Design Standards</td>
</tr>
<tr>
<td>Software Code Standards</td>
</tr>
<tr>
<td>Software Requirements Data</td>
</tr>
<tr>
<td>Design Description</td>
</tr>
<tr>
<td>Source Code</td>
</tr>
<tr>
<td>Executable Object Code</td>
</tr>
<tr>
<td>Software Verification Cases and Procedures</td>
</tr>
<tr>
<td>Software Verification Results</td>
</tr>
<tr>
<td>Software Life Cycle Environment Configuration Index</td>
</tr>
<tr>
<td>Software Configuration Index</td>
</tr>
<tr>
<td>Problem Reports</td>
</tr>
<tr>
<td>Software Configuration Management Records</td>
</tr>
<tr>
<td>Software Quality Assurance Records</td>
</tr>
</tbody>
</table>

Figure 7. The V-Model of Product Derivation

Table 1 summarises the responsibilities regarding the generation of certification lifecycle data. Most of such data are provided by the product line team, including planning, development and verification (available in full for low-level verification and partially for software/hardware integration and verification). There are a number of software/hardware integration tests that need to be performed by individual projects (Figure 7). Although guidance on the execution of these tests is provided by the product line team, the actual work should be carried out in the context of a specific platform environment. Once the certification package is completed for any EMU software product, it is the responsibility of the project team to liaise with the certification authorities.

In short, product line artefacts should ideally be certified once and reused in different products without further certification effort. However, in practice, the certification authorities consider only completed
8. Discussion and Conclusions

In this paper we have deliberately focused on configuration management and certification as their impact is sometimes underestimated in the overall product line lifecycle. Even if a product is derived from a well-defined product line architecture with constrained variable points, flaws in the configuration consistency can weaken any claims about the integrity of the product. In the EMU product line, we had to define a configuration management plan which could integrate smoothly and systematically with the configuration management system of individual projects. We believe that the shared responsibility for the maintenance of the product line assets will result in great benefit as any EMU software product can take advantage of any problem fixing or in-service experience of the rest of the EMU software products.

The issue of certification seems to pose a greater challenge as it involves external stakeholders, namely the certification authorities. More worrying, it is constrained by highly prescriptive software certification standards that lack a clear published rationale. To overcome this, software engineers have to justify that the product line approach satisfies certification requirements, for example, by showing that the product line process and that recommended by the certification standards are equivalent. This is complicated particularly for product line development. Product line development relies on variable and discontinuous processes, underlying core asset development, that are only integrated at the time of development. Process discontinuity may complicate further the already challenging integration between the system development process and the analysis process (especially for analysing qualities such as performance and reliability).

In this paper we have suggested that the effort of certifying EMU software systems, which are part of the product line, may be reduced by attaching certification evidence to each reusable asset. Subsequently, the certification argument and evidence (i.e. the assurance case) of the entire software can be created by composing the items of evidence associated with each reusable asset. Unfortunately this is insufficient, as the correspondence between the overall evidence and the software architecture is not necessarily one-to-one. Arguing the dependability of the whole based on the dependability of the parts is fallacious as failures may occur as a consequence of unreliable interactions between these assets. In short, the correspondence between modularity in reasoning and modularity in software is not necessarily one-to-one. One area of our future research work will investigate the feasibility of creating a dependability case architecture for the product line which ensures the logical integrity of reusable items of evidence in the same way that a product line architecture ensures the integrity of composable software components.

Finally, the EMU product line is a first step towards the establishment of a product line of engine control systems. A number of research areas need to be explored as a prerequisite for the establishment of a control system product line. These areas include, but not limited to:

- Understanding the relationship between hardware product lines and software product lines
- Investigating the impact of product line variability on code analysis
- Understanding the relationship between software reuse and unreachable code or dead code
- Checking the feasibility of encapsulating analyses and hence treating them as reusable product line assets
- The acceptability to the dependability community of the concept of incremental certification where pre-certified software elements can be composed with each other with little re-certification effort.

9. References