Product Line Analysis and the System Safety Process

Zoë Stephenson, Savita de Souza, John McDermid; University of York; Heslington, York YO10 5DD, UK

Andrew G. Ward; Rolls-Royce plc.; P.O. Box 31, Derby DE24 8BJ, UK

Keywords: safety, software, product line, dependencies, requirements

Abstract

Product lines (groups of highly similar products) have recently received a great deal of attention in the software engineering community, and especially in organisations that develop products that contain embedded software. The product-line concept provides a constrained production environment that facilitates change and reuse between members of the product line. The reported successes of the last few years concentrate on the embedded software product, but do little to address the wider issues of integrated systems engineering, and certainly do not address the safety process.

We believe that there is an enormous benefit to be gained by applying the successful disciplines of software product-line engineering to the system safety process. The results thus far indicate that these techniques can help in enhancing the completeness and robustness of a product line's safety-related requirements. This paper focuses on the phases of the system safety process that affect system requirements.

Introduction

The safety process is normally concerned with the behaviour of a single system within its context of operation. The process focuses on the generation of models of the behaviour of the system that can be analysed to show their safety properties, and uses methods such as Markov modelling and fault-tree analysis (ref. 10). For each new system, a complete new analysis is undertaken, with new models based on the physical system design, its functional design, and its context of use. This process is fundamental to the success of a safety-critical product, but it takes a large proportion of the project effort to generate and maintain the safety analysis information.

In the field of software engineering, and particularly in embedded systems, a great deal of research is devoted to product lines (refs. 13, 2) and reuse (ref. 5). Instead of treating each product as a separate endeavour, as was common in earlier software engineering processes, product line processes operate on a set of related software products. The membership of this set of products is controlled by an explicit definition of the product line scope. The scope describes the features that characterise members of the product line (ref. 9). The constrained scope of a product line provides a basis for cost reduction through the opportunity to reuse or auto-generate design process artefacts.

To reinforce this notion of a constrained product scope, product line processes often make a distinction between asset engineering, in which reusable assets are developed for the product line, and application engineering, in which assets are composed and combined to create products (ref. 15). A decision model is created, during the asset engineering phase, to describe the available selections and choices among product line assets. Because it effectively defines the set of products that the product line contains, the decision model can also be considered as a *de facto* expression of the product line scope.

The experience from software product lines indicates that a constrained product scope and a prescribed decision model are beneficial in securing a return on investment in reusable design process artefacts (ref. 1). The work reported in this paper is a preliminary investigation into the effects of applying these principles to the safety assessment process. If successful, this should provide a constrained environment in which to reuse and automate the generation of safety analysis information for products that are related within the product line. This would reduce the effort associated with producing the analysis models, allowing the engineers to focus on correctly deriving safety requirements from those models and evaluating conflicting requirements and feature interactions. As an additional benefit of this approach, it is expected that there will be a uniform structure for safety information across the whole product line within which the various analysis results fit, hence improving the readability of the analysis information.

Related Work

The work presented in this paper draws on existing asset engineering analysis processes. The following five steps represent a summary of the key aspects of asset engineering processes (refs. 8, 4, 3, 14):

Scoping: The product line scope identifies the range of products that are considered to be part of the line. The products are defined in terms of features, which are domain-specific statements that identify those products to the various stakeholders. Getting a representative scope is extremely important — too small a scope, and the organisation will be unable to make some of its products; too wide a scope, and there will be a smaller opportunity to make use of common, reusable assets and a greater investment in assets that never get used.

Domain Analysis: Domain analysis describes the environment within which the software operates — typically including the different stakeholders that are involved in the development and operation of the software such as maintenance engineers, users and computer hardware systems. The resulting domain model encompasses the pertinent details of that environment, taking in such diverse information as domain-specific terminology, constraints and device data sheets. This provides the information necessary to build systems within the product line.

Commonality Analysis: The commonality analysis stage defines the features of the product line in more detail, and associates them with a decision model. The decision model identifies the permitted selections among the features with the selection keywords "Common" (all products), "Variable" (only some products), "Choice" (mutually exclusive list) and "Select" (select at least one). There may also be dependencies on particular parameter ranges or analyses. Each decision in the decision model is a *point of variation*, and those decisions select, deselect and customise the requirements for a particular product.

Architecture Definition: The architecture definition stage derives a product-line architecture that accommodates the variations in the requirements. This can be considered as a standard architectural trade-off process, with the requirements for each individual product trading off against the flexibility needed to support the whole product line.

Asset Definition: Given a decision model, a set of features and a software architecture, the final phase of the asset engineering process creates specific assets for use in the individual products. There may be some overlap between application teams and core asset teams in producing these assets, and assets may be created afresh or mined from existing products. This phase uses standard software engineering techniques, guided by the product-line scope.

These techniques work well when there are few stakeholders, and a single boundary between a small domain and a well-defined single-technology product. For use in an embedded safety-critical system, there are some important differences that must be accommodated:

- Embedded systems have system-level and software-level stakeholders, and they often use different terminology and place emphasis on different characteristics when defining features. This arrangement is not accommodated well in existing product-line processes, especially those that focus on generation technology.
- Embedded systems are typically highly integrated, with many indirect (causal) dependencies between different parts of the system. The boundaries between different technologies and different stakeholder responsibilities may vary from product to product just as easily as functionality. Product line processes are aimed at functionality and assume that the other factors will be constant across the product line.
- Safety-critical system development processes do not have a single step between requirements and design stages. Instead, the design process runs concurrently with safety assessment processes, and successive stages of analysis introduce derived requirements for different system and subsystem scopes.

Despite these challenges, the issue of product lines in safety-critical software engineering has been addressed in recent work. Stephenson (ref. 14) proposes an abstract metamodelling technique, *decision tracing*, as a way of abstracting from the details of the different stakeholder concerns and levels of design. This means, however, that the abstract metamodels must be customised to a particular domain before being used, and there are currently no criteria by which an effective set of concrete elements is to be determined. As an example, the domain of safety-critical systems engineering would require decision elements for preliminary safety assessment, fault-tree analysis, introducing FMEA data and so on — but there is no established method for identifying precisely the required elements.

The issue of safety in product line software has been addressed in other work by Lutz (ref. 12). The analysis of a product line of astronomical instruments led to the identification of a product line hazard list. A safety assessment of the existing product line software requirements showed that a number of those hazards were not yet adequately addressed by any system. The use of the product line approach led to the expression of derived software requirements to address these hazards; these requirements were reusable across all of the products. The analysis process as a whole is intended to provide assurance that the architecture contributes correctly to the safety and reliability of the system while remaining flexible with respect to the product-line scope. Lutz' work concentrates on small software systems, and focuses on the contribution of software functionality to system-level hazards.

Unlike many of the properties typically addressed in product line processes, the safety of a complete system is not a straightforward compositional property. Even if each part of the system is in itself safe to use, this is no guarantee that the combination of those parts will be safe in its context of use. Safety must be addressed as a property of the integrated system (ref. 10).

The work presented in this paper focuses on analysis at the level of system requirements for a product line of systems that contain an embedded software controller. The standard product line techniques can be used to treat the embedded software as a product line; the aim of this work is to investigate the potential benefits of applying the same characterisation of product lines to the system safety assessment process. If the same type of product line technique can be usefully applied, it will bring a number of potential benefits:

- Safety analyses and results will be reusable between members of a product line, reducing the cost of the safety process and reducing the opportunity for error in the safety assessment.
- The process systematically identifies variation in derived safety requirements, thus reducing the risk that the safety process will undermine the reuse of the design artefacts in the product line.
- Applying product line techniques to safety assessment reduces the barriers to adoption of a complete product line process.

The rest of this paper presents the results that we have obtained through the application of product line techniques to aspects of the safety analysis process in use at Rolls-Royce Controls.

Scoping

The initial stage of the study was the scoping of the product line to be addressed. The scoping was performed in two separate stages. In the first stage, a set of recognisers was drawn up to identify the types of product and function, documentation, level of detail and expertise present in the product line. Each recogniser is a sentence that is true of elements within the product line, and false of elements outside the product line. As an example, a functionality recogniser could be as follows:

> The FADEC is responsible for auto-relight of the engine.
> The EEC is responsible for control of the ignition system.

Systems for which these statements are not all true are not members of the product line. Together, these statements give a definition of the boundary of the systems that form the product line.

The second stage of the scoping study was an analysis of a set of safety-related requirements. The intent of the analysis is to arrive at a list of hazards and design elements to represent the product line scope. A reading technique was used by two different researchers to extract the various noun-phrases from the high-level system requirements for the starting functionality within the product line. These phrases were compared with one another to eliminate redundancy, and then classified as hazards at the boundary of the system, design elements, or related causes.

Domain Analysis

In extracting the hazards, design elements and related causes, some domain-specific terminology was encountered (e.g. "Hydraulic Offload", "FMU", "HPSOV"). These terms were fed into a conventional domain analysis process to build a glossary and define relationships for the various domain-specific items. The process took information from

the glossaries, safety assessments and system architecture descriptions of the products that fit within the product line scope. This was straightforward except for generalisations, which were were found in two different ways. The first type was a special-case scenario that could be found in some products. For example, a product may make a distinction between an inability to start while the aircraft is on the ground and when the aircraft is in the air. The second type of generalisation arose when there was a mismatch in terminology or concept between the different products. For example, one product may use electrical actuation for a component, whereas another product uses hydraulic actuation. For this kind of situation, it is up to the domain analyst to invent a unifying generalisation to include these different specific concepts.

Commonality Analysis

Commonality analysis processes are typically focused on the requirements and design artefacts. Here, we have applied the same types of technique but to the safety assessment concepts of hazard and cause. To begin, a variation analysis was performed on a set of derived safety requirements. The analysis used a conventional reading technique, identifying the different parts of the requirements that could undergo change. This kind of search uses a simple key-phrase search such as that found in the HAZOP technique:

- Could this concept take a different value?
- Could the concept be weakened (generalised) or strengthened (specialised)?
- Could this concept be of a different type?
- Could this concept begin or end at a different time?

In each case, the only valid variations are those that lie within the scope of the product line. As an example, consider the following requirement:

> **R.StartingAutomatic**
> The FADEC System shall have the capability to initiate Automatic Engine starting on the ground and in flight in response to pilot command.

The following concepts were found in this requirement: ‘The FADEC System’, ‘Having a capability’, ‘Initiating’, ‘Automatic engine starting’, ‘On ground’, ‘In flight’, ‘In response to’, ‘Pilot command’.

Next, each concept was evaluated against the key-phrases to determine the possibility of variation within the product line scope. This gave the following possibilities:

- VARIATION: The concept of initiating can be extended to initiating and terminating.
- VARIATION: Automatic engine starting can be generalised into a generic engine start concept with the automatic type as a specialisation.
- VARIATION: On ground and in flight can be generalised to an abstract concept of flight phase.
- VARIATION: Pilot command can be strengthened into the specific commands that may initiate an automatic engine start.

In addition, the issues of replication, removal, extension and constriction were considered at the level of the complete requirement:

- Could this requirement be used elsewhere?
- Could this requirement be removed?
- Could this requirement be weakened?
- Could this requirement be strengthened?

These larger requirements-level questions apply to the example requirement as follows:

- VARIATION: A similar requirement is used for manual starting.

- VARIATION: The requirement can exist in a weaker form: The FADEC system shall initiate starting.
- VARIATION: The requirement can exist in a stronger form: The FADEC system shall initiate, control and terminate starting.

The analysis thus far is based solely on the requirements. To apply this to the safety process, the variation in the requirement is traced forward to safety information that is based on that requirement, using standard impact analysis techniques. For the example requirement, this is as shown in table 1. The entries in the first column correspond to the analysed variations. The safety variations are the impact that the requirements variation has on the safety analysis information. For example, the table shows that a change to the definition of flight phase impacts on the analysis of scenarios where the wrong type of start is used.

This obviously does not form a complete safety assessment. Instead, it identified areas of vulnerability based on particular requirements. These areas are the places where the safety analysis information is most likely to change when moving from one product to another. It is important to ensure that the safety assessment process and its results take into account the effects of changes in these areas.

Application to Safety Analysis

To help in visualising the various hazards, causes, design elements and relationships between them, a dependency matrix was used. The matrix records the dependencies between changes in one element and any consequent changes in other elements. That is, a dependency from A to B exists when a change to B could have a subsequent effect on A. The typical changes that would be dealt with in a product line include:

- A redefinition of a parameter or some other minor detail.
- A change from one version of a component to another.
- A change of selection from a list of entries.
- A change that completely includes or excludes an element of the product.

These dependencies are recorded in a matrix, structured as follows:

- The rows and columns of the matrix represent the different design elements, hazards and causes to be considered.
- Each entry is present as both a row and a column, so that any dependency between entries may be recorded.
- A dependency from element A to element B (i.e. one in which a change to B may ripple to A) is recorded in the cell at row A and column B. It is recorded by writing the type of dependency into the cell.

The intent is that the matrix can record the variations among an entire product line, and identify areas that suffer a high degree of change between members of the product line. The dependency matrix can be used to perform automated impact analysis, identifying all of the hazards affected when a change occurs in a design element or in the way events arise from design elements. This analysis is based solely on the recorded dependency information, and so it should be considered to be an initial change set, and subjected to human review.

Table 1 — Correspondence Between Requirements and Safety Variation

Requirement Variation	Safety Variation
Initiating and terminating	Incorrect sequencing, uncommanded start, inability to terminate start
Generic engine start	Incorrect start type
Flight phase	Incorrect start type
Pilot command	Uncommanded operation, incorrect start type
Manual start	No impact
FADEC initiation	Incorrect start type
FADEC control and termination	No impact

For product-line safety, the dependency matrix contains hazards, causes and design elements. The matrix can be summarised as follows:

	Design	Cause	Hazard
Design			
Cause			
Hazard			

The matrix is divided into nine compartments, each representing dependencies between particular types of information. Information from the analysis stages is added into the appropriate compartments in the matrix to show the relationships between the elements. The different relationships are represented with two-letter abbreviations to denote the reason for the relationship, as shown in table 2. An example of these relationships is shown in table 3, focusing on the design-to-design compartment.

The matrix provides all of the information needed to extend the safety analysis from a single product to a product line. A systematic process of scenario generation is used; each scenario represents a point of variation that must be accounted for in the safety assessment. This gives a constrained set of variation points as a basis for product line safety analysis. Each scenario is linked to a particular hazard, and identifies the most vulnerable areas of variation for that hazard. The process is shown diagramatically in figure 1. The diagram depicts a dependency matrix, with an extra column to the left that identifies variant design entities. The stages are as follows:

1. Identify the row that represents the hazard to be addressed, and check for more general hazards.
2. Continue checking until there are no more general hazards.
3. Identify corresponding causes for the hazards, again checking for more general causes.
4. Identify design elements that may change and check for sub-element dependencies.
5. Repeat until a full set of elements is found.
6. Causes that belong to both the hazard and the design elements that change are the vulnerable causes for that hazard.

This process was applied to hazards within the engine starting functionality, based on system safety assessment data from an existing engine controller project within the product line. The system safety assessment identifies sixteen events of interest, classifying severity levels of five as hazardous, six as major severity, four as minor severity and one as not severe. Of these sixteen events, twelve correspond to events that are not at the system boundary but are of importance to the customer, or events that do not involve the engine starting system. The remaining four events are listed as hazards in the bottom compartment of the dependency matrix.

The analysis data from the system safety assessment documentation was used to create a set of simple fault trees. This process allowed the authors to visualise the information and abstract away from the various domain-specific acronyms that characterise the different events. Then, the top-level hazards from each of these trees were used as input to the scenario-generation process. The resulting causes and design information were mapped back into the fault trees to identify the overall effect on the tree of the vulnerable causes. The results are as shown in table 4. In this table, the "Fault Tree Impact" column identifies the type of change to the fault tree. An *optional contribution* is a change that inserts an extra branch under a node in the tree. A *change of rate* is a change to the failure rate assigned to a node, without changing the tree structure. The "Gate" column identifies the gate immediately above the change, and the "Analysis" column identifies the type of analysis that was used in producing that gate in the original tree.

Some of the variations in this table are related to the way in which different subsystems are arranged in the aircraft design. For example, some aircraft designs treat the starter system as a separate subsystem to the engine, whereas others treat the starter as an integral part of the engine system. This is reflected in the table as optional contributions to the engine-level hazards from the starter system. When the starter system is considered to be part of the engine system, those contributions are present, and when the starter system is a separate system, those contributions are absent (and present in the separate starter system analysis instead).

Table 2 — Dependency Types for Each Compartment

Compartment	Abbreviation	Dependency
Design A to Design B	CN	A controls B
	ST	A reports status of B
	PE	A addresses physical effects of B
	SE	A is a sub-element of B
Cause A to Design B	EX	A is exhibited by B
Cause A to Cause B	SC	A is a special case of B
	CB	A is caused by B
Hazard A to Cause B	CB	A is caused by B
Hazard A to Hazard B	SC	A is a special case of B

Table 3 — Tabular Representation of Dependencies

	Ignition System	Control System	Starter	Starter Cooling	Cockpit Display	...
Ignition System						
Control System	CN		CN	CN		
Starter						
Starter Cooling			PE			
Cockpit Display	ST	ST	ST	ST		
...						

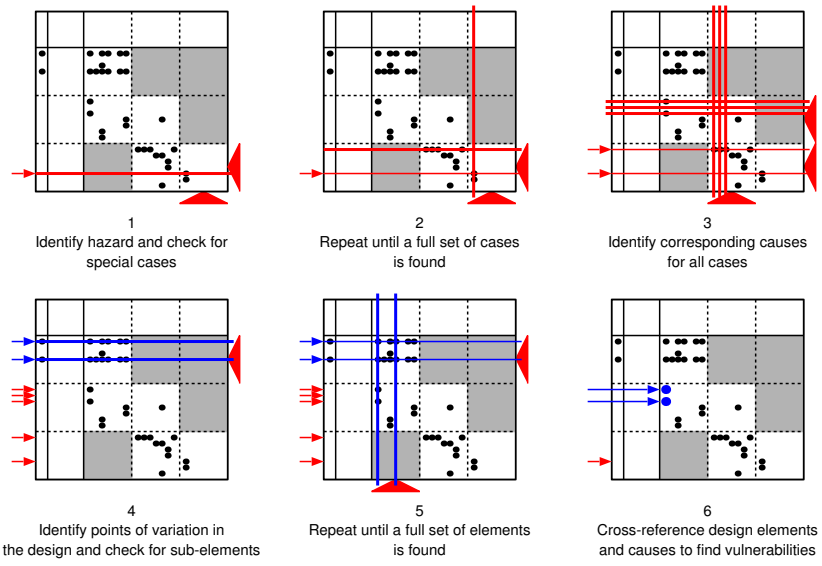

Figure 1 — Scenario Generation Process

Table 4 — Fault-Tree Impact Analysis Results

Hazard	Cause	Fault Tree Impact	Gate	Analysis
Inability to start	Starter overspeed	Optional contribution	OR	design
	SAV unable to open	Change of rate	OR	design
Inability to shut down	HPSOV unable to close	change of rate	OR	causal
Loss of thrust control	Inadvertent energisation of SAV	Optional contribution	OR	causal
	SAV unable to close	Optional contribution	OR	causal
	Uncontained starter overspeed	Optional contribution	OR	causal
	Non-restartable engine flame out in flight	Optional contribution	OR	causal
Engine fire	SAV unable to close	Optional contribution	AND	causal
	HPSOV unable to close	Optional contribution	OR	causal

Conclusions and Further Work

From the small sample studied in this preliminary investigation, several useful conclusions can be drawn:

- It had been assumed at the start of this work that variation in design would only impact those parts of a fault tree that are based on design structure. The other parts of the fault tree, such as those based on causal analysis, were expected to remain the same. However, from the data in the table of results, there is no such restriction. Changes to the design can have an impact on any part of a fault tree.
- A majority of the impact was optional contribution rather than a change of failure rate. This may be a bias due to the choice of a function that is sometimes treated as a separate aircraft subsystem. However, the presence of optional contribution must be taken into account when analysing these fault trees.
- Most of the variations manifested themselves directly below an OR gate. This is good from the point of view of modifiability — a change to the variable event will at most produce a change in the rate assigned via that gate. One of the variations was found to be directly below an AND gate, however. The removal of a branch under an AND gate may mean the removal of the entire gate and any higher-level events until an OR gate is reached. A simple observation would be that OR gates are good for variability, and AND gates are good for safety.
- The use of an inspection and analysis process such as that described in this paper provides an additional level of validation of the safety analysis. During the analysis, many assumptions about the nature of the starting domain were challenged and documented. The dependency matrix served as a useful overview of the structure of the domain and helped to identify areas of concern (e.g. identical rows or columns in the table indicating the same concept under a different name). This in itself supports the completeness and robustness of the analysis.

This whole analysis process helps in managing the reusability of fault trees in two complementary ways:

1. If the analysis process shows that there is a vulnerable cause, it can also link back to the design change that leads to that cause. From this, the analyst can determine a strategy for handling the impact — for example, the tree could be restructured using suitable abstractions, or a generator system could be used to create trees for different products. If the impact cannot be accommodated in any of these ways, then the hazard must be analysed separately for each product, and this can be planned for accordingly.
2. If the analysis process shows no vulnerable causes, the fault tree is likely to have the same structure and content across the product line. These fault trees can be set out for the whole product line, and that common structure reused for each product.

The findings of this preliminary investigation coincide with recent findings in similar work by Lutz and colleagues (refs. 11, 6). They describe two related approaches — Fault Contribution Trees and Product-Line Software Fault Tree

Analysis. A Fault Contribution Tree (FCT) is a decision model that describes the dependencies between hazards and their possible causes. The causes exhibited by a particular system select variabilities in the FCT; the dependencies then identify which of the hazards are to be addressed for that product. The FCT does not record the causal dependencies of the fault tree, merely the selection dependencies of the decision model.

The product-line software fault tree analysis method takes a different approach. Here, product-line variability analysis is used to describe variations in the events of the fault trees for software system hazards. The analysis is performed on the tree itself, rather than being based in the design model. Analysis on the tree leads to variation specifications under leaf-nodes in the fault tree; for a specific product, the variations for that product are used to prune the tree to match the features present in the product. This pruning process navigates selection dependencies in a similar fashion to the scenario generation process, but requires additional domain knowledge to complete the analysis — the dependency types and design information in the dependency matrix provide a way of capturing this type of domain knowledge in advance, and provide a broader and more complete view of the product line.

The FCT approach performs a similar function to the cause and hazard compartments of the dependency matrix, in that it describes product-line selection dependencies at that level. It would be appropriate to use the FCT approach as a modelling technique to derive the entries in those compartments of the table, or to visualise the content of those compartments.

To continue this work, then, there are a number of strategies that we would like to pursue:

- Produce a dependency matrix for other functional areas within the engine controller. This will give a better coverage of the system functionality, and may uncover further dependency types. This will also test the scalability of the approach to a large dependency matrix.
- Build tool support for the manipulation of the dependency matrix and associated analyses. Without some kind of tool support, the method is unlikely to scale to a full project. The tool support would ideally integrate with safety analysis tools.
- Evaluate the scope of the existing product line analysis against system safety analysis information from the other products. This validates the scope that has been chosen, and identifies areas on the border of that scope that need further deliberation.
- Extend the process to the full lifecycle. The safety process continues alongside the development process all the way through design, implementation, verification, validation and delivery. The additional detail from these stages should be traced back to the requirements-level dependency matrix.
- Consider alternatives to the use of fault trees. The fault tree structure is relatively independent of the design structure, and small changes to one are not necessarily reflected as small changes in the other. A safety modelling technique based on the design structure, such as FPTN (ref. 7), may be preferred.

A considerable amount of work will be needed to produce an industrially usable method for product line safety analysis, but these initial steps confirm that the idea has potential.

References

1. J. Bayer, O. Flege, P. Knauber, et al. PuLSE: A Methodology to Develop Software Product Lines. In Proceedings of the Fifth Symposium on Software Reusability, pages 122–131, May 1999.
2. J. Bosch. Design and Use of Software Architectures. Addison-Wesley, 2000.
3. J. Bosch, G. Florijn, D. Greefhorst, et al. Variability Issues in Software Product Lines. In F. van der Linden, editor, Proceedings, PFE-4 2001 (LNCS 2290), pages 13–21, 2002.
4. J. Coplien, D. Hoffman, and D. Weiss. Commonality and Variability in Software Engineering. IEEE Software, 15(6):37–45, Nov. 1998.
5. K. Czarnecki and U. W. Eisenecker. Generative Programming. Addison-Wesley, 2000.
6. J. Dehlinger and R. R. Lutz. Software Fault Tree Analysis for Product Lines. In IEEE International Symposium on High Assurance Systems Engineering, 2004.
7. P. Fenelon, J. McDermid, M. Nicholson, et al. Towards Integrated Safety Analysis and Design. ACM Applied Computing Review, Aug. 1994.

8. H. Gomaa and G. A. Farrukh. Methods and Tools for the Automated Configuration of Distributed Applications from Reusable Software Architectures and Components. IEE Proceedings in Software, 146(6):277–290, Dec. 1999.
9. M. L. Griss. Implementing Product-Line Features with Component Reuse. In Software Reuse: Advances in Software Reusability (LNCS 1844), pages 137–152, June 2000.
10. N. G. Leveson. Safeware: System Safety and Computers. Addisson-Wesley, 1995.
11. D. Lu and R. R. Lutz. Fault Contribution Trees for Product Families. In International Symposium on Software Reliability Engineering, pages 231–242, Nov. 2002.
12. R. R. Lutz. Toward Safe Reuse of Product Family Specifications. In Proceedings of the Fifth Symposium on Software Reusability, pages 17–26, May 1999.
13. D. L. Parnas. On the Design and Development of Program Families. IEEE Transactions on Software Engineering, 2(1):1–9, Mar. 1976.
14. Z. R. Stephenson. Change Management in Families of Safety-Critical Embedded Systems. PhD thesis, The University of York, Mar. 2002.
15. D. M. Weiss and C. T. R. Lai. Software Product-Line Engineering: A Family-Based Development Process. Addison-Wesley, 1999.

Biography

Zoë Stephenson, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432749, fascimile – +44 1904 432708, email – zoe.stephenson@cs.york.ac.uk

Zoë Stephenson graduated from the University of York with a first-class honours degree in Computer Science. She has previously worked as a research student funded through an EPSRC CASE studentship with Rolls-Royce plc., and she is now working in the Rolls-Royce UTC on flexible product-line technology and processes.

Savita de Souza, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432735, fascimile – +44 1904 432708, email – savita.souza@cs.york.ac.uk

Savita De Souza graduated with a First Class Honours degree in Electrical Engineering and a PhD in fractal-based digital watermarking at De Montfort University, Leicester. Since November 2002, she has been working as a Research and Teaching Fellow in the High Integrity Systems Engineering (HISE) group in the Department of Computer Science at the University of York. More recently, she has been working on requirements engineering and product line analysis as a research activity at York.

John McDermid, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432726, fascimile – +44 1904 432708, email – john.mcdermid@cs.york.ac.uk

John McDermid has been Professor of Software Engineering at the University of York since 1987 where he runs the high integrity systems engineering (HISE) research group. HISE studies a broad range of issues in systems, software and safety engineering, and works closely with the UK aerospace industry. Professor McDermid is the Director of the Rolls-Royce funded University Technology Centre (UTC) in Systems and Software Engineering and the BAE SYSTEMS-funded Dependable Computing System Centre (DCSC). He is author or editor of 6 books, and has published about 280 papers.

Andrew G. Ward, Functional Lead — System Safety & Reliability Group, Controls OBU, Rolls-Royce plc., P.O. Box 31, Derby DE24 8BJ, United Kingdom, telephone – +44 1332 247189, facsimile – +44 1332 247427, email – andrew.ward@rolls-royce.com

Andy Ward is the Functional Lead in Safety and Reliability for the Control Systems Business Unit within Rolls-Royce plc., and is the Process Owner for Control Systems Safety. He has a BSc degree in Mathematics, a Postgraduate Diploma in Reliability Analysis, and an MSc degree in Safety Critical Systems Engineering. He was previously a member of SAE (US Society of Automotive Engineers) S-18 committee and a co-author of ARP 4761.