INTERNATIONAL DESIGN CONFERENCE - DESIGN 2006
Dubrovnik - Croatia, May 15 - 18, 2006.

DESIGN 2006

# CHANGE IMPACT ANALYSIS AT THE INTERFACE OF SYSTEM AND EMBEDDED SOFTWARE DESIGN

M. S. Kilpinen, P. J. Clarkson and C. M. Eckert

*Keywords: change impact analysis, change propagation, systems engineering, embedded software*

## 1. Introduction

As designers incorporate more and more computing technology to create "smart" products, the design process must ensure the seamless coordination of contributing design fields, including, mechanics, hardware electronics, and software. Systems engineering design processes recognise this multidisciplinary nature of products and suggest means to partition and then integrate designs from different domains. System designers particularly synchronise the embedded software requirements with the other disciplinary design requirements to provide the required control functionality and logic of the product, tending to leave the embedded software field only to interact with system design for these development tasks, rather than to work directly with engineers from other fields. Studies suggest this melding of embedded software with the system design during design iterations or redesign can cause difficulty meeting development costs and schedules, and, in severe cases, instigates operational errors or failure; the impact of such design changes may not be fully understood between the system and embedded software disciplines [Maier and Rechtin 2000] [Lindvall *et al.* 2003].

Systems engineering often prescribes design iterations and redesign through requirements; the design of a system should begin with determining the high-level requirements, followed by deriving the relevant lower level specifications and finally creating the detail design. In turn, this process of development constructs a requirement hierarchy linked to the component designs. The introduction of component rework or a new feature of a system similarly should follow this top-down approach by first revising the requirements and then the design. Estimating these alterations to all the requirements and design, also known as *change impact analysis*, should occur through evaluating the existing requirement traceability relationships within the hierarchy [INCOSE 2004] [Bohner and Arnold 1996]. Theoretically, this requirement-based change impact analysis process should initiate the appropriate requirement changes then passed from system design to the involved disciplines, such as embedded software. However, the system design requirements at this embedded software interface might not always exist or be up-to-date. In this case, the prescribed requirement-based change impact analysis might be impossible or prohibitively difficult and replaced with a more *ad hoc* process, leading to an inaccurate estimation of the required rework, misunderstanding of design tradeoffs, and unexpected knock-on effects caused by design changes.

This paper examines this apparent incongruence between the prescribed systems engineering top-down redesign processes through requirements and the industrial practise of change impact analysis without complete requirements at the system and embedded software interface. A discussion of the observations from an empirical study exemplifies this misalignment, suggests the factors influencing the lack of such impact analysis, and proposes future directions of inquiry.

## 2. Methodology

This research consists of a systematic literature review and a corresponding empirical study. The following section discusses how the discovery of a gap in the literature review spawned the exploration of industrial practise.

### 2.1 Systematic literature review

This study performed a systematic literature review of research in software, hardware, mechanic, mechatronic, and system design for references and approaches to impact analysis. The literature search space additionally included design processes, requirements engineering, and modelling capabilities in order to understand the context of the spread of change. The search identified the systems engineering requirement-based techniques, applicable at the software interface, and other software-specific change impact analysis methods and tools implemented at the component level (see figure 1, first column, and [INCOSE 2004] [Bohner and Arnold 1996]). Both the system and software research report a conflict between the prescribed requirement-based impact analysis technique and the reported frequent lack of software requirements and suggest a couple coping strategies within these disciplines (see figure 1, second column, and section 3 for a detailed discussion). Although systems engineering research has briefly addressed the general difficulties surrounding the implementation of system design process improvement strategies and has offered generic mechanisms to circumvent these barriers (as discussed in [Sheard *et al.* 2000]), no literature to date has been found to explore this misalignment of prescribed and practised impact analysis at the systems and software engineering interface. Given the high costs and reported project delays associated with embedded software redesign (see [Clark *et al.* 1999]), this research aims to bridge this gap (summarised in figure 1) by highlighting this incongruence and the missing research on the barriers of requirement-based impact analysis adoption at the system and embedded software design interface.

| | Prescribed Impact Analysis Technique | Practiced Impact Analysis Technique | Barrier Identification / Process Improvement Strategies |
|---|---|---|---|
| **System Engineering** | Top-down requirement traceability analysis | **[CONFLICT]** Lack of requirements; *Ad hoc* analysis | **[LIMITED]** General barriers of and strategies to deploying sys eng techniques |
| **Software Engineering** | Top-down requirement traceability or component-level dependency analysis | **[CONFLICT]** Lack of requirements; Analysis through experience | **[LIMITED]** Specific strategies to implement dependency analysis |
| **System-Software Interface** | Top-down requirement traceability analysis | **[SUGGESTED CONFLICT]** Lack of requirements; Traceability analysis difficulty | **[NONE]** *Missing: barriers to requirement traceability analysis* |

**Figure 1. Literature search space highlighting of conflicting and missing research**

### 2.2 Empirical Study

In order to further investigate this apparent literature gap, a 7-week empirical study on the interaction of the system and embedded software design groups was conducted within the civil aero engine control system division of Rolls-Royce plc in Derby, UK. Twenty-three semi-structured interviews, numerous informal discussions, daily observation, and documentation review with key stakeholders, including system designers, embedded software designers, design process engineers, and control system management, obtained a holistic perspective on the interaction of these domains. Interviews were timely transcribed and analysed by identifying and revising overarching concepts and themes in order to inform the subsequent interviews. The stabilisation of these themes in the later interviews suggests a common understanding had been reached. The phases of the study included:

1. The exploration of the change processes and requirement management between systems and embedded software in practise,
2. The inspection of the Rolls-Royce prescribed iterative design and requirement management processes, and

3. The examination of the systems and embedded software change impact analysis processes in detail.

The first phase specifically began by gathering background information and perceived problems of the change process between the system and embedded software groups. From this initial inquiry, a further investigation of the requirement management processes ensued based on the necessity for the company to provide such information to pass safety-critical software certification for their delivered product. The second phase aimed to contrast these change and requirement processes observed in practise with the prescribed processes by the company. These two phases provided the context for the third phase to elicit the details of change impact analysis techniques between the system and embedded software teams implemented in practise. This stage of the study acquired impressions of the differences between the actual and prescribed design processes, the role and timing of requirement generation, revision, review and implementation, and the means of interaction between the system and embedded software groups. A fourth and final phase of the study elicited input on the results obtained from the on-going analysis as a means of validation. These phases of research yielded an understanding of industrial practise for a single company and also suggest that additional research is required to substantiate further discussion on the prescribed and practised impact analysis techniques.

## 3. Prescribed versus practised system and software design

The systems engineering requirement-based style of redesign appears to conflict with the reported lack of requirements during embedded software design. This section explores the prescribed and practised approaches of impact analysis and discusses the misalignment and gap found in literature.

### 3.1 Systems engineering redesign processes and impact analysis through requirements

According to the International Council on Systems Engineering (INCOSE), the systems engineering process views a product as a whole, guides the design of the product subsystems to produce the desired functionality, and integrates the design activities and changes. Thus, system designers must closely monitor the design activities and resolve interface issues between disciplines [INCOSE 2004]. Although systems engineering process models may illustrate the design process slightly differently, the system design discipline tends to be described as the primary interface between the disciplines [Kossiakoff and Sweet 2003].

The V-model (shown in figure 2) particularly represents an example of a prescriptive systems engineering design process at the interface between the system and software designs. This representation illustrates the division between systems and software engineering and assigns responsibilities to these disciplines based on the phase of product development. Although the software discipline should have input during the user requirement and preliminary design phases, this model implies that software requirements can only be derived from the functional requirements of the system and, thus, suggests that system design is the primary interface for software design [Blanchard 2004].

The V-model acknowledges the decomposition and integration of disciplines, but does not explicitly specify iteration loops between the disciplines. System design may have to revise requirements before the finalisation of the design based on input from the software team. Other systems engineering process models, such as a spiral model, capture this iterative redesign process [Kossiakoff and Sweet 2003]. These practises tend to specify that design iterations should begin at the highest level of system requirements and changes should flow down through the relationships in the requirement hierarchy into the lower-level requirements given to the disciplinary teams by system design. The disciplinary groups can then perform the detail specification and design changes to fulfil the modifications by system design. This top-down style of redesign lends itself to maintain requirement and design traceability, or the mapping of design requirements to their respective detail designs [INCOSE 2004]. Theoretically, the impact of an initiating change may be analysed before redesign commences from the existing network of requirement relationships. Understanding the full potential impact of such a change between disciplines can make the design process more effective through mitigating the risks of late delivery, exceeded design budget, or product failure. If a side effect of an iterative or redesign requirement change is unexpected and discovered late, the development time and cost can increase;

moreover, if it is simply undetected, the modification can cause system failure or errors. Several commercially available software tools enable this top-down requirement style of impact analysis.

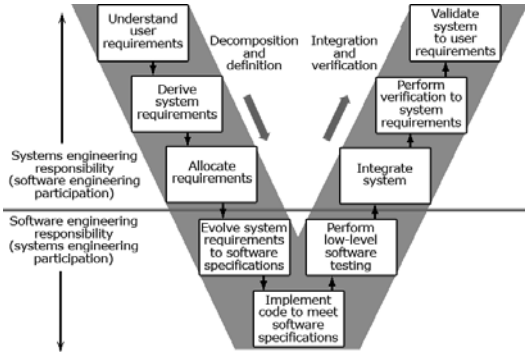


**Figure 2. The V-model of the systems and software boundary adapted from [Blanchard 2004]**

### 3.2 System design impact analysis in practise

Although systems engineering processes and tools tend to prescribe that system redesign due to an initiating change should begin with a top-down examination of the requirements, literature suggests that essential modifications are identified through experience and often are implemented in an *ad hoc* manner without any requirement analysis or revision [Maier and Rechtin 2000]. Additionally, research found in the embedded software domain indicates that software requirements may not exist or be updated regularly and that design changes are often implemented directly at a component level. Software requirements and traceability relationships only might be retrofitted to match the modified detail design [Taramaa *et al.* 1996]. Without the complete existence and maintenance of the requirement hierarchy between the system and embedded software design disciplines, a complete requirement-based impact analysis may prove difficult, if not impossible, because the requirement relationships remain incomplete. Thus, the systems and embedded software engineering literature both suggest that requirement-based impact analysis approaches at this disciplinary interface may not always be used in practise. However, the literature found does not directly address this incongruence between the prescribed and practised change impact analysis process at the system and software design interface or capture influences for the industry impact analysis practise at this interface.

### 3.3 Impact analysis specific to the software engineering discipline

Similar to the systems engineering discipline, software engineering suggests requirement-based impact analysis to improve the understanding of the risk associated with domain design changes. Within this specific domain, this technique investigates the relationships between software life-cycle objects (i.e. requirements, specifications, design documentation, etc.). In contrast to systems engineering, software engineering also has developed component-level impact analysis techniques. These dependency analysis methods deduce the linkages between variables, logic, modules, and the like within the software code and can perform predictive change impact analyses [Bohner and Arnold 1996]. However, both the traceability and dependency techniques tend to focus on particular domain-specific changes rather than modifications to the functionality of the entire system design, which may not always be directly visible through these linkages.

Although software impact analysis, nevertheless, may yield some insight into low-level system design change, there is no overwhelming evidence that the software industry has adopted either the software traceability or the dependency style of impact analysis. According to case studies conducted in eleven different industry sectors, Lindvall *et al.* (2003) observe that software engineers tended to perform impact analyses intuitively. Despite this common practise, many software engineers do not predict the complete change impact [Lindvall and Sandahl 1998]. Lindvall *et al.* further suggest that software impact analysis of embedded software systems is likely to be more difficult than non-embedded

change prediction because of the additional semantic and time-sensitive dependencies involved in embedded software development and that these types of dependencies are rarely recorded and can cause hidden or knock-on side effects [Lindvall *et al.* 2003]. This tendency for the inaccurate prediction of the impact on the embedded software discipline when lacking the recording of dependencies and determined through intuition or experience may also cause similar difficulties in impact analysis at the system-software interface and contribute to the risks of project delays, elevated development costs, and design failures and errors. Although this field of literature indicates the lack of adoption of software impact analysis, this research does not address the contributing factors to this practise, particularly, at the interface of the high-level embedded software requirements with other domains, such as system design. By exploring the influences instigating the disuse of the impact analysis techniques at the system and embedded software interface, suitable measures to improve the interdisciplinary design practises can be taken and, thus, reduce the risks associated with redesign.

## 4. Empirical study results

Since the literature found only suggests that designers do not implement requirement-based impact analysis between system design and embedded software, this research conducted an empirical study at Rolls-Royce to gain insights into the prescribed versus the practised process and explore the influences on the misalignment of these processes at this interface.

### 4.1 System design and embedded software discipline interaction structure

The embedded software within a jet engine controller implements algorithms and logic interacting with mechanical actuators and electronic sensors to manage the thrust, check for failures, and ensure safe operation. Although engine software must implement these general functions, the control strategy may differ between engines, necessitating in design variations. Rolls-Royce organises separate engine design teams with similar disciplinary breakdown structures in order to develop several engine projects concurrently. Disciplinary teams within an engine project design specific engine components, such as the compressor, turbine, piping, control system (including hardware and embedded software), etc., and a high-level systems engineering team coordinates these disciplinary designs to meet the specifications of a particular engine project. Additional lower-level system design teams synchronise a section of the whole engine design. For instance, within the controls group, control system designers serve as the primary interface for the other mechanically driven project teams and project stakeholders, such as the airframe manufacturer and suppliers, as well as internally between the hardware and embedded software control disciplines. Although some of the embedded software designers may work directly with the hardware engineers to develop the on-board *operating system software*, a majority of the embedded software team only interfaces with the control system design group to develop the *application software*, which controls the functionality of the engine. Rolls-Royce essentially implements the V-model of system design between these control system and embedded software groups; the control system designers write the software system requirements, which are passed to the embedded software team, initiating the detail design work. The following discussion focuses on the interaction of the control system designers and embedded software developers for the application software design during an engine project.

### 4.2 The practised design iteration or change process

Rolls-Royce operates a concurrent engineering framework within engine projects; in other words, the design of the embedded software occurs at the same time as the mechanical and hardware component designs. Change is inherent to such a process. As disciplines negotiate the details of a design, each domain may have to adapt to the design being produced by another team. The control group manages design changes through an informal change control process early in the detail design and a more formal change control process after a baseline design has been set. Both of these processes catalogue and require a review of proposed design changes, which may require either rework or the addition of a new feature. Only upon acceptance of a proposed design change by the change control board may the implementation of the change occur. Figure 3 illustrates the change implementation process in the simplest case of redesign between the system and embedded software disciplines.

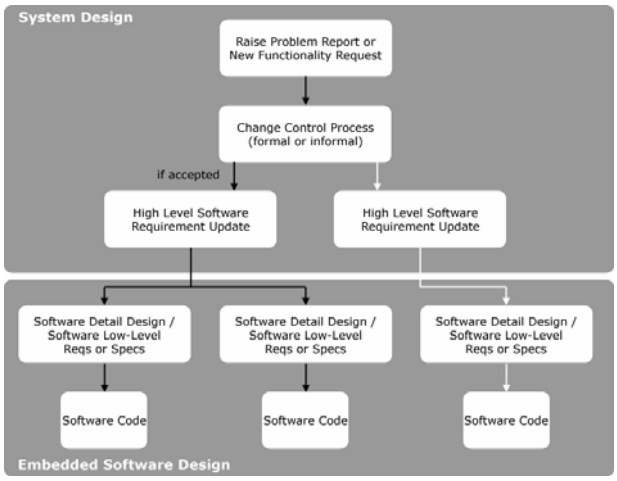


**Figure 3. The practised change implementation process – a change may cause a cascade of rework of multiple requirements and the detail design**

If the change control board accepts a design change, the system designers will first update the system requirement documents related to the software design. After receiving these revised or new requirements and models, the embedded software designers may change their various detail design specifications, also known as low-level requirements, and models and finally modify the related modules of code. However, the timing of the requirement and code updates within software may vary; the code may be updated before the requirements are formally agreed. Although not depicted in figure 3, the embedded software designers may consult with the system designers to iterate on the modified requirements for clarification on the desired functionality. At the end of this redesign process, the requirements, models, and code should be synchronised with the initially proposed design change.

As suggested in figure 3, a single change request may cause multiple system requirement changes and multiple code modules to be reworked. Although the change control processes require an assessment of the impact of the initiating, proposed change by the system designers assigned to the initially affected functionality, the process of assessing the functional impact of such a design change is not documented or performed consistently by system designers across any engine project; particularly, no requirement-based impact analysis is performed. The system designers tend only to use their experience and knowledge of the system rather than any systematic method to brainstorm on the impact on the system requirements given to embedded software, as suggested in the literature review.

This method of impact analysis may identify all the system design areas affected by the change. However, if the system designers misjudge the system requirement impact of a proposed change during the initial assessment, a series of early requirement reviews of the system requirements and embedded software detail design may catch the error. Design errors may also be found later during the various software, hardware, and engine testing procedures and rarely emerge during flight operation.

### 4.3 The effects and breakdown of the practised impact analysis process

Based on historical data from several large and small engine projects over the last decade, most functional failures and errors of the embedded software can be traced back to errors in the system requirements. Given that most of these system requirements are reworked to some extent suggests that requirement errors frequently occur during modification. Although some of these errors might be identified during the review and low-level embedded software testing process, a significant number of these errors are found during rig and engine testing, when the control system is integrated with the hardware and mechanical parts, respectively. If an error is found during this integration testing phase, the fix to the error typically is much more costly than finding it earlier during design. Subsequently, the work to remove system requirement errors accounts for a substantial amount of the embedded software development cost for engine projects. Moreover, knock-on effects, initiated by the incorrect

assessment of the impact of a change, produce almost a third of all of these requirement errors. Although some requirement errors arguably may be difficult to prevent by the control system design discipline due to erroneous or lack of information from stakeholders, knock-on requirement errors, generated by system design, are theoretically preventable through rigorous change process implementation and impact analysis techniques.

Despite the lack of implementation of requirement-based impact analysis techniques, the review and testing process should reduce the number of these system requirement errors passed to embedded software. If a system designer fails to recognise a knock-on effect based on their knowledge and experience, the review process should identify the error early on in the redesign process and have minimal associated rework. Should this review, for whatever reason, also not be carried out effectively, the breakdown of the prescribed design process contributes to the cost of these knock-on requirement errors. The software implementation may unknowingly design to erroneous requirements that slipped through the review process. Although the low-level software testing may catch any implementation or coding errors, requirement errors and knock-on effects tend to be found later during the "requirement testing" or engine testing since the testing scheme is often developed directly from the level of written requirements or specifications, as described by the V-model. This redesign due to requirement errors tends to be more costly than the rework of coding errors. Thus, the misjudgement of the side effects caused by a modification ripples through the design process and contributes to the observed high percentage of costs due to change, as also suggested in literature.

### 4.4 The perceived barriers to the prescribed systems engineering impact analysis process

The control project group studied implements a systems engineering top-down requirement traceability management software tool. This software serves as a means to track design requirements in order to prove compliance to the standards for safety-critical software certification. Although the tool allows for requirement-based impact analysis at the system and software interface, the system designers resist using this feature based on the perception of two primary barriers. Firstly, this style of impact analysis is not required within the published, prescribed design process used by the control design teams. Consequently, the systems engineers may not implement or realise that such an analysis technique exists. Secondly, the requirement and traceability relationships are not necessarily complete or up-to-date during the detail design phase of the project. During the beginning of the design within the concurrent engineering framework, not all high-level system requirements may be agreed to by the various stakeholders. Meanwhile, the low-level embedded software requirements and design has already begun in order to meet the target schedules developed to design engines efficiently. Thus, the traceability relationships among requirements will not exist since some of the requirements do not exist, preventing a top-down requirement analysis. Furthermore, as the system design progresses and implements numerous embedded software design changes, the requirement traceability inherently may not always be completely accurate since the changes are not executed instantaneously, which is at least partially due to the requirement configuration management procedures. The coordination of multiple changes within the fast-paced style of concurrent engineering must be addressed. Although a rigorous adherence to a top-down requirement redesign process could potentially correct for this lag in traceability relationships, the designers tend to mistrust complete loyalty to such a process across the system and embedded software teams based on their experience with the breakdown of the current processes. The culmination of these perceived barriers has paused the implementation of the requirement-based impact analysis tool within the company.

The designers suggest several improvements within the system and embedded software groups could allow for the support of a process implementing this impact analysis technique. They propose: improving training on writing unambiguous, testable requirements of uniform style; tailoring the implemented design processes to prevent the urge to "dive into the detail" or "fire fight" individual changes and promote initially hunting for all side effects; better balancing the resources between the "softer" requirement side of the deliverable and the "hard" system models and software code, especially for late software changes; and, enhancing the capture process of informal communication regarding system and software interactions between engineers that contribute to the formal design rationale, requirement documentation, or traceability relationships in real-time. The incorporation of

these suggestions into a defined process for impact analysis through requirements could potentially mitigate the preventable change costs caused by requirement knock-on effects.

## 5. Conclusion

The empirical study confirms the suggestion by the literature review regarding the lack of requirement-based impact analysis adoption by industry at the system and embedded software interface and indicates that the lack of a rigorous change impact analysis process at this interface can have a significant effect on the design process, as illustrated by the high software development change costs observed. This research has outlined a gap in the research concerning the influences contributing to the disparity between the prescribed and practiced impact analysis process and has indicated several perceived barriers to the adoption of the prescribed impact analysis technique within a single company and industry sector. Although the study may indicate several generic factors accounting for this difference and suggest areas for potential process improvement, further studies in different industry sectors are being conducted to substantiate discussion and the development of theory.

### Acknowledgement

The authors wish to acknowledge the support of Rolls-Royce plc.

### References

*Blanchard, B., "System Engineering Management", John Wiley & Sons Hoboken New Jersey USA, 2004.*

*Bohner, S. and Arnold, R., "Software Change Impact Analysis", IEEE Computer Society Press Los Alamitos California USA, 1996.*

*Clark, E. K., Forbes, J. A., Baker, E. R. and Hutcheson, D. W., "Mission-Critical and MissionSupport Software: A Preliminary Maintenance Characterization", Crosstalk: The Journal of Defense Software Engineering, Vol. 12, 1999, pp. 17-22.*

*INCOSE, "Systems Engineering Handbook", INCOSE, 2004.*

*Kossiakoff, A. and Sweet, W., "Systems Engineering Principles and Practice", John Wiley & Sons Hoboken New Jersey USA, 2003.*

*Lindvall, M., Komi-Sirviö, S., Costa, P. and Seaman, C., "A State of the Art Report: Embedded Software Maintenance", Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland, College Park Maryland USA, 2003.*

*Lindvall, M. and Sandahl, K., "How Well do Experienced Software Developers Predict Software Change?", Journal of Systems and Software, Vol. 43, No. 1, 1998, pp. 19-27.*

*Maier, M. and Rechtin, E., "The Art of Systems Architecture", CRC Press LLC Boca Raton Florida USA, 2000.*

*Sheard, S., Lykins, H., Armstrong, J., "Overcoming Barriers to Systems Engineering Process Improvement", Systems Engineering, Vol. 3, 2000, pp. 59-67.*

*Taramaa, J., Seppanen, V. and Makarainen, M., "From Software Configuration to Application Management - Improving the Maturity of the Maintenance of Embedded Software", Software Maintenance: Research and Practice, Vol. 8, 1996, pp. 49-75.*

Malia Kilpinen
PhD Research Student
Univeristy of Cambridge, Engineering Design Centre
Trumpington Street, Cambridge, United Kingdom
Tel.: +44 1223 332828
Email: mk432@cam.ac.uk