Chapter 6
Product-Line Models to Address Requirements Uncertainty, Volatility and Risk

Zoë Stephenson, Katrina Attwood and John McDermid

Abstract Requirements uncertainty refers to changes that occur to requirements during the development of software. In complex projects, this leads to task uncertainty, with engineers either under- or over-engineering the design. We present a proposed commitment uncertainty approach in which linguistic and domain-specific indicators are used to prompt for the documentation of perceived uncertainty. We provide structure and advice on the development process so that engineers have a clear concept of progress that can be made at reduced technical risk. Our contribution is in the evaluation of a proposed technique of this form in the engine control domain, showing that the technique is able to suggest design approaches and that the suggested flexibility does accommodate subsequent changes to the requirements. The aim is not to replace the process of creating a suitable architecture, but instead to provide a framework that emphasises constructive design action.

6.1 Introduction

In a conventional development process, a requirements writer creates an expression of his needs, to be read by a requirements reader. The requirements reader then creates a system to meet that need [1]. A particular problem faced by developers is the gradual squeezing of implementation time as deadlines become tighter and requirements are not fully agreed and validated until late in the development programme. Our experience with product-line modelling and architecture suggests that an uncertain requirement may be treated as a miniature product line, albeit one that varies over time rather than between different products. In the ideal case, this will derive flexibility requirements that help to accommodate subsequent fluctuation in the requirements, allowing the engineer to commit to the design even though the requirements are still uncertain. In this chapter, we take inspiration from uncertainty analysis, product-line engineering and risk management to synthesise an approach that provides insight into the flexibility needs of implementations driven by uncertain requirements.

Our approach, which is described in detail in Section 6.2 below, has two stages. Firstly, we use a combination of lightweight linguistic indicators and domain expertise to identify potential uncertainties in natural-language requirements, either in terms of the technical content of a requirement or of its precise meaning. We then use these uncertainties as the basis of a process of restatement of the original requirement as one or more ‘shadow’ requirements, which take account of changes which might arise in the interpretation of the original as the development proceeds. Each ‘shadow requirement’ is tagged with an indication as to the likelihood of its being the actual intended requirement. System design can then proceed with a clearer understanding of the risk of particular implementation choices. We refer to the approach as ‘commitment uncertainty’, to reflect the trade-off between requirements *uncertainty* and the need for (a degree of) design *commitment* at an early stage in the development process.

Our claim is that our particular choice of techiques provides up-front information about flexibility needs such that the resulting implementation is better able to cope with the eventual agreed requirements. Our contribution includes details of the choice of techniques, their specialisation and evaluation of the effectiveness of the resulting approach.

The remainder of the introduction provides a review of the areas of literature that inspire this work. Then, in the following section, we describe our approach to commitment uncertainty, followed by an evaluation of the approach in two separate studies.

6.1.1 Product-Line Engineering

Product-line engineering enables the provision of flexibility constrained to a particular scope [2]. The principal processes of product-line engineering are domain engineering, in which the desired product variation is modelled and supported with reusable artefacts and processes; and application engineering, in which the predefined processes configure and assemble the predefined artefacts to create a particular product [3]. The ability to rapidly create products within the predefined scope offsets the up-front cost of domain engineering, but it relies on a high degree of commonality between products [4] to reduce the size and complexity of the product repository.

Several of the technologies specific to product-line engineering are useful when dealing with uncertainty. Feature models [5] provide a view of the configuration space of the product line, documenting the scope of available products and controlling the configuration and build process. These models present a simple selection/dependency tree view of the underlying product concepts [6]. Topics in feature modelling include staged configuration [7] in which a process is built around partially-configured feature models that represent subsets of the available products, and feature model semantics [8] in which the underlying propositional structure of the feature model is examined. The former is useful in uncertainty analysis as it provides a way for the design to track the gradual evolution of changes to the requirements; the latter is problematic because it generally normalises the propositional selection structure into a canonical form. Such a normalised model could make it difficult to precisely identify and manage variation points that exist because of uncertainty.

Domain-specific languages [9] often complement feature models; while a feature tree is good for simple dependencies and mutual exclusions, a domain-specific language is better able to cope with multiple parameters with many possible values. Domain-specific languages are typically used along with automated code generation and assembly of predefined artefacts [10]. Given suitable experience and tool support for small-scale domain-specific languages, it may be feasible to use such approaches in making a commitment to a design for uncertain requirements.

In addition to these techniques, some more general architectural strategies are often used with product-line engineering, and would be suitable candidates for design decisions for uncertain requirements. These include explicitly-defined abstract interfaces that constrain the interactions of a component; decoupling of components that relate to different concerns; and provision of component parameters that specialise and customise components to fit the surrounding context [11].

6.1.2 Requirements Uncertainty and Risk

Requirements uncertainty is considered here to be the phenomenon in which the requirement as stated is believed by the requirements reader not to be the requirement that is intended by the requirements writer; that once the system is delivered, the requirements writer will detect the discrepancy and complain about the mismatch [12]. In dealing with requirements uncertainty, approaches take into account both organisational and linguistic concerns.

Uncertainty is considered to be present throughout a project, and presents a risk both to that project and to the organisation as a whole [13]. To deal with uncertainty from an organisational perspective, it must be identified, managed and controlled (see e.g. Saarinen and Vepsäläinen [14] for a more complete overview of risk management in this area) in tandem with other technical and programme risk activities. More pragmatically, in a study of U.S. military projects, Aldaijy [15] identified a strong link between requirements uncertainty and task uncertainty. That is, when faced with the prospect that the requirements may change, engineers often lack clarity on what tasks to perform.

Techniques to deal with uncertainty vary widely in their nature and scope. A significant body of work is aimed at linguistic techniques, to control language and identify problems in ambiguous requirements for feedback to the requirements writer [16,17]. From this literature, we recognise the important trade-off between the “weight” of the language analysis and the effort involved in obtaining useful information, as evidenced by the use of lightweight techniques that only use a shallow parse of the requirements [18] or targeted techniques that ignore ambiguities that are easy to detect in favour of highlighting those that are difficult to work with [19].

Once uncertainty is detected, it should at a minimum be recorded, e.g. as a probability distribution [20]. While uncertainty remains, there is an increased possibility of the requirements being inconsistent; this should be respected and maintained throughout the lifecycle and only forced to be resolved when necessary [21]. A further step is to use the information in negotiating for clearer requirements with the requirements writer, a strategy that is coloured by the project's conceptualisation of the requirements writer (e.g. as explained by Moynihan [22]).

6.2 The Commitment Uncertainty Process

Our approach to requirements uncertainty assumes that it is possible to explicitly analyse requirements and context for potential future changes and provide insight into the design phase so that the design accommodates those changes. In this respect, the approach is identical to a conventional product-line engineering approach (albeit with temporal variation rather than population variation) and is similar in its structure to other requirements-uncertainty approaches that aim to influence the design directly rather than waiting to negotiate further with the requirements writer. For example, Finkelstein and Bush report [23] on an uncertainty approach that considers scenarios in different versions of future reality as a basis for stability assessment of goal-based requirements representations. Within this structure of suggesting derived requirements to control flexibility, we use a classification of requirements language issues that is broadly similar to the checklist decomposition proposed by Kamsties and Paech [17]. In addition to classifying the problem in the requirement, we also classify the situation of the requirements writer that might have led to the uncertainty, building on the insights found in Moynihan's requirements-uncertainty analysis of IS project managers [22]. Finally, we introduce one important terminological distinction: in addition to requirements uncertainty, which is associated with the problems in communicating the requirement, we also refer to requirements volatility, the change that could occur to the requirement.

6.2.1 Process Overview

The commitment uncertainty process is shown in Figure 6.1. In sub-process p_1 we examine requirements for indicators of uncertainty, taking into account both the requirement and its assumed development context. The techniques employed in sub-process p_1 are described in detail in Sections 6.2.2 and 6.2.3 below. In p_2 we create flexibility requirements by factoring the uncertainty specification into volatile and non-volatile parts. This sub-process is detailed in Section 6.2.4. Finally, in Section 6.2.5, we describe sub-process p_3, in which we suggest possible implementation strategies for the requirements based on a predefined prompt list for the scope of the product in which the volatility lies.

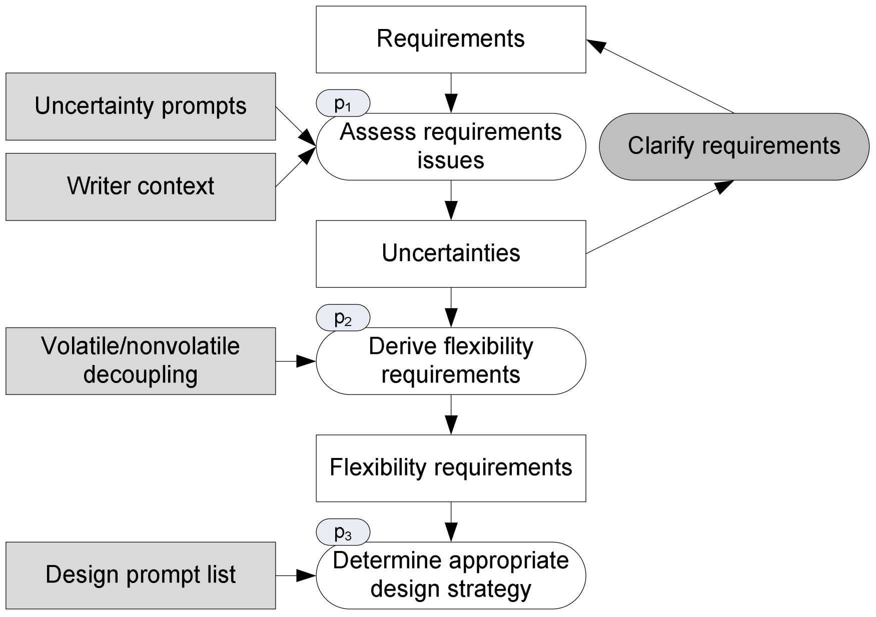


Fig. 6.1. Overview of the commitment uncertainty process. Specific materials on the left-hand side are explained in this chapter. The clarification process on the right-hand side is outside of the scope of this chapter.

6.2.2 Requirements Uncertainty Prompts

The checklist for requirements issues is as shown in Table 6.1. The list contains issues that are related to the linguistic structure of the requirement as well as issues that relate to the technical content of the requirement. In this analysis technique, we recommend an explicit record of uncertainty, linking to relevant supporting information, to enable effective impact analysis. This is similar to the use of domain models and traceability in product-line engineering, and is essential to effective uncertainty risk management.

In practice, a single requirement may include many forms of uncertainty, which may interact with one another. Rather than trying to classify all such interactions, the analysis prompts the reader into thinking about the requirement from different standpoints. In some situations, no particular reason for the uncertainty will emerge. A pragmatic trade-off must be made between the effort of explaining the uncertainty and the costs saved by performing the analysis.

As an example, consider this sample requirement:

110. The system shall provide an indication of the IDG oil temperature status to the aircraft via ARINC.

This requirement suffers from (at least) two different uncertainties. Firstly, the phrase "an indication of the IDG oil temperature status" contains words ("indication", "status") that are either redundant (meaning 'provide the IDG oil temperature') or poorly-defined (meaning to get the status of the oil temperature, and then transmit an indication of the status).

In all likelihood, the real requirement is the following:

110a. The system shall provide its measurement of the IDG oil temperature to the aircraft via ARINC.

The second issue is that the requirement does not directly identify where to find out information about the message format or value encoding. The link could be written into the requirement:

110b. The system shall provide its measurement of the IDG oil temperature to the aircraft via ARINC according to protocol P100-IDG.

It is more likely that the link would be provided through a tool-specific traceability mechanism to an interface control document.

6.2.3 Identifying Requirements Writer Context

We recognise that there is also value in trying to understand the context within which a requirement is written; given a set of such requirements, it may be possible to infer details of the context and hence suggest specific actions to take to address uncertainty. Table 6.2 explains possible reasons for problems appearing in requirements. Rather than capturing every requirements issue, we instead present possible issues for the engineer to consider when deciding on how much information to feed back to the requirements writer and when. We make no claim at this stage that the table is complete; however, it covers a number of different aspects that might not ordinarily be considered, and on that basis we feel it should be considered at least potentially useful.

In practice, it will rarely be possible to obtain a credible picture of the requirements writer context. Nevertheless, it can be useful to consider the possible context to at least try to understand and accommodate delays in the requirements. Explicitly recording assumptions about the writer also facilitates useful discussion among different engineers, particularly if there are actually multiple issues behind the problems in a particular requirement. Finally, the overall benefit of this identification step is that it gives clear tasks for the engineer, reducing the so-called "task uncertainty" and improving the ability to make useful progress against the requirements.

Table 6.1. Uncertainty indicator checklist.

Area	Uncertainty	Form
Incompleteness	Unfinished requirement	A part of the requirement has not yet been written. There could be a trailing unfinished clause or an ellipsis. There will usually be little to indicate how to fill the gap.
	Placeholders	A placeholder is used for part of the requirement. This is typically some metasyntactic expression – perhaps "TBD" or "[upper limit]". In some organisations, placeholders may be given as information paragraphs or marked-up notes.
	Missing counterpart	In many cases, requirements come as a set. For example, there may be a startup requirement for each mode of a system. Even with little domain knowledge, it should be apparent when part of the set is missing.
Ambiguity	Under-specification	An underspecified requirement constrains the implementation to some extent, but leaves options open. In some cases, this is a careful abstraction to avoid over-constraining the implementation. In others, the requirement is simply not concrete enough.
	Terminology	Some terminology in the requirement is not well-defined; it needs qualification, better definition or replacement.
	Syntactic structure	The sentence has a structure that can be read in more than one plausible way. This areas is well-studied in linguistic approaches to requirements ambiguity [18,24]
Commitment	Incorrectness	There is some detail in the requirement that is demonstrably incorrect, through a scenario or some logical inference.
	Overspecification	The requirement includes more detail than necessary, giving awkward or infeasible constraints.
Mislabelling	Misplaced requirement	The positioning of the requirement (section heading, informative context) conflicts with its content.
	Mislabelled domain information	The statement is presented as a requirement but it contains only definitions, therefore technically not requiring any action.

6.2.4 Recording and Validating Uncertainties

To record the uncertainty in a way that is useful to all interested stakeholders, we advocate a multi-stage recording process shown in Figure 6.2. First, the original requirement and uncertainty analysis are identified. Then, the requirement is re-stated by identifying volatile parts and presenting a miniature product line view of the requirement. The exact process here is one of engineering judgement based on the form of the uncertainty and the reason behind it; the interested reader is referred to a more comprehensive work on product lines (e.g. Bosch [3] or Weiss and Lai [9]) for further elaboration.

With the requirement volatility captured, the requirement is then restated into the same form as the original requirement. This is the shadow requirement, and represents what the engineer will actually work to. The final step is to double-check the result by checking that the original requirement, as stated, is one possible instantiation of the shadow requirement.

Consider the sample requirement 110 again:

110. The system shall provide an indication of the IDG oil temperature status to the aircraft via ARINC.

The volatility might be specified as follows:

Stable part:
- Sending data over ARINC, part of overall ARINC communications.
- Sending IDG oil temperature or derived values, depending on reading or synthesising value

Volatile part:
- Data to be sent
- Message format / encoding

Likely changes:
- Data to be sent is one or more of:
- IDG oil temperature reading
- IDG oil temperature limits, rate of change
- Details of faults with that reading
- Presence/absence of faults
- Details of current sensing method
- Value encoding will depend on data to be sent

Instantiations:
110a. The system shall provide its measurement of IDG oil temperature to the aircraft via ARINC.
110b. The system shall provide a count of current IDG oil temperature faults to the aircraft via ARINC.
110c. The system shall provide IDG oil temperature, operating limits and rate of change to the aircraft via ARINC.

One possible shadow requirement for the IDG oil temperature requirement would be:

110x. The system shall provide (IDG oil temperature|IDG oil temperature faults|IDG oil temperature presence/absence of faults|IDG oil temperature, operating limits and rate of change) to the aircraft via ARINC [using protocol (P)].

An alternative approach is to derive explicit flexibility requirements to guide the implementation:

F.110a. The impact of changes to the feature of the IDG oil temperature to send on the delivery of IDG information in requirement 110 shall be minimised.

F.110b. The impact of changes to the communication format on the delivery of IDG information to the aircraft shall be minimised.

Table 6.2. Writer context checklist.

Reason	Details
Novelty	The requirements writer, and perhaps also the reader, is unfamiliar with this area of requirements. This is either slowing the writer down (unfinished requirements) or causing premature commitment (incorrect requirements). Uncertainty should decrease over time. Detailed feedback on the requirements may be unwelcome early on.
Complexity	The requirements specify something complex, and the difficulty of dealing with the complexity leads to unfinished, ambiguous or conflicting requirements. They may also be copied from previous requirements to reuse a successful structuring mechanism.
Concurrent Delay	The requirements writer has yet to perform the work towards the requirement; the details depend on the results of processes that are incomplete. This often happens in large projects with multiple subsystems and complex interfaces. Eventually, the requirement will be properly defined. In this case, feedback is likely to be welcome.
Pressure	The requirements writer is under pressure; the final requirement has not yet been defined. This could arise from areas such as inter-organisational politics, financial arrangements or resourcing. Feedback to suggest clarifications may not be effective.
Language Gap	The requirements writer uses language differently to the requirements reader. He may be writing in a non-native language but using native idioms and grammar, or he may apply the rules of the language differently to the expectation of the reader. This subject is studied at length in linguistic approaches to requirements [16].
Context Gap	The uncertainty arises from the difference in information available to the reader compared to the writer. The writer may make assumptions that are not made explicit, or the reader may know more about the target platform. It will be useful to document explicit assumptions to help support decision-making.
Intent Gap	It may be unclear how the requirements writer intends to constrain the implementation. This can occur with abstractions; distinguishing between deliberate and accidental generality can be difficult. Another possibility is a set of “soft” requirements to trade off, presented as hard constraints. This may happen if the contract does not allow for appropriate negotiation. Issues such as these indicate that extra effort in design trade-offs and flexible architecture may be appropriate.
Medium	The communications medium between writer and reader may constrain the information that may be represented in a cost-effective way. The medium includes the tools and experience available to the writer and reader. For example, the reader may be able to read text and diagrams, but not view links between the two.

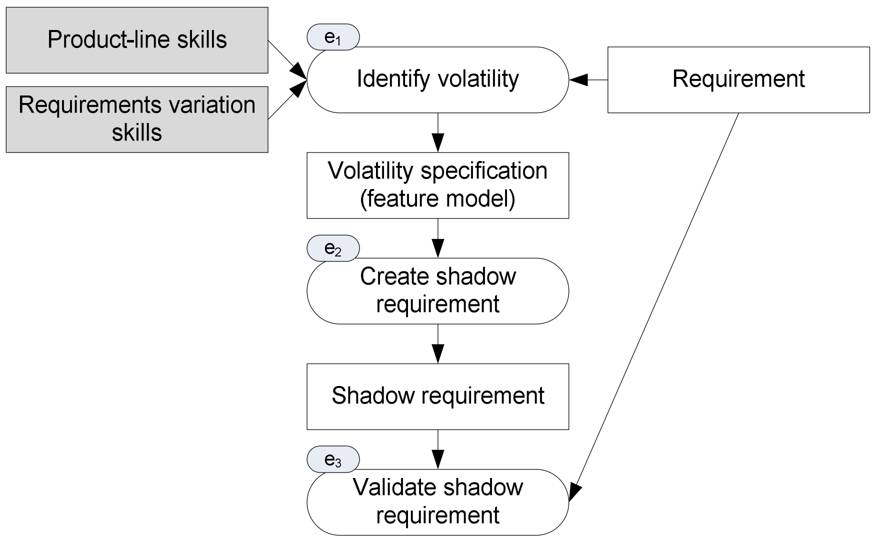

Product-line skills
Requirements variation skills
e1
Identify volatility
Requirement
Volatility specification (feature model)
e2
Create shadow requirement
Shadow requirement
e3
Validate shadow requirement

Fig. 6.2. Overview of deriving flexibility requirements. Volatility (expected changes) is specified and then factored into the requirement to create a family of related requirements

6.2.5 Suggesting Design Approaches

It is not intended that the commitment uncertainty approach should constrain the type of implementation chosen to accommodate the identified uncertainty. Nevertheless, it is useful to give advice on the type of design approach that is likely to be successful, as a way to further overcome task uncertainty and improve the likelihood of quickly arriving at a suitable design.

The advice is based on a recognition that a design approach will typically respond to a group of requirements. We take as input the scale of the volatility in that group of requirements and produce a suggested list of design approaches to consider, in a particular order. The intent is not to restrict the engineer to these design approaches, nor to constrain the engineer to select the first approach for which a design is possible; the aim is simply to help the engineer to quickly arrive at something that is likely to be useful.

Our approach is therefore much coarser than more considered and involved approaches such as those of Kruchten [11] or Bosch [3]. The mapping is shown in Table 6.3. In this table, the scale of design volatility is broadly categorised as “parameter” when the volatility is in a single parameterisable part of the requirements; “function” when the behaviour changes in the volatile area; and “system” when the volatility is in the existence or structure of a whole system. The engineer is encouraged to choose whichever of these designations best matches the volatility, and then use his existing engineering skills to arrive at designs that are prompted by the entries under that heading: “Parameterisation” to include suitable data types and parameters in the design; “Interfaces and Components” to consider larger-scale interfacing and decoupling; and “Auto-Generation” to build a domain-specific language or component configuration scripting system to accommodate the volatility.

Table 6.3. Mapping from volatility scale to suggested design approaches.

Parameter	Function	System
Parameterisation	Interfaces and Components	Interfaces and Components
Interfaces and Components	Parameterisation	Auto-Generation
Auto-Generation	Auto-Generation	

6.2.6 Ordering Design Decisions

In software, major design decisions are traded off and captured in the software architecture; functionality is then implemented with respect to this architecture. In some complex design domains, however, there are multiple competing design dependencies that can be difficult to resolve. To assist in making progress in this context, we provide a framework that tracks design dependencies and resolves design decisions hierarchically to produce the complete design. The intended effect is that the areas that are most volatile are those that are least fundamental to the structure of the design. This technique makes use of product-line concepts to represent optionality.

In addition to dependencies between design decisions and (parts of) requirements, any part of a design may depend on part of an existing design commitment. This includes both communicating with an existing design element and reusing or extending an existing element. These dependencies are the easiest to accommodate with indirection and well-defined interfaces. Contextual domain information is also important, and most design commitments are strongly related to domain information. The dependency on domain information can be managed through parameterisation or indirection. The process of uncertainty analysis provides additional exposure of contextual issues, helping to reduce the risk associated with missing context.

In our prototype modelling approach, we explicitly represent dependencies between design elements in a graphical notation, shown in Figure 6.3. This example represents the decision to add the IDG oil temperature parameter to the ARINC table as defined in the interface control document. The decision is prompted by the requirement to send the information, the availability of the information, and the availability of a suitable communication mechanism. The result of the decision is a new entry in a data table to allow the communication of the appropriate value. While this example is perhaps trivial, it illustrates the important distinction between decisions (processes that the user may engage in) and designs (the artefacts that result from design activity).

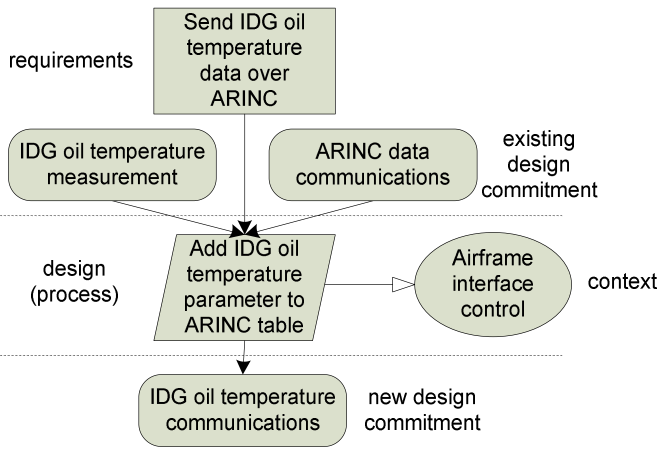
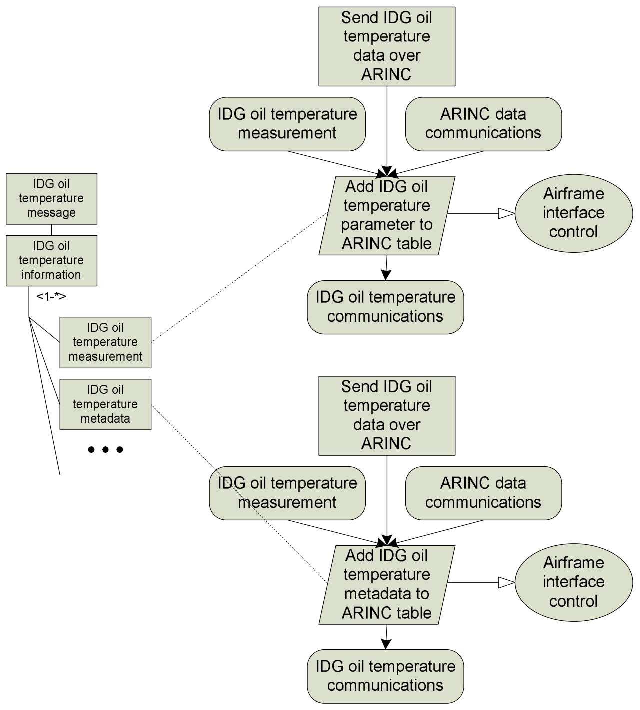

requirements
Send IDG oil temperature data over ARINC
IDG oil temperature measurement
ARINC data communications
existing design commitment
design (process)
Add IDG oil temperature parameter to ARINC table
Airframe interface control
context
IDG oil temperature communications
new design commitment

Fig. 6.3. Representing dependencies between design elements.

Commitment uncertainty analysis associates volatility with context, requirements and design. This may be annotated alongside the decision tracing diagram. To retain familiarity and compatibility with existing approaches, we base this representation on conventional feature modelling notations, as shown in Figure 6.4. A feature model view of design decision volatility is a powerful visual tool to help appreciate the impact of volatility on the design approach. It is expected that this type of visualisation will be of most benefit when communicating involved technical risk deliberations to interested stakeholders.

Once a body of design decisions has been produced, individual design commitments may be resolved together to create design solutions. The granularity of resolution is not fixed; it will depend on the complexity of the design decisions, their dependencies and their volatility. Similarly, the order of design decisions is not fixed. Any design decision that changes will have an impact on later design decisions, so the intent of design decision resolution is to defer more volatile decisions as late as possible in the decision sequence. Ordinarily, the design decision sequence is chosen by creating a partial order from the design decision dependencies, adding volatility information and then creating a total order from the resulting dependencies. When adding the volatility information, it is important to start from the most volatile decision and then descend in order of volatility, adding volatility relationships where they do not conflict with existing relationships.



Send IDG oil temperature data over ARINC
IDG oil temperature measurement
ARINC data communications
Add IDG oil temperature parameter to ARINC table
Airframe interface control
IDG oil temperature communications
IDG oil temperature message
IDG oil temperature information
<1-*>
IDG oil temperature measurement
IDG oil temperature metadata
Send IDG oil temperature data over ARINC
IDG oil temperature measurement
ARINC data communications
Add IDG oil temperature metadata to ARINC table
Airframe interface control
IDG oil temperature communications

Fig. 6.4. Representing volatility with design decisions. The annotation on the left-hand side shows features (rectangles) with dependencies (arcs) and selection constraints (arc label <1–*> in this example)

In extreme cases, some reengineering will be needed to arrive at an appropriate design. For example, it may be advantageous to break a design dependency in order to accommodate a volatility dependency. This will typically prompt a refactoring of the existing commitments to accommodate the additional variation from the volatile design and allow for the design dependency using dependency injection and inversion of control.

6.3 Empirical Evaluation

6.3.1 Quantitative Analysis of Effectiveness

In this section, we present the design and results of an experiment to test the theoretical effectiveness of the commitment uncertainty approach. For this experiment, we used four instances of the requirements from an engine controller project; a preliminary (draft) document *P* and issued requirements I_1, I_2 and I_3, from which we elicited changes made to the requirements over time. Since the requirements in this domain are generally expressed at a relatively low level – particularly with respect to architectural structure – we consider that the requirements are, for the purposes of this experiment, equivalent to the design.

In the experiment, we created two more flexible versions of *P*: P_t using conventional architectural trade-off techniques, and P_u using commitment uncertainty techniques. The hypothesis is that, when faced with the changes represented by $I_{1\text{–}3}$, P_u accommodates those changes better than P_t, and both P_u and P_t are better at accommodating changes than *P*. We consider a change to be accommodated if the architecture of the design provides flexibility that may be used to implement the required change. Table 6.4 shows the data for P_u, and Table 6.5 is the equivalent data for P_t. In each table, the **ID** column gives the associated requirement ID, then the **Scope** column identifies whether the requirement was in scope for commitment uncertainty and the **Derived** column indicates whether a derived requirement was produced from the analysis. The remaining columns evaluate the two sets of requirements – the original set and the set augmented with derived requirements from the additional analysis. At the bottom of the table, the totals are presented as both raw totals and a filtered total that excludes the requirements that were outside the scope of architectural flexibility provision.

Table 6.4. Uncertainty analysis on randomly-selected requirements.

ID	Scope	Derived	I_1-P	I_1-P_u	I_2-P	I_2-P_u	I_3-P	I_3-P_u
6	Y	Y	Y	Y	Y	Y	Y	Y
20	Y	Y	Y	Y	Y	Y	Y	Y
32	Y	Y	Y	Y	Y	Y	Y	Y
99	Y	Y	Y	Y	Y	Y	Y	Y
105	Y	Y	Y	Y	Y	Y	Y	Y
22	Y	Y	Y	Y	Y	Y	Y	Y
127	Y	Y	N	Y	N	Y	N	Y
5	Y	N	Y	Y	Y	Y	Y	Y
39	Y	Y	N	Y	N	Y	N	Y
49	Y	Y	Y	Y	Y	Y	N	Y
26	Y	Y	Y	Y	Y	Y	Y	Y
16	Y	Y	Y	Y	Y	Y	Y	Y
113	Y	Y	Y	Y	Y	Y	Y	Y
118	Y	Y	N	Y	N	Y	N	Y
119	Y	Y	Y	Y	Y	Y	Y	Y
50	Y	N	Y	Y	Y	Y	Y	Y
107	Y	Y	N	Y	N	Y	N	Y
56	N	N	Y	Y	Y	Y	Y	Y
12	Y	N	Y	Y	Y	Y	Y	Y
60	Y	Y	Y	Y	Y	Y	Y	Y
64	Y	N	Y	Y	Y	Y	Y	Y
70	Y	Y	Y	Y	Y	Y	Y	Y
71	Y	Y	Y	Y	Y	Y	Y	Y
110	Y	Y	N	Y	N	Y	N	N
2	N	N	Y	Y	Y	Y	Y	Y
73	N	N	Y	Y	Y	Y	Y	Y
76	Y	Y	Y	Y	Y	Y	Y	Y
123	Y	Y	N	Y	N	Y	N	Y
137	N	N	Y	Y	Y	Y	Y	Y
140	N	N	Y	Y	Y	Y	Y	Y
148	N	N	Y	Y	Y	Y	Y	Y
151	N	N	Y	Y	Y	Y	Y	Y
155	N	N	Y	Y	Y	Y	Y	Y
159	N	N	Y	Y	Y	Y	Y	Y
162	N	N	Y	Y	Y	Y	Y	Y
93	Y	Y	N	Y	N	Y	N	Y
94	Y	Y	N	Y	N	Y	N	Y
120	Y	Y	Y	Y	Y	Y	Y	Y
Y Total	28	24	30	38	30	38	29	37
Filtered	28	24	20	28	20	28	19	27

Table 6.5. Trade-off analysis on randomly-selected requirements.

ID	Scope	Derived	I1-P	I1-Pt	I2-P	I2-Pt	I3-P	I3-Pt
19	Y	Y	Y	Y	Y	Y	Y	Y
30	Y	Y	N	N	N	N	N	N
32	Y	N	N	N	N	N	N	N
102	Y	Y	N	N	N	N	N	N
106	Y	N	N	N	N	N	N	N
127	Y	N	N	N	N	N	N	N
10	Y	N	Y	Y	Y	Y	Y	Y
38	Y	N	Y	Y	Y	Y	N	N
48	Y	Y	Y	Y	Y	Y	Y	Y
131	Y	N	Y	Y	Y	Y	Y	Y
16	Y	Y	Y	Y	Y	Y	Y	Y
113	Y	Y	Y	Y	Y	Y	Y	Y
118	Y	Y	N	Y	N	Y	N	Y
42	Y	N	Y	Y	Y	Y	Y	Y
50	Y	N	Y	Y	Y	Y	Y	Y
81	Y	N	Y	Y	Y	Y	Y	Y
27	Y	N	N	N	N	N	N	N
62	Y	Y	Y	Y	Y	Y	Y	Y
67	Y	Y	Y	Y	Y	Y	Y	Y
71	Y	Y	Y	Y	Y	Y	Y	Y
111	Y	Y	N	Y	N	Y	Y	Y
3	Y	N	Y	Y	Y	Y	Y	Y
31	N	N	Y	Y	Y	Y	Y	Y
76	Y	Y	Y	Y	Y	Y	Y	Y
6	Y	N	Y	Y	Y	Y	Y	Y
56	N	N	Y	Y	Y	Y	Y	Y
109	N	N	Y	Y	Y	Y	Y	Y
78	N	N	Y	Y	Y	Y	Y	Y
136	N	N	Y	Y	Y	Y	Y	Y
142	N	N	Y	Y	Y	Y	Y	Y
148	N	N	Y	Y	Y	Y	Y	Y
151	N	N	Y	Y	Y	Y	Y	Y
154	N	N	Y	Y	Y	Y	Y	Y
160	N	N	Y	Y	Y	Y	Y	Y
88	Y	N	Y	Y	Y	Y	Y	Y
92	N	N	Y	Y	Y	Y	Y	Y
96	Y	Y	Y	Y	Y	Y	Y	Y
108	N	N	Y	Y	Y	Y	Y	Y
Y Total	26	13	30	32	30	32	30	31
Filtered	26	13	18	20	18	20	18	19

To analyse the hypothesis, we compare the ability of one design to accommodate change with the ability of another. This produces contingency tables, displayed in Tables 6.6–6.8.

The results indicate that the commitment uncertainty analysis made a significant improvement in the ability to accommodate change, while the more conventional trade-off analysis was not as capable. Some caveats should, however, be stated here. The significance measure here provides an indication of internal validity. In this particular engineering domain, the use of requirements to stand for designs is appropriate, since the design process is characterized by the iterative decomposition of requirements. We note, however, that our findings may not be applicable in other domains. In terms of the external validity of the study, it is important to note that our experiment differed from real-world practice in that we were external observers of the project, with access to the entire lifecycle history. In practice, it may be more difficult for interested parties to make appropriately flexible commitments early on in a project. It would be interesting to repeat the study in a live project, deriving flexibility requirements to which designers were prepared to commit their choices and then observing the degree to which these requirements proved useful in accommodating later change. It should also be noted that this work concerns an embedded software system, where there are considerable constraints on the design and implementation, and there are objective tests of system functionality and effectiveness. Design drivers in other domains may, of course, differ markedly: for example, a successful design might be one which opens up a new market or incorporates some innovative functionality. In these cases, the nature of the requirements and design processes are likely to differ markedly from the aerospace domain, and the effectiveness of our proposed approach may be less clear.

Table 6.6. Summary of uncertainty analysis against original requirements. Data show significant (χ^2,p<0.05) improvement over original system.

	Y	N
I_1-P	20	8
I_1-P_u	28	0

	Y	N
I_2-P	20	8
I_2-P_u	28	0

	Y	N
I_3-P	19	9
I_3-P_u	27	1

Table 6.7. Summary of trade-off analysis against original requirements. No significant improvemen**t.**

	Y	**N**
I_1-P	**18**	**8**
I_1-P_t	**20**	**6**

	Y	**N**
I_2-P	**18**	**8**
I_2-P_t	**20**	**6**

	Y	**N**
I_3-P	**18**	**8**
I_3-P_t	**19**	**7**

Table 6.8. Summary of uncertainty analysis against trade-off analysis. Data show significant (χ^2,p<0.05) improvement of uncertainty analysis against trade-off analysis.

	Y	N		Y	N		Y	N
I_1-P_u	28	0	I_2-P_u	28	0	I_3-P_u	27	1
I_1-P_t	20	6	I_2-P_t	20	6	I_3-P_t	19	7

6.3.2 Qualitative Evaluation of Design Selection

In this study, we investigated the ability of the design prompt sequence approach to correctly identify appropriate design targets for the implementation of uncertainty-handling mechanisms. The study is based on an internal assessment of technical risk across a number of engine projects conducted in 2008. We extracted 8 areas that had been identified as technical risks that were in scope for being addressed with architectural mechanisms. For each identified technical risk, we elicited uncertainties and then used the design prompt sequence to generate design options. Finally, we chose a particular design from the list and recorded both the position of the design in the list and the match between the chosen design and the final version of the requirements.

As an example, consider the anonymised risk table entry in Table 6.9. The individual uncertainties for this particular instance are elaborated and documented in a custom tabular format shown in Table 6.10.

The design prompts for “Function, Concurrent” are presented in the order, components/interfaces, parameterisation and then auto-generation.

The options identified are:

1. Use an abstract signal validation component with interfaces that force the designer to consider raw and validated input signals, power interrupts and signals for faults. This design ensures that each component encapsulates any uncertainty regarding its own response to power interrupt.
2. Use a parameterised signal validation component that selects its response to power interrupt from a list of possible responses. The list should be based on domain expertise and experience in designing robust power interrupt management schemes. This is applicable if the range of possible responses can be captured easily in such a list, and as long as the use of the response selection mechanism is consistent with certification guidelines for configurable components.
3. An auto-generation system may be appropriate for complex parameterisation. The input configuration is derived from the range of possible input validation responses to power interrupt. The input language to the auto-generation system should be easy to use and should be similar to other auto-generation input languages in use. For this option, the language itself captures and manages the uncertainty.

Assessment of these available choices shows that the abstract interface is the easiest to implement; the parameterisation and auto-generation approaches carry more specific details of the available interrupt-management schemes, which is not necessarily a net benefit at this time.

With such a small study, it is difficult to quantify the net benefit of the design prompt sequence; nevertheless, we feel it is useful to present some observations on the outcome of the study. The results are shown in Table 6.11.

Table 6.9. Example risk table entry.

Risk Area	Specific Risk	Particular Instances
Requirements – Flow-down	Have system-level requirements for reaction to power supply interrupts been decomposed into software requirements?	Project Hornclaw refit software.

Table 6.10. Custom tabular format for documentation of uncertainty.

Certainties		
After a power interrupt, the system initialises afresh and its RAM and program state no longer represent the state of the environment. The system can determine some information about the state of the environment from non-volatile memory. The only part of the system that will be out of sync after a power interrupt is input validation.		
Uncertainties		
Type	Definition	Rationale
Function Concurrent	Required signal validation after a power interrupt	Derivation from the technical risk concept “requiremens for reaction to power supply interrupts”

Firstly, the study showed that the prompts were able to suggest multiple design oprions for each technical risk area. Contrary to expectations, the first design alternative was not necessarily the alternative that was eventually chosen; moreover, the last design choice was never selected for use in this study. This suggests that it may not be directly beneficial to create too many different design choices, although there may be an indirect benefit from the comparison of the second design choice to the third choice in establishing its relative merit. It should also be noted that, by forcing designers to consider multiple alternative design solutions (contrary to their usual practice), the technique potentially reduces the danger of the “first plausible design syndrome”, whereby designers commit themselves to the first apparently workable solution, unwilling to move on from it even when problems are identified with the design. Put another way, this could be seen as a way of encouraging engineers to delay design commitments [25] which are viewed by some as underpinning lean processes [26].

Secondly, it was useful during the study to note the applicability of the design choices to communicate contextual assumptions from the design phase for validation when changes occur. For example, some design options could result in unused inputs once changes are made, which could impact on testability.

Lastly, we found that in some areas one uncertainty would lead on to further uncertainties. This was particularly the case in novel design areas, where an uncertainty structure arose from consideration of the suggested design alternatives. We expect that this is more closely related to a pattern-based approach to product-line feature modelling than directly to uncertainty analysis, and the phenomenon would merit closer study.

Again, we should express some caveats concerning the wider applicability of these observations. The study reported in this chapter is small, and, because of this, it is difficult to extrapolate its findings to a wider context. However, we do believe the construct validity to be appropriate − that the experiment is able to show, in principle, relationships between the scale of uncertainty and the type of design solution that is most appropriate to address the requirement. It should also be noted that the application domain is a very stable one: it is relatively easy to draw on previous experience to derive alternative design solutions. This may be more difficult in a less stable domain, although it may be that more useful alternative designs can in fact be identified in such environments. There is then a trade-off between the evaluation of alternatives and the technique's capacity, ultimately, to help designers in the derivation of better design solutions.

In terms of the practical application of the ideas presented here, it would be most appropriate to view the approach as a contribution to process architecture, perhaps in the context of product-line development or a framework such as BAPO [27]. Our approach should sit as one of a range of mechanisms that allow the designers to make commitments at appropriate points in the process.

Table 6.11. Outcome of qualitative design prompt sequence study.

Technical Risk Area	Uncertainties	Design Options	Chosen Design	Match to Final Requirements
new feature	2	3	1	estimated good
architecture scope	4	3, 3	2, 2	perfect
comprehensive requirements	7	3	none	n/a
comprehensive requirements	3	3	2	estimated good
incremental development	1	0	n/a	n/a
power interrupt requirements	1	3	1	estimated good
state transitions	2	4	1	perfect
new feature	2	3	2	good

6.4 Summary and Research Outlook

We have presented a synthesis of concepts from uncertainty, risk management and product lines to address the issue of requirements uncertainty that prevents the engineer from making a design commitment. The intended use of the technique is to rapidly suggest possible risk areas and highlight options for a lower-risk implementation that includes flexibility to accommodate particular variations from the requirement as stated. These prompts aim to inspire the engineer into creating a solution that is engineered for specific potential uncertainties, rather than forcing either a brittle implementation that cannot respond to change or an over-engineered solution that is difficult to manage over time.

In our evaluation of the technique, we found that there is significant potential for this type of analysis to suggest design flexibility that is warranted by subsequent changes between early project requirements and final issued project requirements. Several questions still remain unanswered, however. Most importantly, how much effort is involved in creating the derived requirements and flexible design versus the time taken to refactor the design at a later stage? It is this comparison that is most likely to persuade engineers that the technique has merit. In support of this, we emphasise the positive results that have been obtained so far and focus on the practical aspects of the approach – its lightweight nature and the ability to apply it only where immediate risks present themselves. Similarly, in how many cases is flexibility added to the design but never used later on? The presence of a large amount of unused functionality may be a concern particularly in the aerospace industry and especially if it prevents the engineer from obtaining adequate test coverage.

For future work in this area, we have identified four themes. Firstly, we are interested in integration of the concepts of commitment uncertainty into a suitable metamodelling framework such as decision traces [28] or Archium [29]. Second, it would be interesting to deploy appropriate tool support based on modern metamodelling [30]. Third, there is potential benefit in a richer linguistic framework to support more detailed uncertainty analysis and feedback to requirements stakeholders, and lastly, further experimentation is needed to understand the nature of appropriate design advice, design patterns and commitment-uncertainty metrics.

Acknowledgment The research presented in this chapter was conducted at the Rolls-Royce University Technology Centre in Systems and Software Engineering, based at the University of York Department of Computer Science. We are grateful to Rolls-Royce plc. for supporting this research.

References

1. Stokes D A (1990) Requirements Analysis. In: McDermid J (ed) Software Engineer's Reference Book, Butterworth-Heinemann, Oxford, UK
2. Schmid K, Gacek C (2000) Implementation Issues in Product Line Scoping. In: Frakes W B (ed) ICSR6: Software Reuse: Advances in Software Reusability. Proceedings of the Sixth International Conference on Software Reusability, Lecture Notes in Computer Science (LNCS), vol 1844, Springer, Heidelberg, Germany, pp170–189
3. Bosch J (2000) Design and Use of Software Architectures. Addison-Wesley, Reading, Massachusetts, USA
4. Weiss D, Ardiss M (1997) Defining Families: the Commonality Analysis. In: ICSE'97. Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, May 1997. ACM Press, New York, USA
5. Kang K, Cohen S, Hess J et al (1990) Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University Software Engineering Institute
6. Czarnecki K, Kim C H P, Kalleberg K T (2006) Feature Models are Views on Ontologies. In: SPLC2006. Proceedings of the 10th International Conference on Software Product Lines, Baltimore, Maryland, USA. IEEE Computer Society Press, Los Alamitos, California, USA
7. Czarnecki K, Helsen S, Eisenecker U. (2004) Staged Configuration Using Feature Models. In: Nord R L (ed) SPLC 2004. Proceedings of the 9th International Conference on Software Product Lines, Boston, Massachusetts, USA. Lecture Notes in Computer Science (LNCS), vol 3154, Springer, Heidelberg, Germany
8. Bontemps Y, Heymans P, Schobbens P-Y et al (2004) Semantics of FODA Feature Diagrams. In: Proceedings of the International Workshop on Software Variability Management for Product Derivation
9. Weiss D M, Lai C T R (1999) Software Product-Line Engineering: A Family-Based Development Process. Addison-Wesley, Reading, Massachusetts, USA
10. Czarnecki K, Eisenecker C (2000) Generative Programming, Addison-Wesley, Massachusetts, USA
11. Kruchten P (2004) A Taxonomy of Architectural Design Decisions in Software-Intensive Systems. In: Proceedings of the Second Groningen Workshop on Software Variability Management
12. Parnas D L (1979) Designing Software for Ease of Extension and Contraction. IEEE Transactions on Software Engineering, 5(2):128–138
13. Ebert C, de Man J (2005) Requirements Uncertainty: Influencing Factors and Concrete Improvements. In: ICSE'05. Proceedings of the 27th International Conference on Software Engineering, St Louis, Missouri, USA. ACM Press, New York
14. Saarinen T, Vepsäläinen (1993) Managing the Risks of Information Systems Implementation. European Journal of Information Systems, 2(4):283–295
15. Aldaijy A Y (2004) An Empirical Assessment of the Impact of Requirements Uncertainty on Development Quality Performance. In: Command and Control Research and Technology Symposium
16. Berry D M, Kamsties E, Krieger M M (2003) From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Technical Report, University of Waterloo, Canada
17. Kamsties E, Paech B (2000) Taming Ambiguity in Natural Language Requirements. In: Proceedings of the International Conference on System and Software Engineering and their Applications
18. Gervasi V, Nuseibeh B (2002) Lightweight Validation of Natural Language Requirements: A Case Study. Software Practice and Experience, 32(3):113–133
19. Chantree F, Nuseibeh B, de Roeck A et al (2006) Identifying Nocuous Ambiguities in Natural Language Requirements. In: RE2006. Proceedings of the IEEE International Requirements Engineering Conference. IEEE Computer Society Press, Los Alamitos,

20. Maynard-Zhang P, Kiper J D, Feather M S (2005) Modeling Uncertainty in Requirements Engineering Decision Support. In: REDECS'05. Proceedings of the Workshop on Requirements Engineering Decision Support of the 13th IEEE International Requirements Engineering Conference
21. Nuseibeh B, Easterbrook S, Russo, A (2001) Making Inconsistency Respectable in Software Development. Journal of Systems and Software, 58(20):171–180
22. Moynihan T (2000) Coping with 'Requirements-Uncertainty': the Theories-of-Action of Experiences IS/Software Project Managers, Journal of Systems and Software, 53(2):99–109
23. Bush D, Finkelstein A (2003) Requirements Stability Assessment Using Scenarios. In: RE2003. Proceedings of the 11th IEEE International Conference on Requirements Engineering
24. Stephenson Z (2005) Uncertainty Analysis Guidebook. Technical Report YCS-2005-387, University of York Department of Computer Science
25. Thimbleby H (1988) Delaying Commitment. IEEE Software 5(3):78–86
26. Poppendieck M, Poppendieck, T (2003) Lean Software Development: An Agile Toolkit. Addison-Wesley, Reading, Massachusetts, USA
27. van der Linden F, Bosch J, Kansties E, Känsälä K, Obbink, H (2004) Software Product Family Evaluation. In: Nord R L (ed) SPLC 2004. Proceedings of the 9th International Conference on Software Product Lines, Boston, Massachusetts, USA. Lecture Notes in Computer Science (LNCS), vol 3154, Springer, Heidelberg, Germany
28. Stephenson Z (2002) Change Management in Families of Safety-Critical Embedded Systems. PhD Thesis YCST-2003-03, Department of Computer Science, University of York
29. van der Ven J S, Jansen A J G, Nijhuis J A G, Bosch J (2006) Design Decisions: The Bridge between Rationale and Architecture. In: Dutoit A H, McCall R, Mistrik I, Paech B (eds) Rationale Management in Software Engineering
30. Gonzalez-Perez C, Henderson-Sellers B (2008) Metamodelling for Software Engineering. Wiley and Sons, Hoboken, New Jersey, USA