Requirements Eng (1997) 2: 102–113


Requirements Engineering

# Ten Steps Towards Systematic Requirements Reuse

**W. Lam[a], J.A. McDermid[b] and A.J. Vickers[b]**

[a]Department of Computer Science, University of Hertfordshire, Hatfield, Herts, and, [b]Rolls-Royce University Technology Centre, Department of Computer Science, The University of York, York, UK

*Reusability is widely suggested to be a key to improving software development productivity and quality [1,2]. It has been further argued that reuse at the requirements level can significantly increase reuse at the later stages of development [3,4]. However, there is little evidence in the literature to suggest that requirements reuse is widely practised. This paper describes ten practical steps towards systematic requirements reuse based on work at the Rolls-Royce Systems and Software University Technology Centre (UTC) for Rolls-Smiths Engine Controls Ltd (RoSEC) in the domain of aero-engine control systems. We believe these steps have made a significant overall contribution to the 50% reuse figure quoted by the management at RoSEC for current projects within the BR700 family of engine controllers.*

**Keywords:** Aero-engines; Case study; Practical; Requirements; Reuse

## 1. Introduction

Reusability is widely suggested to be a key to improving software development productivity and quality [1,2]. It has been further argued that reuse at the requirements level can significantly increase reuse at the later stages of development [3,4]. However, while there have been successful cases of reuse in companies such as Digital, Motorola and Hewlett Packard [5,6], there is little evidence in the literature of requirements reuse as part of the normal systems development process.

*Correspondence and offprint requests to:* W. Lam, Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, Herts AL10 9AB, UK. E-mail: w.lam@herts.ac.uk

This paper describes ten practical steps towards systematic requirements reuse, based on work at the Rolls-Royce UTC which has been involved in institutionalising reuse within Rolls-Royce since 1993 [7–11]. Our experience is largely drawn from the close working relationship we have with RoSEC, a company jointly owned by Rolls-Royce and Smiths Industries Ltd, which was set up to develop and market aero-engine controllers.

## 2. A Brief History of Requirements Reuse

Why should one attempt to reuse requirements? Although the argument has no documented empirical foundation, it would seem logical that the reuse of requirements in functionally similar systems will bring economic savings. Certainly in the domain of aero-engine control systems, which the UTC has been involved in, where development costs are high, even a small amount of reuse may convert into large financial savings. It can also be argued that by reusing the same set of requirements again and again, one is likely to 'trust' them more than requirements written 'from scratch'.

Requirements reuse has been examined from a number of different perspectives: analogy [12,13], case-based reasoning [14] and generic modelling [15–18]. Unfortunately, the above ideas have been restricted to small-scale academic examples, and remain largely untested in a genuine industrial or commercial capacity. More recently, however, the reuse community has reported success in the use of domain-specific approaches to reuse within certain organisations [19,20]. Central to these domain-specific approaches is the use of domain analysis, which Prieto-Diaz [21] defines as '*a process by which information used in developing software systems is identified, captured, and*

*organised with the purpose of making it reusable when creating new systems*'. Although early domain analysis work appeared more relevant to code reuse [22], there is evidence that domain analysis techniques are now being used during earlier stages of development [23–25].

In sum, the notion of reuse at the requirements stage is accepted by many within the community as a desirable aim. However, what appears to be missing from the literature is pragmatic advice on achieving requirements reuse as part of regular project practice. This paper addresses this concern.

## 3. Ten Steps Towards Systematic Requirements Reuse

The theme of the 4th annual workshop on Software Reuse Education and Training was 'Making Reuse Happen – Factors for Success' [26], emphasising the need for the reuse community to consider the practical implications of their ideas. Such a theme accords with the technology transfer goals of the UTC [27,28] which has been involved in the reuse of requirements for FADECs (Full Authority Digital Engine Controllers) at Rolls-Royce and RoSEC.

A FADEC is a control system for an aero-engine, taking inputs from sensors located on the engine and aircraft, and producing output in the form of electrical signals to actuators such as fuel valves and igniters. The FADEC is embedded and safety-critical.

In the following, we describe ten practical steps which we feel have brought RoSEC towards systematic requirements reuse, backed up with examples from our experiences. The steps we describe are not meant to be carried out in a fixed sequence – each step represents an idea that can be taken onboard and implemented within an organisation separately in its own right. We have classified the steps into one of two categories: orthodox and nonconformist. Orthodox refers to a step which conforms with generally accepted reuse principles (but which may not have much empirical foundation). Nonconformist refers to a step which suggests revisions or new openings to the current way the community thinks about reuse.

### 3.1 'Beware of Seductive Generalisations' (nonconformist)

Generic modelling, in one form or another, is often the cornerstone technique of most approaches to reuse. While we do not dispute that generalisations are important in reuse, we do feel that there is a need to examine the usefulness of generalisations more critically (a point also raised in [29]).

To illustrate, in one study of the functional requirements for three different aero-engine starting systems, we calculated (on the basis of a simple count) that only about 30% of requirements could be considered reusable, despite the systems having comparable functionality. The reason why the reuse figure is lower than expected is probably because the high level requirements between systems were similar, giving an overall impression of similarity. In fact, most of the requirements were low-level requirements, often tied in with design, more detailed in nature, and therefore difficult to reuse.

Our point here is that a view of reuse based solely on generalisations can be deceptive, and need to be examined more closely to reach a more realistic estimation of the true amount of reuse that is possible. Within RoSEC, the UTC has encouraged a broader view of reuse beyond that of generalisations, for example, the use of 'pluggable' requirement parts described in step 3.9 promotes the use of optional and configurable requirements as well as generic ones.

*Contribution to RoSEC: A realistic view of requirements reuse is taken at all levels of management within the company. Reuse education via the UTC has helped ensure that manageable but progressive reuse targets have been set for new projects within the BR700 family of engine controllers.*

### 3.2. 'Identity System Families to Maximise Reuse' (orthodox)

Code libraries were once the mainstay of efforts to promote and achieve reuse, and indeed, many have achieved significant success, e.g. those identified in [44]. Unlike code, however, requirements are context sensitive and are specific to a problem or set of problems. In addition, requirements are often 'knitted' together as part of an overall model, unlike code fragments which can be more modular and 'stand-alone' in nature. A library of individual requirements, therefore, is likely to be difficult to construct and use.

A more promising organisation for requirements from a reuse point of view is that of system 'families' as proposed in [23]. Within a system family, it may be possible to:

- Identify commonalties between the 'parent' system and 'child' system.
- Impose a common or standard requirements engineering process within the organisation.

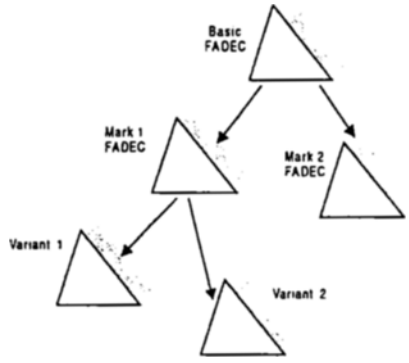


**Fig. 1.** System families for the FADEC.

- Anticipate certain kinds of change and specialisations.
- Reuse domain knowledge.
- Recognise working patterns which aid project planning.

A simple tree diagram can be used to depict a family of systems, and help identify sub-families where the requirements between family members may be even more closely aligned. Figure 1 shows the family structures for FADECs which is composed of marks and variants.

A mark is a FADEC for a specific engine within a series, such as the medium thrust BR710 within the BR700 series. Clearly, there is reuse potential between the 'basic' (or parent) FADEC and its marks (or children). A variant is a mark that is produced to the specific requirements of an airframer (such as Boeing or Airbus). Again, there is further reuse potential between a mark and its variants.

*Contribution to RoSEC: The development of a reuse programme which maximises reuse based upon a 'parent' and 'child' view of engine controllers. This is best shown in the document structure used by RoSEC for the BR700 family of engine controllers. Here, a set of generic requirements documents currently exists for the BR710 engine controller which are referenced by variants of the BR710 engine controller.*

## 3.3 'Evaluate Reuse Technology in Terms of Process Change, not Just on Reuse Potential' (nonconformist)

Numerous reuse technologies – application generators, patterns, high-level languages and cookbooks – have been described in the literature (see [4] and [30] for more details). However, rather than 'leaping into' the technology, it is important to assess the likely impact of the technology. This involves being clear about:

- *Current Practice* – how requirements are currently engineered.
- *Reuse Strategy* – how one envisages reuse will be 'implemented' in the current requirements engineering process.
- *Effects on Current Practice* – how the reuse strategy will change current practice in terms of methods, organisation structure, finance and other facets.

We have identified a number of evaluation criteria within each of these three areas, and used these as the basis for assessing different reuse technologies (described in Table 3 at the end of the paper). The framework acts as a 'checklist', encouraging one to think more deeply about the way in which requirements reuse is to be achieved and sustained in a commercial setting, emphasising the mix of technical, organisational and financial issues. It should be noted, however, that the technology offering the highest reuse potential is not necessarily the most suitable for the organisation (it may be seen as too costly or risky for example).

*Contribution to RoSEC: the UTC has produced an assessment document for RoSEC which prescribes a set procedure for evaluating different kinds of reuse technologies, which takes into account both technical and non-technical concerns.*

## 3.4. 'Domain Issues Act as Requirements Focal-points, and Can be Used to Organise and Structure Reuse Products and Processes' (nonconformist)

The process of creating reusable requirements is aided by having a road-map for structuring the domain and organising reusable requirements knowledge. In this respect, we have found the notion of 'issues' a useful structuring mechanism. We view an issue as an area where requirements, in a particular domain, are typically focused. Table 1 describes issues for thrust reverser systems and the key questions, which we call trigger questions, pertaining to each issue.

Issues can be compared to the notion of 'touchstones' proposed in [31] as a way of structuring and controlling the process of knowledge elicitation. The trigger questions in an issue hint or point to individual requirements, for which there may be a corresponding template requirement. At RoSEC, we have explicitly recorded issues and trigger questions, and used them as a basis for developing reusable, domain-specific requirements engineering processes (described in more detail in sub-section 3.8).

**Table 1.** Issues for thrust reverser systems

| Issue | Trigger questions |
|---|---|
| Deploy thrust reverser | How does a pilot activate the thrust reverser system? Is the activation related to the position of the thrust reverser doors? What safety provisions are made if the doors are jammed or inhibited (such as automatic thrust limitation)? |
| Stow thrust reverser | How does a pilot deactivate the thrust reverser system? Is the deactivation related to the position of the thrust reverser doors? What safety provisions are made if the doors are jammed or inhibited? |
| Thrust reverser maintenance | How is the thrust reverser deployed and stowed under aircraft maintenance? What safety measures are in place with respect to the operation of the thrust reverser under maintenance? |
| Thrust reverser interlock | Is interlock provided to give the pilot a tactile indication of thrust deployment? If so, during which period does the interlock take place, and at what point is it released? |

*Contribution to RoSEC: For the domain of aero-engine starting, we have identified 16 different issues and their associated questions. These are recorded in a 'Domain issues' document forming part of a wider reusable document set which the UTC is developing aimed primarily for the BR700 engine family.*

## 3.5. 'Reasoned Abstraction is Effective for Developing Template Requirements' (orthodox)

Template (or parameterised) requirements encourage reuse by factoring out system-specific details as parameters of the requirement. We have found that template requirements provide a quick and cost-effective route to reuse – the notion of a template is easy to comprehend and does not require a change in the way requirements are described. The method for creating template requirements is one of reasoned abstraction, and is described as:

- Identify commonly re-occurring issues and trigger questions between similar projects.
- Use the trigger questions to locate equivalent 'concrete' requirements in each project.
- Use the similarity between concrete requirements to formalise the 'constant' part of the template requirement.
- Use the differences between concrete requirements to formalise the 'variable' part of the template requirement as parameters.
- Validate the template requirement with an expert.
- Re-use the template requirement in future projects, and refine it as necessary

Table 4 shows an example of the results of this process with respect to a developing template requirements for dry cranking an engine (rotation of the engine without ignition or fuel). Note that the abstraction process is a reasoned one; in any abstraction process, we need to ask a number of important questions:

- Do we have 'equivalent' exemplar requirements?
- What part of the requirement is constant, what part is variable, and how can the two be separated?
- What is the explanation for the separation?
- Is the resulting template requirement meaningful and sufficiently flexible that it can be considered reusable? Is it possible to test this by applying the template requirement to a separate exemplar?
- Do other requirements engineers understand the template requirement and the abstraction process from which is has been derived?

Answering such questions is not straight-forward, which is why we believe abstraction of this nature will be difficult to automate, despite recent work in the area of computational matching [32]. Abstraction is clearly important in reuse, and the ideas here can be compared to work in artificial intelligence – [33] for example, describes the formation of general plans and heuristics from exemplars. However, some decision must be made as to the most appropriate level of abstraction. Over-abstraction will strip away essential parts of a requirement, while under-abstraction will retain system-specific details which will reduce overall reusability. There is a balance here where the most optimal level of abstraction is ultimately the one which requires the minimum amount of effort to re-use the abstract artifact (template requirements). It is likely that this balance will be reached with usage over time rather than as something we can 'calculate' beforehand.

*Contribution to RoSEC: The UTC has created 30 template requirements in the domain of aero-engine starting based on the functional requirements documents (FRDs) from four different engine controllers. We believe these 30 template requirements represent the core*

*30% of a typical FRD. As well as the ongoing work of refining these, the UTC is also involved in the process of developing template requirements in the domain of thrust reverse.*

### 3.6. 'Requirements Patterns Often Emerge After Working in a Particular Domain' (nonconformist)

Domain analysis is based on the idea that if one studies the systems in a particular domain, patterns can be identified. Substantial interest in patterns at the levels of design and code has been shown by those in the object-oriented community [34]. However, patterns can be found much earlier in the development process. An analysis of different requirements documents for aero-engine starting systems and signal validation systems revealed patterns of requirements. For example, we noticed a pattern of requirements for engine relight, a common feature of modern FADECs (Fig. 2).

In short, engine relight refers to the relighting of the engine when a flameout condition occurs (such as in the case of severe water ingestion). The pattern depicts five different kinds of requirements, which are often found together. For example, the requirements for engine relight will always include a requirement for how flameouts will be detected. The arrows shown in Fig. 2 provide additional information about the pattern, and indicate the order in which the requirements are usually addressed. For example, an expert in starting will usually ask questions about the operation of the ignition system before considering any associated timing requirements.

By capturing patterns, we have moved closer to formalising the structures of requirements knowledge in a particular domain. In doing so, there are a number of issues which need to be addressed:

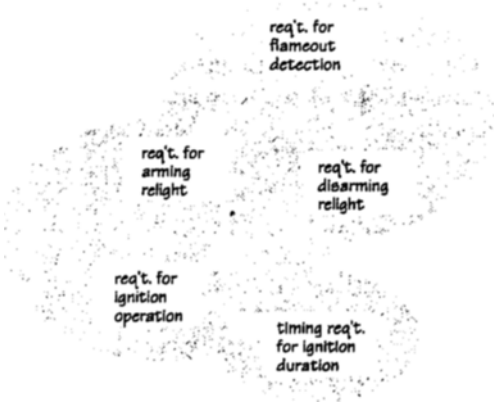


**Fig. 2.** A requirement pattern for engine relight.

- *Pattern content.* What knowledge does a pattern impart? For example, the pattern in Fig. 2 tells us about 'expected' requirements in a particular area, and something about the order in which they are to be elicited. However, we have often found the existence of dependencies between requirements along the lines of 'requirement B is possible only if requirement A is true'. It is possible therefore that other kinds of patterns might capture this type of knowledge.
- *Representing patterns.* How are patterns represented? In some respects, this is dependent upon the pattern content. However, we have found the use of simple diagrams (as in Fig. 2) supplemented with more detailed explanations in natural English, sufficient for our purposes here. It should be noted it is not the formality of the pattern representation which is important, but the knowledge which the pattern communicates to the pattern reader.
- *Dealing with exceptions.* Are there exceptions to the pattern? Encountering exceptions may indicate the need to revise a pattern, or to further delineate the context in which the pattern can be applied.

In practice, recognising a pattern is the most difficult part of the process, which will only be possible after studying several similar systems. The important point, however, is to aware that patterns exist and to document them so that they can reuse to guide future systems.

*Contribution to RoSEC: A Windows-based prototype tool, known as COMPASS (COMPonent ASSistant) [8], has been developed which records patterns in the area of aero-engine starting and links them with template requirements stored in a local reuse database. Using COMPASS, a user (RoSEC engineer) is able to browse and select patterns and instantiate template requirements into project requirements. At present, COMPASS has 13 patterns including patterns for cranking, various starting modes, continuous ignition and engine relight. RoSEC is currently evaluating COMPASS.*

### 3.7. 'Making Explicit the Context of Reuse to Prevent Misuse' (nonconformist)

One of the most striking cases of reuse misuse, reported by [35], concerned the traffic control system used by the Civil Aviation Authority (CAA) in the UK. The software, designed by IBM's Federal Systems Division, contained a model of the airspace it controls. However, the software was designed for air traffic control centres in the US, and the CAA had not taken account of a zero longitude when reusing the software in the UK.

This oversight caused the computer to fold its map of Britain in two at the Greenwich meridian.

The case clearly demonstrates the need to reuse with care, especially in the case of high-integrity systems. Work at the UTC suggests that the explicit documentation of context is a step towards the prevention of reuse misuse. In doing so, one is forced to think about the (often hidden) assumptions behind a reuse artefact. For example, in the domain of aero-engine starting systems, we have explicitly defined three typical contexts, shown in Fig. 3.

Each context depicts an abstract design model of the (often physical) components in a starting system. Civil aircraft often have an air turbine starter, which provides a high torque-to-weight ratio. Military aircraft on the other hand, will usually have a solid propellant starter which provides rapid starting. Small, light aircraft finds the electric motor starter more suitable because of the ease of maintenance.

We argue that it is important to associate reusable requirements with a particular context, i.e. a particular abstract design model, and to ensure that the *intended* context that a set of reusable requirements is created for matches the *actual* context in which they are going to be reused. For example, requirements which concern the ignition of a solid propellant cartridge are only 'valid' or meant to be reused in the context of a solid propellant starter – attempting to reuse them in the context of an air turbine starter or electric motor starter

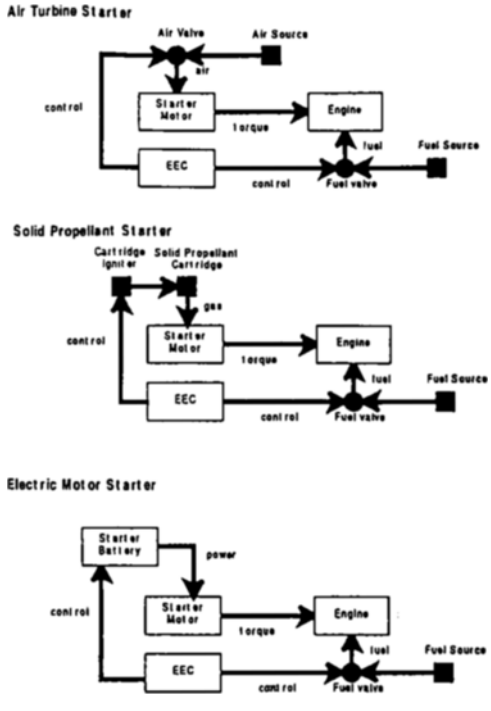


**Fig. 3.** Three different contexts for aero-engine starting.

is inappropriate, potentially dangerous and is likely to lead to poor levels of reuse.

To help record the differing types of context, we make use of five separate levels of description based on the physical organisation of these types of system. We recognise descriptions at the generic level of Environment, Platform, System, Subsystem, and Unit. Within the domain of aero-engines (and in particular towards FADECs) these correspond to Airplane, Engine, FADEC, EEC, and Software. From RoSEC documentation we have been able to identify design and requirements descriptions at each of these levels of abstraction. Requirements for a low-level component are in part derived from design descriptions at the higher levels, and these in turn are presented in response to higher level requirements. We are seeking to establish reuse libraries, for requirements and context, at each of these different levels of abstraction.

*Contribution to RoSEC: There is now raised awareness within RoSEC with respect to the problems of inappropriate reuse. The UTC is currently in the process of writing a 'safe reuse' guidebook for RoSEC which includes a description of safe reuse guidelines.*

## 3.8. 'Parts of the Requirements Engineering Process is Also Reusable' (nonconformist)

It was Osterweil [36] who first suggested that 'processes are programs too'. Since then, there has been an increasing interest in the explicit modelling of software processes [37,38]. In the context of requirements engineering at RoSEC, we observed similar sequences of questions and routines being followed by domain experts working on the requirements of systems in the same domain. We believe that if parts of this process is modelled in an abstract manner, it can be reused to guide future requirements engineering exercises. For example, we have modelled the process by which requirements are elicited for 'aborting a start' (Fig. 4) using our own variant of Role Activity Diagrams (RAD) [39].

In short, an aircraft engine in the process of being started can be aborted for safety reasons such as engine overheating. In our process description, a square box represents a questioning or elicitation activity, described as a set of trigger questions for the RoSEC requirements engineer to consider. For example, the questioning activity labelled 'Manual fuel shutoff' includes the trigger question: 'is there a facility in the cockpit for the pilot to abort the start by switching off the fuel supply to the engine?'. A circle represents a checking activity, i.e. some analysis of the information

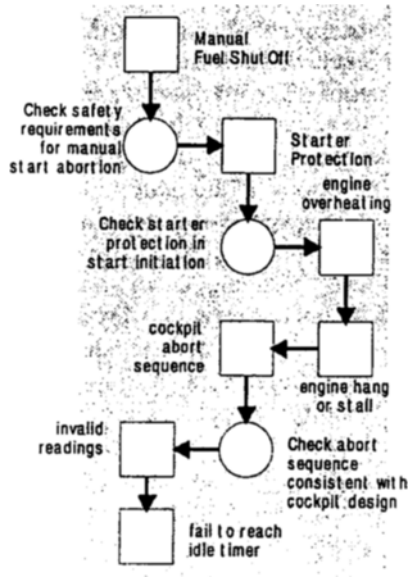


**Fig. 4.** The requirements process for 'aborting a start'.

gathered from a questioning activity. For example, the checking activity labelled 'Check safety requirements for manual start abortion' involves following a procedure for finding out the safety and certification requirements for safe manual start aborting and ensuring that they are met in this case. The arrows in our process description suggest a logical order for elicitation.

Our experience shows that process modelling in the manner shown here has a number of benefits:

- A reused process will facilitate a reused product (in this case, a reusable requirement specification).
- The process model can act as a checklist for ensuring all the requirements for a particular area are elicited.
- The 'dynamic nature' of requirements information can often be more clearly seen in the context of a process model (for example, a process model can indicate at what point in requirements engineering a particular piece of information should be elicited, cross-checked with other information, expanded upon etc.).
- Process models can be matured and refined to a point where they become key educational aids for novice engineers learning new areas.

Preliminary process models can be developed by 'watching' the expert at work first, and then refining the model with the expert as a form of validation. In addition, a relationship was often observed between requirement patterns (cf. Section 3.6) and process reuse: as patterns 'matured' it became possible to derive the associated process. The UTC is currently investigating the use of process modelling in establishing domain-specific requirements guide-books for RoSEC engineers. More details of our work on process reuse can be found in [40].

*Contribution to RoSEC: So far the UTC has informally modelled 7 domain-specific requirements engineering processes in the domain of aero-engine starting using the RAD notation. We have documented these processes as a kind of 'workplan' that an engineer can pickup and follow. These processes are documented in a requirements guide-book for the aero-engine starting domain.*

## 3.9. 'Factor Requirements Variance into Pluggable Requirement Parts' (nonconformist)

Generalisation is often used to capture the commonality between all systems in a domain. As such, there is always a danger that the resulting generalisation is so devoid of detail that it becomes of little practical value (cf. Section 3.1: 'Beware of seductive generalisations'). However, reuse can be increased if we consider the isolation of requirements which, although are not common to all systems in a domain, re-occur on a frequent basis.

One approach we have used at RoSEC is that of the 'pluggable' requirement part. To illustrate, consider the following template requirement:

*Fuel and Ignition Template Requirement*:
'Fuel and ignition will only be switched on when a start has been requested by the pilot and [fuel and ignition conditions] are true.'

The square brackets indicate the variable part of the generic requirement. However, although the fuel and ignition conditions for different aircraft systems may differ, it is likely that the same conditions will re-occur across many different aircraft. Therefore, we captured the variability of the generic requirements in the form of pluggable requirement parts (this is something we feel that the domain expert should do because actually recognising variability can be difficult). Table 2 shows a list of six pluggable requirement parts for our generic requirement.

The use of pluggable requirement parts enables engineers to construct requirements quickly, with a degree of flexibility, by choosing relevant requirement parts and plugging them into the generic requirement. The requirement below illustrates this:

'Fuel and ignition will only be switched on when a start has been requested by the pilot and the engine speed >

**Table 2.** A list of pluggable requirement parts

| Pluggable requirement part |
|---|
| 1. Engine speed > preferred fuel-on speed |
| 2. Engine speed > minimum fuel-on speed |
| 3. Engine acceleration < maximum engine acceleration |
| 4. Turbine gas temperature (TGT) < starting TGT |
| 5. TGT < on-ground pre-start TGT |
| 6. Fuel-on timer has not expired |

minimum fuel-on speed and the turbine gas temperature < starting TGT and the fuel-on timer has not expired.'

Here, we have two reuse concepts working together. The first is that of the template requirement mentioned earlier in the paper. The second is the formalisation of typical parameter values associated with a template requirement, i.e. a set of pluggable requirement parts.

More generally, we are seeking to extend this notion of pluggable requirements to characterise more fundamental aspects of requirements specification. In particular we are seeking to identify formats, based on controlled natural language, that recognise pre-condition, action, and post-condition. We are using keywords

**Table 3.** Framework for evaluating reuse technology

| Evaluation criteria | Description | RoSEC-related examples |
|---|---|---|
| Current practice | | |
| • Document | The type of report | Functional requirements doc., System concept doc. |
| • Document content | The type of requirements included in the document | System, FADEC, hardware design or aircraft interface requirements |
| • Notation | The representation of the requirements | Structured English and statecharts |
| • Methods used | Any particular methods or techniques used during the requirements engineering process | Statecharts |
| Reuse strategy | | |
| • Coverage | An estimate of how much reuse is possible | 20% of a typical functional requirements document |
| • Reuse artefacts | The form of what is to be reused | Generalised structured English requirement statements |
| • Scope | Any limitations of the reuse artefacts | Only applicable to systems in the BR700 engine series |
| • Reuse frequency | How often the reusable artefact would be used | On all engine projects, or just on BR710-related engine projects |
| • Envisaged process | If reuse was adopted, how the process would look from an engineer's point of view | The engineer logs onto the reuse library, which is based around a World Wide Web browser. Requirements given as textual statements are cut and pasted into a requirements document. Other requirements not taken from the reuse library can be directly added to the document |
| • Startup actions | What is needed in order to make reuse possible | A domain analysis of the thrust reverser domain, followed by the setting up a reuse library and its population with generic requirements |
| • Critical success factors | What are seen to be the most important factors in order for the reuse strategy to succeed | Engineers actively involved in the domain analysis; the reuse library to be accessible via the engineer's PCs and project manager providing 1 extra week in the project budget to allow engineers familiarisation |
| Effects on current practice | | |
| • Notation | How a reuse strategy is likely to affect the way in which requirements are represented | None |
| • Methods used | How a reuse strategy is likely to affect the methods and techniques already in use | Engineers will be able to select requirements from the reuse database and automatically import them into a RoSEC document |
| • Organisation | The organisational impact that a reuse strategy will have | The assignment of a person to maintain a reuse library and to perform a domain analysis |
| • Financial | The financial consequence that a reuse strategy is likely to entail | Initial up-front costs in producing a reusable requirements library |
| • Other | Other consequences a reuse strategy is likely to bring | General training for engineers on how to use the reuse library |

**Table 4.** Creating a template requirement using abstraction

| Element in abstraction process | Example |
|---|---|
| Concrete requirement from system A | When engine not in process of being started, cranked or run, if fuel switch in OFF position and master crank switch in ON position, and engine start switch then turned to ON position then dry crank will be initiated |
| Equivalent concrete requirement from system B | When engine not in process of being started, cranked or run, if fuel switch in OFF position and engine start switch turned to CRANK position, then dry crank will be initiated |
| Constant requirement part | When engine not in process of being started, cranked or run, if (X) and then (Y), dry crank will be initiated |
| Variable requirement part | X and Y are cockpit-specific signals |
| Abstraction reasoning | Cockpits are specific to a particular system, and not all systems will have the same cockpit layout. Hence, this aspect is a variable requirement part and must be factored out of the generic requirement |
| Template requirement | When the engine is not in the process of being started, cranked or run, if (cockpit signal 1) and then (cockpit signal 2), a dry crank will be initiated |

such as 'when', 'while', 'if', 'generate', and 'sustain' to characterise the differing roles of events and conditions in the specification of individual functional requirements. Our work shares some similarity with that of [43], and our intention is to develop these formats as aides for systems engineers in producing quality requirements more quickly.

*Contribution to RoSEC: As mentioned earlier, we have developed a tool called COMPASS which enables RoSEC engineers to select template requirements from a reuse database and 'instantiate' them to form project requirements. For many instantiations, the user can select from a parts database which has been built into COMPASS.*

## 3.10. 'Assess Beforehand the Impact of Requirements Reuse on the Development 'Food Chain'' (orthodox)

It is fallacious to think of requirements engineering as a process which is performed in a vacuum. In truth, it is impractical to take a 'purist' view of requirements engineering, as external circumstances such as costs, design implications and even politics will inevitably affect the way in which requirements are shaped. In this respect, an analogy can be made with the food chain – changing a requirement is likely to have profound repercussions down the chain, causing reassessment of the original design, and forcing significant re-testing.

Under the same analogy, changing the way in which requirements engineering is performed by increasing levels of reuse is likely to induce repercussions, good and bad, along the development chain. We need to assess the impact, taking into account a number of viewpoints.

- *Design Options.* Do reusable requirements 'home in' on a standard design or set of designs? Would it be acceptable and worthwhile formalising standard designs based around the reusable requirements? (Work on the AD-AGE project [24] seems to have adopted a reusable design approach)
- *Testing Strategies.* Does reusing requirements lead to more economical testing? If not, can the existing testing strategy be changed to reap such benefits? Does the introduction of novel requirements nullify the potential savings possible with reuse?
- *Certification Pitfalls.* Certification is a crucial hurdle in the development of safety-critical systems such as those often found in avionics. We therefore need to make some honest judgements: is reuse likely to affect the quality of the delivered system? If so, in what way and will this lead to pitfalls during certification? Is the integrity of the delivered system in any way comprised, and if not, how sure can we be?

Attempting to localise reuse, without examining changes along the development food chain, can at best only lead to localised benefits. Our advice here is to establish a small working group, consisting of individuals representing different areas of the software development process. The objective of the working group should be to critically examine the impact of reuse on other aspects of systems development, and to explore potentially advantageous and/or harmful reuse 'spin-offs'.

*Contribution to RoSEC: Under the guidance of the UTC, RoSEC has established a 4 man reuse group within the company comprised of individuals from different engine projects. The objective of the reuse group is to identify and exploit opportunities for reuse across the different projects. One achievement of the group has been recent work on the reuse and automatic generation of test scripts.*

## 4. Lessons for Others

In this paper we have described a number of steps that we have used to help facilitate requirements reuse within RoSEC. These steps have been identified through industrial collaboration and they are therefore expressed in terms of the industry involved with that collaboration – aerospace. Although the actual patterns we have identified are specific to RoSEC, we feel it is very possible that they apply to other aero-engine control manufacturers outside of Rolls-Royce. In theory, this could lead to standardisation within the industry as discussed in [46]. However, given the extremely competitive nature of the industry, there are serious commercial barriers here.

Perhaps to be of real use and value to the community, it should be possible to see their application in other domains. We believe that the lessons described in this paper can be generalised and that there is nothing within this paper that is inherently aerospace-oriented. We believe that in all domains it should be possible to identify useful and appropriate generalisations and abstractions (steps 1 and 5), identify families (step 2), consider evaluation more broadly than just the product alone (steps 3 and 10), identify patterns and context (steps 6 and 7), reuse processes (step 8), and identify focal points and variance (steps 4 and 9). These are general concepts which we believe are generally applicable. After all, whilst in Section 2 we said that there was little published in the way of industrial-scale requirements reuse, our steps clearly draw on the general work of the patterns community, analogical reasoning, case-based reasoning, and generic modelling.

## 5. Conclusions

The main contribution of this paper is a description of ten practical steps towards systematic requirements reuse, distilled from three years work at the UTC in institutionalising reuse within RoSEC. These ten steps have helped considerably in taking RoSEC to a point where they are able to begin populating a requirements reuse library, and where requirements reuse is becoming an integral part of their development process. An overall figure of 50% reuse between engine controllers within the BR700 family has been quoted by the senior management at RoSEC (even though the exact breakdown of this figure for requirements reuse is not yet available to the UTC, we expect it to be close to the 50% mark). Although most of our steps have a distinct technical focus, we cannot over-emphasize the importance of organisational and managerial factors in institutionalising reuse [41, 42].

Perhaps the most encouraging lesson from our work is that an increase in the level of requirements reuse can be achieved using a number of relatively cheap and simple measures which do not require radical organisational changes. While we believe that each of the steps described in this paper can stand alone in its own right, it is the synergy between steps working in parallel which is likely to bring the most significant benefits of reuse.

## 6. Future Work

The work presented in this paper has been of the 'action-based' or 'case-study' style and therefore may appear ad-hoc. This was necessitated by the exploratory nature of our investigation into this area. Having undertaken this exploratory work, we are now in a position to undertake the investigation of large-scale reuse in a much stronger and more methodical position.

In the future, we intend to build upon our existing models of requirements and code reuse and consider, from an industrial perspective, the role of domain-specific architectures [45] (systems and software) in the facilitation of reusable systems. We intend to identify a reusable architecture for our domain and then investigate the relationships between our reusable requirements and our reusable code.

Finally, we observe that our overall aim is to help RoSEC define technology that will enable them to maintain the high quality of their product, but at a fraction of the cost. We gratefully observe that the enthusiasm shown by the managers and the many engineers at RoSEC towards reuse has played a critical role in our work. Without them, our ideas would remain unused on the academic bookshelf.

## Acknowledgements

We are grateful for the support of our sponsors, RoSEC and Rolls-Royce, in funding our work. We would also particularly like to acknowledge the technical support of Mike Bardill, Paul Essam, and Graham Tanner, as well as past and present colleagues of the UTC; all of whom have helped shape our ideas.

## References


1. Lim WC. Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, 19941 11(5): 23–30.
2. Sommerville I. Software Engineering (5th Edition), Addison Wesley, ISBN 0-201-42765-6. 1996
3. SPC (1992) Software Productivity Consortium Reuse Adoption Guidebook, Version 01.00.03, SPC-92051-CMC, November, 1992
4. Biggerstaff T, Ritcher C. Reusability framework, assessment and directions, IEEE Software, 1987; 41(3): 41–49 March
5. Guerrieri E. Case study: Digital's application generator, IEEE Software, 1994; 11(5): 95–96
6. Joos R. Software reuse at Motorola, IEEE Software, 11(5): 42–47
7. Kelly TP, Lam W, Whittle BR. Diary of a domain analyst: a domain analysis case-study from avionics, In Proceedings of IFIP Working Groups 8.1/13.2 Conference, Domain Knowledge for Interactive System Design, Geneva, May 8-10, 1996
8. Lam W, Whittle BR, McDermid J, Wilson S. An integrated Approach to Domain Analysis and Reuse for Engineering Complex Systems, In Proceedings of International IEEE Symposium and Workshop on Engineering of Computer-Based Systems (ECBS '96), Friedrichshafen, Germany, March 11-15, 1996
9. Lam W, Whittle BR. A Taxonomy of Domain-Specific Reuse Problems and their Resolution – Version 1.0, Software Engineering Notes 21(5): 72–77 1996
10. Whittle BR, Lam W, Kelly TP. A Pragmatic Approach to Reuse Introduction in an Industrial-Setting, In Proceedings of the International Workshop on Systemmatic Reuse, Liverpool John-Moores University, Springer-Verlag, 1996
11. Whittle BR, Vickers AJ, Lam W, McDermid J, Hill JA, Rimmer R, Essam P. Structuring Requirements Specifications for Reuse, International Journal on Applied Software Technology 1(3-4), 1995
12. Finkelstein A. Reuse of formatted requirements specifications, Software Engineering Journal, 1988; 3(5): 186–197
13. Maiden N, Sutcliffe A. Exploiting reusable specification through analogy, Communications of the ACM, 1993; 35(4): 55–64
14. Lam W. Reasoning about requirements from past cases, PhD thesis, Kings College, University of London, 1994
15. Bolton D, Jones S, Til D, Furber D, Green S. Using domain knowledge in requirements capture and formal specification construction, In Jirotka, M. and Goguen J. (Eds.), Requirements Engineering: Social and Technical Issues, Academic Press, London, 1994
16. Reubenstein HB. Automated Acquisition of Evolving Informal Descriptions, Report No. AI-TR 1205, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, 1990
17. Miriyala K, Harandi TH. Automatic derivation of formal software specifications from informal descriptions, IEEE Transactions on Software Engineering, 1991; 17(10): 1126–1142
18. Ryan K, Mathews B. Matching conceptual graphs as an aid to requirements reuse, In Proceedings of the IEEE International Symposium on Requirements Engineering, ISBN 0-8186-3120-1, page 112-120, 1993
19. Griss ML, Favaro J, Walton P. Managerial and Organisational Issues – Starting and Running a Software Reuse Program, In Software Reusability, eds W. Schaefer, R. Prieto-Diaz and M. Matsumoto, Ellis Horwood, Chichester, GB, 1994; pp.51–78
20. WISR. Proceedings of the 7th Annual Workshop on Software Reuse, St. Charles Illinois, August 28–30, 1995
21. Prieto-Diaz R. (1990) Domain analysis: an introduction, ACM Software Engineering Notes, 1990; 15(2): 47–54
22. Wartik S, Prieto-Diaz P. Criteria for comparing reuse-oriented domain analysis approaches, International Journal of Software Engineering and Knowledge Engineering, 1992; 2(3): 403–431
23. Gomaa H. Reusable software requirements and architectures for families of systems, Journal of Systems and Software, 1995; 28: 189–202
24. Tracz W. DSSA (Domain-Specific Software Architecture) pedagogical example, ACM SIGSOFT Software Engineering Notes, 1995; 20(3): 49–62
25. Tracz W, Coglianese L, Young P. A domain-specific software architecture engineering process outline, ACM SIGSOFT Software Engineering Notes, 1993; 18(2): 40–49
26. WSRET. Proceedings of the 4th International Workshop on Software Reuse Education and Training, Morgantown, West Virginia, 14–18th August, 1995
27. Bate IJ et al. Technology Transfer: An Integrated 'Culture-Friendly' Approach, In Proceedings of Workshop on Technology Transfer, 18th International Conference on Software Engineering, Berlin, Germany, 25–29 March, 1996
28. Vickers AJ, Whittle BR, McDermid JA. Technology Transfer by Case Study; An Experience Report, Presented at 3rd International Conference on Concurrent Engineering & Electronic Design Automation, Poole, 1996
29. Kramer J. Generalisations are false?, In Proceedings of the IEEE International Symposium on Requirements Engineering, ISBN 0-8186-3221-1, IEEE Computer Society Press, Los Alamitus, CA 1993
30. Mili H, Mili F, Mili A. Reusing Software: issues and research directions, IEEE Transactions on Software Engineering, 1995; 21(6): 528–561
31. Littman DC. Modeling human expertise in knowledge engineering: some preliminary observations, International Journal of Man-machine Studies, 1997; 26: 81–92
32. Spanoudakis G, Constantopoulos P. Analogical Reuse of Requirements Specifications: A Computational Model, Applied Artificial Intelligence (to appear) 1996
33. Carbonell JG. Learning by analogy: Formulating and generalising plans from past experience, In Michalski RS, Carbonell JG, Mitchell TM (Eds.) Machine Learning: an Artificial Intelligence approach, Los Altos, CA, Kaufmann, 1983
34. Coplien JO, Schmidt DC. Eds. Pattern Languages of Program Design, Addison-Wesley, ISBN 0-201-6073-4, 1995
35. Wray T. The everyday risks of playing it safe New Scientist, September 8, pp. 61–65, 1988
36. Osterweil L. Software processes are software too, In Proceedings of the International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitus, CA 1987

37. Curtis B, Kellner MI, Over J. Process modelling, Communications of the ACM, 35(9): 91–100, 1992
38. McChesney IR. Towards a classification scheme for software process modeling approaches Information and Software Technology, 1995; 37(7): 363–374
39. Holt AW, Ramsey HR, Grimes JD. Co-ordination System Technology as the Basis for a programming environment, Electrical Communication, 1983; 57(4): 307–314
40. Lam W. Process reuse using a template approach: a case-study from Avionics, Software Engineering Notes 22(2): 35–38, 1997
41. Fafchamps D. Organisational factors and reuse, IEEE Software, 1994; 11(5): 31–41
42. Frakes WB, Isoda S. Success Factors for systematic reuse, IEEE Software, 1994; 11(5): 15–19
43. Fuchs NE, Schwitter R. Attempt to Controlled English (ACE), CLAW 96, First International Workshop on Controlled Language Applications, University of Leuven, Belgium, March 1996
44. Software Reuse Success Stories prepared by: Reuse Acquisition Action Team (RAAT), Association for Computing Machinery (ACM) SIG on Ada Reuse Working Group, 1994
45. Tracz W. DSSA (Domain-Specific Software Architecture) pedagogical example ACM SIGSOFT Software Engineering Notes, 1995; 20(3): 49–62
46. Lam W, McDermid JA, Vickers AJ. Reuse Through Standardisation: Towards Industry-Standard Engine Control Software, In Proceedings of the 1996 Avionics Conference, Ramada Hotel Heathrow, UK, 20-21st November, 1996. (Publication available from ERA Technology, Cleeve Road, Leatherhead, Surrey, KT22 7SA, UK), 1996