A Summary of Domain Analysis Experience By Way of Heuristics

W. Lam
Dept. of Computer Science, University of Hertfordshire
College Lane, Hatfield, Herts AL10 9AB, UK
W.Lam@herts.ac.uk, Phone: +44 1707 284337, Fax: +44 1707 284303,

J.A. McDermid
Rolls-Royce University Technology Centre
Dept. of Computer Science, University of York,
Heslington, Y01 5DD, UK.

Abstract
Domain analysis is seen by some in the reuse community to be a key process for achieving systematic, large-scale, reuse. However, the success of a domain analysis is largely dependent upon how well the domain analysis process is carried out. This paper describes a set of heuristics for domain analysis, which summarises our experience of domain analysis in a palatable way for others. The aim of these heuristics is to provide an inexperienced domain analyst with practical advice about how to cope with problems during the domain analysis process. We explain when and how to apply each heuristic, illustrated with examples taken from the domain analysis case-studies we have performed at RoSEC, a company which manufactures electronic controllers for aircraft engines.

Keywords: Domain analysis, Domain engineering, Requirements reuse.

1. Domain Analysis: Background
Reuse is suggested to be a key to improving software development and productivity [1], particularly where one can identify a *family* of systems [2]. In recent years, domain analysis has emerged as a central process in achieving systematic reuse (see for example, the number of references to domain analysis in the 7th Annual Workshop on Software Reuse [3] and the 4th Annual Workshop on Software Reuse Education and Training [4]). Prieto-diaz [5] first defined domain analysis 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"*.

SSR '97 MA,USA

Although early domain analysis methods such as Feature Oriented Domain Analysis (FODA) [6] were criticised for being too code-oriented [7], more recent methods such as Organisational Domain Modeling (ODM) [8] are aimed across all levels of software development. In addition, several companies have reported reuse success stories using domain-specific techniques [9], [10].

2. Paper Contribution
Given the perceived optimism for domain analysis and domain-specific reuse in general, many organisations will be attracted towards this technology. Unfortunately, much of the existing literature on domain analysis concentrates on the results of domain analysis studies, with little in-depth critique of the actual domain analysis process carried out. There is also a scarcity of detailed case-studies describing the practical, day-to-day problems and decision-making which are part of a domain analysis project. To the potential domain-analyst, with no prior knowledge of domain analysis techniques, it would be hard to prepare for the practical issues involved in domain analysis from the literature alone.

We have done several industrial domain analysis case-studies in the avionics domain at RoSEC, a company which manufactures electronic engine controllers for aircraft. Like in a typical organisation setting, we learnt what we could from the literature, but developed a better understanding of domain analysis through practical experience. We feel such experience will be of value to others, and so in this paper, we have attempted to write down some of the heuristics which help to guide the way we now do domain analysis. First, however, we briefly describe the case-studies upon which our experience is based.

3. Case-Studies
Modern aircraft engines are fitted with Electronic Engine Controllers (EEC) - essentially a computer which runs engine control software (Figure 1).

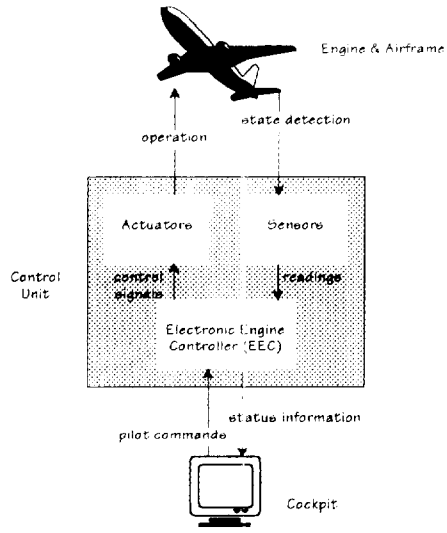

Figure 1 The Electronic Engine Controller (EEC)

The EEC uses sensors to monitor the engine and other components on the airframe (the aircraft body). The EEC controls the engine via the operation of actuators, such as valve, igniters and pumps. The EEC also accepts pilot commands, and provides status information about the engine back to the cockpit. The EEC is embedded and safety-critical, and is physically located as a 'black-box' on the engine casing.

Engines are manufactured as part of a series. Each engine in a series may have one or more variants which reflect the requirements of a specific customer (an airline company) and the interfacing to a specific airframe. We therefore view engines as part of an engine family, and apply a similar notion of family to the development of the control software for the EEC.

The EEC is required to provide control in a number of different domains. Three typical control domains are outlined below (a good technical guide to how aero-engines work can be found in [11]):

- *Engine Starting:* The rotation of the engine shaft from a still position to a self-sustaining idle speed. This includes control of fuel-flow into the engine and subsequent ignition.
- *Reverse Thrust:* Slowing down the aircraft quickly after landing by reversing the direction of engine thrust. This includes control of the thrust-reverser doors which deflect the engine thrust.
- *Signal Validation:* Checking readings from sensors for accuracy. This may include comparison with a value taken from a pre-defined model of normal engine operation, or cross-checking with similar values.

We have performed domain analysis case-studies over these three domains, a summary of which is shown in Table 1. A distinctive feature of our case-studies is that we aimed our domain analyses at the level of requirements, as we believe this will give us greater leverage in any later design and code level analysis. This follows a similar theme to work in domain-specific software architectures [12]. We also felt that because of the complex nature of the systems in these domains, it would not have been sensible to have attempted to identify reuse at the design and code level without first understanding requirements. The 'size of typical system document' field provides some indication as to the size of the domain. The 'key output' addresses the question of what tangible artifacts were produced from the domain analysis – essential for making a business case for domain analysis which can be presented to senior management.

4. Choice of Domain Analysis Method

How did we carry out the domain analysis in these case-studies? We used the Organisational Domain Modeling (ODM) method, as the method has a background of industrial usage and is well-documented [8]). Very briefly, the main stages in ODM are:

- *Define the Domain.* Bound the domain of focus, and put it in context by defining its relationship with other domains.
- *Acquire Domain Information.* Gather information about the domain from examining system documentation and talking to domain experts.

Table 1 Summary of domain analysis case-studies

Domain	*Time Spent*	*Level of Focus*	*Number of Domain Experts Involved*	*Size of Typical System Document (single-spacing)*	*Key Output from Domain Analysis*
engine starting	1 man-month	system functional requirements	3	System functional requirements document is 30 sheets of A4	30 generic functional requirements
reverse thrust	3 man-weeks	system functional requirements	2	System functional requirement document is 21 sheets of A4.	18 generic functional requirements
signal validation	2 man-weeks	software functional requirements; architectural software design	1	Software requirements document is 15 sheets of A4	Generic DFD diagram; 10 generic software functional requirements

- *Develop Descriptive Models.* Develop different models of the domain, paying particular attention to those aspects of the domain which might be considered common to all systems in the domain, and those aspects which are more variable.
- *Refine the Domain Model.* Integrate the separate descriptive models into a single, consistent domain model.
- *Scope the Asset Base.* Prioritize the variations, so that the most used variations, or those pertaining to the most important customer, are given a higher priority.
- *Architect the Asset Base.* Determine how assets — the reusable components — are to be parameterised and linked together.
- *Implement the Asset Base.* Create the assets, and develop an infrastructure, such as a tool, for organising assets.

The full ODM includes two initial stages for selecting project objectives and defining candidate domains for domain analysis; we have omitted them here as our choice of domain was already determined.

5. Heuristics for Domain Analysis

In this section, we describe the heuristics we have formalised for domain analysis. The heuristics are intended to complement the ODM method, although they could work equally well with other domain analysis methods. We have tried to think deeper beyond the scope of our own case-studies and eliminated what we see as possible bias towards the avionics domain (although there may be an unavoidable element of bias towards embedded control systems). This is essential if the heuristics are to be of use to domain analysts working in non-avionics-related domains. In order to facilitate ease of understanding, we have characterised each heuristic using the following pattern:

- *Triggers:* When and in what situations to use the heuristic. We also indicate the ODM stage or stages the heuristic is most applicable to.
- *Guidelines:* The steps the domain analyst should take.
- *Example:* An illustrative example of the heuristic being applied (taken from one of our case-studies).

The heuristics also form part of a domain analysis guidebook that we are currently developing.

5.1 Heuristic 01 — Delineate overlapping domains

Triggers

- *Unclear domain boundaries:* The first step in domain analysis is the definition of domain boundaries. However, domain analysts who themselves may be new to a domain will often find it difficult to identify clear-cut domain boundaries.
- *Shared or distributed functionality:* One domain may use or provide 'services' to another domain. This can confuse where the domain boundaries actually are.
- ODM stage: Define the domain.

Guidelines

- Sketch out 'rough' domains pictures using simple Venn-like diagrams (as shown in Figure 2) which show the selected domain and its relationship with other related domains.

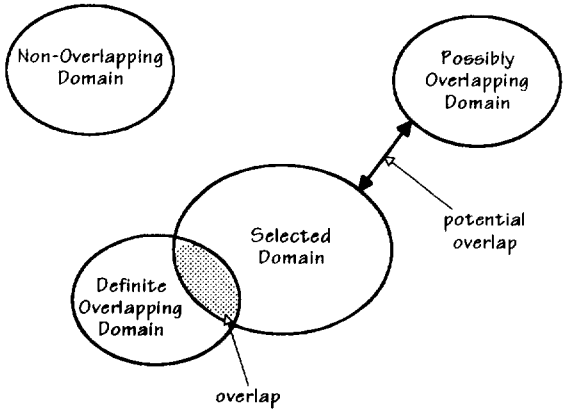

Figure 2 Venn-diagram of domains

- Identify definite regions of overlap between domains. Note that overlap can be characterised in a number of ways: for example, shared functions, functions used by one domain but 'provided' by another and objects used in both domains.
- Indicate any possible regions of overlap, where it is unclear if there is some cross-over or interaction between domains.
- Decision time: decide if the regions of overlap, both definite and possible, should be included as part of the domain in the domain analysis. This should take the form of a consultation process with the domain experts.
- Avoid taking on 'large' domains (in our experience, these tend to be domains which are beyond the knowledge of a single individual) which are generally beyond which often results in an unwieldy and difficult to manage domain analysis process. Instead, treat a large domain as the individual analysis of sub-domains.
- Note that once the domains have been 'sketched out' as above, one can then proceed to define an individual domain more thoroughly (for example, using ER diagrams).

Example

Figure 3 shows the use of Venn-diagrams in defining domain boundaries. The domain of engine starting has a definite overlap with the domain of fuel management — fuel needs to be properly scheduled into the engine for a successful engine start. There is also potential overlap with the domain of signal validation as all signals used by the EEC for engine starting are first validated before being used.

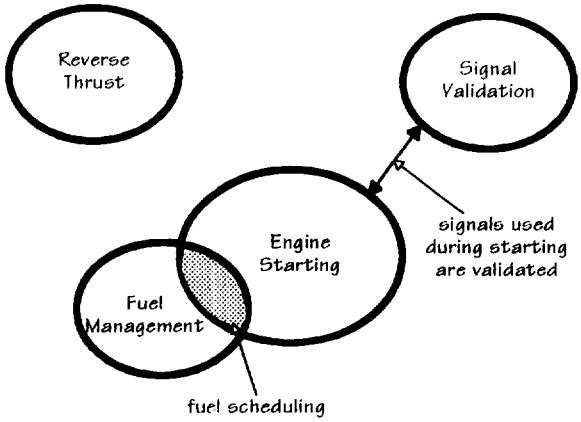

Figure 3 Venn-diagram for engine starting and related domains

The domain analyst needs to be aware and alert for such subtle interactions when attempting to define the domain at the start of domain analysis. We decided in the end to treat the detailed aspects of fuel scheduling and signal validation as areas *outside* of the engine starting domain to keep our analysis as focused as possible.

5.2 Heuristic 02 — Consider the "style" of the domain model

Triggers

- *Confusion about the nature of the domain model:* The main output from domain analysis is often a domain model, but what do we mean by a domain model? Does a domain model contain all or any of the following: a taxonomy of domain concepts, an object model of the domain (cf. object-oriented analysis [13]), an object-composition model (Figure 4), a set of abstract system requirements, a semantic network of concepts and relations.

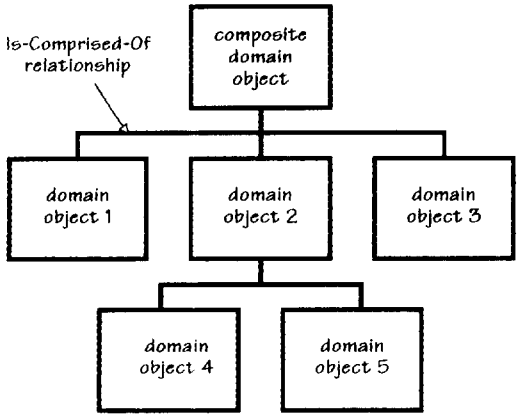

Figure 4 Skeleton object-composition diagram

- *Domain knowledge representation:* What notations are to be used to represent the domain model? If the domain model contains different kinds of domain knowledge, it is unlikely that a single representation will be sufficient.
- *Guide during domain knowledge acquisition:* What domain knowledge is being elicited from the domain knowledge ? What kind of questions should be asked?
- ODM stage: Acquire domain information.

Guidelines

- Review what the objective of the domain analysis is: Requirements reuse? Code reuse? Or to produce pedagogical material for non-experts?
- Determine the information needed to create a domain model which supports the stated objectives.
- Think about the representations that are most appropriate for the kind of domain model desired.

Example

The objective of our domain analysis of the reverse thrust domain (and the other domains) was to achieve the reuse of requirements. Our domain model therefore is essentially a model of requirements. In parallel, we also found it useful to record in the reverse thrust domain, which we did using an object-composition model (Figure 5).

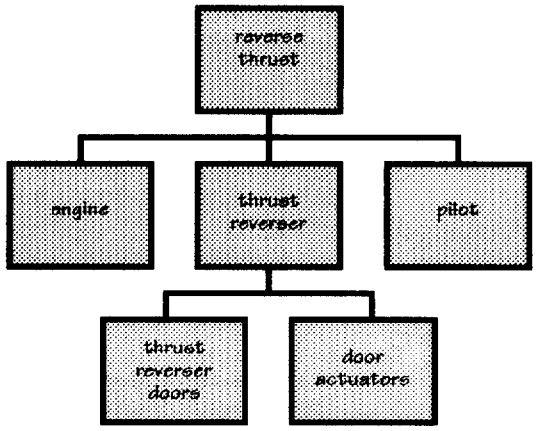

Figure 5 Partial object-composition model for reverse thrust

5.3 Heuristic 03 — Identify the domain issues early on

Triggers

- *Lack of domain road-map:* Unless the domain analyst is also a domain expert, he or she will also need to learn about the domain in order to perform a domain analysis effectively. In highly technical domains, a good starting point is often a high-level picture of the domain.
- *Information overload:* At the start of domain analysis, a domain analyst may be faced with a large amount of system documentation and domain-related materials.
- *Focus during knowledge acquisition:* The acquisition of knowledge, particularly in interview situations with domain experts, needs to be coordinated in order to ensure there is proper focus for discussion.
- ODM stage: Acquire domain information

Guidelines

- Identify the main functional areas of the domain, as these are likely to be central domain issues. This can often be done by looking at the structure of system

documentation, and picking out sections which address key topics.
- Write candidate issues down, and discuss them with a domain expert: "can you explain this issue to me?". More often than not, explanation of an issue will raise further questions and uncover new terminology, providing further material for discussion.
- Probe further: Have some issues been left out? Are some issues more important than others? Are some issues mandatory and some optional? Look to produce a map of the domain using these issues.

Example
Reading the functional requirements documents for several different starting systems provided pointers to the important issues in the engine starting domain; some of these issues are listed Table 2.

Table 2 Some issues in the engine starting domain

Issue	*Brief description*
cockpit signals	How a pilot controls the engine starting process is different due to variation in cockpit layout.
Possible starting modes	Depending upon where the aircraft is, its altitude, speed, and the receipt of the appropriate cockpit signals, an engine can be started in one of many different starting 'modes'.
continuous ignition	Continuous ignition enables an engine to be continuously lit for a period of time.
engine re-light	Engine re-light prevents a 'flame-out' condition when the engine becomes unlit.
engine shut-down	Shutting down an engine requires that certain engine components become disabled.
cranking	'Cranking' is an engine maintenance feature, and refers to the starting of an engine but without the engine actually becoming ignited.
rotor-bow protection	Rotor-bow protection prevents excessive damage being done to the motor in the event of the rotor-blades becoming locked.

We used the domain issues are subject areas which we could 'expand' on, for example, as a way of devising questions before interviewing domain experts.

5.4 Heuristic 04 — Recognise knowledge specialisms

Triggers
- *Search for rationale and explanation*: It may not be obvious why certain behavior occurs in the domain. The search for rationale and explanation is expected to form a large part of the work performed by the domain analyst.
- *Partial domain picture and bias*: Speaking to a single domain expert or using only a single source of domain knowledge is likely to give only a partial picture of the domain. A domain analyst should seek to construct a broader picture of the domain.
- ODM stages: Acquire domain information and develop descriptive models.

Guidelines
- Use system documentation and text books to find detailed knowledge about a domain. Note down any ideas which are difficult to understand.
- Make effective use of the domain experts time. Use the analytical expertise of the domain expert to help explain hard-to-understand concepts (rather than just regurgitate details readily found in system documentation).
- Recognise that it may be necessary to consult a number of domain experts, each with their own expertise or viewpoint on the domain. In avionics-related domains for example, there are typically functional, safety and performance experts.

Example
In the signal validation domain, we noted from the system documentation that timing delays were 'built into' the validation procedure. Initially, this was confusing, as it was not apparent why such timing delays were needed. We noted this point down for explanation at the next scheduled meeting with one of the domain experts. The domain expert informed us that timing delays are needed to 'ride over' momentary surges produced by the engine.

5.5 Heuristic 05 — Achieve a consistent view

Triggers
- *Inconsistent information*: Information from one source conflicts with information from another source.
- *Conflicting opinions*: Domain experts provide domain knowledge which appears to be in conflict with existing domain knowledge.
- ODM stages: Acquire domain information and develop descriptive models.

Guidelines
- Question conflicts: Is there really a conflict? If so, what actually is at conflict?
- Identify the type of conflict. Is the conflict based on factual information or a difference in personal opinion? In practice, we have found that conflicts are usually a result of personal opinion rather than based on direct facts.
- Is it necessary to resolve the conflict? Conflicts in terminology for example, may be acceptable (and perhaps desirable) in multi-project environments.
- In situations where one expert directly disagrees with another, it may be wise to seek a further opinion.

Example
With multiple domain experts being involved in all our domain analyses, there was a need to cross-check the information given by each domain expert for consistency.

Opinions differed, for example, on the naming convention applied to engine start modes. We had chosen to include experts from many different projects to give a large basis of expertise, and found that the terms used varied with the definitions adopted on different projects. However, we recognised this to be a point of consistency management rather than conflict resolution.

5.6 Heuristic 06 — Domain knowledge and system knowledge is entwined

Triggers

- *Developing a generic model of the domain*: Reusable artifacts are usually generic in nature with application across a family of systems in a domain. Deciding on an appropriate level of generality, however, requires an appreciation of the interaction between generic and system-specific information.
- *Information Weeding*: Given a body of information, one of main tasks of a domain analyst is to weed out system-specific information from more generic information, or abstract out generic information from system-specific information.
- ODM stages: Develop descriptive models and refine the domain model.

Guidelines

- A useful way to view domain modeling is as the separation of three layers of information (Figure 6).

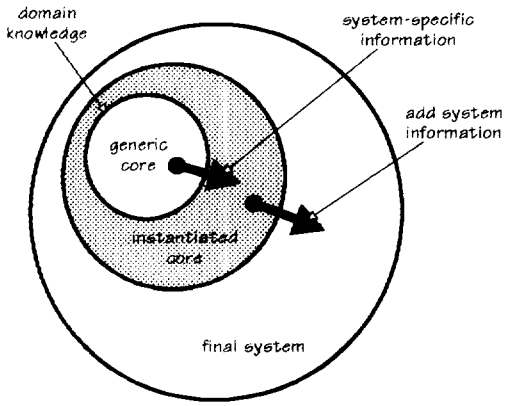

Figure 6 Information layering

- The generic core is what we might equate as the domain model. Concepts in the generic core should be common or optional to all systems in the domain (or a selected portion of the domain).
- The instantiated core represents the specialisation of the generic core. The specialisation is specific to a particular system.
- The 'final system' includes novel or unusual features which are not subject to reuse and therefore outside of scope of the domain model.

Example

In the engine starting domain, the notion of 'flight envelope' is often used to express the relationship between altitude and engine speed. Within a certain range of altitude and engine speeds, it is preferable to perform a starter-assisted start rather than a windmill start (i.e. without the starter motor), and vice-versa. We might consider the basic idea as explained above, as domain knowledge and part of the generic core. The specific values for these ranges is considered as system knowledge and part of the instantiated core, specific to an individual aircraft engine. Note, however, that in this case, the system knowledge can not be understood in isolation from the domain knowledge; we need both to appreciate the full picture of flight envelopes. Perhaps we should say that we have to know how to instantiate domain knowledge in order to generate system knowledge, and that the process of refining domain knowledge is aided by an analysis of the relationship between the two. We also consider that the symbiotic relationship between the domain and system knowledge is a characteristic of the safety-critical systems domain.

A novel feature, such as the ability to prevent the starter-motor operating under normal operating conditions, would be an aspect of the final system.

5.7 Heuristic 07 — Familiar notations facilitate understanding

Triggers

- *Domain knowledge representation*: How should domain knowledge be described?
- *Difficulty understanding new notations*: The domain expert has difficulty understanding the domain model.
- ODM stage: Develop descriptive models.

Guidelines

- Think about the notations that are most appropriate for representing domain knowledge.
- If possible, chose notations which are already in use.
- Recognise that new notations or languages will incur a learning overhead, and may not be conducive to the overall domain analysis process.

Example

The domain experts we dealt with were already acquainted with notations such as data-flow, entity-relationship and state-transition diagrams. It seemed sensible to use these for developing the descriptive models wherever appropriate. For example, we used state-transition diagrams to model the operation of different starting modes, and entity-relationship diagrams to describe the components of an ignition system (note here that there is some overlap in the sense that states in the starting modes may be related to states of the entities in the ER diagram). In the signal validation domain, we used data-flow diagrams to model the validation schemes for individual signals.

5.8 Heuristic 08 — The right way to generalise

Triggers

- *Multiple generalisation options*: An inherent part of domain analysis is generalisation across a family of systems. However, there are often different ways in which concepts can be generalised.
- ODM stages: Develop descriptive model and refine the domain model.

Guidelines

- Identify the different possible generalisation options.
- Balance the level of reuse afforded by each option against other factors such as ease of understanding and efficiency of reusable components.

Example

An engine starting system is often characterised as a number of starting modes. There is a great deal of overlap between the functionality provided by the modes (all of them control fuel flow, igniters and valves in similar ways). The core control (the parameters on which the decisions to control the start are based) is different for each starting mode. Figure 7 shows two different ways of generalising a starting mode.

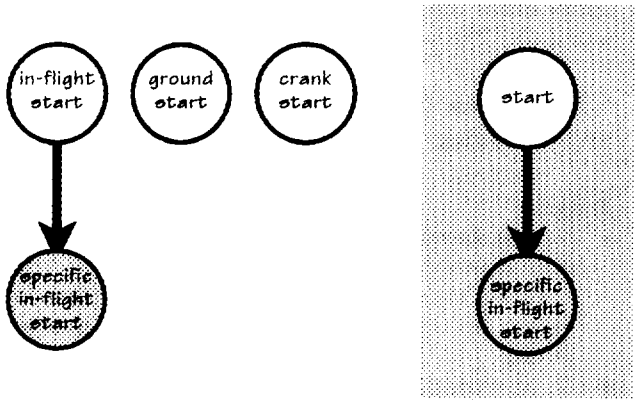

Figure 7 Two different generalisation strategies

On the left, a generic state-transition diagram exists for each starting mode. The Ground Start mode is instantiated with specific information about ground start to form a specific ground start module. On the right, a single generic state component applies to all starting modes, combining the common functionality wherever possible, and is again instantiated with information about ground start. It is important to remember we are talking about a requirement specification here not a design - whilst we wish to avoid repetition and to highlight common functionality within the specification, we also need to make each part of the specification complete and easy to understand.

The structure with separate generic modules for the different starting modes was eventually adopted, even though there was a high degree of commonality between the functionality provided by the different starting modes. The common functionality was factored out into a number of starting utilities (one for each kind of fuel valve and air valve for example) which could be used by all modules in the specification. The main reason for this was readability, the number of switches needed to specify the requirements as a single generic module made the specification over complex and hard to understand.

5.9 Heuristic 09 — Organise and structure the domain knowledge

Triggers

- *Framework for organising domain knowledge*: One output from a domain analysis is a domain analysis report which includes documentation of the domain model and associated artifacts. Some thought needs to go into how the domain analysis report should be organised so that the document becomes a resource to others in the organisation, not just the domain expert.
- *Need for managerial or non-expert view of a domain:* Managers and engineers who are non-domain experts may want to view information about the domain. This information needs to be pitched at an appropriate level: not too detailed and not assuming existing knowledge about the domain.
- ODM stage: At the end of domain analysis.

Guidelines

- We mentioned in heuristic 03 the importance of identifying domain issues early on during the domain analysis process. Issues are also a logical way of structuring domain information.

Example

Figure 8 shows a simplified view of how we have structured the reverse thrust domain into issues.

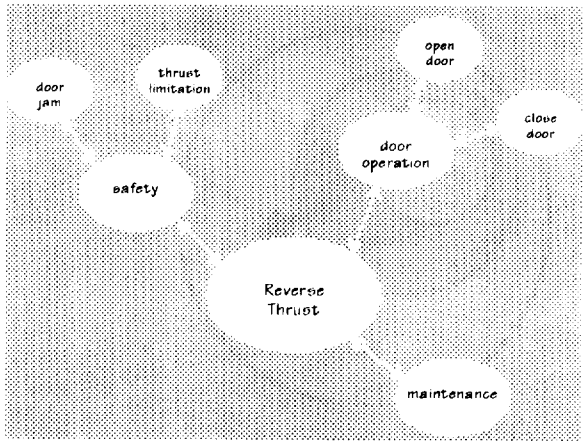

Figure 8 Using patterns to structure the aero-engine domain

We have identified three main issues in this domain: safety, door operation and maintenance. Within each main issues, we have identified a number of sub-issues (not all of them shown). For example, the safety issue has sub-issues concerning the jamming of door and thrust limitation (when the doors are in the process of being opened or closed, there is a maximum thrust limit depending upon the position of the doors). We have organised the sections of our domain analysis reports around the issues in the domain and we feel this has greatly improved readability, both from the perspective of domain experts and non-experts.

5.10 Heuristic 10 — Parametisation can help factor our variability

Triggers

- *Creating reusable components*: Creating reusable components which can be used by application developers is one of the most tangible and visible outputs from the domain analysis process.
- ODM stages: Architect the asset base and implement the asset base.

Guidelines

- Isolate the optional or variable elements of a concept, and build these in as parameters for reusable components.
- The common and more stable elements of a concept should not be parametised.
- Avoid, however, large components with an excessive number of parameters as this can lead to internal and external problems. Internal problems in relation to the efficient implementation of the component, and external problems in relation to developers understanding the component.

Example

We have developed a number of reusable requirements for signal validation by identifying and parametising the elements which vary in system-specific requirements. The reusable requirement below is concerned with the synthesised model check and range check typically found in specific validation schemes:

IF (N2V is less than N2P30ModLim) AND (N2VStat is not 'Failure') THEN
  P30RangeHigh equal P30RangeHigh1
ELSE
  P30RangeHigh equals P30RangeHigh2
ENDIF
SET P30OwnFault RANGECHECK (P30OwnRaw, P30RangeLow, P30RangeHigh)

where:

N2V = The value of validated N2 (the middle engine shaft speed)
N2P30ModLim = The value of N2, above which a model of P30 will be used for validation
N2VStat = N2V status
P30RangeHigh = The Variable which takes the value of the selected upper range limit
P30RangeHigh1 = The upper limit of P30 range check, when N2 below P30N2ModLim
P30RangeHigh2 = The upper limit of P30 range check, when N2 above P30N2ModLim
P30OwnFault = A flag indicating if P30Own is in range
P30OwnRaw = The Value of raw P30Own
P30RangeLow = The lower limit for P30 range check

Note how we have factored out the variable elements as parameters or requirement variables. We have also found that timing and iteration requirements can be readily parametised:

The time allowed to perform one single process of input validation shall not exceed ExecutionTimeLimit milliseconds.

The iteration rate for the process of input validation shall be IterationRate milliseconds.

where:

ExecutionTimeLimit = Maximum single execution time
IterationRate = Iteration rate

5.11 Heuristic 11 — Model mandatory and optional concepts to understand choice in the domain

Triggers

- *Documenting commonality and optionality relationships*: One emerging principle of domain analysis is to separate aspects which are common to all systems in a domain from those which are not. In practice however, we discovered that concepts often 'interacted', and resided at different levels of abstraction, making it difficult to establish a clear-cut boundary between what was common and what was optional.
- ODM stages: Develop descriptive models, refine the domain model, scope the asset base and architect the asset base.

Guidelines

- Use the framework shown in Figure 9 as a starting point for modeling mandatory and optional concepts (a concept may be a requirement, an object or anything of importance in the domain).

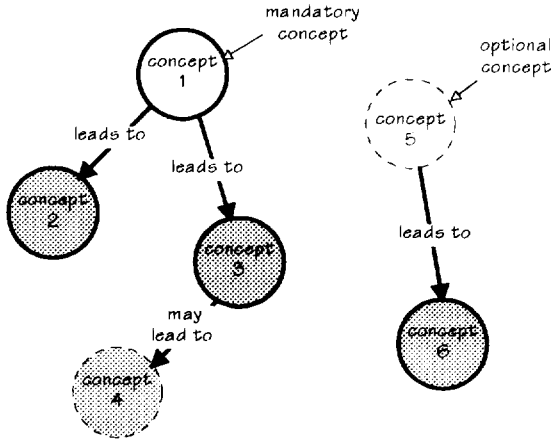

Figure 9 Framework for modeling mandatory and optional concepts

- Model the main mandatory concepts, represented as a white bubble, first. Mandatory concepts are the concepts which are common to all systems in the domain).
- Model mandatory concepts which are dependent upon a main mandatory concept using the 'leads to' relationship.
- Model optional concepts (represented as a bubble with a dotted line) using the 'may lead to' relationship.

Example

The engine starting domain has over 70 individual 'leads to' and 'may lead to' relationships - this accounts for a significant portion of domain knowledge. Figure 10 shows the relationship between a number of concepts in the domain.

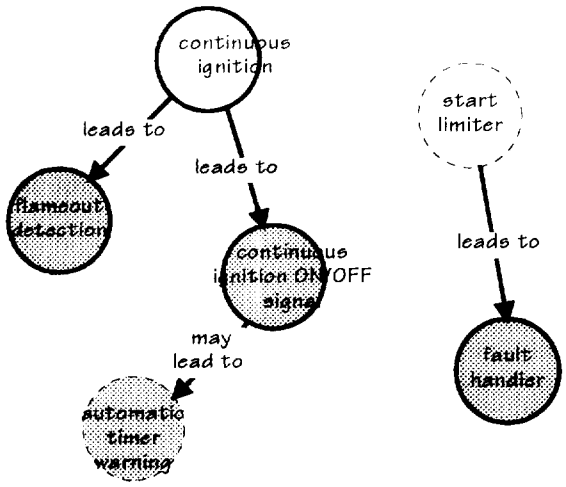

Figure 10 Modeling mandatory and optional concepts in the starting domain

Here, the concept of 'continuous ignition' must lead to a way of detecting that the ignition has been extinguished (known as a flameout) and a continuous ignition signal from the cockpit (we call these 'follow-on' concepts). Note, however, that a continuous ignition signal may lead to an automatic timer warning (as excessive use of continuous ignition significantly reduces the normal life of an igniter).

5.12 Heuristic 12 — Clarify Assumptions

Triggers

- *Querying the domain model*: During the development of a domain model, the domain analyst and domain expert will need to review the domain model at various checkpoints to check the accuracy, completeness and usefulness of the domain model.
- *Validating the domain model*: At the end of domain analysis, there should be a formal period of validation where the domain model can be inspected by all the domain experts.
- ODM stages: Develop descriptive models, refine the domain model, scope the asset base, architect the asset base and implement the asset base.

Guidelines

- Critique the assumptions made by the domain model and reusable artifacts: Are the assumptions valid for all systems in a domain or just a selected portion? Are such assumptions realistic? Are there any known cases which refute the assumptions? Are there any technological developments in the near future which will supersede or render such assumptions void (this is of particular importance in domains which might be considered as 'high-tech')?

Example

Figure 11 shows a generic dataflow diagram (DFD) which represents the software structure for a signal validation scheme. In short, inputs to the EEC need to be validated to ensure that inaccurate measurements (when a sensor is fault for example) are not used for control purposes. We developed the generic DFD by studying several 'concrete' DFDs for particular validation schemes and identifying common elements. However, the generic DFD does make a number of assumptions. For example, assumptions are made about the basic architecture of the EEC, such as existence of a fault system, aircraft maintenance system and health system. We have also assumed that there will be at least three kinds of validation strategies in operation: range check input, cross-check input and model-check input. In addition, the existence of a cross-check input implies a non-single lane EEC architecture (a lane may be thought of as a 'mirror' processor and databus).

We feel such assumptions are valid for EECs on civil aero-engines, but are probably at question in the context

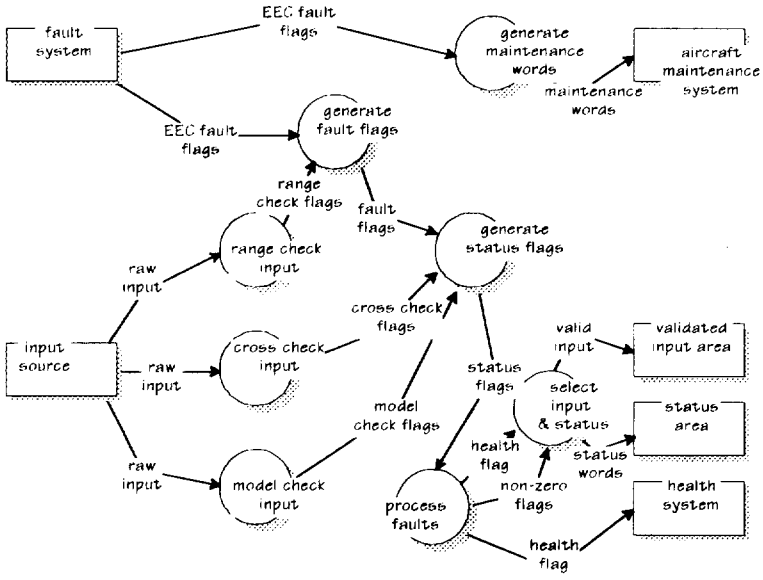

Figure 11 Generic DFD for signal validation

of military aero-engines. Such assumptions need to be clarified when validating the domain model and again when using the domain model.

5.13 Heuristic 13 — Group together related options

Triggers

- *Concepts with optionality*: A domain concept may have a number of associated options. For example, different ways to abort from a ground start might include: turning the fuel off, various signal failure messages, engine over-heating, starter over-heating, hang/stall signal, engine start off, invalid speed or temperature signals or any combination of these.
- ODM stages: Develop descriptive models, refine the domain model and scope the asset base.

Guidelines

- Identify the core, basic concept; these are typically issue or function related.

- Identify all the options associated with a concept.
- Question the option set: Have all the options been included? Decide also, if all the options need to be declared, for example, some options may be viewed as too arcane or unprofitable to support.

Example
Emergency re-light is a feature of some engines which quickly re-ignites the igniter should it become extinguished. This is especially important when full control over the engine is needed such as during take-off. The variable element of emergency re-light is how long the igniters are energised for (excessive energisation reduces the life of the igniter which would be hazardous). We have separated out this variable element in the form of options which are available from the basic requirement (Table 3).

Table 3 A basic requirement and its options

Basic Requirement	When the emergency re-light comes operational, then:	
Options	1	The igniters are energised for a minimum specified period of time.
	2	The igniters are energised for a maximum specified period of time.
	3	The igniters are energised for a minimum and maximum specified period of time.
	4	The igniters are energised until manually switched off.

5.14 Heuristic 14 — Note domain constraints

Triggers
- *Modeling constraints*: A domain analyst is often concerned with modeling constraints in the domain. For example, When is concept A valid? When is concept A not valid? What is the relationship between concept A and concept B? etc. We have found in the safety-critical systems domain that such constraints needed to be treated as 'first-class' domain concepts.
- ODM stages: Develop descriptive models, refine the domain model, architect the asset base and implement the asset base.

Guidelines
- A useful starting point is to consider the general kinds of constraints that may be worth recording in the domain analysis: For example, functional, performance and safety constraints. This may help to highlight important constraints in the domain, which can then be checked over with other domain experts.
- Record the constraints in a way that they can be tested. This may be in natural English as a manual test by a human, or in a more formal notation which can be automatically checked.

Example
We have an informal notion of 'rules' to model constraints between requirements in the engine starting domain; Table 4 illustrates some of the rules we have used.

Table 4 Illustration of rules

Rule	*Example Use of Rule in Requirements*
AND	All starting systems have requirements for [Ignition Layout] *and* [Ignition Fault Accommodation]
	For an [in-flight windmill start mode], [the engine must not be already running] *and* [the aircraft must be in-flight] *and* [air-speed > windmill start threshold]
OR	When [continuous ignition is switched on], either [one igniter is energised] *or* [two igniters are energised] depending on [number of igniters]
IF	*If* the starting system has [continuous ignition], then there must be a requirement for [continuous ignition signal]

We feel that once a rule base is established it can be used as a means of proving the domain model through providing direct statements which we can use when questioning engineers. The rules can then be used (probably hidden behind a tool) when instantiating the model to guide or automatically select certain domain options depending on the result of related parts of the instantiation. For example, once we have decided how many igniters we have on a system then we can hide (grey out) all of the related options relating to single igniters.

Note that we have yet to formalise the syntax and semantics of the kind of rules in the table above or to fully appreciate the way in which they can be used to express the commonality and variability relationships. The rule system is part of our on-going research.

5.15 Heuristic 15 — Do a variability analysis to anticipate potential change in the domain

Triggers
- *Understand variability in the domain*: One principle of reuse is to distinguish elements in the domain which are constant from those elements which are variable. Understanding potential variability in a domain is a central pillar to mastering reuse.
- *Validation of domain model*: How can we assess if the domain model is accurate, complete and usable?
- *Select market focus*: There may be much variability in a domain. However, from a business perspective, it may be more advantageous to focus on selected variability to address a particular market segment.
- ODM stages: Acquire domain information, develop descriptive models, refine the domain model and scope the asset base.

Guidelines
- Gather together a group comprising of domain experts, systems analysts, end-users and marketing people. Brainstorm to produce a list of variabilities in the domain, i.e. things in the domain which can change.
- Select and prioritise the most important variabilities (this may be guided by a market analysis to identify the most profitable market segments).

- Use the variabilities identified as a checklist against the domain model: is the domain model flexible enough to handle all desired variabilities? If not, domain artifacts may need to be revised.

Example
We did a variability analysis for the reverse thrust domain in the form of a 20 minute brainstorm, the results of which are shown in Table 5.

Table 5 Results of variability analysis

1. Engine design
   1.1 Thrust reverser
      1.1.1 Type of thrust reverser
      1.1.2 Geometry
      1.1.3 Thrust reverser movement mechanisms
   1.2 Engine power management
   1.3 Geometry
   1.4 Electrical system
2. Airframe
   2.1 Cockpit
      2.1.1 Thrust reverser deployment
      2.1.2 Thrust reverser stow
   2.2 Aircraft Communications
      2.2.1 Communications protocol
      2.2.2 Status messaging
3. Devices
   3.1 Actuators
      3.1.1 Deployment actuators
      3.1.2 Stow actuators
   3.2 Sensors
      3.2.1 Deployment sensors
      3.2.2 Stow sensors
4. Safety mechanisms
   4.1 Safe deployment
   4.2 Safe stow
   4.3 Auto re-stow
   4.4 Thrust reverser movement impediment
   4.5 Manual inhibition of thrust reverse (for maintenance)
   4.6 Ground safety switch

The results are presented in a hierarchical form, where large-grain areas of variability have been decomposed into finer-grain areas of variability. We used the variability analysis as a means of scoping our domain analysis. For example, in 1.1.1. Type of thrust reverser, there are 3-4 main different designs of a thrust reverser; we selected one which was typically used on civil aircraft engines as our customer focus was towards passenger-based airline companies. The variability analysis also helped us pinpoint areas in our domain model where greater flexibility was needed.

6. Conclusions: The human element of reuse

This paper has described a set of heuristics for domain analysis based on the industrial domain analysis case-studies we have applied in the area of aero-engine control. We have argued that the success of a domain analysis is largely dependent upon how well the domain analysis process is carried out by the domain analyst. The focus of our work therefore has been on the capture and dissemination of the principles which can be identified with successful domain analyses. The heuristics we have presented go some way towards this, and draw attention to the human aspects of domain analysis which have been overshadowed by largely technical accounts of methods and techniques.

Acknowledgments
The author acknowledges the financial support given by Rolls-Royce Plc, the insightful discussions with other members of the University Technology Centre and engineers at RoSEC, and the help of Dr. Ben Whittle.

References

[1] Poulin, J.S., Caruso, J.M. and Hancock, D.R. (1993), The Business Case for Software Reuse, IBM Systems Journal, 32(4):567-594, 1993.

[2] Gomaa H. (1995) 'Reusable Software Requirements and Architectures for Families of Systems' Journal of Systems and Software, 28:189-202.

[3] WISR (1995), Proceedings of the 7th Annual Workshop on Software Reuse, St. Charles, Illinois, August 28-30, 1995.

[4] WSRET (1995), Proceedings of the 4th International Workshop on Software Reuse Education and Training, Morgantown, West Virginia, 14-18th August, 1995.

[5] Prieto-Diaz R. (1990) 'Domain Analysis: an Introduction', Software Engineering Notes, 15:47-54.

[6] Kang K., Cohen S., Hess J., Novak W. and Peterson S. (1990), 'Feature-Oriented Domain Analysis Feasibility Study' CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University.

[7] Wartik S. and Prieto-Diaz P. (1992), 'Criteria for Comparing Reuse-Oriented Domain Analysis Approaches', International Journal of Software Engineering and Knowledge Engineering, 2(3):403-431.

[8] STARS (1995), 'Organisation Domain Modelling Guidebook', STARS-VC-A023/011/00, March 1995.

[9] Lim, W.C. (1994), Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, 11(5):23-30, 1994.

[10] Joos, R. (1994), Software reuse at Motorola, IEEE Software, 11(5):42-47, 1994.

[11] Treager, I.E. (1994), Aircraft Gas Turbine Engine Technology (2nd Edition), ISBN 0-07-065158-2, Glencoe, Ohio.

[12] Tracz W., Coglianese L. and Young P. (1993), 'A Domain-Specific Software Architecture Engineering Process Outline', Software Engineering Notes, 18(2):40-49.

[13] Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W., 'Object-Oriented Modelling and Design' Prentice-Hall ISBN 0-13-630054-5, 1991.