7. Reubenstein H.B. & Waters R.C., 1991, 'The Requirements Apprentice: Automated Assistance for Requirements Acquisition', IEEE Transactions on Software Engineering 17(3), 226-240.
8 Riesbeck C.K. & Schank R.C., 1989, 'Inside Case-based Reasoning', Lawrence Erlbaum Associates, Hillsdale NJ.
9. Rosch E., 1983, 'Prototype Classification and Logical Classification: the Two Systems', New Trends in Conceptual Representation: Challenges to Piaget's Theory, edited K. Scholnick, Lawrence Erlbaum Associates, Hillsdale NJ.
10. Sutcliffe A.G. & Maiden N.A.M., 1994, 'A Theory of Domain Knowledge for Requirements Engineering', Technical Report CU-94-00B, Centre for Human-Computer Interface Design, City University.

7: Implications for the space of formal problem abstractions

The results led to revision of the problem space. The financial exchange system model has been removed from the highest level of the hierarchy (P2b). There was also little evidence for the resource containment system model (P2b). Experts tended to give categories which could be mapped to the resource hiring system model, so resource containment is replaced with resource returning, resource supplying and resource usage system models which currently specialise the resource containment system model. Indeed, the goal state for resource containment was difficult to express given its abstract nature, therefore removal of this model is in keeping with the importance of system goals.

The space for the lower-level object system models is shown in Figure 7. Domain structure cardinalities have been removed (P1c) and goal states have been merged in line with the results (P2c). Event triggers have been retained (P1d). Object properties are used to discriminate between financial and physical resource hiring at the lowest level of the hierarchy (P1d).

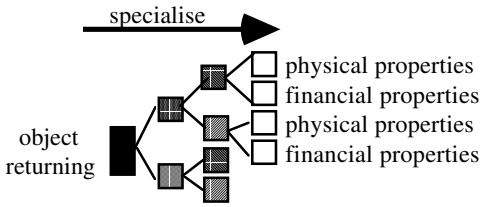

Figure 7. Revised segment of problem space

Many expert and inexperienced software engineers recognised categories which mapped to most object system models regardless of past experience, or the lack of it. Therefore, formal problem abstractions can be recognised by less experienced software engineers, thus enhancing their reusability (e.g. [3]). One reason for this finding may be that the software engineers were not complete novices but had been exposed to sufficient real systems and examples to categorise. Similar card sorts will examine how naive software engineers categorise problems. An alternative explanation is that the problem abstractions are simple software engineering artefacts which do not include complex technical knowledge possessed by more experienced software engineers. As such the abstractions are more natural and accessible to more participants in requirements engineering such as end-users. Problem structuring and scoping, which are difficult requirements engineering activities, can be aided through provision of non-technical, understandable problem abstractions.

Reuse of problem abstractions, natural or otherwise, can also be aided through explanation. The card sorts investigated the naturalness of problem categories with no reliance on explanation. Tool-based explanation [4] can improve the reusability of natural problem abstractions and enable reuse of more artificial object system models at lower-levels of the formal problem space.

Finally, results have wider implications for effective object-oriented abstraction. Attaching goals to objects can enable object-oriented analysis to become more natural and hence usable. There is evidence that application of object-oriented analysis makes requirements more difficult to understand, especially because requirements are often first expressed as functions or tasks. Our approach is similar to that of use-cases which model chains of behaviour triggered by agents in the domain. Object system models describe behaviour to attain a goal state, therefore these models combine advantages from both object-orientation and use-cases as a central reuse technique during requirements engineering.

Acknowledgements

The authors wish to thank colleagues, participants and the Atlantic Systems Guild, MacDonald Dettwiler and BT for their participation. This work is part of the European Union's ESPRIT III 6353 NATURE basic research action.

References

1. Chi M.T.H., Glaser R. & Rees E., 1982, 'Expertise in Problem Solving', Advances in the Psychology of Human Intelligence, ed. R. Sternberg, Lawrence Erlbaum Associates, 7-75.
2. Jarke M., Bubenko Y., Rolland C., Sutcliffe A.G. & Vassiliou Y., 1993, 'Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis', Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 19-31.
3. Lee A. & Pennington N., 1994, 'The Effects of Paradigm on Cognitive Activities in Design', International Journal Human-Computer Studies 40, 577-601.
4 Maiden N.A.M. & Sutcliffe A.G., 1994, 'Requirements Critiquing Using Domain Abstractions', Proceedings of IEEE Conference on Requirements Engineering, IEEE Computer Society Press, 184-193.
5. Maiden N.A.M. & Sutcliffe A.G., 1992, Exploiting Reusable Specifications Through Analogy', Communications of the ACM. 34(5), 55-64.
6. Mylopopoulos J., Borgida A., Jarke M. & Koubarakis M, 1990, 'Telos: Representing Knowledge about Information Systems', ACM Transactions on Office Information Systems 8(4), 325.

Categorisation by inexperienced subjects: again, to our surprise, inexperienced subjects generated similar categories to experts. Goal state categories contained 91.9% of all predicted problems (versus 91.5% for experts). Again a two-way ANOVA with expert/inexperienced and predicted/not predicted categories as factors revealed no effect of experience on categorisation, $F(1,59)=159.4$, $p<0.3$, $MSe=1.573$. Furthermore, as with the experts, predicted categorisations were achieved using object properties but not domain structure cardinalities or event triggers. Again, it was difficult to determine possible links between criteria, categories and past experience for both groups of subjects.

6: Predictions revisited

Two card sorting sessions acquired software engineers' criteria and categories for partitioning two sets of problems for requirements reuse. Results are reviewed in light of whether each prediction was satisfied in full, in part, or rejected.

P1 & P3: software engineers discriminated using criteria mapped to knowledge types in the modeling language:

P1a: partial satisfaction: system type/goal discriminates between the four broad classes of requirements engineering problem used in the first session. It was used to discriminate between more function-oriented problems in which monitoring and allocation take place;

P1b: accept: object returning problems are discriminated according to goal state;

P1c: reject: object returning problems are not discriminated using domain structure cardinalities;

P1d: partial satisfaction: some object returning problems are discriminated using event triggers.

One other criterion not mapped to the domain modeling language (safety criticality) was used by more than half the experts.

P2: software engineers' categories were mapped to object system models in the two segments of the problem space:

P2a: satisfaction: object sensing and resource allocation are natural categorisations for requirements engineering problems;

P2b: rejection: resource containment and financial exchange are not natural categorisations. Resource containment appeared to be an artefact of our categorisation because it is too abstract;

P2c: satisfaction: resource returning problems are discriminated using the goal states *return of resources before a deadline* and *no damaged resources present in the container* into natural categorisations;

P2d: rejection of predictions that resource returning problems are discriminated by other goal states into natural categorisations. Goal states for *return of augmented resources* and *no unserviced resources with clients* were subsumed into the goal state defining *no damaged resources present in the container*, so one implication for the formal problem space might be the need for fewer goal states. This might be because all these problems involve tracking the location of resources regardless of the goal of that tracking;

P2e: partial satisfaction: event triggers discriminate between some resource returning problems.

Object properties were also effective categories for lower-level object system models but were used by fewer experts.

P4: most experts' criteria and categories could not be linked to past experience.

P5 & P6: there were few significant differences between expert and inexperienced software engineers' criteria and categories, in contrast to findings in other disciplines [1]. The software engineers exhibited 3 types of abstraction for the 25 problems. First, purpose-driven abstraction was linked to states to be attained or avoided. Functional abstraction was linked to system designs, for example monitoring, allocating, scheduling and queueing functions. This provides some evidence for a set of design abstractions orthogonal to the problem abstractions. Third, simple problem attribution linked to system functions occured, for example some problems are prototypically more safety critical than others.

It is important to note that given criteria and categories partition only the two investigated segments of the space. These conclusions cannot be extrapolated to other parts of the space. Similar card sorting exercises have been undertaken to investigate the wider space, and it will interesting to compare categorisations and structures for the entire formal problem space.

In support of our method, there was little evidence of lexical cueing from cards. Knowledge types in the domain modeling language were mapped to only 31% of all criteria used the software engineers, and few criteria or categories could be linked to syntactic cues.

These results have implications for the categorisation and structure of the space of formal problem abstractions.

criteria	number of criteria users		average criteria ranking	
	exps (/9)	inexps (/10)	exps	inexps
system goal	8	6	1	1.3
triggering event	4	3	2.28	1.6
object property	3	3	3.33	2.6
no. of transactions objects	2	1	3.5	
safety criticality	2	2	2.5	3
domain structure cardinal.	1	3		3
system complexity	1	2		3

Figure 5. Number of subjects who used criteria to sort problems (only criteria used by more than one subject in either group are shown), and averages of subject ranking of criteria (1 = most powerful discriminator)

Average ranking of these criteria by their power of discrimination is also shown in Figure 5. Again, all experts ranked system goals as the most powerful discriminator while object properties was one of the weakest. Inexperienced subjects also ranked system goals as the most powerful discriminator but this selection was less clear cut and object property was again not the weakest discriminator. There was no evidence for domain structure cardinalities as a powerful domain discriminator.

To recap, the theoretical model discriminates between the 17 problems using domain structure cardinalities, goal states and triggering events. Each was examined in turn.

Categorisation using cardinalities: only one expert categorised problems using cardinalities of the client. These categories were not as predicted, so domain structure cardinality was also not an effective discriminator.

Categorisation using goal states: on the other hand 8/9 experts claimed system goals to be the most powerful discriminant for requirements reuse. Mappings between expert system goal categories and object system models revealed that categorisation of 94.6% of all problems was as predicted. Categorisation is shown in Figure 6:

- *resource hiring* was a category used by 7/8 experts. These 7 experts included 96.4% of all predicted problems in resource hiring. Furthermore, 4/8 experts also included all 3 financial exchange problems in this category;
- *financial exchange* was used by 2 experts who both categorised all 3 financial exchange problems as predicted;
- *resource repairing* was used by 5 experts, all of whom included all predicted problems, although 3/5 also included additional problems;
- *checking resources* was used by 3 experts, all of whom included the predicted aircraft safety checking and dental check-up problems but not recall of faulty goods;
- *upgrading resources* was also used by 3 experts, 2 of whom included all predicted problems while the third omitted the antiques restorer problem.

hypothesised categories	number of experts	%age correct	total incorrect
resource hiring	7/8	96.4%	19
resource repairing	5/8	100%	14
resource augmenting	3/8	91.6%	1
resource checking	3/8	66.7%	0
financial exchange	2/8	83.3%	0

Figure 6. Use of categories within the predicted goal state criterion

Two experts also introduced categories such as monitoring, scheduling and queueing which had no mapping to the predicted object system models.

Categorisation using event triggers: 4/9 experts used event triggers as a criterion. However, these subjects used diverse categories, such as one-off versus continuous behaviour, customer versus system-initiated processes and triggers beyond the system's control. It was impossible to map these categories to object system models.

Other categories: the theoretical model also includes object properties as a weak discriminator between object system models. Three out of 9 experts categorised problems using object properties. For these 3, categorisation of 90.2% of all problems using this criterion was as predicted, and no unpredicted categorisations were made. All 3 experts categorised the "financial" problems as having financial objects and other problems as having physical objects, although for one expert, this was an aggregation of electrical goods, publications, fleets, cars and repairing multi-component objects. Therefore, object properties, when used, were an effective but weak discriminator between problems.

To validate predicted object system models, categories of 5/9 experts who used the system/type goal criterion were mapped to resource allocation, resource containment, financial exchange and object sensing system models, see Figure 4. Of these 5, all recognised categories for resource allocation and object sensing. This categorisation was as predicted, in that 90% of resource allocation problems were categorised as "allocation" problems and 80% of object sensing problems were categorised as "monitoring" problems. Six problems which were not predicted as resource allocation or object sensing system models were also categorised as "monitoring" or "allocation" problems, but no pattern was discernible.

hypothesised categories	number of experts	%age correct	total incorrect
resource allocation	5/5	90%	3
object sensing	5/5	80%	3
resource containment	2/5	50%	2
financial exchange	1/5	100%	0

Figure 4. Use of categories within the predicted criterion

On the other hand, expert categorisation of resource containment and financial exchange problems was not as predicted. Only one expert categorised the mortgage lending and ATM problems as "financial exchange" problems. Furthermore, only two experts used the resource containment category to categorise the book lending and mortgage loan problems as "lending" problems. Indeed, the stock control problem was placed in an unpredicted category by 4 of these 5 experts, which was twice as often as any other problem.

Experts also used other criteria which could not be mapped to knowledge types in the domain modeling language. Five out of the 9 experts used safety criticality as one discriminating criterion. Categorisation for this criterion was consistent across experts. All 5 attributed the scheduling airport gate allocation, monitoring train safety and aircraft flight plan adherence problems to a high safety criticality category while other problems were medium or low criticality categories. The computational type criteria, used by 3/9 experts, divided problems into real-time or various not real-time categories.

Although the experts used diverse criteria and categories, it was difficult to determine possible links between them and experts' past experience of real-time or business domains and systems.

To our surprise, selection of criteria by inexperienced subjects was similar to that of experts. All 10 subjects used goal states as a criterion. All 10 also used resource allocation as a category, 8 used object sensing, 5 used resource hiring and none used financial exchange. Categorisation also mirrored that of experts, for example 16 out of the maximum possible 20 resource allocation problems were categorised as "allocation" problems and 13/16 object sensing problems were categorised as "monitoring" problems. A two-way ANOVA with expert/inexperienced and predicted/not predicted categories as factors revealed no effect of experience on categorisation, $F(1,59)=51.733$, $p<0.82$, $MSe=0.544$. However, the number of categorisations which did not map to the theoretical model for all problems was greater for inexperienced software engineers (19 versus 8 for experts).

To conclude, results from the first session provided empirical evidence for natural categorisation of resource allocation and object sensing system models. However, there was little evidence for the financial exchange system model. Instead, financial problems were categorised as either lending or monitoring domains. Furthermore, experts preferred to use resource hiring rather than resource containment system models for the mortgage lending problem.

5: Results from second sort session

As in the first session, experts used more criteria than did inexperienced subjects (average 3.56 versus 3) but fewer average categories per criteria (2.53 versus 3.33). However, unlike session one, neither difference was significant. Criteria used by most experts were system goals (8/9), triggering events (4/9) and object properties (3/9). Criteria used by inexperienced subjects were system goals (6/10), triggering events (3/10), object properties (3/10) and domain structure cardinalities (3/10), see Figure 5. Again, experts showed greater diversity in their use of criteria. Experts used 10 criteria unique to one person while inexperienced subjects used only 3.

the time limit. At the beginning of each sort, all the cards were shuffled before being given to the software engineer;

2) after each sort, the experimenter recorded the software engineers' chosen criteria and categories using written notes and recorded verbal descriptions;
3) after all sorts in either session were completed, the software engineer was requested to rank each of the criteria selected by them in the previous 5 sorts according to their power of discrimination for requirements reuse;
4) at the end of both sessions, a questionnaire elicited the software engineer's previous experiences and training.

In addition, during the second session, each software engineer was given a text and pictoral description of the behaviour and structure of the object returning system model which was common to all problems given in that session. This was expected to aid comprehension of the 17 problems without biasing software engineers' selection of criteria which specialised the core resource returning system model.

All software engineers completed both sort sessions. They tended to use different names for criteria and categories, therefore a blind judge was requested to determine semantic equivalence between these criteria and categories to enable a cross-subject analysis of results. A second blind judge then mapped these criteria and categories to the formal problem space. This process was in 4 stages:

1) the first judge, with no knowledge of the problem space, determined semantically-equivalent criteria used by different software engineers;
2) this judge also determined equivalent categories used by different software engineers. This was done for categories within equivalent criteria, so two categories provided by two software engineers were equivalent if the criteria for these categories were also equivalent;
3) the second judge, who was familiar with but not an author of the formal problem space, mapped common criteria to knowledge types in the domain modeling language;
4) the second judge also mapped semantically-equivalent categories to object system models in the formal problem space. As with the first judge, categories were mapped to object system models if the criterion for the categories was mapped to relevant knowledge types in the domain modeling language.

These two mapping processes were central to interpretation of results and hence acceptance or rejection of the 6 predictions. For this reason, the use of two independent judges was adopted.

4: Results from first sort session

Broad analysis revealed that experts used significantly more criteria than did inexperienced subjects (average 4.67 versus 3.8, unpaired t-test, $t=2.51$, $p<0.05$) but there was little difference in their totals of average categories per criteria (2.74 versus 2.92).

Five out of 9 experts used system type/goal (e.g. "monitoring", "lending") as a criteria, 5/9 used safety criticality as criteria and 3/9 used the computational type. There was little cross-expert agreement on other criteria, see Figure 3. Surprisingly, inexperienced subjects showed more agreement over criteria. All 10 inexperienced subjects used system type/goal and 6 used organisation type. Seventeen criteria were used which were unique to one expert while the 10 inexperienced subjects used only 9 unique criteria, thus revealing greater diversity in the criteria used by experts.

criteria	number of criteria users		average criteria ranking	
	exps (/9)	inexps (/10)	exps	inexps
system type/goal	5	10	1	1.2
safety criticality	5	3	2.6	2.67
computational type	3	1	2	1.5
organisation type	2	6	4.5	3.16
factors activating system	2	0	3.5	
automated or manual	2	3	3.2	3.2
system complexity	2	0	2.5	
domain structure cardinal.	1	3		2.67

Figure 3. Number of subjects who used criteria to sort problems (only criteria used by more than one subject in either group are shown), and averages of subject's ranking of criteria (1 = most powerful discriminator)

Subject's average ranking of each criterion according to this power of discrimination for requirements reuse is also shown in Figure 3. All experts who chose system type/goal ranked it as the most powerful discriminator while organisation type was the weakest. Inexperienced subjects also ranked type/goal as the most powerful discriminator but this selection was less clear cut and organisation type was not the weakest discriminator.

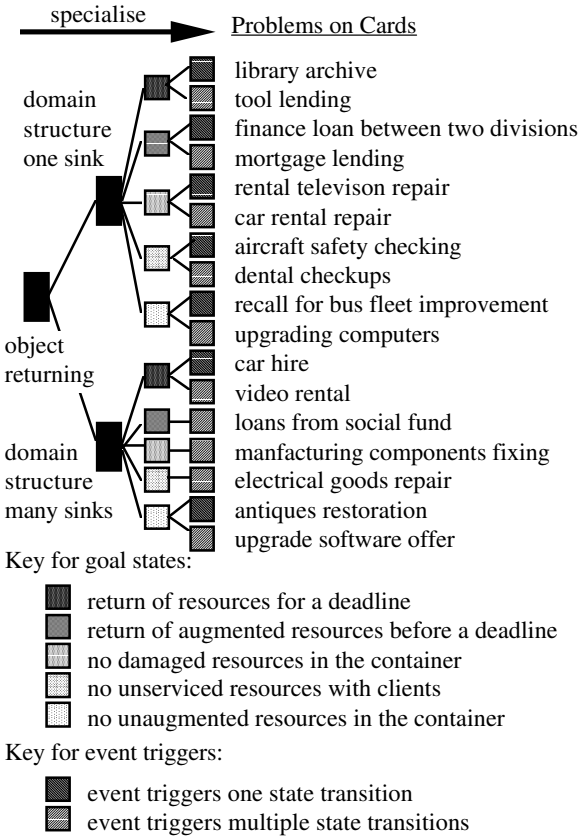

Figure 1. Segment of the formal problem space used in the second session

3: The card sorting method

Mental categorisations were elicited using card sorts, a knowledge acquisition technique which acquires criteria and categories of objects, or in this case, requirements engineering problems. Experts were requested to sort problems described on cards into categories according to their own criteria (e.g. [1]). Card sorts are quick, systematic and easy to use, which was important to ensure the participation of experienced software engineers.

The subjects were 9 experienced and 10 inexperienced software engineers. Experienced software engineers were 8 practicing systems analysts and consultants with a minimum of 10 years systems analysis experience and one university senior lecturer in software engineering. Inexperienced software engineers were final-year undergraduate students in business computing with a maximum of one year's industrial training. Each of the 25 cards used in the two sessions described a software engineering problem in one paragraph of English on a 3"x5'" card. Each described that problem's objects, agents, behaviour, structure, goals, constraints and functions, see Figure 2.

Monitoring Train Safety

A computerised system is required to improve attainment of goals in the following domain. The domain is composed of a railway line divided into many track sections. Trains travel along the railway line and must be in only one track section at a time. The driver makes the train move along the track sections to get from one destination to another. The aim of the domain is to monitor trains to ensure that at no time should there be more than one train on each track section. A train must only travel into the next track section when no train is present in that section.

Figure 2. An example card

It is important to recognise several limitations of our card sorting method. To minimise undesirable lexical cueing from the cards during categorisation, each problem was described using application-specific terms which were not shared by most problems. These descriptions were also produced by someone familiar with the object system models but not an author of the formal space. Therefore, it was deemed unlikely that useful criteria and categories could be determined from the syntax of problem descriptions. Rather, software engineers needed to understand the problems to sort them.

Selection of criteria can also be biased by the semantic contents of problem descriptions. All cards had to provide sufficient detail to ensure understanding of problems. A simple application name was insufficient. Each problem was therefore described using familiar software engineering features, some of which overlapped with features of the domain modeling language. Therefore, criteria obtained for sorting the cards had to be interpreted with care.

Another weakness is potential bias from the space of problems to be sorted. Each sort session examines one segment of the problem space rather than a broad range of requirements engineering problems, for several reasons. First, inclusion of 'noise' problems risked the semantic balance of the problem space which is a prerequisite for card sorting. Second, inclusion of additional cards would have increased session duration. Therefore, 'noise' problems were not included.

Each software engineer undertook the two sort sessions back to back. The same procedure was used for both:

1) each software engineer was requested to sort the problems into categories according to their own criteria. The instructions were "*to maximise reuse of functional requirements and specifications of solutions to problems in the same categories*". Software engineers were requested to do up to 5 different sorts per session, to keep the exercise within

and: (ii) use these criteria to categorise problems according to categories in the formal problem space.

2.2: Two segments of the problem space

Card sorts were used to investigate how experienced software engineers categorise two segments of the problem space. The first sort session elicited mental criteria and categories to validate theoretical predictions about 4 high-level object system models. Eight cards described 8 different real-world problems, one on each card. Two of these problems instantiated the resource containment system model (*stock control* and *library lending*), two instantiated the resource allocation system model (*scheduling aircraft to airport gates* and *theatre ticket allocation*), two instantiated the object sensing system model (*train collision avoidance* and *sensing for flight plan adherence by aircraft*) and two instantiated the financial exchange system model (*mortgage lending* and *cash withdrawl from an automatic teller machine*). The theoretical model predicts that these four object system models are discriminated between according to key behaviour into the types "*allocating*", "*monitoring*", "*containing*" and "*financial transaction*".

The second session was more complex and elicited mental criteria and categories to validate theoretical predictions about lower-level object system models. The 17 problems sorted in the second session were all instances of specialisations of the resource returning system model, which is a specialisation of the resource containment system model. Object returning represents the removal and return of resources from a container to a client. Specialisation is achieved through addition of different criteria which were domain structures, goal states and triggering events. First, the 17 object system models were categorised using cardinalities of domain structure. Analogical retrieval has been shown to need structure in the form of object relations, however it was not known whether the cardinalities of these relations are also powerful discriminators between problems. To investigate this, each problem had either one or many 'client' objects. Second, these object system models were specialised using five different goal states: (i) return of resources before a deadline (e.g. *book lending*); (ii) return of augmented resources before a deadline (e.g. *mortgage lending*); (iii) no damaged resources present in the container (e.g. *book repair*); (iv) no unserviced resources with clients (e.g. *vehicle servicing*), and; (v) no unaugmented resources in the container (e.g. *computer system upgrading*). Third, each object system model was categorised according to its behaviour with respect to time. A resource-removal event triggers either one or many transitions of the resource from the container to the client (e.g. *one loan event can trigger loan of many books to one borrower or loan of one car to a customer*). Furthermore, these 3 criteria were ranked according to their power of discrimination for reuse. It was this combination of criteria, ranking and categorisation which defined the hierarchical problem space. This space, and problems instantiating the 17 object system models and described on the cards are listed in Figure 1.

2.3: Experimental predictions

One card sort session was undertaken for each problem space segment. Both sessions had the following experimental predictions, which were proposed to test the theoretical model of domain knowledge:

P1: experienced software engineers would discriminate between categories of problem using criteria which can be mapped to knowledge types in the domain modeling language;

P2: experienced software engineers would use these criteria to categorise problems according to predicted object system models;

P3: experienced software engineers would rank the power of each discriminating criterion as the knowledge type is ranked in the theoretical model;

P4: elicited criteria and categories would depend on their previous experience;

P5: experienced and inexperienced software engineers would use different criteria and categories;

P6: experienced software engineers will use abstract discriminators linked to behaviour, structure and purpose while inexperienced software engineers use surface features of objects and organisations [1].

Domain analysis is one means of delivering domain knowledge reuse but it has failed to indicate what knowledge to reuse outside the boundaries of a single application area. The answer, so far, has been a pragmatic decision to analyse all applications within an organisation's field of operation. However, such analysis is difficult and time-consuming. To overcome this, some researchers have modeled generic domain knowledge (e.g. [7]) for reuse in many different applications.

The ESPRIT NATURE project [2] proposes a theoretical model of domain knowledge to determine the semantic content and boundaries of software engineering problems [10]. This enables more systematic reuse of domain knowledge in requirements engineering.

2.1: NATURE's problem abstractions

Problem abstractions are composed of object and information system models. Object system models define the fundamental behaviour, structure, goals and constraints which discriminate between instances of problem classes. Information system models represent abstractions of functional requirements which do not discriminate between problem classes. Object system models can be viewed as a reusable network composed of several interlinked objects. These models are expressed within the framework of object-oriented languages, so that abstractions can be exploited by software engineers with no knowledge of the underlying formalism.

Instances of object and information system models can be aggregated for large-scale domains, thus scaling our approach to real-world requirements engineering problems. The formal domain modeling language and a subset of the 150 problem abstractions have been implemented using the knowledge representation language Telos [6] and ConceptBase, an object-oriented deductive data base. Problem abstractions are retrieved from the data base using a computational analogical reasoning mechanism and explained to requirements engineers using predetermined strategies [10].

Object system models are defined in a hierarchical class structure [10]. The theoretical justification for this draws on assumptions from cognitive models of memory [1], and in particular hierarchical models of natural categories [9] and memory schemata [8] which assert that human memory is organised as an informal hierarchy of classes. There is good evidence that memory for several different types of knowledge (e.g. objects, procedures and plans) is organised hierarchically. Mental representations of causally connected actions, properties and participants involved in common activities also exploit hierarchical classification of knowledge structures (e.g. [8]).

Hierarchical levels of problems are discriminated by a small set of different criteria or knowledge types. These include:

- state transitions of objects with respect to structure, and states within structure, to attain a goal state;
- goal states to be achieved or avoided;
- domain or time events which trigger state transitions;
- object properties such as physical or financial.

The 9 highest-level object system models define the fundamental domain behaviour and structure in terms of state transitions to attain a goal state. They indicate the breadth of the formal problem space [10]:

- *resource containment* is characterised by the unidirectional transfer of key objects from a holding to a client structure, for example stock control problems;
- *item composition* aggregates objects to synthesise new ones, for example manufacturing problems which compose raw materials into a product;
- *item decomposition* is the opposite of composition;
- *resource allocation* models resource state changes prior to other transactions, for example booking problems;
- *object sensing* monitors object movement with respect to a spatial/physical structure, for example radar detection in air traffic control;
- *object messaging* communicates objects which have some information value between structures representing a communication topology;
- *agent object control* represents command and control problems in which agents pass messages which cause changes in object behaviour, for example ship control within a major port;
- *work item manipulation* represents incremental changes in object states, for example computer based tools for word processing or computer-aided design;
- *financial object exchange* represents the behaviour of financial and accounting rather than physical or conceptual objects, for example mortgage lending.

Specialisation of these high-level object system models is achieved through addition of further knowledge such as goals and triggering events, as shown in the two described segments of the problem space.

Therefore, according to our predictions, effective reuse should be more feasible if mental criteria for categorisation map to knowledge types in the domain modeling language and mental problem categories map to object system models. This paper investigates the two basic predictions that experienced software engineers: (i) discriminate between problems using a small set of criteria which are part of our domain modeling language,

How People Categorise Requirements for Reuse: a Natural Approach

N.A.M. Maiden, P. Mistry & A.G. Sutcliffe

Centre for Human-Computer Interface Design
City University, London, UK

Abstract

This paper reports a knowledge acquisition exercise which elicited experienced software engineer's knowledge about domains for which requirements engineering takes place. Card sorts were used to acquire software engineers' mental categorisations of these domains to inform categorisation of a set of formal, reusable problem abstractions intended to assist requirements engineers.

1: Introduction

Requirements engineering is problematic and needs intelligent support for capturing complete and consistent requirement specifications. One solution advocated by the ESPRIT NATURE project [1] is to populate tools with problem abstractions to assist requirement specification [5,7]. Problem abstractions are formal models of the fundamental behaviour, structure, goals, constraints and functions of all instances of a problem class. Leverage from reuse of this domain knowledge can be maximised if the formal abstractions are equivalent to mental abstractions recalled by software engineers with extensive requirements engineering experience. Such experience refines the categorisation, size and contents of these mental abstractions to optimise the knowledge reuse which is possible. Furthermore, reusing formal abstractions [4] is easier if these abstractions map to natural categories of problems [9] which are familiar to software engineers. This paper reports acquisition of experienced software engineers' mental categorisations to inform the categorisation and structure of the formal problem abstractions held in an intelligent requirements engineering toolkit [4].

NATURE's domain theory [10] claims that most software engineering problems belong to a tractable set of hierarchical classes which share general features. It defines the semantic content, categorisation and structure of a set of formal problem abstractions to maximise domain knowledge reuse. Problem abstractions are distinguished using a small set of knowledge types. The current set of 150 abstractions has been derived from extensive analyses of the problem abstractions underlying many software engineering domains. These analyses have used domain analysis, case studies and examples from software engineering textbooks. However the categorisation and structure of these abstractions remain to be validated through empirical investigation.

Experienced software engineers recall and reuse mental abstractions when specifying new requirements. Rosch [9] argues that peoples' mental representations are accounted for by the structure of their environment, so these mental categorisations should provide an accurate categorisation of software engineering problems. This paper focuses on acquisition of these mental categories to test their naturalness as predicted by the formal problem space, and to inform the hierarchical structure and categorisation of the formal problem abstractions.

This paper also reports how inexperienced software engineers categorise software engineering problems. Similar studies [1] revealed that experienced physicists categorised physics problems using underlying principles (e.g. laws) while novices used surface features (e.g. pulleys). Surface similarities have also provided the basis for partial analogical transfer of structured specifications by less experienced software engineers [5]. Therefore, categorisation by inexperienced software engineers was predicted to contrast to categorisation by experts and indicate how surface features can lead to misuse of domain knowledge.

This paper is in three parts. First, the theoretical model of domain knowledge and two segments of the problem space are described. These segments provide predictions about the mental categorisation of software engineering problems. Second, two card sorting exercises are reported to investigate these predictions. Third, implications for reuse of domain knowledge during requirements engineering are discussed.

2: A model of formal problem abstractions