Patterns in Software Requirements Reuse

Jacob L. Cybulski

Department of Information Systems
The University of Melbourne
Parkville, Vic 3052, Australia
Phone: +613 9344 9244, Fax: +613 9349 4596
Email: j.cybulski@dis.unimelb.edu.au
Internet: http://www.dis.unimelb.edu.au/staff/jacob/

Abstract

Requirements reuse is an emerging field of software engineering research. This article introduces its fundamental concepts. It begins with a brief review of the selected approaches to reusing software requirements. Then, the article builds a reuse framework for the requirements engineering process. Subsequently, it evaluates various methods and techniques that can be used to assist the process of requirements reuse.

In our analysis, we look at the way requirements documents are created, manipulated and used across the entire cycle of requirements engineering. We consider activities preceding formulation of software requirements, such as elaboration of needs and objectives in the software project. We include requirements acquisition, specification and modelling. Generation and evaluation of alternative interpretations of requirements is also taken into consideration. Finally we review the tasks associated with the verification and validation of requirements specifications. In the process we describe reuse of enterprise and domain information, reuse of information sources, raw and formalised requirements, reuse of document structure, text and annotations, reuse of various requirements engineering by-products, and finally, reuse of techniques and processes associated with the management of software requirements.

Introduction

There are many different ways of improving the requirements engineering process. Some approaches aim at assisting requirements acquisition (Christel and Kang 1992). Some propose the use of tools, such as CASE, to support requirements specification and modelling (Davis 1990, Rock-Evans 1989). A number of projects emphasise the issues of requirements reconciliation and viewpoint analysis (Nuseibeh, Kramer, et al. 1994), verification and validation of requirements (Boehm 1984). Still others investigate the use of artificial intelligence techniques to capture domain knowledge assisting the subsequent construction of requirements documents (Lowry and Duran 1989). In this work, we suggest yet another

method of enhancing the cycle of requirements engineering, i.e. by reusing previously analysed and refined requirements documents and their parts.

The reuse of software products and their components in the early phases of software development life-cycle (SDLC) is claimed to have a positive impact on software projects (Agresti and McGarry 1988). In particular such early reuse is said to lead to :-

- better utilization of available resources (Kang, Cohen, et al. 1992);
- improvement in the quality of requirements specifications (Johnson and Harris 1991);
- encouragement of systematic reuse across the entire software development (Frakes and Isoda 1994, Matsumoto 1989); and,
- developmental assistance early in the life-cycle (Maiden and Sutcliffe 1989).

As requirements engineering sets off the entire development process, consequently reuse of its life-cycle products will offer the most significant benefits (Poulin 1993, Yglesias 1993). At the same time, being the main form of communication between analysts and the clients, software requirements are produced in the notation and media form that are not appropriate for the subsequent computer representation and processing, not to mention their reuse. To cope with these problems, we suggest that software requirements need unique processing methods to accommodate the tasks leading to their reuse, e.g. analysis of informal requirements, their organisation into a repository of reusable artefacts, and the synthesis of the new requirements documents of reusable components.

Approaches to Requirements Reuse
parsing of formal specifications (Allen and Lee 1989)
maintenance as a reuse-based process (Basili 1990)
meta-models, and their tracing (Bubenko, Rolland, et al. 1994)
evolutionary development (Bellinzona, Fugini, et al. 1995, Fugini and Faustle 1993)
taxonomic knowledge representation (Johnson and Harris 1991)
reuse-based development process (Kang, Cohen, et al. 1992)
knowledge-based reuse system (Lowry and Duran 1989)
wide-spectrum reusability (Lubars 1988)
schema selection and instantiation (Lubars 1991)
analogical reasoning (Maiden and Sutcliffe 1991)
natural languages processing (Naka 1987)
integrated CASE environments (Poulin 1993)
logic-based specification formalisms (Puncello, Torrigiani, et al. 1988)
tree of requirements abstractions (Wirsing, Hennicker, et al. 1989)
knowledge-based refinement (Zeroual 1991)

Table 1: Summary of Previous Approaches to Requirements Reuse

Previous Work in Requirements Reuse

There are a number of software engineering environments, techniques, methods, and methodologies that are capable of supporting reuse of software requirements specifications in software development projects (cf. Table 1).

The most promising approaches to software reuse in general focus on the wide-spectrum reuse across the entire development process (Lubars 1988), ranging from the project conception, writing requirements and specifications, through the stages of software design, down to coding and testing.

Reuse of early artefacts, such as domain knowledge, requirements and abstract designs offers a considerable gain due to leveraging reuse very far upstream in the software process (Arango 1994, Moore 1991, Prieto-Diaz 1988, Schafer, Prieto-Diaz, et al. 1994, Sutcliffe and Maiden 1994). The reuse-based software development methodology, in which reuse is at the crux of the development life-cycle, is a natural extension of wide-spectrum reuse efforts (Kang, Cohen, et al. 1992). Such reuse-based methodologies focus on identifying reusable resources in an application domain, rather than, as in other approaches, the applications in the domain. They start with setting a reuse strategy, identify domain experts, and then iteratively construct a structure of reusable artefacts, down to design and code level. The resulting collection of reusable object transformations of specification and design components facilitates easier controlling of software evolution and change due to maintenance, making it also a reuse-based process (Basili 1990). The systematic reuse also promotes development of software development tools and environments that promote effective storage of requirements in the reuse library and their subsequent reuse in the new software projects (Poulin 1993).

Requirements specifications commonly encode knowledge of the problem domain and development processes. Due to the complexity of this knowledge, simple reuse techniques, such as those based merely on information classification and retrieval, are ineffective. Instead, a number of researchers suggested employment of knowledge representation (Lowry and Duran 1989) and analogical reasoning (Maiden and Sutcliffe 1991) to requirements reuse. Such approaches have been tested in several working reuse systems. One of such systems is Aries (Johnson and Harris 1991), where the reuse of software requirements is achieved by the means of capturing requirements into a taxonomic knowledge structure which subsequently allows specialising and instantiating representational abstractions into new software requirements. Another knowledge-based reuse system is KBRAS (Zeroual 1991), which automates the acquisition and refinement of requirements through knowledge-based operations. Although software reuse is not the main focus of the ASPIS project (Puncello, Torrigiani, et al. 1988), intelligent support of reuse has also been put on equal footing with the support for analysis, design, and prototyping tasks. Knowledge-based requirements representation techniques are frequently matched with intelligent identification of and search for software requirements - this usually implies application of compiler-writing techniques to parse formal specifications (Allen and Lee 1989) or the use of natural language processing (Naka 1987) to acquire requirements from informal requirements texts.

Other approaches to requirements reuse advocate the use of meta-models (Bubenko, Rolland, et al. 1994), evolutionary development (Bellinzona, Fugini, et al. 1995, Fugini and Faustle 1993), schema selection and instantiation (Lubars 1991), or the use of formal methods (Wirsing, Hennicker, et al. 1989).

It can be seen that the majority of approaches to requirements reuse focus on the semantics of requirements documents, hence, they promote the utilisation of knowledge-based and linguistic techniques to acquire and represent knowledge contained in the requirements. Some approaches favour controlling the process of developing and maintaining requirements. Others look at the use of tools to support such processes and methodologies. Few of the proposals, however, consider the structural properties of requirements documents and the requirements engineering tasks that are performed in the document processing - this is the approach taken in this paper.

Requirements Engineering Phases vs. Reuse

In our analysis, we look at the way requirements documents are created, manipulated and used across the entire cycle of requirements engineering (cf. Table 2). We consider activities preceding formulation of software requirements, such as elaboration of needs and objectives in the software project. We include requirements acquisition, specification and modelling. Generation and evaluation of alternative interpretations of requirements is also taken into consideration. Finally we review the tasks associated with the verification and validation of requirements specifications. In the process we describe reuse of enterprise and domain information, reuse of information sources, raw and formalised requirements, reuse of document structure, text and annotations, reuse of various requirements engineering by-products, and finally, reuse of techniques and processes associated with the management of software requirements. Some of the proposed methods have already been implemented and tested within an experimental requirements management system - RARE IDIOM (Reuse-Assisted Requirements Engineering with Informal Document Interpreter, Organiser and Manager) (Cybulski 1998).

Our findings are collected in the form of a pattern language for requirements reuse. In this article, the patterns are described very informally with a particularly strong emphasis on the description of the problem solution rather than the problem itself (as is the case with design patterns).

Requirements engineering stage vs. **Reusable component**
Elaboration of Needs and Objectives
Enterprise Model
Requirements Acquisition
Information Sources
Elicitation Skills and Resources
Document Templates and Standards
Reuse Across Requirements Documents
Reuse Within Requirements Documents
Raw Source Information
Domain Concepts
Text Phrases
Requirements Specification and Modelling
Analytic Techniques
Analytic Processes
Analysis By-Products
Refinement Artefacts
Adaptation and Integration Processes
Comments and Annotations
Requirements Model
Analysis and Specification Patterns
Generation and Evaluation of Alternatives
Restricting Problem Domain
Restricting Solution Domain
Evaluation and Selection
Requirements Verification and Validation
Verification Rules
Validation Trace

Table 2: Reuse across various stages of requirements processing

Reuse of Enterprise Model

It is not uncommon for a single development group to be involved in a number of projects for the same organisation. All such projects address the same community of users, who share similar problems and who could benefit from the solutions to these problems. It may, hence, be beneficial for developers to construct a systematically organised and reusable *enterprise model* that will collect and organise business knowledge about the client organisation, its structure, management style, controls and communication. Subsequently, developers may consistently and completely specify different aspects of enterprise systems by utilising common elements of the enterprise model, e.g.

- developers could analyse the structures of authority and business affinities to define the system architecture that will provide distinct services for different business units;

- they could study the composition and flow of information in the organisation to model the database structure and to manage access to corporate data contained in it;

- they must gain understanding of various business rules to determine the validity of the selected information processing methods;

- they will have to model existing business processes to determine the conformance of the proposed software solution with the rest of the business operation;

- they may have to respect business audit procedures and to design system controls to mirror current business practices.

Once created, elements of the enterprise model can be repetitively used and reused to solve a range of problem within the context of the same organisation. At the same time, however, due to the high level of organisation-specific information in the model, reuse of enterprise model across different organisations may not be beneficial or even appropriate at all.

Reuse of Information Sources

Identification of useful information sources in the enterprise requires considerable knowledge of an organisation and its respective application domain. It also necessitates awareness of the group dynamics, experience in social interaction, political and technical skills. Project information sources include key personnel in the organisation, the procedures regulating company operation, technical, management and financial reports, minutes of meetings, professional literature, databases of client data, repositories of electronic documents, newsletters, or logs of electronic discussions. Amassing knowledge about reliable sources of information in a given organisation can, hence, be a considerable investment of time and money. Such information can also have significant value to other business units in the development organisation or other corporations. This preliminary stage in the acquisition of requirements may, hence, decide on the success or the failure of the development project. However, having established a repository of knowledge about corporate information sources (cf. Figure 1), such information can potentially be reused (R) in many software development projects for this particular client organisation. Furthermore, reuse of the same information elicited (E) across projects may also assist in the identification of requirements commonalties and similarities (S) that could potentially lead to the reuse of specification and design components downstream (F) the system development. (Note that broken lines in the diagram indicate activities avoided due to the reuse of components.)

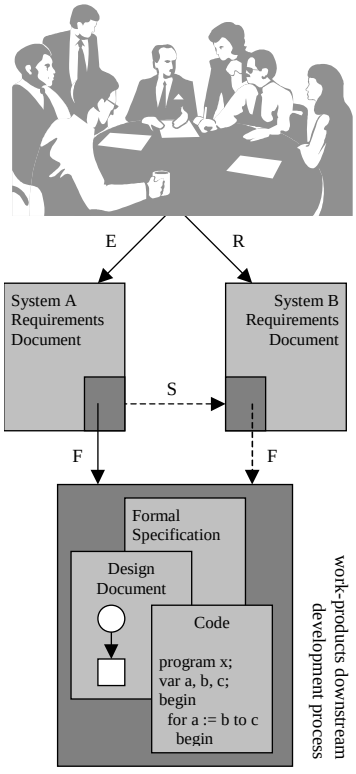

Figure 1: Reuse of information sources

Reuse of Elicitation Skills and Resources

There are great many elicitation techniques useful in acquiring information from various sources. Such techniques may demand expertise in :-

- the use of software tools and prototype building (e.g. simulation and rapid prototyping);
- inter-personal communication (e.g. interviews, tutoring and critiquing);
- group management (e.g. discussion, brainstorming and JAD);
- observation and analysis (e.g. protocol, document and data analysis); or,
- social investigation and testing (e.g. polling and surveying).

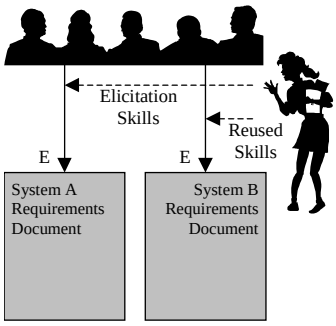

Figure 2: Reuse of elicitation skills

Each of the approaches to requirements acquisition demands specialised knowledge and experience. Few requirements engineers, however, would be equally skilled in all of the techniques that may be useful in a given problem area. In large enterprises, such as government, public service, defence, banking, insurance or telecommunication corporations, effective utilisation of reusable human resources, of which requirements elicitation skills is only an example, is an important management consideration. This means that reuse of elicitation skills (cf. Figure 2) could utilise standard management techniques, such as pooling of development staff, construction of skill databases or planning human resource utilisation. Such reuse is already common and necessary in any sizeable organisation.

Reuse of Standards and Document Templates

Requirements documents are often written in compliance with requirements engineering or quality standards (Dorfman and Thayer 1990). Such standards provide details of the requirements engineering process, techniques to be used and the structure and contents of documents produced. The guidelines prescribed by the standard are usually voluminous and complicated.

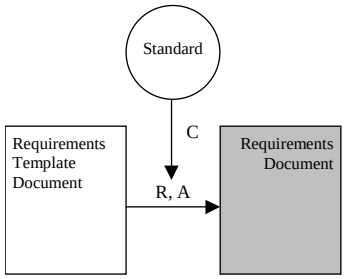

Figure 3: Reuse of templates and standards

To better facilitate the use of standards, most of the methodology providers distribute on-line documentation, interactive help facilities and examples. Some providers may issue the users of such standards (cf. Figure 3) with templates that allow reusing (R) document structure, look and contents, and which also facilitate adaptation of requirements templates (A) by filling in the missing details (Arnold and Stepoway 1988, Frakes and Nejmeh 1988, Volpano and Kieburtz 1989). Others may supply verification rules and compliance checklists (C), or even CASE tools that enforce the methodology and hence result in adherence to the standard adopted (e.g. Rational Rose, Oracle, Cadre or System Architect). Once, automated, the standard and its various associated documents, such as requirements templates, can be reused in many development projects more effectively.

Reuse Across Requirements Documents

Comparable applications and system utilities, programs in the same product line, or packages in the same application domain usually share similar requirements (cf. Figure 4). Similarity in requirements (S), hence, indicates the possibility of their reuse (R), as well as, the

possibility of sharing requirements analysis work-products, i.e. requirements interpretation, classification and refinement decisions, and the consequent development by-products (F), e.g. specifications, designs and code.

In practice, however, detection of requirements similarity may be extremely difficult, as similar requirements can be expressed using distinct writing styles, different grammatical structures and inconsistent terminology, varying semantics and pragmatics. Active tool support of requirements reuse is, therefore, needed to assist developers in finding and understanding similar requirements that have already been processed, and reapplying these requirements to the new project, while at the same time, fetching and reusing all artefacts that were previously derived from the reused requirements.

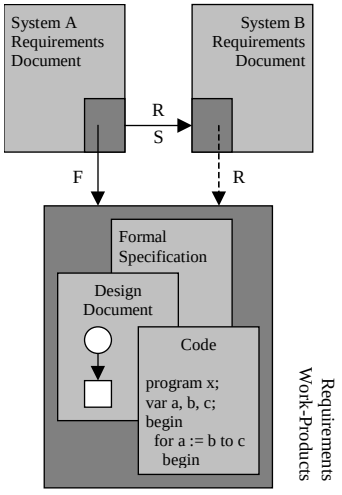

Figure 4: Reuse across requirements documents

Reuse of Requirements References

In large requirements documents written by many people over a long period of time, a number of similar, and potentially reusable, requirements may be entered into different document sections. In practice, however, the very existence of overlapping or "reusable" requirements in a single specification document is an indication of its weakness due to redundancy or bad document organisation. In a single document, different requirements should describe different software functions or quality attributes. What is possible and acceptable (cf. Figure 5), however, is that different parts of a single requirements document rely on and refer to the same product function (F). In such a case, requirements references could be identical or matching (S) - in a sense they may all indicate the existence of a potentially reusable requirement (R).

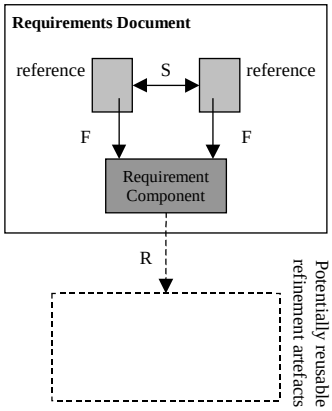

Figure 5: Reuse of references

Unfortunately, as it was the case with requirements reuse across documents, it is an equally difficult task to determine that two parts of the same specification refer to the same requirement.

Reuse of Source Information

In the process of requirements elicitation, analysts collect large volumes of raw source information, such as interview transcripts, surveys, experimental data, simulation and observation notes. All such information eventually proves to be relevant or irrelevant to the project at hand. Should the previously collected information not be turned into requirements, it may still be of some use to as yet unspecified system to be developed for the same group of users.

As the process of collecting raw source information is very laborious and expensive, so its reuse value should also be considered as very high (cf. Figure 6). Sharing elicited information (E) between several projects may indicate the possibility of requirements similarity (S), their potential reuse (R) and the reuse of their derived work-products (F). Reusing raw requirements information, due to the common information sources, can also assists the

process of requirements traceability, identification of enterprise affinity, semantic association or cross system dependence.

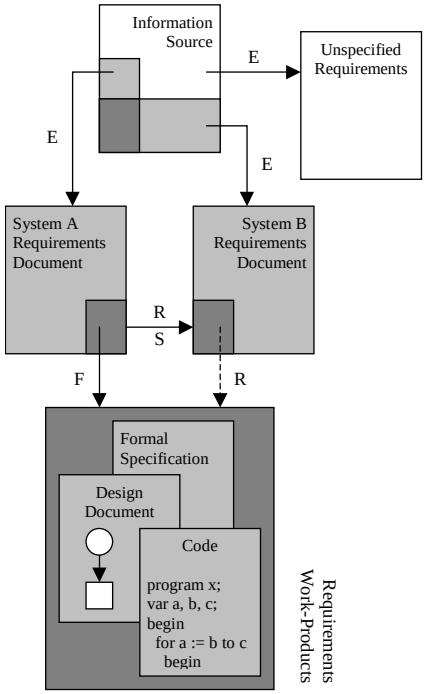

Figure 6: Reuse of raw source information

Reuse of Domain Concepts

Software requirements written for the same problem domain will repetitively utilise similar terminology, concepts, structures, relationships and process descriptions. Creating a model of a given application domain has been proven to provide an effective basis for the systematic reuse of systems and their components (Arango and Prieto-Diaz 1991, Moore 1991, Prieto-Diaz 1988, Simos 1995, Sutcliffe and Maiden 1994). For this reason, some advocates of software reuse consider domain engineering as an essential and inseparable part of software reuse.

With time, a domain model (cf. Figure 7) is gradually refined (F) in terms of specifications, designs and implementation of its elements. Use of domain components in the construction of requirements documents leads to the systematic reuse (R) of all work-products descended from these components. Requirements that use the same domain concepts are usually also similar (S) or related.

Reuse of Text Phrases

During the construction of any large document, to include a requirements specification, it is necessary to ceaselessly repeat terms commonly used in a given organisation or in a specific product line. Such terms include acronyms, names of well-known corporations and people, project and module identifiers, references to software components, or database data. Some of the popular word processing and text editing software already provide the facilities to "reuse" text, e.g. glossaries of terms, predefined and user-defined fields and macros, auto-text or text auto-completion. As reuse of text phrases can be regarded as purely editorial convenience, the facility should, hence, be regarded as independent of a domain or enterprise model.

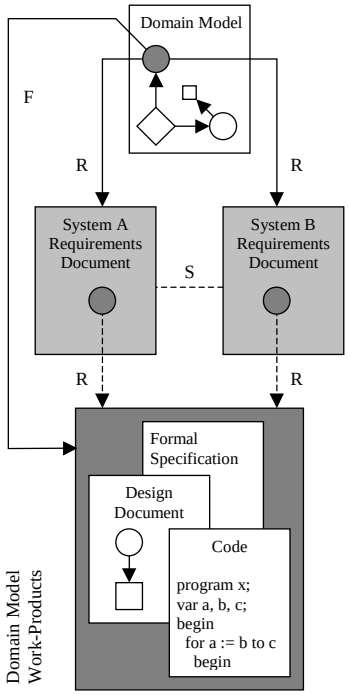

Figure 7: Reuse of domain concepts

Reuse of Analytic Techniques

There are numerous problem-solving techniques that may lead to the analysis of requirements documents, their structure and contents. For example,

- by identifying section headers and studying document outline, we can detect hierarchical relationships

between requirements;

- by locating nouns and verbs in the requirements text, we can identify important entities and actions performed on them;
- by scanning requirements for some well-known keywords, we can spot uses of the familiar domain and enterprise concepts;
- we can also enrich requirements representation by finding frequent terms, eliminating noise words, word stemming, determining semantic concept similarities, or looking for speech markers[1] in interview transcripts.

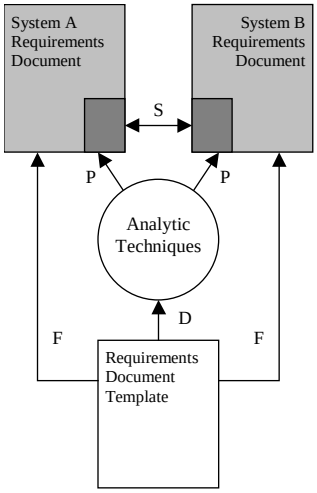

Figure 8: Reuse of analytic techniques

Document analysis techniques are specific to the type of requirements document, syntax and semantics of the requirements statements, and the position of the requirement text in relationship to the rest of its embedding document. Such techniques may be implied by the requirements template or they may be discovered in the midst of the process of requirements analysis. Requirements analysis techniques should (cf. Figure 8), hence, be planned (D) to be reusable so that they could be applied (P) to many different requirements of similar form and contents (S, F).

Reuse of Analytic Processes

Analysis of requirements documents requires a skilled requirements engineer, who chooses the best methods of representation for the requirements, decides on requirements classification, selects matching refinement artefacts, and who conducts necessary transformations of the requirements text.

Having analysed the contents of some requirements (cf. Figure 9), it should be possible to track, record and store (T) all of the actions performed during analysis (A). Such sequences of analytic events can subsequently be parameterised and generalised to make them reusable in the processing of similar (S) requirements texts (akin transformation systems - Cheatham 1989, Feather 1989, Johnson and Feather 1991). Their reuse can then be achieved by instantiating process parameters and replaying the recorded events (P). Similar approach can also be used to track and reanalyse changing requirements (C).

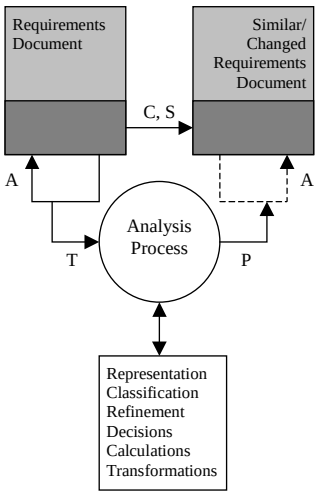

Figure 9: Reuse of analytic processes

Reuse of Analysis By-Products

During requirements analysis, requirements texts are enriched with interpretation information (I) to facilitate requirements classification (C), storage (S), search and retrieval (R), refinement (F), and traceability (T), all of which enable more efficient manipulation of

[1] For example, "firstly", "unfortunately", "summarising", etc.

requirements in the subsequent processing (cf. Figure 10). Reusing requirements from one document to another should transfer not only the text of these requirements but also all other by-products obtained from the previously conducted analysis.

A distinction should be drawn here between reuse of analytic techniques, processes and the reuse of analysis by-products. Reuse of requirements text results in the same requirement to appear in two different contexts. Reuse of analytic techniques and processes over these two instances of the same requirement, may or may not produce similar interpretation results - this will depend on the context and on the human element in the process. However, whenever the analyst is certain of the similarity of two contexts and the identity of the reused requirements, he may opt to reuse not the analytic processes that lead to certain interpretations but rather the result of these processes, i.e. the interpretation by-products themselves. This approach will reduce the analysis effort as no human interference would be required to analyse the requirements text again.

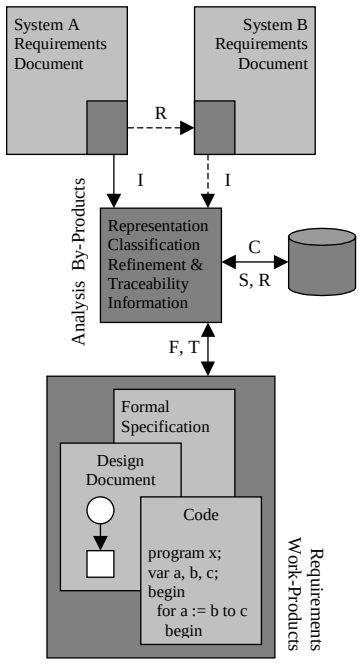

Figure 10: Reuse of analysis by-products

Reuse of Refinement Artefacts

In the process of analysing, formalising and refining software requirements we frequently face a difficult task of finding, understanding and selecting existing specifications and designs that could be used in the construction of new software products. Should the text of requirements be well described and classified, and specifications and designs be reusable (cf. Figure 11), it would then be possible to use requirements classification terms to issue a query (Q) against the repository of reusable refinement artefacts. The returned candidates would then be inspected, selected, adapted (A) and integrated (I) into a new software product matching (S) the original requirement.

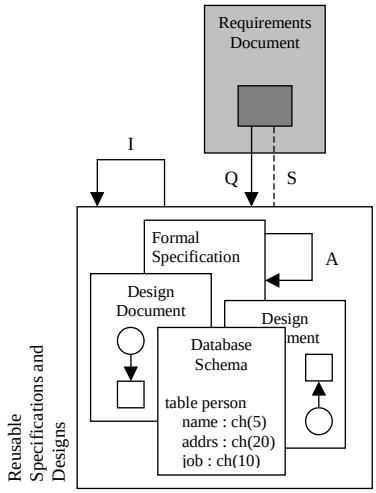

Figure 11: Reuse of refinement artefacts

Reuse of Adaptation and Integration Processes

In the process of requirements analysis (cf. Figure 12), requirement texts are mapped (F) into a collection of refinement artefacts. Such artefacts are then adapted and integrated (A, I) to form a new software component. Should a similar requirement (S) be found in the future, it is likely that a similar set of adaptation and integration steps would be needed in its processing. It is hence apparent that events leading to the refinement of the requirement text should be tracked and recorded (T) and subsequently reused by modifying (M) and replaying (P) the event trace over similar requirements texts.

Reuse of Comments and Annotations

Requirements engineers frequently annotate requirements text with additional information that is not part of the main deliverable documentation (cf. Figure 13), e.g. justifications, details of calculations and reasoning, references to user remarks or some technical documents. Such annotations could be very useful in assisting clients during requirements validation or helping designers in better understanding the outcomes of the requirements analysis process. Therefore, reuse of requirements text (R) should be accompanied by the reuse of associated comments and annotations (A). As annotations could be complex and expensive to produce (they may include text, graphics, sound and video) and they may be applicable to the whole collection of similar or related requirements (S), hence, they too should be treated as reusable artefacts in their own right.

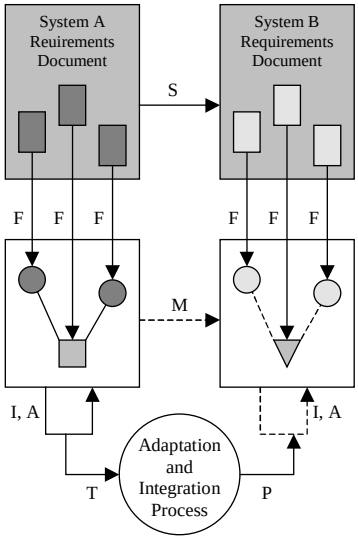

Figure 12: Reuse of adaptation and integration processes

Reuse of Requirements Model

Most often, requirements engineers prefer to work with formal specification documents rather than the informal requirements. In such cases, transfer of reusable information is likely to occur at the level of a formal requirements model (cf. Figure 14) instead of the requirements text from which the model was derived (F). The model of an application may be reused (M), adopted or adapted (A) in its entirety to produce a new software system. As the validation of a formal model may prove difficult for the clients, most of the modern CASE tools support automatic generation of textual requirements (G) in a simple and natural form, compatible with some requirements standard.

It should be noted, however, that reuse of requirements models is quite different from the previously described method of reusing refinement artefacts. While requirements model can be reused with no reference to the informal requirement statement, reuse of refinements was discussed in the context of a process assisting analysts in the task of analysing and formalising the informal requirements with the aid of existing specifications and designs.

Reuse of Analysis and Specification Patterns

Requirements models, whether derived from informal requirements or not, always incorporate great many problem and solution components that are arranged into recognisable structured forms - analysis and specification patterns (cf. Figure 14) (also see Fowler 1997, Gamma, Helm, et al. 1995). Such patterns define entities and their relationships as commonly found in the application domain, they link and structure domain concepts, they aggregate entities, form higher abstractions, and they specify interfaces between well-known classes of components. Patterns can be used as a dictionary of

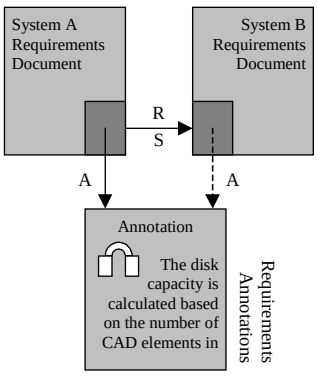

Figure 13: Reuse of comments and annotations

problem-solving approaches, they act as a guide to specification methods, or as the means of enforcing good specification and design principles. In all of the above-mentioned situations, patterns are treated as reusable resource for the effective construction of requirements models.

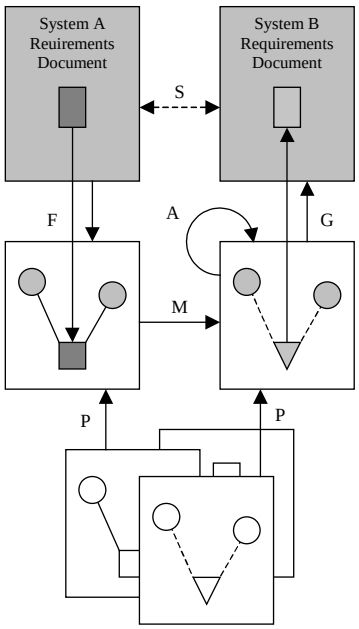

Figure 14: Reuse of requirements model and modelling patterns

Generation and Evaluation of Alternatives

Having collected and analysed software requirements, we may have to list and evaluate many possible interpretations of these requirements. Alternatives in requirements interpretation may find their source in differing views by the project stakeholders, incompleteness of requirements, ambiguous statement of requirement, or the lack of design criteria. All such alternate interpretation need to be inspected and assessed for their suitability for use in the problem solution, requirements may have to be corrected and elaborated, invalid alternatives rejected, and multiple refinements evaluated and reduced to a single problem solution. Rejected alternatives are worth keeping and maintaining for the entire duration of the project, they may have to be reviewed at a later stage in the project, and in some cases, they may even resurface as the only valid choice amongst many alternatives. Although, the criteria and methods of selecting requirements interpretations and refinements are themselves not reusable (there are few such methods only), such methods are central to the reuse process itself.

Buschmann et al. (Buschmann, Meunier, et al. 1996, pp 368ff) provides some insight as to the method of selecting architectural components to match the problem being solved. We adopted this method and adapted it to the selection of refinement artefacts matching given requirements. Hence, we propose the following steps to be taken during artefact selection while analysing and refining requirements:

1. Classify requirements and reusable specification artefacts in precise terms so that classification terms could be used in comparing the attributes of requirements against those of reusable artefacts;

2. Narrow the search for candidate artefacts by selecting the requirements category in terms of a problem domain, requirement type and its semantics, e.g. use of viewpoints (Nuseibeh, Kramer, et al. 1994);

3. Further reduce the scope of artefact search by selecting artefact category in terms of the problem solution, i.e. addressing the system function, data or development activity, e.g. use of architectural patterns (Shaw 1996, p 257);

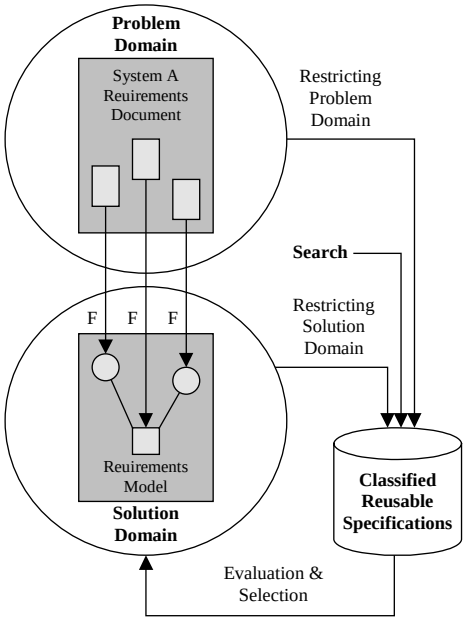

Figure 15: Selection of refinement artefacts

4. Compare problem described by the requirements text against a structured collection of reusable artefacts, aspect by aspect;

5. Compare benefits and liabilities of selecting each of the candidate artefacts to refine a given requirements statement;

6. Select the variant that best matches a given requirement;

7. Continue selecting alternative requirement categories by generalising the requirements until a suitable artefact is found or none are left.

The main aspects of the selection process are, hence, classification of requirements and refinement artefacts, narrowing of the search space by restricting the problem and solution categories, matching of requirements and artefacts and evaluating the validity of the choice. Making the actual selection of one of many possibilities is a problem related to the human cognitive ability to choose the most appropriate problem solution (Slovic 1990). There are several theories of rational choice, e.g. by listing, ranking and comparing of available options, by finding an optimal solution, by identifying the first solution to satisfy our search criteria, by eliminating options that do not fit our criteria, or by comparing artefacts in a common criteria dimension.

Reuse of Verification Rules

Once requirements documents have been analysed, refined and the formal models constructed (F), the documents need to be verified (V) against their prescribed semantics (cf. Figure 16). Requirements text may have to be checked for its lexical and syntactical correctness. Informal requirements texts may have to be checked for their compliance with the requirements standard. Completeness rules may have to be invoked to determine whether or not all the necessary concerns have been addressed. Formal models may have to be traced back to raw source information or the informal requirements documents. Semantics of embedded objects, such as decision tables, pseudo-code or diagrams, may have to be reviewed.

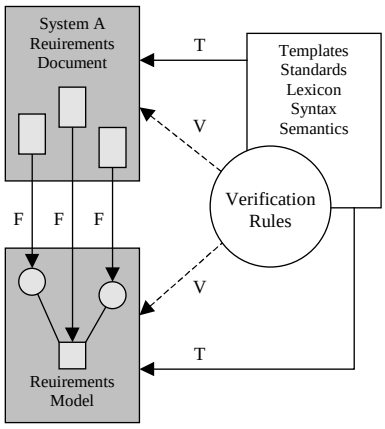

Figure 16: Reuse of verification rules

All such verification checks are determined by the type of the requirements document, its use in the requirements engineering process and the analyst's past experience in the practical management of such documents and processes. Verification rules do not come in huge volumes, nevertheless, they may have to be collected and managed, they need to be allocated to document types or their parts, and be described in a great detail to be used effectively. They may form a repository of reusable rules linked with document templates (T), and be reused together with other components of a template, i.e. its text, adaptation rules and the standards the template complies with.

Reuse of Validation Trace

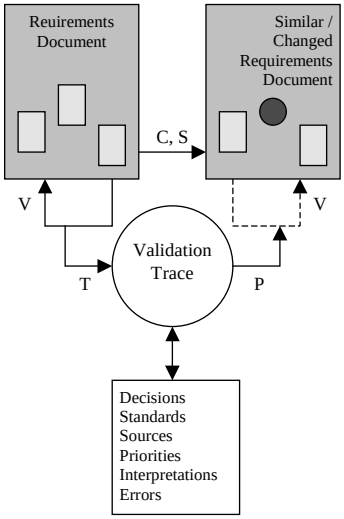

Figure 17: Reuse of validation trace

Requirements documents that are considered to be correct and complete have to be validated with the clients to confirm their true needs, wishes and preferences. Such validation may involve identification and clarification of misinterpreted requirements, re-negotiation of initial requirements, referencing of the raw material used as the basis for the construction of requirements documents, re-modelling of certain aspects of the system, duplication of requirements representation, classification and refinement.

In the future development efforts, the recorded process requirements validation could be used by requirements engineers as an exemplar for the practical validation of requirements expressed by a specific class of users. A collection of such exemplars may result in itemised checklist of validation decisions. It may provide a demonstration the client's priorities and preferences. It will guide the requirements engineer in determining the standards used by the project stakeholders. It may also draw maintenance engineer's attention to the long-forgotten origins of certain requirements. It will point out sources of developer misinterpretations, mistakes and omissions. A trace of validation sessions (cf. Figure 17 - V), at an appropriate level of abstraction, should be reviewed in respect to similar projects (S) sponsored and owned by a given client community. Such a trace should be recorded and stored (T) and replayed (P) to avoid client's dissatisfaction, to make validation sessions more effective and to increase the quality of produced requirement documents.

Similarly to the reuse of analytic processes, requirements validation trace can be applied to revalidate changing requirements (C).

Summary

This article presented an approach to reusing software requirements that focuses not on the syntax of the requirements specification formalism, nor on the semantics of reusable artefacts, neither it looks at the knowledge contained in the requirements. Instead it describes a method of requirements reuse based on the structural properties of the requirements documents themselves and the processes used in their production. The method aims to be independent of the syntax and the semantics of requirements texts, hence, it is should be equally applicable to both formal and informal statements of need. The method offers several different ways of reusing requirements texts and their associated artefacts, all described in terms of patterns in requirements reuse. The patterns also provide some guidelines to practical approaches of automating requirements reuse.

Table 3 summarises the proposed approaches to requirements reuse. Its rows list all aspects of requirements reuse considered in this paper, and described as reuse patterns. The columns identify the type of reused information. Such information can be seen to be classified in terms of the problem context, the methods of requirements processing, the observed processes, document structures used to organise requirements, requirements contents, and finally inter-document references.

Requirements Reuse Pattern \ Requirements Aspect	Context	Method	Process	Structure	Contents	Reference
Elaboration of Needs and Objectives						
Enterprise Model	X	X	X	X	X	X
Requirements Acquisition						
Information Sources	X	X	X			
Elicitation Skills and Resources		X	X	X		
Document Templates and Standards	X	X	X	X		
Reuse Across Requirements Documents				X	X	X
Reuse of Requirements References					X	X
Raw Source Information	X				X	X
Domain Concepts	X	X	X	X	X	X
Text Phrases					X	X
Requirements Specification and Modelling						
Analytic Techniques	X	X				
Analytic Processes			X		X	
Analysis By-Products		X	X	X		
Refinement Artefacts					X	X
Adaptation and Integration Processes			X		X	
Comments and Annotations	X					X
Requirements Model				X	X	
Analysis and Specification Patterns	X	X		X		
Generation and Evaluation of Alternatives						
Restricting Problem Domain	X	X		X	X	
Restricting Solution Domain			X	X	X	X
Evaluation and Selection	X	X	X	X	X	X
Requirements Verification and Validation						
Verification Rules		X		X	X	X
Validation Trace	X		X	X	X	X

Table 3: Summary of Requirements Reuse Patterns

The table provides a good indication as to the suitability of requirements reuse patterns to handling different types of development information, e.g. we can find out,

- which of the approaches give a coverage of all aspects of requirements reuse (enterprise and domain modelling, evaluation and selection of alternatives);

- which of the reuse patterns complement each other, and hence, which of them could be combined in different phases of requirements development, e.g.

- during acquisition stage, reuse of information sources, elicitation skills and standards could be combined with the reuse of requirements information and references in different form;

- during specification and modelling, reuse of design patterns and models could be combined with the reuse of analysis and refinement processes and methods;

- during verification and validation, reuse of verification rules must be complemented with the use of validation traces to give good quality of the final product.

The requirements reuse patterns, their process coverage, and their reuse power have been considered as important design elements in building of the RARE IDIOM system - an experimental requirements management system (Cybulski 1998).

Bibliography

Agresti, W.W. and F.E. McGarry (1988): "The Minnowbrook Workshop on Software Reuse: A summary report," in *Software Reuse: Emerging Technology*, W. Tracz, Editor. Computer Society Press: Washington, D.C. p. 33-40.

Allen, B.P. and S.D. Lee (1989): "A knowledge-based environment for the development of software parts composition systems." in *11th International Conference on Software Engineering*. Pittsburgh, Pennsylvania: IEEE Computer Society Press, p. 104-112.

Arango, G. (1994): "Domain analysis methods," in *Software Reusability*, W. Schafer, R. Prieto-Diaz, and M. Matsumoto, Editors. Ellis Horwood: London, Great Britain. p. 17-49.

Arango, G. and R. Prieto-Diaz (1991): "Part1: Introduction and Overview, Domain Analysis and Research Directions," in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 9-32.

Arnold, S.P. and S.L. Stepoway (1988): "The reuse system: Cataloging and retrieval of reusable software," in *Software Reuse: Emerging Technology*, W. Tracz, Editor. Computer Society Press: Washington, D.C. p. 138-141.

Basili, V.R. (1990): "Viewing maintenance as reuse-oriented software development." *IEEE Software*: p. 19-25.

Bellinzona, R., M.G. Fugini, and B. Pernici (1995): "Reusing specifications in OO applications." *IEEE Software. **12***(2): p. 65-75.

Boehm, B.W. (1984): "Verifying and validating software requirements and design specifications." *IEEE Software*: p. 75-88.

Bubenko, J., C. Rolland, P. Loucopoulos, and V. DeAntonellis (1994): "Facilitating "Fuzzy to Formal" requirements modelling." in *The First International Conference on Requirements Engineering*. Colorado Springs, Colorado: IEEE Computer Society Press, p. 154-157.

Buschmann, F., R. Meunier, P. Sommerlad, and M. Stal (1996): *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester: JohnWiley and Sons.

Cheatham, T.E., Jr. (1989): "Reusability through program transformations," in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. ACM Addison Wesley Publishing Company: New York, New York. p. 321-336.

Christel, M.G. and K.C. Kang (1992): *Issues in Requirements Elicitation*, CMU/SEI-92-TR-12, Software Engineering Institute, Carnegie Mellon University.

Cybulski, J. (1998): "Patterns in Software Requirements Reuse." in *AWRE'98*. Deakin University, Geelong, p. Submitted.

Davis, A.M. (1990): *Software Requirements: Analysis and Specification*. Englewood Cliffs, New Jersey: Prentice Hall.

Dorfman, M. and R.H. Thayer, eds. (1990): *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. . IEEE Computer Society Press: Los Alamitos, California.

Feather, M.S. (1989): "Reuse in the context of a transformation-based methodology," in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. ACM Addison Wesley Publishing Company: New York, New York. p. 337-359.

Fowler, M. (1997): *Analysis Patterns: Reusable Object Models*. Menlo Park, California: Addison-Wesley.

Frakes, W. and S. Isoda (1994): "Sucess factors of systematic reuse." *IEEE Software. **11***(5): p. 15-19.

Frakes, W.B. and B.A. Nejmeh (1988): "An information system for software reuse," in *Tutorial on Software Reuse: Emerging Technology*, W. Tracz, Editor. IEEE Computer Society Press: Washington, D.C. p. 142-151.

Fugini, M.G. and S. Faustle (1993): "Retrieval of reusable components in a development information system," in *Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability*, P.-D. Ruben and B.F. William, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 89-98.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*. Readings, Massachusetts: Addison-Wesley.

Johnson, W.L. and M.S. Feather (1991): "Using evolution transformation to construct specifications," in *Automatic Software Design*, M.R. Lowry and R.D. McCartney, Editors. AAAI Press / The MIT Press: Menlo Park, California. p. 65-91.

Johnson, W.L. and D.R. Harris (1991): "Sharing and reuse of requirements knowledge." in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 57-66.

Kang, K.C., S. Cohen, R. Holibaugh, J. Perry, and A.S. Peterson (1992): *A Reuse-Based Software Development Methodology*, Technical Report CMU/SEI-92-SR-4, Software Engineering Institute.

Lowry, M. and R. Duran (1989): "Knowledge-based software engineering," in *The Handbook of Artificial Intelligence*, A. Barr, P.R. Cohen, and E.A. Feigenbaum, Editors. Addison-Wesley Publishing Company, Inc.: Readings, Massachusetts. p. 241-322.

Lubars, M.D. (1988): "Wide-spectrum support for software reusability," in *Software Reuse: Emerging Technology*, W. Tracz, Editor. Computer Society Press: Washington, D.C. p. 275-281.

Lubars, M.D. (1991): "The ROSE-2 strategies for supporting high-level software design reuse," in *Automatic Software Design*, M.R. Lowry and R.D. McCartney, Editors. AAAI Press / The MIT Press: Menlo Park, California. p. 93-118.

Maiden, N. and A. Sutcliffe (1989): "The abuse or re-use: why cognitive aspects of software re-usability are important," in *Software Re-use, Ultecht 1989*, L. Dusink and P. Hall, Editors. Springer-Verlag: London, U.K. p. 109-113.

Maiden, N. and A. Sutcliffe (1991): "Analogical matching for specification reuse." in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 108-116.

Matsumoto, Y. (1989): "Some experiences in promoting reusable software: presentation in higher abstract levels," in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. ACM Addison Wesley Publishing Company: New York, New York. p. 157-185.

Moore, J.M. (1991): "Domain analysis: framework for reuse," in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 179-203.

Naka, T. (1987): "Pseudo Japanese specification tool." *Faset.* ***1***: p. 29-32.

Nuseibeh, B., J. Kramer, and A. Finkelstein (1994): "A framework for expressing the relationships between multiple views in requirements specification." *Trans. on Software Engineering.* ***20***(10): p. 760-773.

Poulin, J. (1993): "Integrated support for software reuse in computer-aided software engineering (CASE)." *ACM SIGSOFT Software Engineering Notes.* ***18***(4): p. 75-82.

Prieto-Diaz, R. (1988): "Domain analysis for reusability," in *Software Reuse: Emerging Technology*, W. Tracz, Editor. IEEE Computer Society Press. p. 347-353.

Puncello, P.P., P. Torrigiani, F. Pietri, R. Burlon, B. Cardile, and M. Conti (1988): "ASPIS: a knowledge-based CASE environment." *IEEE Software*: p. 58-65.

Rock-Evans, R. (1989): *CASE Analyst Workbenches: A Detailed Product Evaluation*. London, England: Ovum Ltd.

Schafer, W., R. Prieto-Diaz, and M. Matsumoto (1994): *Software Reusability*. London, Great Britain: Ellis Horwood.

Shaw, M. (1996): "Some patterns for software architecture," in *Pattern Languages of Program Design*, J.M. Vlissides, J.O. Coplien, and N.L. Kerth, Editors. Addison-Wesley: Readings, Massachusetts. p. 255-269.

Simos, M.A. (1995): "Organisation domain modeling (ODM): formalising the core domain modeling life cycle." *ACM Software Engineering Notes. Proc. Symposium of Software Reusability, SSR'95*: p. 196-205.

Slovic, P. (1990): "Choice," in *An Invitation to Cognitive Science: Thinking*, D.N. Osherson and E.E. Smith, Editors. The MIT Press: Cambridge, Massachusetts. p. 89-116.

Sutcliffe, A.G. and N.A.M. Maiden (1994): "Domain modeling for reuse." in *3rd International Conference on Software Reuse: Advances in Software Reusability*. Rio de Janeiro, Brazil: IEEE Computer Society Press, p. 169-177.

Volpano, D.M. and R.B. Kieburtz (1989): "The template approach to software reuse," in *Software Reusability: Concepts and Models*, T.J. Biggerstaff and A.J. Perlis, Editors. ACM Addison Wesley Publishing Company: New York, New York. p. 247-256.

Wirsing, M., R. Hennicker, and R. Stabl (1989): "MENU - an example for the systematic reuse of specifications." in *2nd European Software Engineering Conference*. Coventry, England: Springer-Verlag, p. 20-41.

Yglesias, K.P. (1993): "Information reuse parallels software reuse." *IBM Systems Journal.* ***32***(4): p. 615-620.

Zeroual, K. (1991): "KBRAS: a knowledge-based requirements acquisition system." in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 38-47.