The Knowledge Engineering Review, Vol. 32, e11, 1–21. © Cambridge University Press, 2017
doi:10.1017/S0269888917000030

Component-based approach for requirements reuse

AZEDDINE CHIKH

Laboratory of Research in Informatics of Tlemcen, Department of Computer Sciences, University of Tlemcen, 13000 Tlemcen, Algeria; e-mail: az_chikh@mail.univ-tlemcen.dz

Abstract

Reusing requirements improves product quality and the productivity of the development process. This paper investigates how the development of new requirements can be made more productive through reuse of the experience gained on similar requirements. This can be facilitated using a component-based reuse approach supported by a framework. Therefore, the central challenge for this research work is double: (1) to define a new concept for the requirement component as the combination of two types of knowledge: reusable knowledge and knowledge of reuse; (2) to define AFR (Analysis For Reuse) that represents the capitalization process by opposition to Analysis By Reuse that represents the process of reuse itself. Finally, we provide a case study related to the requirements of a hotel system to explain how the first process works. Through this approach, we use a framework 'Requirements Repository Framework (R2F)' in order to capitalize existing requirements within a repository for future reuse. We present the results of an experiment with three second-level student sections that used R2F in their IS240 course project; as well as with four student groups that used R2F in their respective senior projects. The objective of this experiment is to measure the usability of the proposed AFR process.

1 Introduction

Although requirements were partially neglected in the previous era of software engineering, they have received greater attention in the last decennia. Indeed, RE (requirements engineering) has become an integrated knowledge field with its own concepts, techniques, tools, methods, activities, and actors. Its importance is due to the role of good requirements in the success of any software product. In general, RE can be split into four activities: (1) requirements elicitation, (2) requirements analysis, (3) requirements specification, and (4) requirements validation. Several techniques for producing high-quality requirements specifications are proposed in each of these four activities (Wiegers, 2003).

In the last years, requirements reuse has become an important issue, and needs to be considered appropriately. Requirements reuse improves the productivity of the software development process and the quality of software products. Extensive requirements use during the entire software lifecycle makes such requirements very strategic information. Individual requirements act as fragments of information that integrate a semantic dimension, supported by the use of metadata. From this perspective, producing, sharing, and reusing requirements between several software projects within distributed environments have become necessary activities. Building requirements specification, by reusing individual requirements, creates new challenges.

The central challenge for this research work is double: (1) to define a new concept for the requirement component by combining two types of knowledge: reusable knowledge (RK) and knowledge of reuse

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

(KR); (2) to define AFR (Analysis For Reuse) that represents the capitalization process by opposition to ABR (Analysis By Reuse) that represents the process of reuse itself.

The rest of the paper is arranged as follows. After this introduction, a brief review of the literature and related work on requirements reuse are surveyed in Section 2. Our main contribution is presented first through the new component-based approach of the AFR process in Section 3, and second, through the requirements reuse framework in Section 4. Afterwards, Section 5 validates our approach through experimentation. Finally, Section 6 concludes the paper and suggests future directions for this research.

2 Related work

Requirements reuse has been investigated in various research directions. Table 1 displays works from four of these directions: (1) approach (principles, recommendations, vision, etc.): Akers (2008), Mussbacher and Kienzle (2013), and Niu *et al*. (2014); (2) method (framework, systems, etc.): Myklebust *et al*. (2014), Palomares *et al*. (2012), and Pacheco *et al*. (2015); (3) survey (understanding the state of the practice of requirements reuse and the factors that impact reuse effectiveness): Chernak (2012) and Palomares *et al*. (2014); and (4) case study (best practices and successful stories): Hauksdottir *et al*. (2012) and Tastekin *et al*. (2012).

The authors in Akers (2008) discussed the elements that compose a requirement and established a common understanding of how requirements evolve, how that evolution is retained, and how organizations can reuse requirements to speed business innovation, reduce complexity, and control costs. The authors argued that requirements reuse is not for everyone, and organizations first need to know where they lie on the requirements maturity curve. As organizations evolve along the curve, the need for reuse within their requirements management framework exists. Indeed, according to these authors, many companies are still in the infancy of requirements management because they have not yet adopted a requirements management tool. These organizations are not yet at a point of requirements sophistication where reuse support is necessary. However, if an organization has progressed on the maturity curve with respect to requirements management; manages multiple projects and thousands of requirements in parallel; and seeks to reduce complexity, lower the cost of development, and shorten innovation cycles, requirements reuse is a concept that should be investigated. The authors finally concluded that reuse may

Table 1 Research directions in the related work

Direction	References	Outcome
Approach	Akers (2008)	Strategic recommendations
	Mussbacher and Kienzle (2013)	A generic requirements reuse approach
	Niu *et al*. (2014)	A system-oriented approach to extracting functional requirements profiles
	Toval *et al*. (2008)	Eight key issues to be considered for an effective and practical reuse-based RE process
Method	Myklebust *et al*. (2014)	The CoVeR (CoVer all Requirements) method
	Palomares *et al*. (2012)	A catalogue of non-technical patterns included in the PARBRE framework
	Pacheco *et al*. (2015)	A model to reuse the requirements in a catalog
	Toval *et al*. (2002)	The SIREN method
Survey	Chernak (2012)	The results of a survey conducted in the global IT industry in 2010 and discuss the state of the practice for RE
	Palomares *et al*. (2014)	The survey results address the question: what is the current level of adoption of reuse practices during RE processes in organizations?
Case study	Hauksdottir *et al*. (2012)	Two different approaches for requirements reuse at Danfos
	Tastekin *et al*. (2012)	Analytical tool

RE = requirements engineering; IT = information technology.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

not be part of the short-term strategy, but successful strategic companies invest in the future, and this means that a process and tool framework will grow with the organization as it matures over time.

The authors in Mussbacher and Kienzle (2013) proposed arguments in favor of generic requirements reuse rooted in the vision that effectiveness requires a focus on the coordinated composition of reusable artifacts across the entire software development lifecycle. They then outlined for the RE community a research agenda associated with the vision presented for such an approach to reuse those requirements that build on concern orientation, that is, the ability to modularize and compose important requirements concerns throughout the software development lifecycle and model-driven engineering principles. In addition, early research results have been presented briefly that illustrate favorably the feasibility of such an approach.

Product line engineering has become the main method for achieving systematic software reuse. Embracing requirements in a product line's asset base enhances the effectiveness of reuse because engineers can work on abstractions closer to the domain's initial concepts. The authors in Niu *et al*. (2014) proposed a system-oriented approach for extracting functional requirements (FRs) profiles. The validated extraction constructs are amenable to semantic case analysis and orthogonal variability modeling for discovering the variation structure and constraints. To evaluate their approach, they presented an experiment for quantifying extraction effectiveness and a case study to assess their approach's usefulness.

The authors in Toval *et al*. (2008) identified and validated eight key issues to be considered for an effective and practical reuse-based RE process based on their experience gained through the SImple REuse of RequiremeNts (SIREN) method definition and application (Toval *et al*., 2002).

In a requirement process for cross-border maintainability and maintenance operations, including renewal, it is important for the infrastructure managers to gain an overview of all applicable regulations and be able to extract relevant requirements with regard to maintainability and maintenance. As an aid to achieving an overview of all requirements, the authors in Myklebust *et al*. (2014) developed the CoVeR method (CoVer all Requirements), which ensures that all the regulatory requirements are extracted and adapted, and the result from the work can be reused. The CoVeR method is a structured approach for ensuring the control of all relevant requirements. A commercial tool has been utilized: IBM Rational DOORS (Dynamic Object Oriented Requirements System), which has been designed to manage large sets of requirements. The tool makes it possible to achieve linking and traceability between requirements on different levels. It also allows linking between requirements and evidence of verification, and it is capable of detecting linking gaps between requirements and sub-requirements, and/or gaps between requirements and evidence of verification.

The authors in Palomares *et al*. (2012) presented SRP (Software Requirement Patterns) as an artifact for fostering requirements reuse. PABRE (Pattern-Based Requirements Elicitation) is a framework that promotes the use of these patterns as a means for requirements elicitation, validation, and documentation. A catalogue of non-technical patterns is included in the framework. The authors presented the overall structure of a set of non-technical patterns integrated in the PABRE framework that are part of the catalogue constituted by 113 patterns (functional, non-functional, and non-technical).

The authors in Pacheco *et al*. (2015) proposed a model for reusing the requirements in a Requirements Catalog. Their model aims to define the guidelines for performing the requirements reuse process through four principal activities: searching, selecting, adapting, and implementing. In addition, the commercial tool Computer-Aided Requirements Engineering (CARE) Rational RequisitePro was tailored to make feasible the implementation of the proposed model.

The authors in Toval *et al*. (2002) proposed the SIREN method. This is a practical approach for selecting and specifying the requirements of a software system based on requirements reuse and software engineering standards. SIREN encompasses a spiral process model, requirements documents templates, a reusable requirements repository organized by catalogs, and a supporting CARE tool called SirenTool.

Implementing reuse in practice remains challenging, and the IT (information technology) community has scant visibility into the state of the practice specifically as it pertains to reusing requirements. The authors in Chernak (2012) presented the results of a survey conducted in the global IT

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

industry in 2010, and discussed the state of the practice for requirements reuse. The survey studied reuse adoption in two different contexts, that is, Software Product Lines and Software Maintenance. Analysis of the survey data focuses on the latter context as a more common case in practice and investigates the impact of various factors on reuse adoption and effectiveness. An important finding of the survey is that most of the survey respondents believe that requirements reuse is important and can help teams reduce time-to-market and development costs. However, implementing reuse remains challenging in practice where the main obstacle is the poor quality of existing requirements.

The same authors in Palomares *et al*. (2012) presented in Palomares *et al*. (2014) the preliminary results of a survey that addressed the question, 'what is the current level of adoption of reuse practices during RE processes in organizations?'.

The survey first investigated requirements reuse in general, and then elaborated by asking about a specific technique SRP. The survey results showed that requirements reuse is not a widespread practice in IT projects, where the most common techniques are those based on the copy and later modification by hand of requirements originated from previous projects. With regard to the use of SRP, the results seem to support the hypothesis that SRP could help ameliorate some common problems related to requirements specifications, such as lack of uniformity and incompleteness.

The study in Hauksdottir *et al*. (2012) described two different approaches for requirements reuse at Danfoss. The first approach reused those requirements envisioned as common between two consecutive projects, and allowed changing and parameterizing parts of the requirements. The second approach organized all requirements into a common model and explicitly managed variability and different requirement variants in this common model. The results showed that both approaches result in significant savings in reduced effort by reusing common requirements. The first approach was found to be effective when the domain maturity is low and the significant set of requirements were changed from project to project. The second approach allowed high reuse potential and significant savings for stable domains, where most requirements tend to be minor changes of existing requirements.

Product development time estimation is important for project management tasks. The study in Tastekin *et al*. (2012) investigated the impact of requirements reuse on product development duration for different products in a similar domain. An analytical tool was proposed for estimating the minimum time to be saved given the percentage of requirements reused from earlier projects. Empirical results of industrial case studies were used as inputs to this study. Three cases from different organizations were studied. The results of the case studies were compared with a study in the literature on product development time. According to the industrial empirical results, a modified product development time estimation method was proposed for projects.

From the previous related works, more especially those that cover the approach and method directions, three critical observations can be concluded: (1) there is a lack of maturity in research in the field of requirements reuse; (2) the two processes of development for reuse and by reuse need to be complementary and not stand-alone activities of requirements reuse. They should interact with each other through a common repository framework; and (3) there has been less emphasis on requirements metadata, which could lead to the aforementioned difficulties in requirements reuse.

Therefore, because of the diversity of requirements and the heterogeneity of their styles, requirements reuse is particularly complex and requires an appropriate methodology. We resolve this problem by:

1. Differentiating between two processes: the AFR process of capitalizing existing requirements from old or current software projects, and the ABR process of reusing them in writing new requirements. Only the first process is covered in this present research, where a sequence of steps is defined.
2. Designing RML (Requirements Metadata Language).
3. Supplementing RK of existing requirements with its KR based on RML metadata in order to facilitate its reuse.
4. Creating a conceptual requirement reuse model that provides a set of features that support their organization within a repository.
5. Developing a requirements reuse framework that supports the AFR process.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

3 Component-based approach

Our research hypothesis is that it is very likely that different software products of the same family have many common requirements. Accordingly, if these requirements were stored, maintained, and shared in a central requirement repository, requirements engineers could partially or entirely reuse them in authoring new requirements in similar products. This reuse is motivated by at least the two following reasons:

1. Reducing elicitation and analysis efforts, and thus increasing productivity:
   i. It is faster to reuse than perform the work from the start ('from scratch');
   ii. The requirements have already been verified and tested, and therefore the project risk is reduced.
2. Facilitating system interoperability by ensuring more common core requirements between products of the same family.

Furthermore, we argue that in order to develop a new reuse approach, we need to perform two specific actions: (1) AFR: to define the adequate way of capitalizing existing requirements as RK within the repository, and qualify them with KR using RML metadata; (2) ABR: to define a process for retrieving the previous RKs from the repository, and adapt them during a new requirements analysis.

In this research, we focus only on the first action. Sections 3.1 and 3.2, respectively, present the concepts of RK and KR. Section 3.3 presents the conceptual requirement reuse model that encompasses both RK and KR. Section 3.4 presents the AFR process. Section 3.5 illustrates the AFR process through a case study.

3.1 Reusable knowledge

We define RK as any entity, atomic or composite, digital or non-digital, that is candidate for long-term reuse by requirements engineers. The reuse of each RK implies the concurrent reuse of other RKs associated with it, and the eventual corresponding design artifacts and code.

3.2 Knowledge of reuse

We define KR as the metadata associated to RK according to RML. Indeed, in order to describe and manage RKs, some domain-specific vocabularies are needed to support their semantic retrieval. In this section, we present the RML structure, which corresponds to the metadata schema building block presented in Haslhofer and Klas (2010). The RML structure is mainly inspired by LOM (Learning Object Metadata), a standard that specifies the syntax and semantics of Learning Object Metadata, defined as the attributes required to describe a Learning Object (IEEE, 2002). In fact, the RML data model is structured as a hierarchy that includes the categories, aggregate data elements, and simple data elements that represent the leaf nodes. Simple data elements have individual values defined through their matching value space and data type. All data elements are optional. A data element could contain a list of ordered or unordered values, rather than a single value. A vocabulary, which is a recommended list of appropriate values, could be assigned to some data elements.

Table 2 lists the RML base schema that identifies the categories and their embedded data elements, explanation, and corresponding value space.

The category 'Meta-Metadata' is domain independent, and thus its composition is similar to that of the corresponding category in LOM (IEEE, 2002).

The categories 'General' 'Lifecycle' 'Technical' 'Rights', and 'Annotations' are almost similar to their corresponding categories in LOM. Nevertheless, their content was adapted to the field of RE.

Finally, the three categories 'Requirements specific', 'Traceability', and 'Taxonomy' which are domain dependent, were completely designed to fit the particularities of the RE domain.

These last categories and some of their most important embedded data elements are described below.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Table 2 The Requirements Metadata Language base schema

No.	Name	Explanation	Value space/example
1	General	This category groups the general information that describes RK	
1.1	Identifier	A globally unique label that identifies this RK	
1.2	Language	The primary human language used within this RK	
1.3	Structure	Underlying organizational structure of this RK	Atomic; composite
2	Lifecycle	This category describes the history and current state of this RK	
2.1	Version	It allows to keep track of all the possible versions of an RK	
2.2	Status	The completion status or condition of this RK	Draft; revised; final
2.3	Contribute	All contributions to the state of this RK during its lifecycle	
2.3.1	Role	Nature of contribution	Elicitation, specification, verification, validation
2.3.2	Actor	The identification of analysts by nature of contribution	
2.3.3	Date	The date of contribution	
3	Meta-Metadata	This category describes this metadata record itself	Given that this category is domain independent, its composition is similar to that of its corresponding category in LOM
4	Technical	This category describes the technical characteristics of this RK	
4.1	Format	Technical data type (s) of the content of this RK	MIME (Multipurpose Internet Mail Extensions) types or 'non-digital'
4.2	Size	This data element shall refer to the size of the content of this RK	
4.3	Location	A string that is used to access this RK	URL; URI
5	Requirements specific	This category describes the key semantic characteristics of the RK	
5.1	Style	The style used to represent RK	Data model; data dictionary; data expression; virtual windows; context diagrams; event list
5.2	Data nature	Whether this RK is quantitative or qualitative	Quantitative; qualitative
5.3	Output type	The target or output of this RK, typically a product or process	Product; process; both
5.4	Derivation	How this RK was obtained	Primary; interpreted; derived
5.5	Scope	The product family where the RK is reusable	Accounting; human resources management
5.6	Commonality	The objective is to explicitly model the commonality and variability of this RK within a given software product family	Common; optional
5.7	Granularity	The RK may range from a high-level abstract statement of a service to a detailed mathematical functional specification	1: RK is typically 'atomic'; 2: RK can include RK of level 1; 3: RK can include RK of level 2
5.8	Reusability	The RK is completely or partially reusable	Completely; partially
5.9	Importance	This RK is assessed as critical if it is important to success of the related product	High; medium; low
5.10	Risk	The assessed risk level of RK in meeting needs	High; medium; low
5.11	Stability	The assessment of the maturity or stability of RK	High; medium; low
5.12	Fit criterion	A fit criterion is a precise and testable statement of RK	A quantitative or qualitative measure
6	Rights	This category describes the rights and conditions of use for this RK	
6.1	Cost	Whether use of this RK requires payment	

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

6.2	Copyright	Whether copyright or other restrictions apply to the use of this RK	
6.3	Description	Comments on the conditions of use of this RK	
7	Traceability	This category describes the relationships of this RK with other RKs and artifacts	
7.1	Related RK	A reference link to a related RK	
7.1.1	Identifier	The identifier of the related RK	
7.1.2	Type	The traceability relationship type	Dependency; refinement; evolution; satisfiability; overlap; conflict; and rationalization
7.2	Related source	A reference link to a source of RK	
7.2.1	Identifier	The identifier of the related source	
7.2.2	Type	The source type regulation	Contract; standard
7.3	Design artifact	A reference link to a related design artifact	
7.3.1	Identifier	The identifier of the related design artifact	
7.3.2	Type	The type of the related design artifact	Class diagram; state machine diagram.; sequence diagram …
7.4	Reusing software	A reference link to the reusing software product	
7.4.1	Identifier	The identifier of software product where the RK was reused	
7.4.2	Family	The family of the related software product	Accounting; human resources management
7.5	Alternative RK	A reference link to an alternative RK	
8	Annotation	This category provides comments on the use of this RK, and information on when and by whom the comments were created	
8.1	Entity	People who created this annotation	
8.2	Date	The creation date of the annotation	
8.3	Description	The content of this annotation	
9	Taxonomy	This category is intended to be used for an arrangement of RKs into groups	
9.1	Requirements level	The level of RK according to the goal level scale	Business; user; system; design
9.2	Requirements type	The software aspect described by this RK	Functional; non-functional; data
9.3	Non-functional type	When the type of RK is ‘Non-Functional’	Product; organizational; external; data
9.4	Product type	When the non-functional type of the RK is ‘Product’	Availability; flexibility; integrity; maintainability; portability; reliability; safety; security; supportability; sustainability; usability
9.5	Organizational type	When the non-functional type of this RK is ‘Organizational’	Environmental; operational; development
9.6	External type	When the non-functional type of this RK is ‘External’	Regulatory; ethical; legislative

RK = reusable knowledge.

3.2.1 Requirements specific

1. Style (RML Metadata 5.1): the style used to represent RKs, such as data models; data dictionaries; data expressions; virtual windows for data requirements (DaR), context diagrams, and event lists; and use cases for FRs.
2. Scope (RML Metadata 5.5): the product family where the RK is reusable, such as accounting and human resources management.
3. Commonality (RML Metadata 5.6): it is interesting to analyze within families of software products those RKs that are common or optional. Common RKs are those that all products of that family must have. Optional RKs are those that some products of that family have.
4. Granularity (RML Metadata 5.7): RKs might range from high-level abstract statements of a service or constraint to a detailed mathematical functional specification (Sommerville, 2011).
5. Reusability (RML Metadata 5.8): it is interesting to analyze within families of software products those RKs that are completely or partially reusable. Indeed, when generic RKs might be reusable without any modification, more specific RKs might need some context adaptation efforts.

3.2.2 Traceability

1. Other RKs (RML Metadata 7.1): they allow linking the current RK with other RKs according to different types of traceability, such as overlap, satisfiability, dependency, evolution, generalization/refinement, conflict, and rationalization (Ramesh & Jarke, 2001).
2. Sources (RML Metadata 7.2): this is concerned with the notion of pre-traceability relationships (Gotel & Finkelstein, 1994). It includes the relationship between RKs and the sources from which the RKs originated, such as regulations, contracts, standards, and software projects.
3. Design artifact (RML Metadata 7.3): this is concerned with the notion of post-traceability relationships (Gotel & Finkelstein, 1994). It includes the relationship between RKs and the design artifacts created in architectural and detailed designs, such as class, state machine, and sequence diagrams.
4. Reusing software (RML Metadata 7.4): RKs should be reused in software products whose family must be the same as their scope (RML Metadata 5.5).
5. Alternatives (RML Metadata 7.5): it is interesting to consider alternative RKs of a given RK. This increases its reusability.

Based on their semantics, traceability relationships present information that can lead to the reuse of system components when these components are related to RKs of existing systems that are similar to RKs of new systems (Spanoudakis & Zisman, 2005).

3.2.3 Taxonomy

1. Level (Metadata 9.1 in RML): RKs might belong to one level based on the goal level scale defined by Lauesen (2002): business, user, system, and design. Business requirements aim to justify why the customer wants to spend money on a given product. This can be verified, although only after some period of operation. URs (user requirements) outline the user tasks involved and require support for these tasks. SRs (system requirements) specify what comes in and goes out of the product. It identifies only the functions and features. DRs (design requirements) specify one of the product interfaces in detail.
2. Typology (Metadata 9.2 and 9.3 in RML): RKs might be either FRs or NFRs (non-functional requirements). We have added DaR to this former classification. What data should the system input and output, and what data should the system store internally? (Lauesen, 2002). NFRs might be classified according to Sommerville (2011) as follows: product, organizational, and external requirements.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

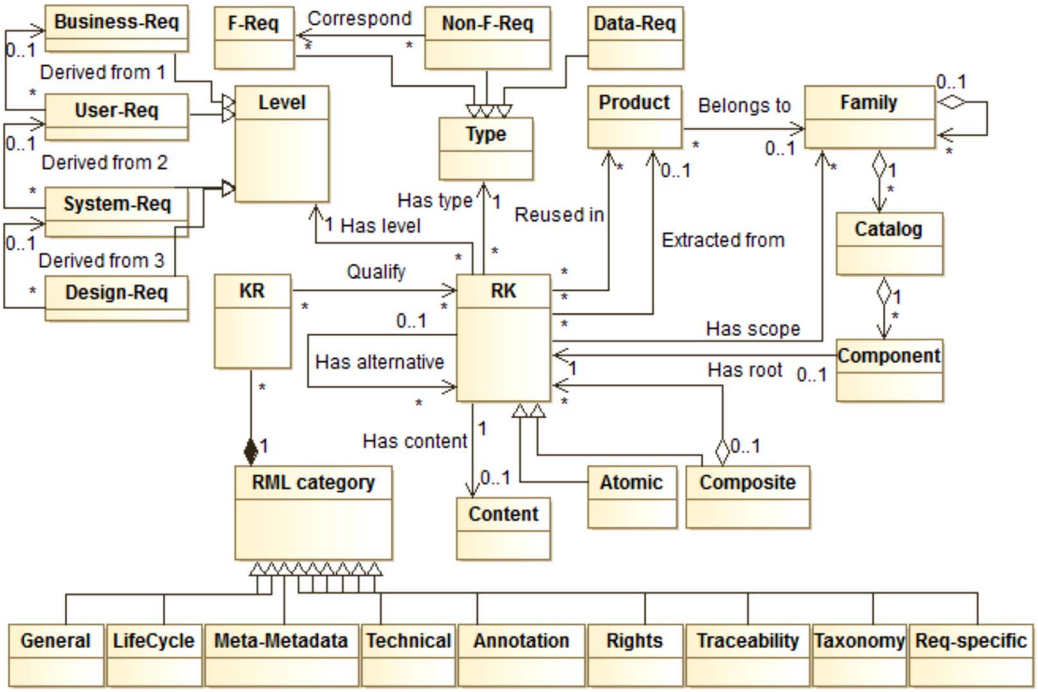


Figure 1 UML class diagram of the conceptual requirement reuse model. KR = knowledge of reuse; RK = reusable knowledge; RML = Requirements Metadata Language

KRs are used in information retrieval for calculating the similarity between components during AFR on one hand, and between components and a requirement query during ABR on the other.

3.3 Repository organization

Awareness of the critical nature of requirements on one side and the experience accumulated by practitioners in reuse determine the adoption of proper best practices that can increase the probability of capitalizing existing requirements during software projects. We aim to help requirements engineers to accomplish their tasks during the AFR process, thus allowing them to share their best practices that reflect their experience and know-how. These best practices, which are materialized with RKs and their corresponding KRs, can be managed, classified, stored, and retrieved from a systematic repository.

The conceptual requirement reuse model provides a set of features that support the organization of the repository.

Figure 1 illustrates this model using a UML (Unified Modeling Language) class diagram. At the center of the diagram, we have the main class 'RK' that corresponds to the RK. It has two subclasses, 'Atomic' and 'Composite' (RML Metadata 1.3). The classes 'KR' and 'RK' are connected through the association 'Qualify'. The use of RML metadata as KR is represented with the composition between the classes 'KR' and 'RML category' on one side and the nine subclasses of the class 'RML category' on the other.

The text description part of RK, such as 'The system should/shall store data according to the following data model' is stored as a value of the description attribute of the class 'RK'. The content part of RK, such as the entity relationship diagram itself, is stored separately from the class 'RK' as an instance of the class 'Content' linked to the former class through the association 'Has content'.

The reflexive association 'Is alternative' attached to the class 'RK' represents the different alternative RKs (RML Metadata 7.5) of a given RK. The association 'Extracted from' between the classes 'RK' and 'Product' shows the product from where the given RK was capitalized (RML Metadata 7.2). Reciprocally, the association 'Reused in' shows the product where RK was reused (RML Metadata 7.4).

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

The association 'Belong to' between the classes 'Product' and 'Family' represents the family of a given software product. The association 'Has scope' between the classes 'RK' and 'Family' represents the RML Metadata 1.3. The commonality characteristic (RML Metadata 5.6) of RKs within a family of software products is represented with an attribute of the former association.

The Player-Role Pattern is used in order to avoid multiple inheritance of the class 'RK', where the class 'RK' is the player and the classes 'Level' and 'Type' are the roles. The class 'Level' is specialized into four subclasses 'Business Req', 'User Req', 'System Req', and 'Design Req' that correspond, respectively, to the business, user, system, and DRs (RML Metadata 9.1). Each of the three-ordered pairs of these subclasses is related by the association 'Derived from'. The class 'Type' represents the 'Functional', 'Non-functional' and 'Data' types (RML Metadata 9.2) represented by the 'F-Req', 'Non-F-Req', and 'Data-Req' classes. The two first former classes are related by the association 'Correspond'.

The repository is structured into three layers: the first layer contains the families of software products (Class 'Family'), such as accounting and human resources management. The second layer includes the potential catalogs (Class 'Catalog') of reusable requirements within a given family of software products. A catalog groups a set of related components (Class 'Component') that correspond to the function of an application domain. Examples of such functions in the human resources management domain could be 'Hiring' and 'Promotion'. The third layer includes the components. A component encompasses a hierarchy of functional RKs (Association 'Has root'), their alternatives, and their eventual corresponding RKs of type 'Non Functional Requirement', along with their associated KRs. It is the application of this organization to all reusable requirements that positions such requirements as valuable assets in full alignment with the strategic goals of the organization.

3.4 Analysis for reuse process

The repository should not be considered a final fixed product, but an evolutionary one to be enriched (Toval *et al.*, 2008). Hence, we should first discover those sources that are relevant for capitalizing new relevant reusable requirements. Among the potential sources, we might identify the following two:

1. Current/past software projects: they might be selected and extracted from software requirements specification of current or past software projects because of their high reusability and quality. These candidates need quality improvement, and more likely neutralization (to hide some specific names or confidential content) and parameterization (to make them more reusable).
2. Other repositories: they might be imported from other existing repositories or catalogs. Naturally, the rules of reuse rights should be respected.

Finally, we mentioned that reusable requirements might also be built, especially for reuse purposes. This could be triggered, for example, by a manager's decision in order to cover certain voids or the absence of such requirements in the repository.

Naturally, we mentioned that it is not necessary to complete KRs for all candidate requirements before the preprocessing step. Instead, only partial information is required. A requirements engineer or any other actor responsible for the AFR process could complete only some pertinent metadata from RML that reflect his/her search goals and whose data are available. Such metadata could be mainly combined from the most important embedded data elements of the categories 'Requirements specific' and 'Taxonomy' described in Section 3.b. A query example could retrieve SRs (RML Metadata 9.1) that are functional (RML Metadata 9.2) and represented in some style, such as use case diagrams (RML Metadata 5.1). The other RML categories and embedded data elements could be used simply for refining results in case their number is very important.

A candidate RK undergoes the three steps described below. These macro-steps are repeated for each candidate RK (RK_x) to be capitalized.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

1. Preprocessing: aimed at checking RK_x (a candidate RK) according to the following algorithm:

$Algorithm : Preprocessing$
$Begin$
$Inputs :$
$R : An\ existing\ requirements\ repository$
$RK_x : A\ candidate\ RK$
$Decide : Boolean$
$/* \ True\ in\ case\ RK_x\ should\ be\ inserted\ in\ R\ either\ as\ an\ alternative\ or\ new\ RK,\ or\ used\ to\ enhance\ an\ existing\ RK \ */$
$Decision : String \quad /* \ Text\ of\ the\ decision \ */$
$/* \ Calculation\ of\ semantic\ similarity\ between\ RK_x\ and\ all\ RKs\ in\ R \ */$
$Decide \leftarrow False$
$RK_y \leftarrow RK \quad /* \ RK_y\ is\ initialized\ with\ the\ first\ RK\ in\ R \ */$
$While\ (Not\ (end\ of\ R)\ and\ Not\ (Decide))$
If (Sem-Simil (RK_x, RK_y))
$Then$
$If\ (Alternative\ (RK_x, RK_y))$
$Then\ Decision \leftarrow$ "$Insert\ RK_x\ in\ R\ as\ a\ new\ alternative\ of\ RK_y$"
$Else$
$Decision \leftarrow$ "$Enhance\ RK_y\ using\ RK_x$"
$Endif$
$Decide \leftarrow TRUE$
$Endif$
$RK_y \leftarrow RK \quad /* \ Go\ to\ next\ RK\ in\ R \ */$
$EndWhile$
$If\ (Not(Decide))$
$Then$
$Decision \leftarrow$ "$Insert\ RK_x\ in\ R\ as\ a\ new\ distinct\ RK$"
$Decide \leftarrow TRUE$
$Endif$
$Outputs :\ Decide,\ Decision$
End

This algorithm is based on the following Boolean functions:

i. Sem-Simil: automatic similarity computation between RK_x and RK_y based on their respective KRs to indicate their semantic overlap. We assume that both KRs for RK_x and RK_y have the same representation structure. When we compare these KRs, we must consider the metadata language, type of metadata, metadata values, and how we can access them all. When we match KR of RK_x with KR of RK_y, it is not efficient to use all metadata from RML because the process of doing so is difficult and hard to maintain, especially in a repository that contains thousands of requirements.We adopt the metric proposed in Menndez-Domnguez *et al*. (2011), which we adapt to serve our purposes with regard to the RE context. The formula then calculates the similarity degree (Sim) between requirements (RK_x, RK_y) and considers the similarity average (simMeta) of the value (v) of the metadata within a RE context (C):

$$Sim(RK_x, RK_y) = \frac{\sum_{i=1}^{n} simMeta(vx_i, vy_i)C_i}{n}$$

where n is the number of metadata pieces to compare, $simMeta(vx_i, vy_i)$ the semantic distance between value v of metadata i for RK_x and RK_y.

$$SimMeta \in [0, 1].$$

C_i is the relevance of metadata i in a particular C context. $Ci \in [0, 1]$.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

ii. Alternative: empirical assessment of the condition about alternatives by a requirements engineer expert. This function aims to check whether RK_x could be considered an alternative to RK_y. Otherwise, its content is used to manually improve RK_y's content.

2. Capitalization: aimed at capitalizing a candidate RK according to the decision made in the previous step, either: as a new distinct RK; as a new alternative to an existing RK; or by using it to improve the existing RK's content.
3. Qualification: aimed at qualifying the new capitalized RK with KR using RML metadata, or editing the existing KR.

Thus, all RKs are maintained in the repository, which increases continuously, thus requiring management efforts. Such management is mainly concerned with performance optimization that must be conducted periodically because analysts require that change according to the demand in software projects. On the other hand, repository evolution is concerned with maintaining the repository content up-to-date with respect to the strategic goals defined by the repository managers. Hence, statistics reports related to the use of RKs are produced in order to support them.

3.5 Case study

This case study is related to the capitalization of reusable requirements from an existing requirements documentation of virtual successful hotel management software. It aims to illustrate the first two steps of the AFR process for capitalizing reusable requirements from the 'Make reservation' use case. Table 3 describes the exchange of information between the three involved roles (the customer who initiates the process and the existing software). Table 4 describes a part of the functional SRs derived by requirements engineers from the three first steps of the previous use case. Table 5 describes part of the non-functional SRs derived by requirements engineers from the previous use case.

We now apply the AFR process for capitalizing reusable requirements from Tables 3, 4, and 5. We suppose that (1) we have initially selected the virtual successful hotel management software project as a potential source for capitalizing reusable requirements; (2) only those requirements related to the 'Make reservation' use case were selected as the more relevant reusable requirements. Only the two main macro-steps of the process, namely, the capitalization and qualification steps repeated for each candidate RK to be capitalized, are described below.

1. Capitalization: in this example, we assume that no previous RK exists in the repository that belongs to the family of product 'Hotel reservation systems'. Accordingly, we assume that we determined during the previous step (preprocessing) to capitalize all candidate RKs, URs (Table 3), and SRs (Tables 4 and 5) considered as new distinct RKs. Table 6 indicates how these RKs were capitalized within the repository according to the conceptual requirement reuse model presented in Figure 2. The first colon 'ID' of Table 6 shows the RK identifier within the repository. The second colon 'No' shows the original identifier within the existing source documentation. This helps manage the traceability between RK and its source. The third colon gives a short textual description stored in the description attribute of the class 'RK'. The fourth colon 'P' contains the parent of the RK. Indeed, all the functional SRs of this case study form one hierarchy with rk1 (that corresponds to the use case UC1) as root. The fifth colon 'D' shows the RK of the upper level from which the current RK is derived. The sixth colon 'A' shows the RK for which the current RK is an alternative.
2. Qualification: aimed at qualifying the previous RK in Table 6 with the KR that uses RML metadata. For clarity, Table 7 presents only the qualification of the part of this RK related to the functional SRs. We use a subset of RML metadata composed of the following metadata: 1.1. Identifier, 1.4. Structure, 5.1. Style, 5.5. Commonality, 5.8. Importance, 5.11. Fit criterion, 9.1. Req. level, and 9.2. Req. type.

We find from our practice during the capitalization step that most of the candidate RKs are from the UR level. This can be explained by the specificities of those from the SR level that make their reusability low. Indeed, URs are at a high level, thus giving a less detailed specification. As a result, there might be several SRs at a low level that give more detailed specifications for more or less of the same UR.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Table 3 A textual description of the functional user requirements (make reservation use case)

Identifier	UC1
Name	Make reservation
Initiator	Customer
Goal	A room in a hotel is reserved for a customer
Precondition	None
Post-condition	A room of the desired type will have been reserved for the customer for the requested period, and the room will no longer be free
Main success scenario	
Step 1	Customer makes a reservation request
Step 2	Customer selects the desired hotel, dates, and type of room
Step 3	Software provides the availability and price for the request, that is, one or more offers are made
Step 4	Customer accepts the offer
Step 5	Software requests identification and contact details
Step 6	Customer provides identification and contact details of the customer for the softwares records
Step 7	Software creates a reservation and gives it a number
Step 8	Software allocates a room to the reservation
Step 9	Software reveals the reservation number to the customer
Extensions	
Step 3.a	No availability in this hotel. The software offers alternatives at a different hotel or hotels within a limited radius
Step 3.b	No availability in this hotel or in any other hotels within a limited radius. The software informs the customer
Step 4.a	Offer not accepted. The software terminates the use case
Step 6.a	Customer already on record. The scenario continues at step 7

Table 4 A textual description of the functional system requirements

No.	Step	Description	Fit criterion
fr1	1	Software shall allow a customer to initiate a reservation	The reservation page shall be displayed
fr2	2	Software shall obtain a hotel name from the user	A valid hotel name shall be accepted
fr3	2	Software shall accept a range of dates from the user	A valid range of dates shall be accepted
fr4	2	Software shall accept a room type from the user	A valid room type shall be accepted
fr5	3	Software shall check the availability of a given room type in a given hotel for a given date	Offers consistent with the availability of a room shall be made
fr5.a	3.a	No availability in this hotel. Software shall check the availability of alternatives at a different hotel or hotels within a limited radius	Offers consistent with the availability of a room shall be made
fr5.b	3.b	No availability in this hotel or in any other hotels within a limited radius	No offer shall be made, and the customer shall be informed

Table 5 A textual description of the non-functional system requirements

No.	Description	Fit criterion
nf1	Software shall make use of a small number of bright colors, to conform to the brand image of the hotel chain	Software shall be designed with black and white forms with a sky blue title bar and a yellow motif in the bottom right-hand corner
nf2	Software shall have uncluttered forms	Each form shall be limited to a max of eight input boxes

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Table 6 Capitalization of the candidate reusable knowledges (Tables 3, 4 and 5)

Id	No.	Description	P	D	A
rk1	uc1	Make reservation (UC1)			
rk2	st1	Customer makes a reservation request	rk1		
rk3	st2	Customer selects the desired hotel, dates, and type of room	rk1		
rk4	st3	Software provides the availability and price for the request, i.e. one or more offers are made	rk1		
rk5	st4	Customer accepts the offer	rk1		
rk6	st5	Software requests identification and contact details	rk1		
rk7	st6	Customer provides identification and contact details of the customer for the softwares records	rk1		
rk8	st7	Software creates a reservation and gives it a number	rk1		
rk9	st8	Software allocates a room to the reservation	rk1		
rk10	st9	Software reveals the reservation number to the customer	rk1		
rk11	st3.a	No availability in this hotel. Software offers alternatives at a different hotel within a limited radius	rk1		rk4
rk12	st3.b	No availability in this hotel or in any other hotels within a limited radius. Software informs the reserves	rk1		rk4
rk13	st4.a	Offer not accepted. Software terminates the use case	rk1		rk5
rk14	st6.a	Customer already on record. Scenario continues at step 7	rk1		rk7
rk15	fr1	Software shall allow a customer to initiate a reservation		rk2	
rk16	fr2	Software shall obtain a hotel name from the user		rk3	
rk17	fr3	Software shall accept a range of dates from the user		rk3	
rk18	fr4	Software shall accept a room type from the user		rk3	
rk19	fr5	Software shall check the availability of a given room type in a given hotel for a given range of dates		rk4	
rk20	fr5.a	No availability in this hotel. Software shall check the availability of a different hotel within a limited radius		rk11	
rk21	fr5.b	No availability in this hotel or in any other hotels within a limited radius		rk12	
rk22	nf1	Software shall make use of a small number of bright colors, to conform to the brand image of the hotel chain			
rk23	nf2	Software shall have uncluttered forms			

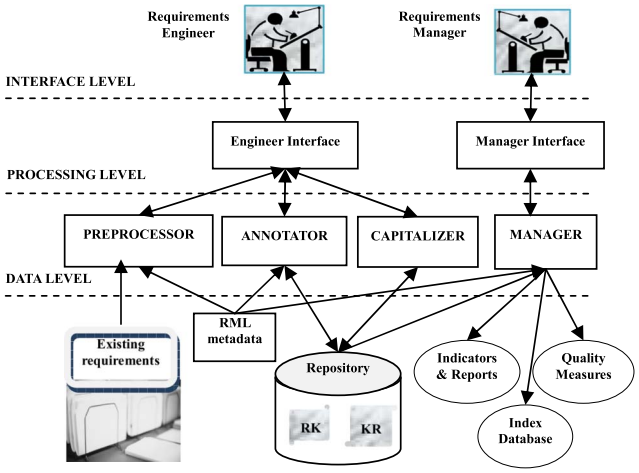


Figure 2 Requirements Repository Framework architecture. RML = Requirements Metadata Language; RK = reusable knowledge; KR = knowledge of reuse

Table 7 Qualification of some reusable knowledges using a subset of Requirements Metadata Language metadata

1.1	1.4	5.1	5.6	5.9	9.1	9.2	5.12
rk15	1	SP	C	H	SR	FR	The reservation page shall be displayed
rk16	1	DFD	C	H	SR	FR	A valid hotel name shall be accepted
rk17	1	DFD	C	H	SR	FR	A valid range of dates shall be accepted
rk18	1	DFD	C	H	SR	FR	A valid room type shall be accepted
rk19	1	AD	C	H	SR	FR	Offers consistent with the availability of a room shall be made

1.1. = identifier; 1.4. = structure (1: Atomic, >1: Composite); 5.1. = style (SP = Screens and Prototypes; DFD = Data flow diagrams; AD = Activity diagrams); 5.6. = commonality (C = Common; O = Optional); 5.9. = importance (H = High; M = Medium; L = Low); 5.12. = fit criterion; 9.1. = Req. level (BR = business requirements; UR = user requirement; SR = system requirement; DR = design requirement); 9.2. = Req. type (FR = functional requirement; NFR = non-functional requirement).

4 Analysis for reuse support framework

The AFR process is complex and requires a substantial supporting framework of services if it is to transform the experience of RE. Frameworks for different phases of RE are available, such as frameworks for requirements analysis, elicitation, traceability, and reuse. The authors in Husnain *et al*. (2009) reviewed frameworks for the different SRE phases. This review was performed according to a conceptual framework that encompasses the (1) problem statement, (2) domain, (3) proposed solution, (4) phases discussed, (5) phases not discussed, and (6) reusability. We emphasize that the adoption of reuse is still extremely limited in real projects, whereas the need for solutions that reduce software analysis efforts is increasing. This gap is more likely due to a lack of commercial AFR tools.

Our proposed system R2F (Requirements Repository Framework) supports the AFR process presented in Section 3.d, and it is intended to be used mainly by requirements engineers and managers. R2F provides guidance and support to requirements engineers when they want to capitalize new RKs and to managers when they want to manage the requirements repository. The architecture shown in Figure 2 indicates the framework built around the four modules described below. The first three modules are mainly used by the requirements engineer when the fourth is used by the manager.

1. Preprocessor: aimed at supporting step 1 of the AFR process. Techniques from information retrieval are used where similarities between RKs are calculated to indicate the semantic overlap between RKs.
2. Capitalizer: aimed at supporting step 2 of the AFR process. It allows also editing an existing RK. This module is clearly necessary for the framework to support the creation of a new RK from an existing

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

requirement. New RKs can also be built by aggregation of lower RKs of the same type and level. The creation and editing process might also need to be aware of the constraints imposed on RKs. Although constraints restrict possible RKs, there is a separate requirement for providing alternatives to an existing RK, thus acting as other variants. In addition to editing the text description part of RK, such as 'The system should/shall store data according to the following data model', it is also necessary to incorporate the content part of RK, such as the entity relationship diagram. This content tends to be created and managed separately from the RK instance, and then needs to be associated to it as shown in Figure 1.

3. Annotator: aimed at supporting step 3 of the AFR process. It exploits the RML metadata in order to annotate RKs within the repository, create new KRs, and edit existing KRs. Semantic annotation adds diversity and richness to the search process and helps resolve the ambiguity of the natural language when translating notions into a formal computational language. By qualifying RKs using RML metadata, it is possible to process complex filters and search operations.
4. Manager: aimed at supporting the management process. It mainly considers storage, change management, and traceability of RKs from the repository, in addition to maintenance issues related to the repository. RKs are indexed within the repository using KRs. An index database is then generated. In addition, different useful statistics are provided for the repository, thus allowing its optimization and reorganization. The manager module also allows its users to filter the repository content. Indeed, the combination of (1) types: 'Functional', 'Non-functional', and 'Data'; (2) levels: 'Business', 'User', 'System', and 'Design'; and (3) styles: such as use case diagrams and DFDs (data flow diagrams), is very expressive and permits a tremendous amount of flexibility for extracting a set of desired RKs. For example, a user might be interested in extracting functional SRs using DFDs.

Figure 3 shows a screenshot of the preprocessor module of R2F for checking the similarity of the candidate RK (RK-HM-RE-034), related to the hotel reservation system, with existing RKs from the repository. We can see the highest resulting similarity rate (0.17) with an existing RK (RK-SA-PO-181) from the catalog 'Purchase Order' of the product family 'Sales management systems'. Because of this low value the recommended decision was to insert RK-HM-RE-034 in the repository as a new distinct RK.

Figure 4 shows a screenshot of the capitalizer module of R2F for editing an existing RK (RK-HM-RE-034). The screen displays the selected RK identifier (left top corner) as well as its corresponding text. Also the user has a set of gray buttons on the left which allow him to browse, edit, save, print, delete, and export the RK content. Finally the 'Qualify' button (left bottom corner) allows to switch to the qualification task using RML metadata (Figure 5).

Figure 5 shows a screenshot of the annotator module of R2F for qualifying editing the previous RK (RK-HM-RE-034) using the 'Requirements specific' category of RML. The screen displays first the RK identifier as well as its corresponding text. The left panel displays the nine metadata categories of RML.

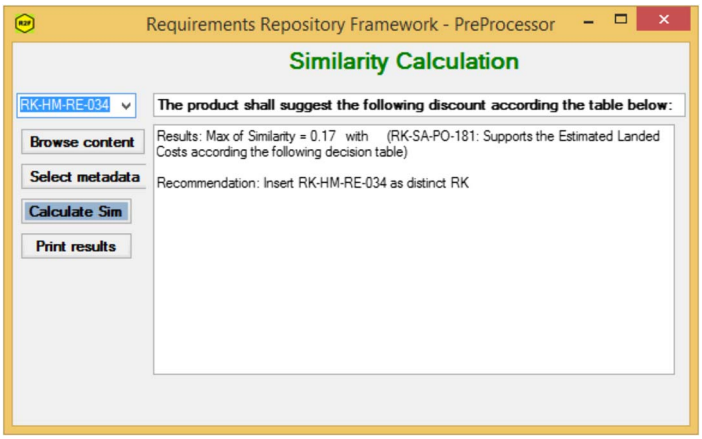


Figure 3 Screenshot of the preprocessor module of Requirements Repository Framework

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

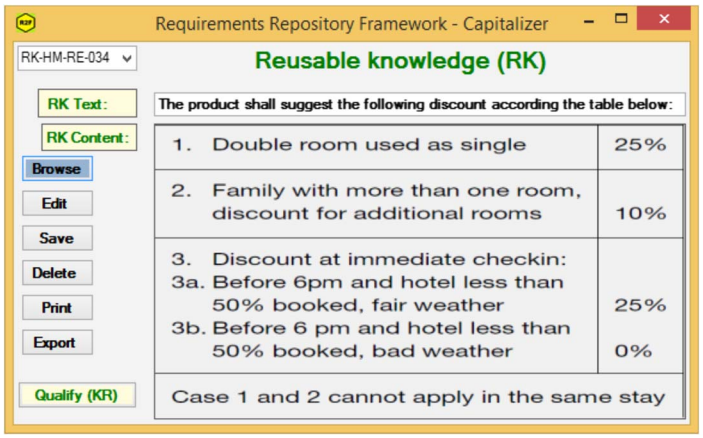


Figure 4 Screenshot of the capitalizer module of Requirements Repository Framework

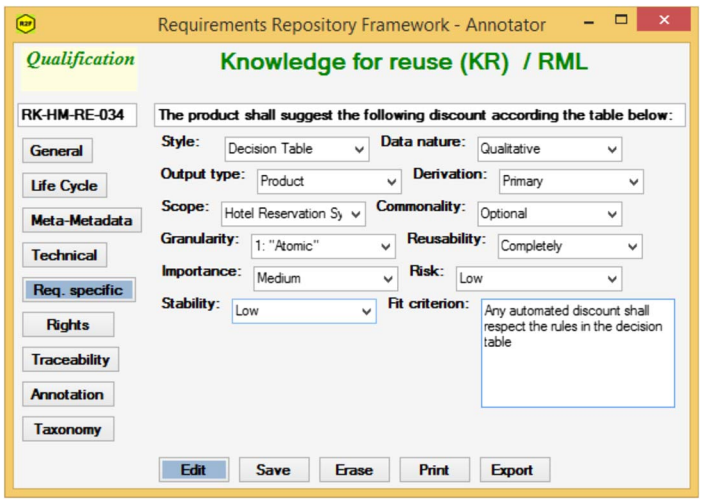


Figure 5 Screenshot of the annotator module of Requirements Repository Framework. RML = Requirements Metadata Language

The central part contains the form used to edit the metadata related to the active category 'Requirements specific'. We remark that all the values have been selected from predefined set of values as explained in the RML base schema (Table 2) except for the value for the fit criterion metadata which has been directly entered by the user. Also the user has a set of gray buttons on the bottom that allow him to edit, save, erase, print, and export the displayed metadata (KR).

5 Experimentation

In order to validate our framework, we opted to conduct a live-user experiment. Therefore, we developed a first prototype of R2F as a Windows application using VB within the .NET visual studio framework.

We started using the R2F prototype locally at the experimental level with our IS240 (Information Systems Analysis and Design) students and our senior project students. Here, we present the results of the experiment with three second-level student sections (composed of 25 students each), that used R2F in their IS240 (Information Systems Analysis and Design) course project; as well as with four Information Systems student groups (composed of three students each), that used R2F in their respective senior projects. The objective of this experiment was to measure the usability of the proposed AFR process. This helped us to examine, through direct observation and survey, the users' feedback with regard to R2F use.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Table 8 Extract of the capitalized reusable knowledges (RKs) for experimenting Requirements Repository Framework

No.	Functional requirement	Family	Catalog
RK-SA-VD-015	Supports user-defined vendor categories	SA	Vendor definition
RK-SA-VD-016	Supports user defined vendor types	SA	Vendor definition
RK-SA-VD-017	Supports user defined vendor payment priority codes	SA	Vendor definition
RK-SA-PO-179	Supports tracking of last price paid for an item	SA	Purchase order
RK-SA-PO-180	Supports the entry of user-defined landed cost labels	SA	Purchase order
RK-SA-PO-181	Supports the estimated landed costs according the following decision table	SA	Purchase order
RK-SA-RP-224	Supports both scheduled and unscheduled blanket purchase orders	SA	Reporting
RK-SA-RP-225	Open requisitions can be released from existing unscheduled blanket orders	SA	Reporting
RK-SA-RP-226	Supports non-stock purchases (no part number set up required)	SA	Reporting
RK-SA-CO-434	Tracks commodity-based purchases	SA	Costing
RK-SA-CO-435	Tracks minority credit purchases	SA	Costing
RK-SA-CO-436	Tracks warranty period by vendor–product combination	SA	Costing
RK-HR-ED-147	Supports entry of a department code for each employee	HR	Employee definition
RK-HR-ED-148	Supports the entry of a pay rate for each employee	HR	Employee definition
RK-HR-ED-149	Supports the tracking skills sets by individual employee	HR	Employee definition
RK-HR-AT-352	Supports the tracking of time and attendance reporting for hourly employees	HR	Attendance
RK-HR-AT-353	Supports the tracking of time and attendance reporting for salary employees	HR	Attendance
RK-HR-AT-354	Supports integration of attendance functionality to the manufacturing schedule	HR	Attendance
RK-HR-PA-579	Supports printing of payroll checks	HR	Payroll
RK-HR-PA-580	Supports the tracking and payment of piece and incentive-based pay rates	HR	Payroll
RK-HR-PA-581	Supports the tracking of federal payroll tax processing	HR	Payroll

SA = sales management systems; HR = human resource systems.

The students used R2F to capitalize some existing RKs in the field of Information Systems. Two families of products were explored: SA (sales management systems) and HR (human resource systems). The projects used for extracting RKs were developed by TGI (Technology Group International), an ERP (Enterprise Resource Planning) vendor that specializes in the development, sale, implementation, and support of their fully integrated ERP system for small and mid-market manufacturing and distribution companies (http://www.tgiltd.com/erp-vendor-tgi). More than 1000 RKs were capitalized for each of the two families of products. Table 8 demonstrates an extract of these RKs, where only three requirements were selected from each catalog within a family of products.

The experimentation was performed in three steps: (1) ask senior project students (12) to use R2F in their senior projects and directly observe them; (2) ask IS240 students (74) to use R2F in their course projects and conduct a survey on them; and (3) analyze the results of the direct feedback from the senior project students on one hand and the indirect feedback from the IS240 students on the other hand. Qualitative data were collected in two different means: direct participant observation and survey.

1. Direct observation: we participated with the four student groups in:
   i. Manually selecting and extracting RKs from the TGI documentation for both families of products (HR) and (SA). At the start of our experiment, because no previous RK that belongs to the two families of products existed in the repository, all the selected RKs were found by the R2F preprocessor module as distinct RKs.
   ii. Automatically capitalizing some of these RKs according to the conceptual requirement reuse model.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Table 9 Requirements Repository Framework questionnaire

No.	Question	Strongly disagree	Disagree	Neither disagree or agree	Agree	Strongly agree
Q1	The similarity calculation process is working well					
Q2	Its easy to upload the content of a new RK from a given folder					
Q3	Its easy to edit an existing RK					
Q4	Its easy build a new RK by aggregation of lower RKs of same type					
Q5	Its easy to assign alternatives to an existing RK					
Q6	Its easy to annotate/qualify an existing RK using RML metadata					
Q7	Its easy to manage existing RKs					
Q8	Its easy to produce statistics related to the content of the repository					

RK = reusable knowledge; RML = Requirements Metadata Language.

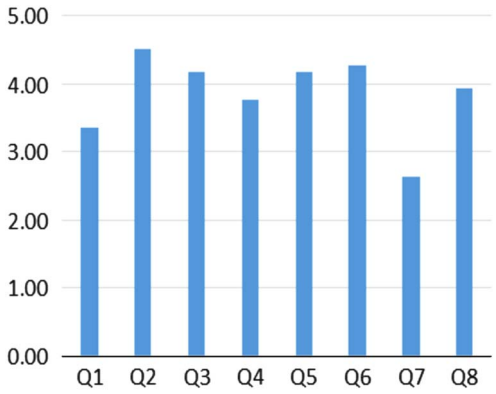


Figure 6 Mean Likert scale values of the eight questions

iii. Qualifying those RKs using some RML metadata (KR).

Finally, in order to check the preprocessor module, a set of alternatives and some very similar RKs were created for some already capitalized RKs.

This initial experiment allowed us to assist the students in learning how to use the prototype. The experiment showed how easy and intuitive is R2F. In addition, the experiment mainly allowed us to improve the graphical interface of R2F, thus making it more intuitive. We note that 66.66% of students (8 out of 12) succeeded in using R2F without any assistance.

2. Survey: we used a Likert scale (Runeson & Hst, 2009) from 1 (strongly disagree) to 5 (strongly agree) to characterize the users' answers.

Table 9 lists the eight asked questions selected for analyzing the following hypothesis: R2F is easy to use. As indicated in Figure 6, the mean values for the eight questions of the questionnaire are between 2.64 and 4.58. Seven mean values are superior to 3 (neutral answer) suggesting R2F users' attitudes to be very positive. Only the mean value (2.64) for question 7 is less than 3. This guides us to improve the manager module.

6 Conclusion

Several of the approaches discussed within the RE community consider as reusable elements many of the different types of requirements that can be assembled, refined, or modified in order to generate new customized requirements. We explored how the development of requirements can be made more

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

productive, and new requirements specifications resolved through reuse of the experience gained from the requirements of previous similar projects.

The main objective of the component-based approach proposed in this paper is to provide a conceptual requirements reuse model. Such model allows the discussion of several requirements reuse-related issues, such as: what is the proper level of reuse based on different parameters? Which types of RKs are potentially more reusable, the coarser, and/or the more incomplete? In addition, we proposed an AFR process that allows requirements engineers to capitalize requirements in the repository from different sources, thus making such requirements reusable in future software projects with the advantage of increasing productivity and quality. Finally, we proposed R2F that supports the previous process. R2F is intended to be mainly used by requirements engineers and managers. It provides guidance and support to requirements engineers in capitalizing RKs according to the AFR process, and helps managers administer the requirements repository.

This framework is currently being evaluated by several developers and researchers within our university. We presented the results of an experiment with three second-level student sections that used R2F in their IS240 course project; as well as with four student groups that used R2F in their respective senior projects. This helped us to examine, through direct observation and survey, the users' feedback with regard to R2F use. Thus far, we have received positive feedback for its effectiveness and usability. One of the key limitations of this research work is that it only covers the AFR process. Therefore, our next step aims to implement the complementary ABR process. As other future work, component reuse can be further improved by investigating other aspects, such as the use of ontologies, to enrich the semantic of components in order to improve their retrieval. Indeed, RK effectiveness can be exploited at best only if RKs can be coupled with both domain and requirements ontologies, such as SWORE (SoftWiki Ontology for Requirements). The main aim is to create a semantic layer on the existing repository that would support the semantic processing of both the AFR and ABR processes. This will foster a component exchange among repositories during the AFR process on one side and a better personalization of the components during the ABR process on the other side.

Acknowledgments

The author would like to thank senior project students who participated in the experimentation of R2F.

References

Akers, D. 2008. Real reuse for requirements. *Methods & Tools* **16**(1), 33–40.

Chernak, Y. 2012. Requirements reuse: the state of the practice. In *Proceedings of IEEE International Conference on Software Science, Technology and Engineering (SWSTE)*, 46–53. https://doi.org/10.1109/SWSTE.2012.12.

Gotel, O. & Finkelstein, A. 1994. An analysis of the requirements traceability problem. In *Proceedings of the 1st International Conference in Requirements Engineering*, 94–101.

Haslhofer, B. & Klas, W. 2010. A survey of techniques for achieving metadata interoperability. *ACM Computing Surveys* **42**(2), 1–42.

Hauksdottir, D., Vermehren, A. & Savolainen, J. 2012. Requirements reuse at Danfoss. In *Proceedings of 20th IEEE International on Requirements Engineering Conference (RE)*, 309–314. https://doi.org/10.1109/RE.2012.6345820.

Husnain, M., Waseem, M. & Ghayyur, S. A. K. 2009. An interrogative review of requirement engineering frameworks. *International Journal of Reviews in Computing* **2**(1), 1–8.

IEEE 1484.12.1-2002 2002. *Draft Standard for Learning Object Metadata*. Technical report, IEEE Learning Technology Standards Committee (LTSC).

Lauesen, S. 2002. *Requirements: Styles and Techniques*. Addison-Wesley.

Menndez-Domnguez, V. H., Zapata, A., Prieto-Mendez, M. E., Romero, C. & Serrano-Guerrero, J. 2011. A similarity-based approach to enhance learning objects management systems. In *Proceedings of 11th International IEEE Conference on Intelligent Systems Design and Applications (ISDA)*, 996–1001.

Mussbacher, G. & Kienzle, J. 2013. A vision for generic concern-oriented requirements reuse re@21. In *Proceedings of 21st IEEE International on Requirements Engineering Conference (RE)*, 238–249. https://doi.org/10.1109/RE.2013.6636724.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press

Myklebust, T., Lyngby, N., Bains, R. & Hanssen, G. K. 2014. CoVeR maintenance and maintainability requirements by using tools. In *Proceedings of Annual Symposium on Reliability and Maintainability (RAMS)*, 1–6. https://doi.org/10.1109/RAMS.2014.6798497.

Niu, N., Savolainen, J., Niu, Z., Jin, M. & Cheng, J. C. 2014. A systems approach to product line requirements reuse. *IEEE Systems Journal* **8**(3), 827–836, https://doi.org/10.1109/JSYST.2013.2260092.

Pacheco, C. L., Garcia, I. A., Calvo-Manzano, J. A. & Arcilla, M. 2015. A proposed model for reuse of requirements in requirements catalog. *Journal of Software: Evolution and Process* **27**, 1–21.

Palomares, C., Quer, C. & Franch, X. 2014. Requirements reuse and patterns: a survey. In *Requirements Engineering: Foundation for Software Quality*, Lecture Notes in Computer Science **8396**, Springer, Cham 301–308.

Palomares, C., Quer, C., Franch, X., Guerlain, C. & Renault, S. 2012. A catalogue of non-technical requirement patterns. In *Proceedings of IEEE Second International Workshop on Requirements Patterns (RePa)*, 1–6. https://doi.org/10.1109/RePa.2012.6359969.

Ramesh, B. & Jarke, M. 2001. Towards reference models for requirements traceability. *IEEE Transactions in Software Engineering* **27**(1), 58–93.

Runeson, P. & Hst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* **14**(2), 131164.

Sommerville, I. 2011. *Software Engineering*. Pearson Education.

Spanoudakis, G. & Zisman, A. 2005. Software traceability: a roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, Chang, S. K. (ed.). World Scientific, 395–428.

Tastekin, S. Y., Erten, Y. M. & Bilgen, S. 2012. Product development time improvement with requirements reuse. In *Proceedings of the 7th International Conference on Software Engineering Advances*, 394–400.

Toval, A., Nicols, J., Moros, B. & Garca, F. 2002. Requirements reuse for improving information systems security: a practitioner's approach. *Springer Requirements Engineering Journal* **6**(4), 205–219.

Toval, A., Moros, B., Nicols, J. & Lasheras, J. 2008. Eight key issues for an effective reuse-based requirements process. *Computer Systems Science and Engineering* **23**(6), 1–20.

Wiegers, K. E. 2003. *Requirements*. Microsoft Press.

https://doi.org/10.1017/S0269888917000030 Published online by Cambridge University Press