Enhancing Requirements Reusability through Semantic Modeling and Data Mining Techniques

Themistoklis Diamantopoulos and Andreas Symeonidis

Electrical and Computer Engineering Dept., Aristotle University of Thessaloniki, Thessaloniki, Greece

ARTICLE HISTORY
Compiled November 2, 2017

ABSTRACT
Enhancing the requirements elicitation process has always been of added value to software engineers, since it expedites the software lifecycle and reduces errors in the conceptualization phase of software products. The challenge posed to the research community is to construct formal models that are capable of storing requirements from multimodal formats (text and UML diagrams) and promote easy requirements reuse, while at the same time being traceable to allow full control of the system design, as well as comprehensible to software engineers and end users. In this work, we present an approach that enhances requirements reuse while capturing the static (functional requirements, use case diagrams) and dynamic (activity diagrams) view of software projects. Our ontology-based approach allows for reasoning over the stored requirements, while the mining methodologies employed detect incomplete or missing software requirements, this way reducing the effort required for requirements elicitation at an early stage of the project lifecycle.

KEYWORDS
Multimodal Requirements Engineering; Web Ontology Language (OWL); Unified Modeling Language (UML); Association Rule Mining; Requirements Reuse

1. Introduction

Contemporary trends in software engineering dictate agile approaches, which include fast feature sprints and short time to market. In this context, requirements elicitation has grown to be a sophisticated phase of software development. According to a recent study (Montequin et al. 2014), inaccurate, incomplete or undefined requirements have been found to be the most common reason of failure for software projects. Furthermore, the need for proper identification of software requirements at an early stage is crucial, since continuous changes to initial requirements can lead to faults (Montequin et al. 2014), while reengineering costs as a result of poorly specified requirements are considerably high (Leffingwell 1997). These issues are quite noticeable in the industry, where current trends involve using complex *Enterprise Information Systems (EISs)* for integrating/mining business processes (Ingvaldsen and Gulla 2012), handling large amounts of data, etc. (Olson and Kesharwani 2009), and thus the stakes of erroneous requirements elicitation are high.

CONTACT Themistoklis Diamantopoulos. Email: thdiaman@issel.ee.auth.gr

Another important concept of contemporary software development is that of *software reuse*. Given the introduction of the open-source software initiatives and the component-based nature of software, developers nowadays rely more and more on reusing components in a rapid prototyping context. The need for reusing existing components has become more eminent than ever, since component reuse can reduce the time and effort spent during all phases of the software development lifecycle, including requirements elicitation, specification extraction and modeling, source code writing, and software maintenance/testing. As a result, there are several approaches towards applying data mining techniques to recommend software components that address the required functionality and are of high quality. Most of these approaches, however, focus on the source code of the components (Thummalapenta and Xie 2007; Sahavechaphan and Claypool 2006; Hummel, Janjic, and Atkinson 2008), on quality information/metrics (Diamantopoulos, Thomopoulos, and Symeonidis 2016), and on information from online repositories (Papamichail, Diamantopoulos, and Symeonidis 2016; Dimaridou et al. 2017). Requirements reuse cannot be easily addressed as requirements are usually expressed in natural language text and UML models, which are often ambiguous or incomprehensible to developers and stakeholders (Mich, Mariangela, and Pierluigi 2004).

The benefits from requirements reuse are evident regardless of the type of software product built and the software development methodology adopted. Software is built based on requirements identified, irrespective of whether they have been collected all at once, iteratively and/or through throw-away prototypes. What is, thus, important is to *design a model capable of storing software requirements* and *develop a methodology that will enable requirements reuse*. This model has to allow for seamless instantiation or even migration of existing requirements, and support reuse regardless of the software engineering methodology applied.

The aforementioned challenges are applicable both to waterfall or iterative development methodologies and to modern component-based or agile development practices. Typical waterfall/iterative scenarios first require accurate and complete requirements identification, and then proceed with specification extraction, software design, development, and testing/quality assurance (Sommerville 2010). Hence, given a sufficient pool of requirements/specifications (Li et al. 2014), mining techniques can be applied to facilitate the requirements elicitation phase of new products, and thus reduce the cost and effort of requirements' modifications at a later stage of the development process. Requirements reuse can further prove advantageous in the context of component-based or agile software development practices, which have lately become popular among different types of companies (Franck 2017). Although requirements in this case may be modeled differently (e.g. as user stories) and may be continuously adapted as part of the development process, having a sufficient pool from internal or even online sources can again offer increased reuse potential. To maximize this potential, one would have to develop a traceable and updatable scheme that shall support storing and indexing requirements for similar projects. Given also current advances in automatically extracting models from requirements and generating source code (Zolotas et al. 2016), reuse may be enabled on different levels, e.g. projects with similar models could be assigned interchangeable requirements.

In an attempt to confront these challenges, research efforts involve storing software requirements in formal models (Kaindl et al. 2007; Smialek 2012; Wynne and Hellesoy 2012; Mylopoulos, Castro, and Kolp 2000), that allow requirements engineers to have full control of the system design and detect errors at an early stage of the project lifecycle, which is usually much more cost-effective than finding and fixing them at a later stage (Boehm and Basili 2001). Furthermore, storing and/or indexing functional requirements as formal representations allows for easy retrieval, either for immediate reuse or for receiving useful recommendations for new (similar) projects. Research efforts in this area are numerous, spanning from functional requirements and stakeholders modeling to UML models mining for recommendations.

Concerning functional requirements elicitation, most approaches involve constructing and instantiating models using domain-specific vocabularies. The models can be subsequently used to perform validation (Kumar, Ajmeri, and Ghaisas 2010; Ghaisas and Ajmeri 2013) or to recommend new requirements (Chen et al. 2005; Alves et al. 2008) by identifying missing entities and relations (Frakes, Prieto-Diaz, and Fox 1998). Further lines of research include identifying dependencies among requirements and distributing them according to their relevance to stakeholders (Felfernig et al. 2010), as well as recovering traceability links among requirements and software artefacts and utilizing them to identify potentially changed requirements (Maalej and Thurimella 2009). Though effective, most of these efforts are confined to specific domains, mainly due to the lack of annotated requirements models for multiple domains.

UML model mining techniques suffer more or less from the same issues, as most semantics-enabled methods (Gomes, Gandola, and Cordeiro 2007; Robles et al. 2012) are based on the existence of domain-specific information. Finally, domain-agnostic techniques (Alspaugh et al. 1999; Blok and Cybulski 1998; Kelter, Wehren, and Niere 2005) can be greatly facilitated by appropriate data handling of requirements models, as their shortcomings usually lie on incorporating structure or flow information, e.g. for use case or activity diagrams.

In this work, we design a model capable of enhancing requirements reuse. Specifically, we facilitate storing requirements from multimodal formats, including semi-structured text and UML use case and activity diagrams. Our methodology effectively models the static and dynamic views of software projects and employs heuristics and NLP techniques to instantiate software ontologies, that can be subsequently employed for reasoning over the models, for mining specifications to provide useful recommendations, and for maintaining and updating the original software requirements (as our methodology is fully traceable). Having successfully annotated requirements, we then employ data mining techniques to extract useful associations. In specific, association rule mining techniques and heuristics are used to determine whether the requirements of a software project are complete and recommend new requirements, while matching techniques are used on UML models in order to find similar diagrams and thus allow the requirements engineer to improve the existing functionality and the data flow/business flow of his/her project.

2. Related Work

2.1. *Modeling and Mining Functional Requirements*

Early research efforts in recommendation systems for requirements elicitation were focused on domain analysis, and thus used linguistics (vocabularies, lexicons) to determine project domains and identify missing entities and relations at requirements' level. An example of such a tool is DARE (Frakes, Prieto-Diaz, and Fox 1998), which utilizes multiple sources of information, including the requirements, the architecture and the source code of a project. The tool extracts entities and relations from several projects and then uses clustering to identify common entities and thus recommend similar artefacts and architectural schemata for each project. Kumar, Ajmeri, and Ghaisas (2010) make use of ontologies in order to store requirements and project domains and extract of software specifications. Ghaisas and Ajmeri (2013) further develop a Knowledge-Assisted Ontology-Based Requirements Evolution (K-RE) repository in order to facilitate software requirements elicitation and resolve any conflicts between change requests.

There are also several approaches that aspires to identify the features of a system using its requirements, and recommend new ones. Chen et al. (2005) used requirements from several projects and constructed relationship graphs among requirements. The authors then employed clustering techniques to extract domain information. Their system can identify features, such as e.g. writing to a file, that can be used to create a feature model of projects that belong to the same domain. Similar work was performed by Alves et al. (2008), who employed the vector space model to construct a domain feature model, and used latent semantic analysis to find similar requirements by clustering them into domains. Dumitru et al. (2011) employed association rule mining to analyze requirements and subsequently cluster them into feature groups. These groups can be used to find projects that are similar or to recommend a new feature for a project.

Finally, there are also approaches that explore the relation between requirements and stakeholders (Lim and Finkelstein 2012; Castro-Herrera et al. 2008; Mobasher and Cleland-Huang 2011); the related systems are given as input ratings for the requirements of individual stakeholders, and use collaborative filtering (Konstan et al. 1997) to provide recommendations and prioritize requirements according to stakeholder preferences. However, these approaches, and any approaches focusing on non-functional requirements (Romero-Mariona, Ziv, and Richardson 2008), deviate from the scope of this work.

Based on the above discussion, one easily notices that most approaches are domain-centered (Frakes, Prieto-Diaz, and Fox 1998; Kumar, Ajmeri, and Ghaisas 2010; Ghaisas and Ajmeri 2013; Chen et al. 2005; Alves et al. 2008), which is actually expected as domain-specific information can improve the understanding of the software project under analysis. On the other hand, as noted by Dumitru et al. (2011), these domain analysis techniques are often not applicable due to the lack of annotated requirements for a specific domain. Although feature-based techniques (Dumitru et al. 2011; Romero-Mariona, Ziv, and Richardson 2008) do not face the same issues, their scope is high-level as they are not applied on fine-grained requirements.

In this work, we construct a system that focuses on functional requirements and provides low-level recommendations, maintaining a semantic domain-agnostic outlook. To do so, we use a state-of-the-art semantic role labeler (Roth et al. 2014; Diamantopoulos et al. 2017) to extract actors, actions, objects, and properties from functional requirements, and design a set of ontologies to effectively index them. The use of ontologies for storing and validating requirements is common (Castañeda et al. 2010; Siegemund et al. 2011; Happel and Seedorf 2006; Dermeval et al. 2015), however we chose to design a new schema that is flexible and supports indexing both the static and the dynamic view of software projects, allows reasoning over the data for validation, as well as mining for recommendation purposes.

2.2. *Modeling and Mining UML models*

Early efforts for UML model mining employed Information Retrieval techniques and were directed towards use case scenarios (Blok and Cybulski 1998; Alspaugh et al. 1999). Typically, such scenarios can be defined as sets of *events* triggered by *actions* of *actors*, and they also include *authors* and *goals*. Thus, research on scenario reuse is mainly directed towards representing event flows and comparing them to find similar use cases (Blok and Cybulski 1998). Other approaches involve representing UML diagrams to graphs and detecting similar graphs (diagrams) using graph matching. In such graph representations, the vertices comprise the object elements of UML use case, class, and sequence diagrams, while the edges denote the associations among these elements. The employed graph matching techniques can either be exact (Woo and Robinson 2002; Robinson and Woo 2004; Bildhauer, Horn, and Ebert 2009) or inexact (Salami and Ahmed 2013). Similar work by Park and Bae (2011) involves using Message Object Order Graphs (MOOGs) to store sequence diagrams, where the nodes and the edges represent the messages and the flow among them.

Despite the effectiveness of graph based methods under certain scenarios (e.g. structured models), applying them to UML diagrams lacks semantics. As noted by Kelter, Wehren, and Niere (2005), UML model mining approaches should focus on creating a semantic model, rather than arbitrarily applying similarity metrics. Thus, UML diagrams (and XML structures) are also commonly represented as ordered trees (Kelter, Wehren, and Niere 2005; Chawathe et al. 1996; Wang, DeWitt, and Cai 2003). The main purpose of these efforts is to first design a data model that captures the structure of UML diagrams, and then apply of ordered tree similarity metrics. These approaches are effective for static diagram types (use case, class), however they cannot efficiently represent data flows or action flows, thus they do not adhere to dynamic models (activity, sequence). Furthermore, they usually employ string difference techniques, thus they are not applicable to scenarios with multiple sources of diagrams. To confront this challenge, researchers have attempted to incorporate semantics through the use of domain-specific ontologies (Gomes, Gandola, and Cordeiro 2007; Robles et al. 2012; Bonilla-Morales, Crespo, and Clunie 2012). Matching diagram elements (actors, use cases, etc.) according to their semantic distance has indeed proven effective as long as domain knowledge is available; most of the time, however, domain specific information is limited.

In this work, we focus on the requirements elicitation phase and propose two mining methodologies, one for use case diagrams and one for activity diagrams. For use case diagram matching, we handle use cases as discrete nodes and employ semantic similarity metrics, thus combining the advantages of graph based and Information Retrieval techniques and avoiding their potential limitations. For activity diagram matching, we construct a model that represents activity diagrams as sequences of action flows. This way, the dynamic nature of the diagrams is modeled effectively, without using graph based methods that could result to over-engineering. Our algorithm is similar to ordered tree methods (Kelter, Wehren, and Niere 2005; Chawathe et al. 1996; Wang, DeWitt, and Cai 2003), which are actually a golden mean unstructured (e.g. Information Retrieval) and heavily structured (e.g. graph based) methods. Our matching methodologies further use a semantic scheme for comparing strings, which is however domain-agnostic and thus can be used in scenarios with diagrams originating from various projects.

3. Modeling and Mining Software Requirements

3.1. *System Overview*

Our approach comprises two parts: the Reqs2Specs module and the requirements mining methodology built within the context of the EU-funded project S-CASE (Scaffolding Scalable Software Services)[1]. S-CASE facilitates rapid application prototyping based on

[1]`http://s-case.github.io/`

software requirements and system models provided in multimodal formats. The conceptual architecture of our requirements elicitation approach with respect to the project is shown in Figure 1.

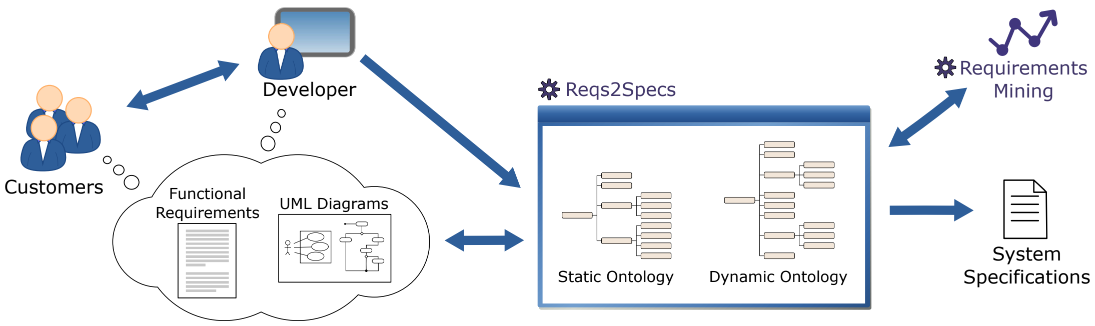


Figure 1. Overview of the Conceptual Architecture of our System.

In a typical scenario, the developer communicates with the customers in order to capture user requirements, document them into functional requirements and draw UML diagrams that describe the envisioned system. Using our approach, these artefacts are syntactically and semantically annotated and stored in two ontologies, the *static ontology* and the *dynamic ontology*, which practically store the static elements and the dynamic elements of the system. Any mining methodologies are performed at this stage using the requirements mining module, while any changes to these representations are propagated back to the functional requirements and the UML diagrams, since our methodology is fully traceable. As a result, the stakeholders are always provided with a clear requirements view of the system, including natural language text as well as standardized (UML-compliant) models.

Having finalized the requirements elicitation process, the Reqs2Specs module parses the ontologies and extracts detailed system specifications, which are suitable for use either directly by developers, or even as input in an automated source code generation engine (as performed in S-CASE (Zolotas et al. 2016)). The following subsections outline the proposed ontologies and the mining process for functional requirements and UML diagrams. The full specification of the ontologies can be found at `http://s-case.github.io/publications/eis2017/`.

3.2. *Functional Requirements Modeling and Mining*

3.2.1. Static Ontology

The static ontology models functional requirements and use case diagrams, and thus revolves around the concept of an acting unit (e.g. user or admin) performing some action(s) on some object(s) (Roth et al. 2014). The ontology class hierarchy is shown in Figure 2.

Any `Concept` of the ontology is further classified into `Project`, `Requirement`, `ThingType`, and `OperationType`. Instances of `ThingType` are acting units (`Actor`) and units acted upon (`Object` and `Property`), while instances of `OperationType` refer to all types of actions, including possession (`Ownership`), passive transformation (`Emergence`), actor status (`State`), and transitive actions that are applied on an object (`Action`). The aforementioned ontology (sub)classes are related using the *properties* that are shown in Figure 3.

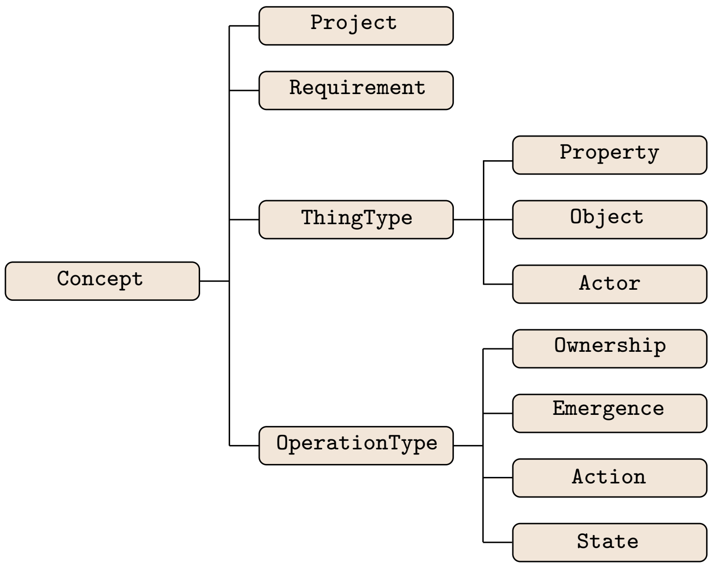


Figure 2. Static Ontology of Software Projects.

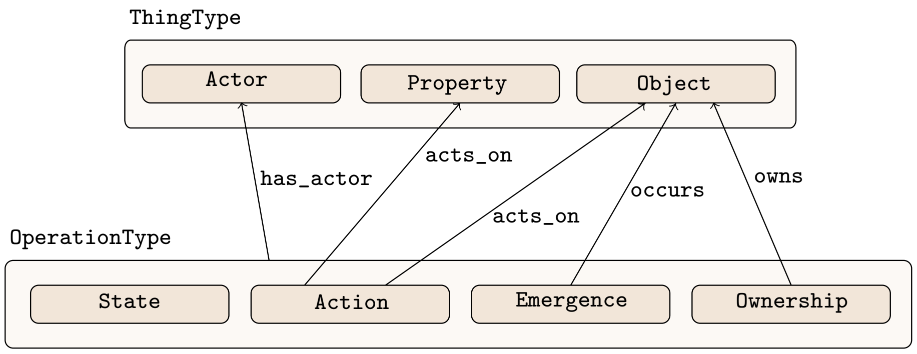


Figure 3. Properties of the Static Ontology.

A `Project` can have many instances of `Requirement` and each `Requirement` has `ThingType` and `OperationType` instances, enabling the tracing and the reengineering of the original requirements. Additionally, `OperationType` connects to `Actor` via the `has_actor` property (and the inverse `is_actor_of` property), and connects to `Object` or `Property` via `acts_on` (for `Action`), `occurs` (for `Emergence`), and `owns` (for `Ownership`). Non-transitive `State` operations do not connect to any `Object` or `Property`.

Populating the ontology requires annotating the concepts of functional requirements (and use case models) that usually follow the Subject-Verb-Object motif. To do so, we employ an NLP parser (thoroughly described in (Diamantopoulos et al. 2017)), which operates in two stages: first it performs *syntactic analysis* to identify the grammatical category of each word and the grammatical relations between them, and then it performs *semantic analysis* to extract semantic features for the terms (e.g. part-of-speech, relation to parent lemma, etc.) and further classify the terms to the relevant ontology concepts.

3.2.2. Functional Requirements Mining

To design and evaluate our requirements mining model, we use a diverse dataset of 30 projects, which includes student projects, industrial prototypes, as well as RESTful prototype applications from S-CASE. Our dataset can be found at `http://s-case.github.io/publications/eis2017/`. In total, these projects have 514 functional requirements, and amount to 7234 entities and 6626 relations among them. An example annotated requirement is shown in Figure 4.

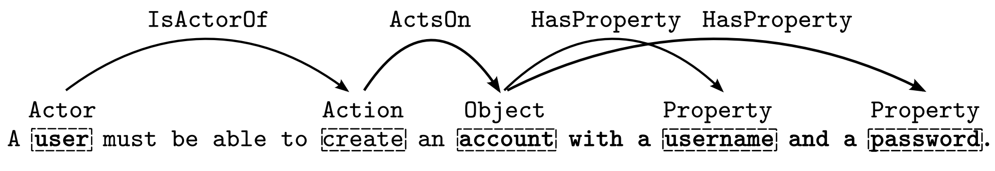

IsActorOf
ActsOn
HasProperty
HasProperty
Actor
Action
Object
Property
Property
A user must be able to create an account with a username and a password.

Figure 4. Example annotated requirement.

Our system can relate domain-agnostic terms between different projects. At first, the entities and relations for each requirement (and thus for each project) are extracted. For instance, for the requirement of Figure 4, we may extract the terms `user`, `create`, `account`, `username` and `password`, as well as the corresponding relations among them: `user_IsActorOf_create`, `create_ActsOn_account`, `account_HasProperty_username`, and `account_HasProperty_password`. Relating two requirements (or two projects in general) requires semantically relating their terms. For instance, if we have another project where each user also has an account, yet the chosen term is `profile`, then these two terms have to be marked as semantically similar.

In order to mark semantically similar terms, we require an index of words and a similarity measure. We use WordNet (Miller 1995) as our index, and employ the MIT Java Wordnet Interface (Finlayson 2014) and the Java Wordnet::Similarity library[2] to interface with it. There are several methods for computing the similarity between two terms (Pedersen, Patwardhan, and Michelizzi 2004). However, most of them either do not employ semantics or they are not correlated to human judgment. As a result, we use the information-content measure introduced by Lin (1998), which and conforms with human judgment more often than other metrics. Thus we define the similarity between two terms (i.e. WordNet classes) C_1 and C_2 as follows:

$$sim(C_1, C_2) = \frac{2 \cdot \log P(C_0)}{\log P(C_1) + \log P(C_2)} \tag{1}$$

where C_0 is the most *specific* class that contains both C_1 and C_2. E.g. the most specific class that describes terms For `account` and `profile` is the class `record`, as shown also in the relevant WordNet fragment of Figure 5.

Upon having found C_0, we compute the similarity between the two terms using equation (1), which requires the *information content* for each of the three WordNet classes. The information content of a WorNet class is defined as the log probability that a corpus term belongs in that class[3]. Thus, for instance, `record`, `account`, and `profile` have information content values equal 7.874, 7.874, and 11.766 respectively, and the similarity between `account` and `profile` is $2 \cdot 7.874/(7.874 + 11.766) = 0.802$. Finally, two terms are assumed to be semantically similar if their similarity value (as determined by equation (1)) is higher than a threshold t. We set this threshold to 0.5, and thus we now have 1512 terms for the 30 projects, out of which 1162 are are distinct.

Hence, given a dataset with one set of *items* per software project, we can extract

[2]`http://users.sussex.ac.uk/~drh21/`

[3]We used the precomputed information content data of the Perl WordNet similarity library (Pedersen, Patwardhan, and Michelizzi 2004), available at `http://www.d.umn.edu/~tpederse/`.

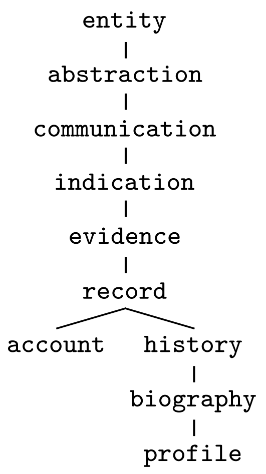


Figure 5. Example WordNet fragment where `record` is the most specific class of `account` and `profile`.

useful *association rules* using association rule mining (Agrawal, Imieliński, and Swami 1993). Let $P = \{p_1, p_2, \ldots, p_m\}$ be the set of m software projects and $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all n items. *Itemsets* are defined as subsets of I. Given an itemset X, its support is defined as the number of projects in which all of its items appear in:

$$\sigma(X) = |\{p_i | X \subset p_i, p_i \in P\}| \tag{2}$$

Association rules are expressed in the form $X \rightarrow Y$, where X and Y are disjoint itemsets. An example rule that can be extracted from the items of the requirement shown in Figure 4 is {`account_HasProperty_username`} → {`account_HasProperty_password`}.

The two metrics used to determine the strength of a rule are its *support* and its *confidence*. Given an association rule $X \rightarrow Y$, its support is the number of projects for which the rule is applicable, and it is given as:

$$\sigma(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{|P|} \tag{3}$$

The confidence of the rule indicates how frequently items in Y appear in X, and it is given as:

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \tag{4}$$

We use the Apriori association rule mining algorithm (Agrawal and Srikant 1994) in order to extract association rules with support and confidence above certain thresholds. For our dataset, we set the minimum support to 0.1, so that any rule has to be contained in at least 10% of the projects. We also set the minimum confidence to 0.5, so that the extracted rules are confirmed at least half of the time that their antecedents are found. The execution of Apriori resulted in 1372 association rules, a fragment of which is shown in Table 1. Several of these rules are expected. For example, rule 2 indicates that in order for a user to login, the system must first validate the account. Also, rule 5 implies that logout functionality should co-occur in a system with login functionality.

Upon having extracted the rules, we use them to recommend new requirements for software projects. At first, we define a new set of items p for the newly added software

Table 1. Sample Association Rules Extracted by the Dataset.

No	Association Rule	σ	c
1	$provide\_ActsOn\_product \rightarrow system\_IsActorOf\_provide$	0.167	1.0
2	$system\_IsActorOf\_validate \rightarrow user\_IsActorOf\_login$	0.1	1.0
3	$user\_IsActorOf\_buy \rightarrow system\_IsActorOf\_provide$	0.1	1.0
4	$administrator\_IsActorOf\_add \rightarrow administrator\_IsActorOf\_delete$	0.167	0.833
5	$user\_IsActorOf\_logout \rightarrow user\_IsActorOf\_login$	0.167	0.833
6	$user\_IsActorOf\_add \rightarrow user\_IsActorOf\_delete$	0.133	0.8
7	$user\_IsActorOf\_access \rightarrow user\_IsActorOf\_view$	0.1	0.75
8	$edit\_ActsOn\_product \rightarrow add\_ActsOn\_product$	0.1	0.75
9	$administrator\_IsActorOf\_delete \rightarrow administrator\_IsActorOf\_add$	0.167	0.714
10	$user\_HasProperty\_contact \rightarrow user\_IsActorOf\_search$	0.133	0.5

σ: Support, c: Confidence

project. Given this set of items and the rules, we extract the *activated* rules R. A rule $X \rightarrow Y$ is activated for the project with set of items p if all items present in X are also contained in p (i.e. $X \subset p$). The set of activated rules R is then *flattened* by creating a new rule for each combination of antecedents and consequents of the original rule, so that the new rules contain single items as antecedents and consequents. Given, e.g., the rule $X \rightarrow Y$ where the itemsets X and Y contain the items $\{i_1, i_2, i_3\}$ and $\{i_4, i_5\}$ respectively, the new flattened rules are $i_1 \rightarrow i_4$, $i_1 \rightarrow i_5$, $i_2 \rightarrow i_4$, $i_2 \rightarrow i_5$, $i_3 \rightarrow i_4$, and $i_3 \rightarrow i_5$. We also propagate the support and the confidence of the original rules to these new flattened rules, so that they are used as importance criteria. Finally, given the set of items of a project p and the flattened activated rules for this project, our system provides recommendations of new requirements using the heuristics of Table 2.

Table 2. Activated rule heuristics for a software project.

Antecedent	Consequent	Conditions	Result
$[Actor1, Action1]$	$[Actor2, Action2]$	$Actor2 \in p$	$[Actor2, Action2, Object]$, $\forall Object \in p :$ $[Actor1, Action1, Object]$
$[Action1, Object1]$	$[Actor2, Action2]$	$Actor2 \in p$	$[Actor2, Action2, Object1]$
$[Actor1, Action1]$	$[Action2, Object2]$	$Object2 \in p$	$[Actor1, Action2, Object2]$
$[Action1, Object1]$	$[Action2, Object2]$	$Object2 \in p$	$[Actor, Action2, Object2]$, $\forall Actor \in p :$ $[Actor, Action1, Object1]$
* (except for the above)	$[Action2, Object2]$	$Object2 \in p$	$[Actor, Action2, Object2]$, $\forall Actor, Action \in p :$ $[Actor, Action, Object2]$
*	$[Any2, Property2]$	$Any2 \in p$	$[Any2, Property2]$

Concerning the heuristics for consequent $[Actor2, Action2]$, which corresponds to an `Actor2_IsActorOf_Action2` item, the recommended requirement includes the actor and the action of the consequent as well as an *Object* that is determined by the antecedent. Given, e.g., an antecedent $[create, bookmark]$ and a consequent $[user, edit]$, the new recommended requirement will be $[user, edit, bookmark]$. Concerning the heuristics for consequent [$Action2, Object2$], which corresponds to an `Action2_ActsOn_Object2` item, the recommended requirement includes the action and the object of the consequent as well as the actor that is determined by the antecedent. Given, e.g., an antecedent [$user, profile$] and a consequent [$create, profile$], the new recommended requirement will be [$user, create, profile$]. Finally, any rule with a `HasProperty` consequent (and any antecedent) leads to new recommended requirements of the form [$Any, Property$]. An example requirement would be [$user, profile$]. Using the static ontology, we are also able to reconstruct requirements, in the formats 'The *Actor* must be able to *Action Object*.' and 'The *Any* must have *Property*.'.

3.3. *UML Diagrams Modeling and Mining*

3.3.1. Dynamic Ontology

The dynamic ontology models dynamic representations found in activity diagrams, and thus represents actions as ontology concepts and flows between them using ontology properties. The class hierarchy of the ontology is shown in Figure 6. Apart from the `Project` and `ActivityDiagram` classes, any `Concept` instances of the ontology include activities (`AnyActivity`), conditions (`Condition`), and transitions (`Transition`), i.e. typically most elements that are present in activity diagrams. Activities are further distinguished into initial diagram states (`InitialActivity`), final diagram states (`FinalActivity`), and all other activities of the model ((`Activity`)), while classes `Actor`, `Action`, and `Object` are used to store the main elements of an activity (e.g. an activity 'create username' would split into action 'create' and object 'username').

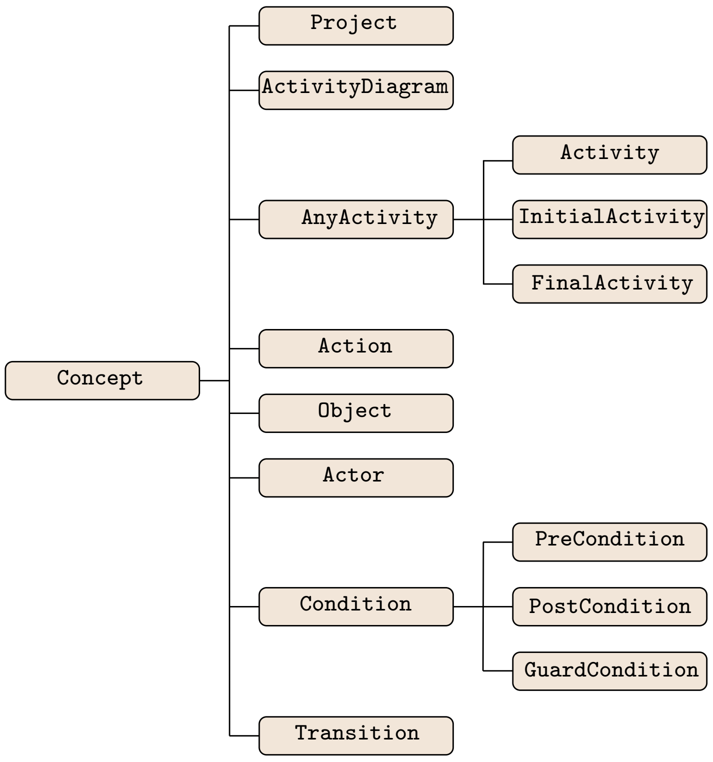


Figure 6. Dynamic Ontology of Software Projects.

`Transition` describes the flow from one instance of `AnyActivity` to the next instance of AnyActivity, and thus connects to these instances via the properties has_source and has_target respectively, as shown in Figure 7. Additionally, each Transition may or may not have a GuardCondition allowing the execution of the target activity given with the corresponding answer (e.g. 'Is the username unique? Yes'). Instances of GuardCondition also have their opposites (e.g. 'Is the username unique? No'), connected via bidirectional property is_opposite_of. Finally, diagrams can also have conditions that have to be met before (PreCondition) or after (PostCondition) the execution of their activity flow. As in the static ontology, all properties also have their corresponding inverse properties (e.g. the inverse of has_source is is_source_of).

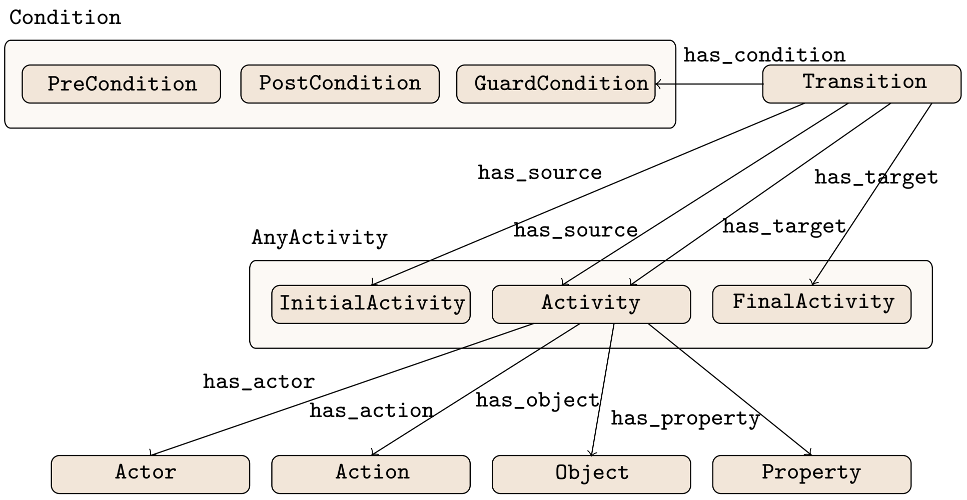


Figure 7. Properties of the Dynamic Ontology.

3.3.2. UML Models Mining

To design our UML diagrams mining model, we use a dataset of use case and activity diagrams, which is available at http://s-case.github.io/publications/eis2017/ (more on the dataset in subsection 4.2) and includes the diagrams of project Restmarks, a demo social service that allows storing bookmarks online, adding tags and sharing them with other users (more on Restmarks in subsection 4.1). An example use case diagram from Restmarks that contains 10 use cases and 3 actors is shown in Figure 8.

As use cases refer to static information, they are stored in the static ontology with a model that is mostly flat. Specifically, the model comprises the actors and the use cases of the diagram. For example, the diagram of Figure 8 has a model that consists of two sets: the set of actors {*User*, *Registered User*, *Guest User*} and the set of use cases {*Add Bookmark*, *Update Bookmark*, *Update Account*, *Show Bookmark*, *Search by Tag*, *Add Tag*, *Login to Account*, *Delete Bookmark*}. Given two diagrams D_1 and D_2, our matching scheme involves two sets for each diagram: one set for the actors A_1 and A_2 respectively, and one set for the use cases UC_1 and UC_2 respectively. The similarity between the diagrams is computed by the following equation:

$$s(D_1, D_2) = \alpha \cdot s(A_1, A_2) + (1 - \alpha) \cdot s(UC_1, UC_2) \tag{5}$$

where s denotes the similarity between two sets (either of actors or of use cases) and α denotes the importance of the similarity of actors for the diagrams. α was set to the

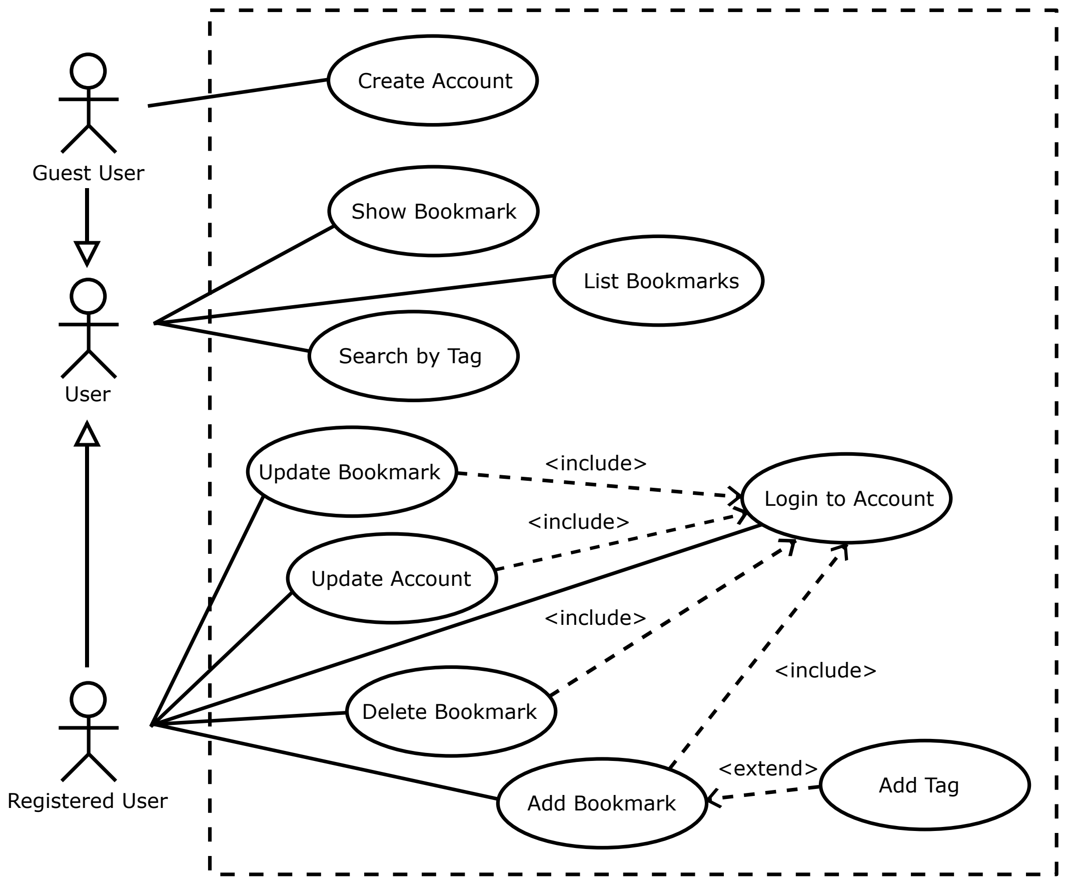


Figure 8. Example use case diagram for project Restmarks.

proportion of the number of actors divided by the number of use cases of the queried diagram. Given, e.g., a diagram with 3 actors and 10 use cases, α is set to 0.3.

The similarity between two sets, either actors or use cases, is given by the combination between all the matched elements with the highest score. Given, e.g., two sets $\{user$, $administrator$, $guest\}$ and $\{administrator$, $user\}$, the best possible combination is $\{(user, user)$, $(administrator, administrator)$, $(guest, null)\}$, and the matching would return a score of $2/3 = 0.66$. We employ the semantic measure of the previous subsection to provide a similarity score between strings. Given two strings S_1 and S_2, we first split them into tokens, i.e. $tokens(S_1) = \{t_1, t_2\}$, $tokens(S_2) = \{t_3, t_4\}$, and then determine the combination of tokens with the maximum token similarity scores. The final similarity score between the strings is determined by averaging over all tokens. For example, given the strings 'Get bookmark' and 'Retrieve bookmarks', the best combination is ('get', 'retrieve') and ('bookmark', 'bookmarks'). Since the semantic similarity between 'get' and 'retrieve' is 0.677, and the similarity of 'bookmark' with 'bookmarks' is 1.0, the similarity between the strings is $(0.677 + 1)/2 = 0.8385$.

Given, for example, the diagram of Figure 8 and the diagram of Figure 9, the matching between the diagram elements is shown in Table 3, while the final score (using equation (5)) is 0.457.

Concerning recommendations, the engineer of the second diagram could consider adding a guest user. Furthermore, he/she could consider adding use cases for listing or updating bookmarks, adding tags to bookmarks, or updating account data.

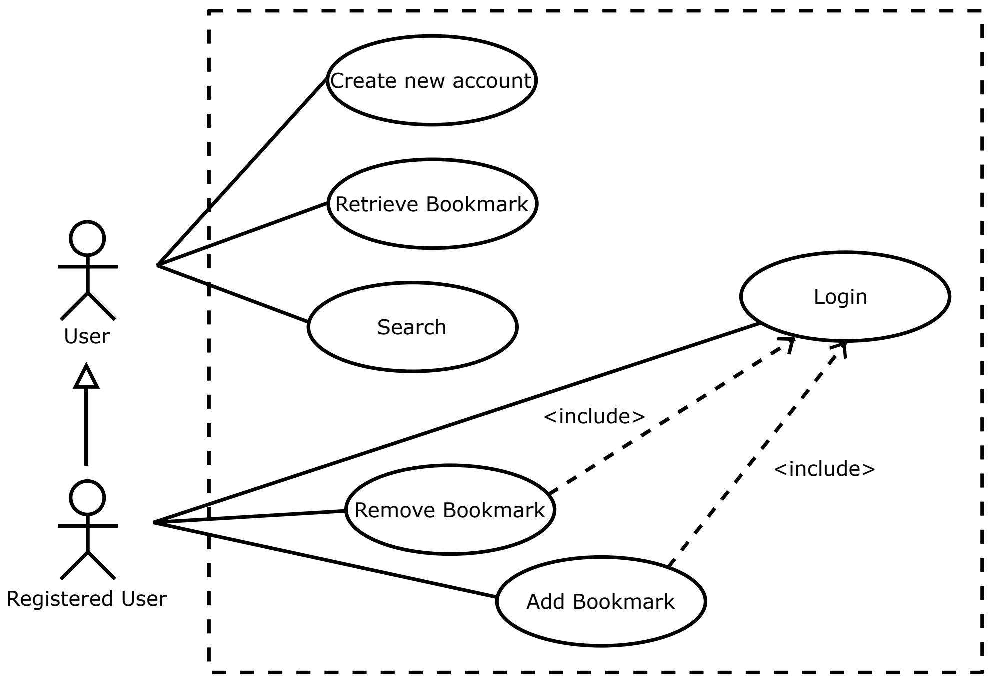


Figure 9. Example use case diagram for matching with the one of Figure 8.

Table 3. Matching between the diagrams of Figures 8 and 9.

Diagram 1	Diagram 2	Score
User	User	1.00
Registered User	Registered User	1.00
Guest User	null	0.00
Delete Bookmark	Remove Bookmark	0.86
Show Bookmark	Retrieve Bookmark	0.50
Add Bookmark	Add Bookmark	1.00
Create Account	Create new account	0.66
Search by Tag	Search	0.33
Login to Account	Login	0.33
List Bookmarks	null	0.00
Update Bookmark	null	0.00
Update Account	null	0.00
Add Tag	null	0.00

Concerning activity diagrams, we require a representation that would view the diagram as a flow model. An example diagram of Restmarks is shown in Figure 10.

Activity diagrams consist mostly of sequences of activities and possible conditions. Concerning conditions (and forks/joins), we split the flow of the diagram. Hence an activity diagram is actually treated as a set of sequences, each of which involves the activities required to traverse the diagram from its start node to its end node. For instance, the diagram of Figure 10 spawns a sequence *StartNode* > *Logged In*? > *Login to account* > *Provide bookmark URL* > *Create Bookmark* > *Add tag* > *User wants to add tag*? > *EndNode*. Upon having parsed two diagrams and having extracted one set of sequences per diagram, we compare the two sets. In this case and in contrast with use case diagram matching, we set a threshold t_{ACT} for the semantic

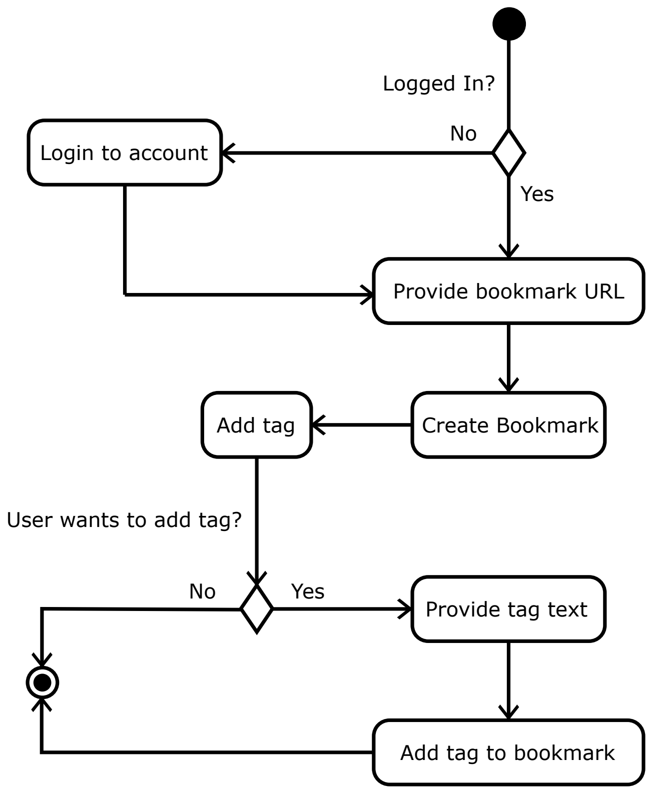


Figure 10. Example activity diagram for project Restmarks.

string similarity metric. Thus, two strings are considered similar if their similarity score is larger than this threshold. We set t_{ACT} to 0.5.

We use the *Longest Common Subsequence (LCS)* (Cormen et al. 2009) in order to determine the similarity score between two sequences. The LCS of two sequences X and Y is defined as the longest subsequence of common (yet not consecutive) elements between them. Given, e.g., the sequences $[a, b, d, e, g]$ and $[a, b, e, h]$, their LCS is $[a, b, e]$. Finally, the similarity score between the sequences is defined as:

$$sim(X, Y) = 2 \cdot \frac{|LCS(X, Y)|}{|X| + |Y|} \tag{6}$$

The similarity score is normalized in $[0, 1]$. Given that we have two sets of sequences (one for each of the two diagrams), their similarity is given by the best possible combination between the sequences, i.e. the combination that results in the highest score. Given, e.g., two sets $\{[a, b, e], [a, b, d, e], [a, b, c, e]\}$ and $\{[a, b, e], [a, c, e]\}$, the combination with the highest score is $\{([a, b, e], [a, b, e]), ([a, b, c, e], [a, c, e]), ([a, b, d, e], null)\}$. Finally, the similarity score between the diagrams is the average of their LCS scores.

For example, let us consider matching the diagrams of Figure 10 and Figure 11. Our system returns the matching between the sequences of the diagrams, shown in Table 4, while the total score, which is computed as the mean of these scores, is $(0.833 + 0.714 + 0 + 0)/4 = 0.387$.

The matching process between the sequences indicates that the engineer of the second diagram could add a new flow that would include the option to add a tag to the newly created bookmark.

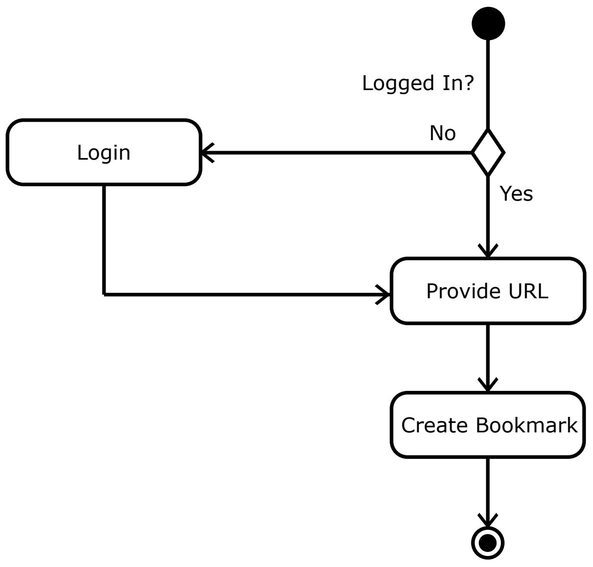


Figure 11. Example activity diagram for matching with the one of Figure 10.

Table 4. Matching between the diagrams of Figures 8 and 9.

Diagram 1	Diagram 2	Score
StartNode > Logged In? > Provide bookmark URL > Create Bookmark > Add tag > User wants to add tag? > EndNode	StartNode > Logged In? > Provide URL > Create Bookmark > EndNode	0.833
StartNode > Logged In? > Login to account > Provide bookmark URL > Create Bookmark > Add tag > User wants to add tag? > EndNode	StartNode > Logged In? > Login > Provide URL > Create Bookmark > EndNode	0.714
StartNode > Logged In? > Provide bookmark URL > Create Bookmark > Add tag > User wants to add tag? > Provide tag text > Add tag to bookmark > EndNode	null	0.000
StartNode > Logged In? > Login to account > Provide bookmark URL > Create Bookmark > Add tag > User wants to add tag? > Provide tag text > Add tag to bookmark > EndNode	null	0.000

4. Evaluation

4.1. *Functional Requirements Mining*

In order to demonstrate the validity of our approach we have performed requirements mining and provide recommendations for a small-scale software project. Project Restmarks is a demo social service for bookmarks created in the context of S-CASE to serve as a scenario for the S-CASE workflow. The users of Restmarks can store online their bookmarks, share them with the Restmarks community and search for bookmarks using tags. The requirements of the service are shown in Figure 12. To apply our methodology, we created the association rules using the remaining 29 projects, and isolated the rules that are activated by the annotated requirements of Restmarks, including their corresponding support and confidence values.

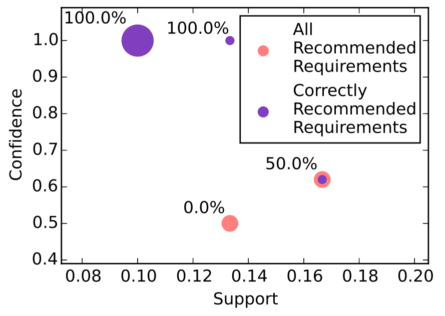
A user must be able to create a user account by providing a username and a password.
A user must be able to login to his/her account by providing his username and password.
A user that is logged in to his/her account must be able to update his/her password.
A logged in user must be able to add a new bookmark to his/her account.
A logged in user must be able to retrieve any bookmark from his/her account.
A logged in user must be able to delete any bookmark from his/her account.
A logged in user must be able to update any bookmark from his/her account.
A logged in user must be able to mark his/her bookmarks as public or private.
A logged in user must be able to add tags to his/her bookmarks.
Any user must be able to retrieve the public bookmarks of any RESTMARKS's community user.
Any user must be able to search by tag the public bookmarks of a specific RESTMARKS's user.
Any user must be able to search by tag the public bookmarks of all RESTMARKS users.
A logged in user must be able to search by tag his/her private bookmarks as well.

(a) Functional requirements of Restmarks.

+ The user must be able to edit bookmark. ($\sigma = 0.138, c = 1.0$)
+ The user must be able to view bookmark. ($\sigma = 0.103, c = 1.0$)
+ The user must be able to view account. ($\sigma = 0.103, c = 1.0$)
+ The user must be able to edit tag. ($\sigma = 0.103, c = 1.0$)
+ The user must be able to edit account. ($\sigma = 0.103, c = 1.0$)
+ The user must be able to logout account. ($\sigma = 0.172, c = 0.62$)
− The user must be able to contact account. ($\sigma = 0.172, c = 0.62$)
− The user must be able to contact bookmark. ($\sigma = 0.138, c = 0.5$)
− The user must be able to stop account. ($\sigma = 0.138, c = 0.5$)

+/−: Correctly/Incorrectly Recommended Requirement
σ: Support, c: Confidence

(b) Recommended requirements for Restmarks.



(c) Visualized requirements for Restmarks.

Figure 12. Example depicting the recommendation of functional requirements for project Restmarks.

The recommended requirements are quite rational. For example, the option to edit one's tags (e.g. for renaming) or to log out of one's account probably has been omitted by the engineers/stakeholders that originally compiled the requirements of Restmarks. Interestingly, the quality of each recommendation seems to be correlated with the corresponding support and confidence values. By visualizing the number of recommended requirements for each different combination of support and confidence values (in red color), including the percentage of correctly recommended requirements for these combinations (in blue color), we conclude that most recommendations exhibit high confidence.

Finally, we used a cross-validation scheme to further evaluate our approach and explore the influence of the support and confidence metrics on the quality of the recommendations. We split the dataset into 6 equal bins of 5 projects. For each bin, we removed the 5 projects of the bin from the dataset, we extracted the association rules from the remaining 25 projects, and recommended new requirements for the 5 removed projects. After this procedure, we examined the recommended requirements for each project and determined whether each of them could be valid. The accumulated results of our evaluation for all projects are shown in Table 5 and visualized in Figure 13.

In total, our system recommended 297 requirements, out of which 177 were correct recommendations. Given that almost 60% of the recommendations can lead to useful requirements, we deem the results as satisfactory. Given a project, the requirements engineer would have been presented with a set of 10 requirements, out of which he/she would have selected 6 to add to the project. Most recommended requirements are extracted from rules with low support, which is actually expected since our dataset is

Table 5. Evaluation Results for the Recommended Requirements.

Support (σ)	Confidence (c)	# Correctly Rec. Requirements	Recommended Requirements	% Correctly Rec. Requirements
0.2	1.0	1	2	50.0%
0.133	1.0	23	37	62.16%
0.1	1.0	43	76	56.58%
0.133	0.8	2	4	50.0%
0.2	0.75	0	1	0.0%
0.1	0.75	64	92	69.57%
0.167	0.71	0	1	0.0%
0.133	0.67	13	14	92.86%
0.167	0.62	1	1	100.0%
0.1	0.6	9	17	52.94%
0.133	0.57	4	7	57.14%
0.167	0.56	4	5	80.0%
0.1	0.5	13	40	32.5%
	Total	177	297	59.6%

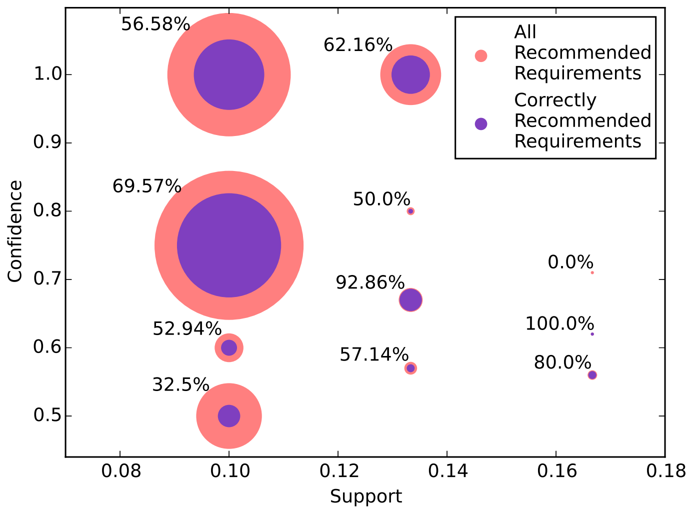


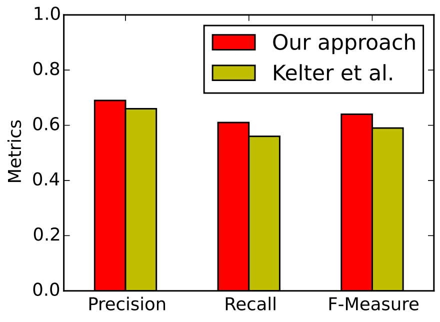
Figure 13. Visualization of recommended requirements including the percentage of the correctly recommended requirements given support and confidence.

largely domain-agnostic. However, low support rules do not necessarily result in low quality recommendations, as long as their confidence is large enough. Indicatively, 2 out of 3 recommendations extracted from rules with confidence values equal to 0.5 may not be useful. However, setting the confidence value to 0.75 ensures that more than 2 out of 3 recommended requirements will be added to the project.

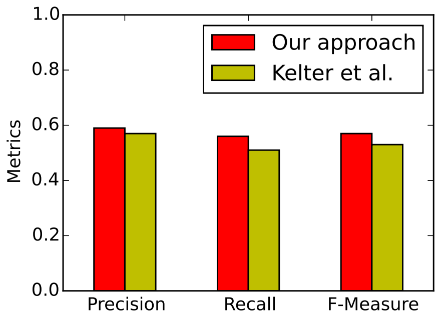
4.2. *UML Models Mining*

To evaluate our UML mining model, we use a dataset of 65 use case diagrams and 72 activity diagrams, originating from software projects with different semantics. Our methodology involves finding similar diagrams and subsequently providing recommendations, thus we initially split diagrams into 6 categories, including health/mobility, traffic/transportation, social/p2p networks, account/product services, business process, and generic diagrams. We construct all possible pairs of use case and activity diagrams, which are 2080 and 2556 pairs respectively, and mark each pair as relevant or non-relevant according to their categories. The dataset of the XMIs and their defined categories can be found at `http://s-case.github.io/publications/eis2017/`.

We compare our approach to that of Kelter, Wehren, and Niere (2005)[4]. Given that the two approaches are structurally similar, their execution on use case diagrams provides an assessment of the semantic methodology. Concerning activity diagrams, the approach of Kelter et al. uses a static data model, thus our evaluation should reveal how using a dynamic flow model can be more effective. Upon executing the approaches on the sets of use case and activity diagrams, we normalized the matching scores according to their average (so that pairs are defined when their score is higher than 0.5), and computed the precision, the recall, and the F-measure for each approach and for each diagram type. The results are shown in Figure 14.



(a) Evaluation of Use Case diagrams.



(b) Evaluation of Activity diagrams.

Figure 14. Classification results of our approach and the approach of Kelter et al. for the dataset diagrams.

It is clear that our approach outperforms that of Kelter et al. when it comes down to finding relevant diagram pairs both for use case and for activity diagrams. The higher recall values indicate that our methodology can effectively retrieve more relevant diagram pairs, while high precision values indicate that the retrieved pairs are indeed relevant and false positives (non-relevant pairs that are considered relevant) are fewer. The combined F-measure also indicates that our approach is more effective for extracting semantically similar diagram pairs.

5. Threats to Validity and Limitations

The limitations and threats to the validity of our approach span along the following four axes: (a) its applicability to component-based and agile development practices, (b) its effectiveness in cases of randomly structured or unstructured data, (c) the potential of its semantics to extract new information from the models through inference, and (d) the assessment of its usefulness in actual scenarios.

Requirements in agile approaches are usually characterized by user stories, while semi-structured text and/or UML diagrams are not typically used (except when it is

[4]An extensive comparison between the two approaches has been made as part of deliverable D2.4 of S-CASE, available at `http://s-case.github.io/publications/eis2017/S-CASE_D2.4.pdf`

considered necessary). As a result, our NLP parsing and UML modules cannot be used directly. However, the underlying modeling of requirements is capable of supporting the elements present in different formats, such as user stories; our ontologies sufficiently capture the static and dynamic view of software projects regardless of the requirements' representations. In relevant work (Zolotas et al. 2016), the dynamic ontology is instantiated using storyboards, which are graphical representations of program flow. Furthermore, as a future extension, the NLP module could be updated to support semi-structured formats that are similar to user stories, such as the Cucumber format (Wynne and Hellesoy 2012). Finally, the traceability of our methodology can prove useful in an agile context, where it is common for requirements to change and system features to be updated. This is particularly useful in cases of automated model (and subsequently code) generation, e.g. as in (Zolotas et al. 2016), since it allows modifying the requirements and maintaining a clear link between them and the produced features.

A relevant threat to validity involves the nature and quality of the data that are entered into the system. That is, although our approach can effectively handle well-defined requirements and standardized UML diagrams, its performance when given as input randomly structured or even unstructured data is not directly assessed by our datasets and evaluation. To address this threat, we have selected a set of projects that come from different sources, different software engineering teams and different domains. As already noted, our dataset contains student projects, industrial prototypes, and RESTful prototype applications from the EU-funded project S-CASE. Most of these projects were developed by third-party people or organizations, while the RESTful prototype applications were developed in the context of S-CASE, yet by the pilot cases and not the authors of this work. This led to testing with different types of requirements (e.g. the average number of tokens for the textual requirements of industrial prototypes was 16.6 as opposed to 11.6 for student projects), all however effectively parsed in our models (see (Diamantopoulos et al. 2017) for more information about the effectiveness of the NLP parser for model instantiation using diverse requirements). Finally, although currently there is no direct support for certain formats (e.g. user stories), these could be supported in future extensions, as discussed in the previous paragraph.

The third limitation, which relates also to the possible inputs of our methodology, is that of the employed semantics. Although our approach involves structural and semantic matching (using ontologies and WordNet (Miller 1995)), it refrains from applying semantic inference on the ontologies, which could prove useful for validation purposes and possibly for revealing missing requirements information. Our approach instead focuses on functionality co-occurrence among requirements of different projects, therefore inference is performed at a data mining level, using association rule mining and model matching techniques. Though not suited for declarative methodologies that may require different level of semantics, our approach is however effective for imperative methodologies, while it can be used as a solid basis for further work. An indicative first effort towards this direction, which we have implemented for our semantic parsing module (Diamantopoulos et al. 2017), involves adding also inferred relationships (e.g. the phrase "the user can create his/her account" includes not only an Action performed on "account" but also Ownership of the "account" by the "user"). Finally, a future extension in this aspect would also involve incorporating semantics in ontology level and/or further strengthening our semantic similarity methodology using methods such as salient semantic analysis (Luo et al. 2016) or by measuring similarity in higher orders of abstraction (Zhang et al. 2014).

The final threat refers to whether our approach is useful in actual scenarios. Although the evaluation presented in this work does not involve a user study or a time/effort measurement on development teams, the dataset used involves projects of different companies. Indicatively, our dataset involves also projects from the pilot cases of S-CASE (Zolotas et al. 2016), including an Internet of Things (IoT) application, a small Social Network and an Internet as a Service (IaaS) application. In any case, a user study or a time/effort measurement would be useful so it is considered as future work.

6. Conclusion

As the need for proper reuse of requirements elicitation is becoming more evident, we have designed a methodology capable of storing software requirements for mining and reuse purposes. Our model is based on ontologies, thus capturing semantic information and allowing flexibility and traceability when it comes to maintaining and updating the original software requirements. We constructed two different mining methodologies for extracting useful associations from functional requirements and UML models. Our case study and evaluation illustrates that our approach can be effective for providing useful recommendations to add to a system or improve on already identified requirements.

Potential future work on our methodology lies in multiple axes, some of which were explored in Section 5. Apart from these, the ontology models can be extended possibly by involving the concept of stakeholders in order to further provide recommendations and corrections according to their preferences. Concerning the recommendations, different techniques can also be explored to minimize the error rates (Khedr et al. 2017). Additionally, the semantic matching of the functional requirements mining can be further improved by incorporating information from online software databases (e.g. GitHub). Finally, the UML model mining methodology can include parsing different diagram types, such as UML class or sequence diagrams.

Acknowledgements

Parts of this work have been supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission.

References

Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami. 1993. "Mining Association Rules Between Sets of Items in Large Databases." In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, New York, NY, USA, 207–216. ACM.

Agrawal, Rakesh, and Ramakrishnan Srikant. 1994. "Fast Algorithms for Mining Association Rules in Large Databases." In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, San Francisco, CA, USA, 487–499. Morgan Kaufmann Publishers Inc.

Alspaugh, Thomas A., Annie I. Antón, Tiffany Barnes, and Bradford W. Mott. 1999. "An Integrated Scenario Management Strategy." In *Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, RE '99, Washington, DC, USA, 142–149. IEEE Computer Society.

Alves, Vander, Christa Schwanninger, Luciano Barbosa, Awais Rashid, Peter Sawyer, Paul Rayson, Christoph Pohl, and Andreas Rummler. 2008. "An Exploratory Study of Information Retrieval Techniques in Domain Analysis." In *Proceedings of the 2008 12th International*

Software Product Line Conference, SPLC ’08, Washington, DC, USA, 67–76. IEEE Computer Society.

Bildhauer, Daniel, Tassilo Horn, and Jurgen Ebert. 2009. “Similarity-driven Software Reuse.” In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM ’09, Washington, DC, USA, 31–36. IEEE Computer Society.

Blok, M. C., and J. L. Cybulski. 1998. “Reusing UML Specifications in a Constrained Application Domain.” In *Proceedings of the Fifth Asia Pacific Software Engineering Conference*, APSEC ’98, Washington, DC, USA, 196–. IEEE Computer Society.

Boehm, Barry, and Victor R. Basili. 2001. “Software defect reduction top 10 list.” *Computer* 34: 135–137.

Bonilla-Morales, Belén, Sérgio Crespo, and Clifton Clunie. 2012. “Reuse of Use Cases Diagrams: An Approach based on Ontologies and Semantic Web Technologies.” *Int. J. Comput. Sci.* 9 (1): 24–29.

Castañeda, Verónica, Luciana Ballejos, Ma. Laura Caliusco, and Ma. Rosa Galli. 2010. “The Use of Ontologies in Requirements Engineering.” *Global Journal of Researches In Engineering* 10 (6).

Castro-Herrera, Carlos, Chuan Duan, Jane Cleland-Huang, and Bamshad Mobasher. 2008. “Using Data Mining and Recommender Systems to Facilitate Large-Scale, Open, and Inclusive Requirements Elicitation Processes.” In *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, RE ’08, Washington, DC, USA, 165–168. IEEE Computer Society.

Chawathe, Sudarshan S., Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. 1996. “Change Detection in Hierarchically Structured Information.” *SIGMOD Rec.* 25 (2): 493–504.

Chen, Kun, Wei Zhang, Haiyan Zhao, and Hong Mei. 2005. “An Approach to Constructing Feature Models Based on Requirements Clustering.” In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, RE ’05, Washington, DC, USA, 31–40. IEEE Computer Society.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition*, 3rd ed., 390–396. The MIT Press.

Dermeval, Diego, JÃĺssyka Vilela, IgIbert Bittencourt, Jaelson Castro, Seiji Isotani, Patrick Brito, and Alan Silva. 2015. “Applications of ontologies in requirements engineering: a systematic review of the literature.” *Requirements Engineering* 1–33.

Diamantopoulos, Themistoklis, Michael Roth, Andreas Symeonidis, and Ewan Klein. 2017. “Software Requirements As an Application Domain for Natural Language Processing.” *Lang. Resour. Eval.* 51 (2): 495–524.

Diamantopoulos, Themistoklis, Klearchos Thomopoulos, and Andreas L. Symeonidis. 2016. “QualBoa: Reusability-aware Recommendations of Source Code Components.” In *Proceedings of the IEEE/ACM 13th Working Conference on Mining Software Repositories*, MSR ’16, 05, 488–491.

Dimaridou, Valasia, Alexandros-Charalampos Kyprianidis, Michail Papamichail, Themistoklis Diamantopoulos, and Andreas Symeonidis. 2017. “Towards Modeling the User-Perceived Quality of Source Code using Static Analysis Metrics.” In *Proceedings of the 12th International Joint Conference on Software Technologies*, ICSOFT, to appear.

Dumitru, Horatiu, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. 2011. “On-demand Feature Recommendations Derived from Mining Public Product Descriptions.” In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE ’11, New York, NY, USA, 181–190. ACM.

Felfernig, Alexander, Monika Schubert, Monika Mandl, Francesco Ricci, and Walid Maalej. 2010. “Recommendation and Decision Technologies for Requirements Engineering.” In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, RSSE ’10, New York, NY, USA, 11–15. ACM.

Finlayson, Mark. 2014. “Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation.” In *Proceedings of the Seventh Global Wordnet Conference*, edited by Heili Orav, Christiane Fellbaum, and Piek Vossen, Tartu, Estonia, 78–85.

Frakes, William, Ruben Prieto-Diaz, and Christopher Fox. 1998. "DARE: Domain Analysis and Reuse Environment." *Ann. Softw. Eng.* 5: 125–141.

Franck, Stefan. 2017. "Going Agile - Does it work for all?" Accessed 2017-08-03. `https://www.netcentric.biz/blog/Does-Agile-Work-For-The-Enterprise.html`.

Ghaisas, S., and N. Ajmeri. 2013. "Knowledge-Assisted Ontology-Based Requirements Evolution." In *Managing Requirements Knowledge*, edited by Walid Maalej and Anil Kumar Thurimella, 143–167. Springer Berlin Heidelberg.

Gomes, Paulo, Pedro Gandola, and Joel Cordeiro. 2007. "Helping Software Engineers Reusing UML Class Diagrams." In *Proceedings of the 7th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ICCBR '07, Berlin, Heidelberg, 449–462. Springer-Verlag.

Happel, Hans-Jörg, and Stefan Seedorf. 2006. "Applications of Ontologies in Software Engineering." In *Proceedings of the 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006)*, 5–9.

Hummel, Oliver, Werner Janjic, and Colin Atkinson. 2008. "Code Conjurer: Pulling Reusable Software out of Thin Air." *IEEE Softw.* 25 (5): 45–52.

Ingvaldsen, Jon Espen, and Jon Atle Gulla. 2012. "Industrial application of semantic process mining." *Enterprise Information Systems* 6 (2): 139–163.

Kaindl, H., M. Smialek, D. Svetinovic, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, et al. 2007. *Requirements specification language definition: Defining the ReDSeeDS languages, Deliverable D2.4.1.* Public deliverable. ReDSeeDS (Requirements Driven Software Development System) Project.

Kelter, Udo, Jürgen Wehren, and JÃűrg Niere. 2005. "A Generic Difference Algorithm for UML Models." In *Software Engineering*, edited by Peter Liggesmeyer, Klaus Pohl, and Michael Goedicke, Vol. 64 of *LNI*, 105–116. GI.

Khedr, Ayman E., Amira M. Idrees, Abd El-Fatah Hegazy, and Samir El-Shewy. 2017. "A proposed configurable approach for recommendation systems via data mining techniques." *Enterprise Information Systems* 0 (0): 1–22.

Konstan, Joseph A., Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. 1997. "GroupLens: Applying Collaborative Filtering to Usenet News." *Commun. ACM* 40 (3): 77–87.

Kumar, Manish, Nirav Ajmeri, and Smita Ghaisas. 2010. "Towards Knowledge Assisted Agile Requirements Evolution." In *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*, RSSE '10, New York, NY, USA, 16–20. ACM.

Leffingwell, Dean. 1997. "Calculating your return on investment from more effective requirements management." *American Programmer* 10 (4): 13–16.

Li, JianQiang, Ji-Jiang Yang, Chunchen Liu, Yu Zhao, Bo Liu, and Yuliang Shi. 2014. "Exploiting semantic linkages among multiple sources for semantic information retrieval." *Enterprise Information Systems* 8 (4): 464–489.

Lim, Soo Ling, and Anthony Finkelstein. 2012. "StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation." *IEEE Trans. Softw. Eng.* 38 (3): 707–735.

Lin, Dekang. 1998. "An Information-Theoretic Definition of Similarity." In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, San Francisco, CA, USA, 296–304. Morgan Kaufmann Publishers Inc.

Luo, Jing, Bo Meng, Changqin Quan, and Xinhui Tu. 2016. "Exploiting salient semantic analysis for information retrieval." *Enterprise Information Systems* 10 (9): 959–969.

Maalej, Walid, and Anil Kumar Thurimella. 2009. "Towards a Research Agenda for Recommendation Systems in Requirements Engineering." In *Proceedings of the 2009 Second International Workshop on Managing Requirements Knowledge*, MARK '09, Washington, DC, USA, 32–39. IEEE Computer Society.

Mich, Luisa, Franch Mariangela, and Novi Inverardi Pierluigi. 2004. "Market research for requirements analysis using linguistic tools." *Requirements Engineering* 9 (1): 40–56.

Miller, George A. 1995. "WordNet: A Lexical Database for English." *Commun. ACM* 38 (11): 39–41.

Mobasher, Bamshad, and Jane Cleland-Huang. 2011. "Recommender Systems in Requirements Engineering." *The AI magazine* 32 (3): 81–89.

Montequin, V. R., S. Cousillas, F. Ortega, and J. Villanueva. 2014. "Analysis of the Success Factors and Failure Causes in Information & Communication Technology (ICT) Projects in Spain." *Procedia Technology* 16: 992–999.

Mylopoulos, John, Jaelson Castro, and Manuel Kolp. 2000. "Tropos: A Framework for Requirements-Driven Software Development." In *Information Systems Engineering: State of the Art and Research Themes*, 261–273. Springer-Verlag.

Olson, David L., and Subodh Kesharwani. 2009. *Enterprise Information Systems: Contemporary Trends and Issues.* River Edge, NJ, USA: World Scientific Publishing Co., Inc.

Papamichail, Michail, Themistoklis Diamantopoulos, and Andreas L. Symeonidis. 2016. "User-Perceived Source Code Quality Estimation based on Static Analysis Metrics." In *2016 IEEE International Conference on Software Quality, Reliability and Security*, QRS, Vienna, Austria, 08, 100–107.

Park, Wei-Jin, and Doo-Hwan Bae. 2011. "A Two-stage Framework for UML Specification Matching." *Inf. Softw. Technol.* 53 (3): 230–244.

Pedersen, Ted, Siddharth Patwardhan, and Jason Michelizzi. 2004. "WordNet::Similarity: Measuring the Relatedness of Concepts." In *Demonstration Papers at HLT-NAACL 2004*, HLT-NAACL–Demonstrations '04, Stroudsburg, PA, USA, 38–41. Association for Computational Linguistics.

Robinson, William N., and Han G. Woo. 2004. "Finding Reusable UML Sequence Diagrams Automatically." *IEEE Softw.* 21 (5): 60–67.

Robles, Karina, Anabel Fraga, Jorge Morato, and Juan Llorens. 2012. "Towards an Ontology-based Retrieval of UML Class Diagrams." *Inf. Softw. Technol.* 54 (1): 72–86.

Romero-Mariona, Jose, Hadar Ziv, and Debra J. Richardson. 2008. "SRRS: A Recommendation System for Security Requirements." In *Proceedings of the 2008 International Workshop on Recommendation Systems for Software Engineering*, RSSE '08, New York, NY, USA, 50–52. ACM.

Roth, Michael, Themistoklis Diamantopoulos, Ewan Klein, and Andreas Symeonidis. 2014. "Software Requirements: A new Domain for Semantic Parsers." In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, Baltimore, MD, June, 50–54. Association for Computational Linguistics.

Sahavechaphan, Naiyana, and Kajal Claypool. 2006. "XSnippet: Mining for Sample Code." *SIGPLAN Not.* 41 (10): 413–430.

Salami, Hamza Onoruoiza, and Moataz Ahmed. 2013. "Class Diagram Retrieval Using Genetic Algorithm." In *Proceedings of the 2013 12th International Conference on Machine Learning and Applications - Volume 02*, ICMLA '13, Washington, DC, USA, 96–101. IEEE Computer Society.

Siegemund, Katja, Edward J Thomas, Yuting Zhao, Jeff Pan, and Uwe Assmann. 2011. "Towards ontology-driven requirements engineering." In *Workshop semantic web enabled software engineering at 10th international semantic web conference (ISWC), Bonn*, .

Smialek, Michal. 2012. "Facilitating Transition from Requirements to Code with the ReDSeeDS Tool." In *Proceedings of the 2012 IEEE 20th International Requirements Engineering Conference (RE)*, RE '12, Washington, DC, USA, 321–322. IEEE Computer Society.

Sommerville, Ian. 2010. *Software Engineering.* 9th ed. Harlow, England: Addison-Wesley.

Thummalapenta, Suresh, and Tao Xie. 2007. "Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web." In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, New York, NY, USA, 204–213. ACM.

Wang, Y., D. J. DeWitt, and J. Y. Cai. 2003. "X-Diff: an effective change detection algorithm for XML documents." In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, March, 519–530.

Woo, Han G., and William N. Robinson. 2002. "Reuse of Scenario Specifications Using an Automated Relational Learner: A Lightweight Approach." In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, RE '02, Washington, DC, USA, 173–180. IEEE Computer Society.

Wynne, Matt, and Aslak Hellesoy. 2012. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf.

Zhang, Hui, Deqing Wang, Li Wang, Zhuming Bi, and Yong Chen. 2014. "A semantics-based method for clustering of Chinese web search results." *Enterprise Information Systems* 8 (1): 147–165.

Zolotas, Christoforos, Themistoklis Diamantopoulos, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis. 2016. "From requirements to source code: a Model-Driven Engineering approach for RESTful web services." *Automated Software Engineering* .