The ReqWiki Approach for Collaborative Software Requirements Engineering with Integrated Text Analysis Support

Bahar Sateli Elian Angius René Witte
Semantic Software Lab
Department of Computer Science and Software Engineering
Concordia University, Montréal, Canada

***Abstract*—The requirements engineering phase within a software project is a heavily knowledge-driven, collaborative process that typically involves the analysis and creation of a large number of textual artifacts. We know that requirements engineering has a large impact on the success of a project, yet sophisticated tool support, especially for small to mid-size enterprises, is still lacking. We present *ReqWiki,* a novel open source web-based approach based on a semantic wiki that includes natural language processing (NLP) *assistants*, which work collaboratively with humans on the requirements specification documents. We evaluated ReqWiki with a number of software engineers to investigate the impact of our novel semantic support on software requirements engineering. Our user studies prove that *(i)* software engineers unfamiliar with NLP can easily leverage these assistants and *(ii)* semantic assistants can help to significantly improve the quality of requirements specifications.**

***Keywords*-Software Requirements Engineering; Natural Language Processing; Semantic Wiki; Quality Assurance**

I. Introduction

Software requirements engineering (RE) encompasses the tasks related to eliciting, evaluating, and recording the needs of various stakeholders of a software project. A software requirements specification (SRS) document containing clear and precise definitions of what stakeholders want is critical to building "the right product" and consequently the success of a project [1].

A large number of these specifications – up to 90% according to industry statistics [2] – are written in natural language, typically with the help of templates, such as use cases or user stories. Quality assurance (QA) is an integral part of requirements engineering [1]. However, due to the unstructured nature of natural language, QA needs to be performed manually and is therefore expensive and time-consuming. Natural language processing (NLP) has long been regarded as a potentially significant tool to support the requirements engineering phase [3], but so far has not been widely adopted.

The overall vision of our work is to support modern software development teams with a sophisticated requirements engineering tool that facilitates collaborative, distributed development, has built-in support for automated quality assurance on natural language text through semantic analysis, is easy to learn, and integrates well with heterogeneous web-based environments. *ReqWiki* is our solution presented here: It is the first open source system that seamlessly integrates a wiki-based platform with semantic knowledge representation for formal queries and reasoning with NLP web services for text analysis. These NLP services embody artificial intelligence (AI) 'assistants' that work collaboratively with the human users on a specification, within a single, cohesive interface. Several assistants are available by default, such as the detection of a number of common SRS defects, and custom NLP pipelines can be easily deployed through a service-oriented architecture (SOA).

To evaluate the usability and effectiveness of ReqWiki, we performed an evaluation with several project groups from two university courses in requirements engineering: one at the undergraduate level, one at the graduate level. In this study, we analyzed the usability of NLP methods in the RE process and their impact on the quality of a specification. The results demonstrate both the usability for software engineers, who are typically not trained in natural language processing techniques, as well as the positive impact on the quality of SRS documents.

Our work is significant for two main reasons: *(i)* we developed the first comprehensive open source architecture that combines a user-friendly wiki system with state-of-the-art semantic technologies including natural language processing. It is made available under a standard open source license at http://www.semanticsoftware.info/reqwiki and can be immediately deployed in a software project; and *(ii)* we present the first user study on the usability of text analysis services for software engineers and their impact on the quality of a requirements specification.

II. Background

In this section, we briefly introduce foundations from software requirements engineering and natural language processing.

A. Software Requirements Engineering

Requirements Engineering (RE) is one of the most important phases of a software project: We know that its success or failure is highly dependent on successful requirements engineering, with industry statistics pointing to insufficient RE as the root cause in more than 50% of all unsuccessful software projects [1].

When an initial set of requirements has been elicited and evaluated, it can be captured in a requirements document.

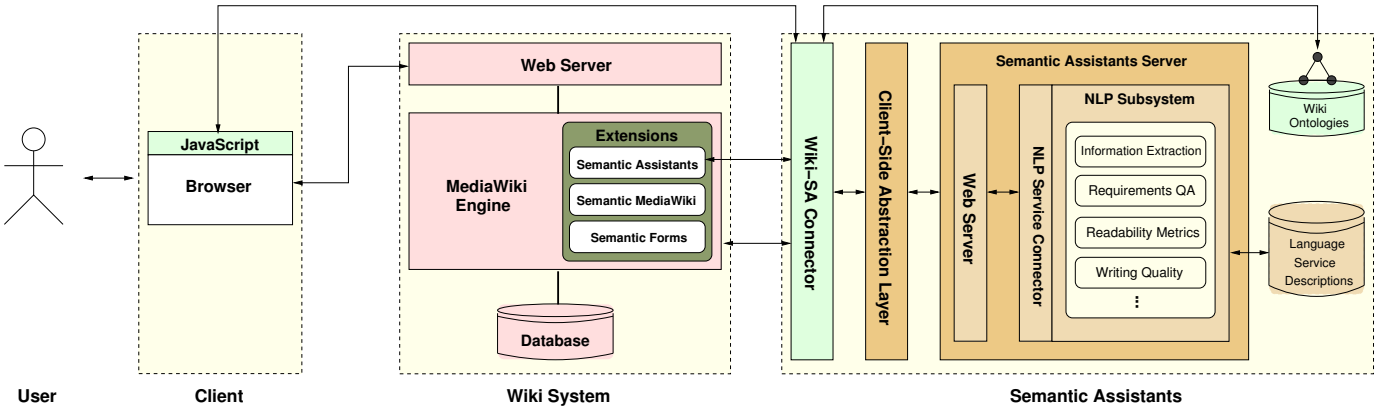

Fig. 1. ReqWiki multi-tier system architecture overview: user interface, semantic wiki system, and Semantic Assistants server for NLP services

Natural language (NL) specifications continue to be the most commonly used form (as opposed to formal models, based on a logical framework), accounting for up to 90% of all specifications [2]. However, natural language specifications are prone to a number of errors and flaws, in particular due to the ambiguity inherent in natural language [1]. This is partially addressed by adopting particular templates and writing styles, such as *use cases* or *user stories*. Use cases [4] have become one of the most widely used forms of requirements specification. Writing good use cases requires an understanding of their core concepts (actors, goals, scenarios, etc.), as well as associated writing rules [5].

However, defects are still frequent and the cost of finding and fixing them increases from each engineering stage to the next (requirements, design, implementation, testing, deployment, maintenance) [5]. This is addressed by requirements quality assurance (QA), also known as requirements validation [1]. The goal of QA is to verify the requirements artifacts produced during elicitation, evaluation, and specification, both in terms of quality (do they follow documentation guidelines and templates, are they complete, free of defects, omissions, etc.) and objectives (do the requirements capture the stakeholders' actual needs and wishes).

While formal specifications can be readily verified with software tools, the quality analysis of natural language requirements has traditionally been performed manually, e.g., by inspections and review boards [1]. Below, we will highlight the most important quality assurance concepts and explain how and where NLP methods can contribute in the software requirements specification process.

B. Natural Language Processing

Natural Language Processing (NLP) is a branch of computer science that employs various artificial intelligence techniques to process content written in natural language (NL). One of the applications of NLP is text mining, the process of deriving structured information from text. Developing text mining systems is facilitated through the use of frameworks, such as the *General Architecture for Text Engineering* (GATE) [6]. Using these frameworks, sophisticated text mining applications can be developed, not only to derive patterns within the unstructured data, but also to enhance information management of a system by finding content based on meaning and context, using technologies from the domain of semantic computing.

Since requirements specifications are most commonly written in natural language, they have recently gained attention by researchers in the field of NLP [2]. For example, the work by Hussain et al. [7] shows how ambiguity in SRS documents can be detected through a decision tree-based classifier; Gervasi and Zowghi discuss the detection of inconsistencies [8]; Sawyer et al. look at supporting concept recognition [9]; and Kof [3] investigates the identification of goals and the construction of domain models from NL documents.

Within this work, we largely rely on existing off-the-shelf NLP components, with some customizations for RE. Our main contributions are *(i)* the seamless integration of these techniques into a wiki-based requirements development system; and *(ii)* investigating how software engineers and other end users, which are not necessarily familiar with NLP, can interact with these techniques in order to improve the quality of a SRS.

III. Design

We now detail the requirements for our ReqWiki system and subsequently describe the design decisions and overall system architecture that we derived from these requirements (Fig. 1).

A. Requirements Analysis

What precisely are the requirements for a RE environment, where users can benefit from semantic capabilities and NLP techniques?

Collaborative Environment (R1). In contemporary contexts, in particular global software development where stakeholders are spatially separated, a strong need exists for a collaborative tool that provides the means for asynchronous communication. Users must be able to author and edit requirements, with clear tracing of their changes in the system.

Consistency (R2). Due to the different backgrounds and perspectives of users, which include diverse groups such as non-technical stakeholders and system developers, as well as different terminologies used by different parties in an

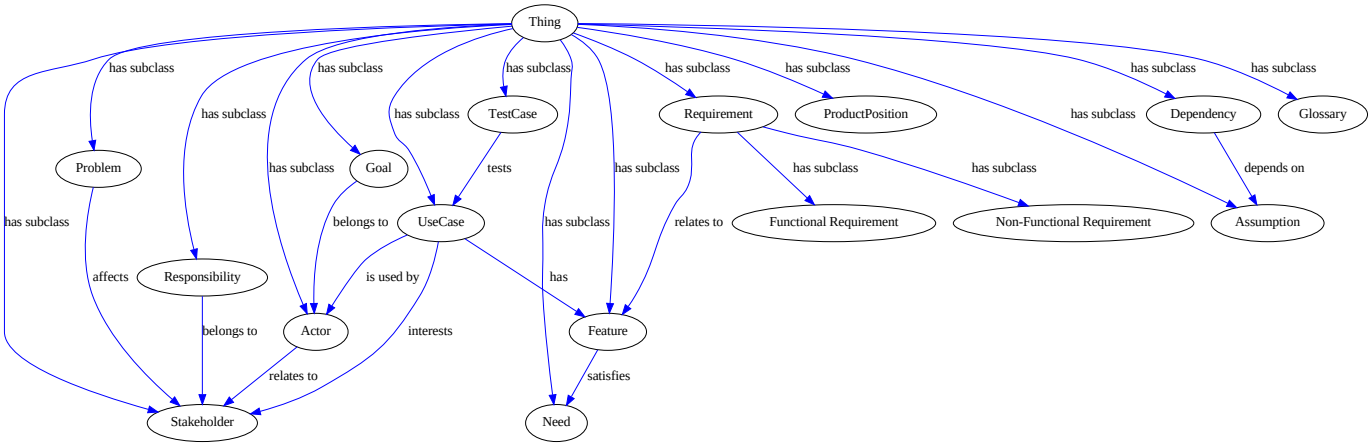

Fig. 2. The ReqWiki ontology modeling the domain of Software Requirements Specifications (SRS)

RE process, the system must provide a consistent approach to unambiguously describe system concepts and their inter-relationships.

Semantic Support (R3). In addition to supporting natural language for the SRS documents, the system shall provide the means to formally describe the SRS entities, both manually and in an automatic manner, in order to make them accessible to machines for automatic processing.

Quality Assurance (R4). Due to the inherent ambiguity and vagueness of natural language used in SRS documents, the system shall provide the users with the ability to perform automatic quality assessment on the documents' content to detect and correct defects.

Seamless Integration (R5). The integration of semantic and NLP capabilities in the system shall be seamless to end users and must not assume any background knowledge in NLP or semantic technologies.

Service Independence (R6). Users must be able to employ any arbitrary NLP service that is best suited for their context. This means that the system must allow various NLP services to be added to the system as needed.

Support for Standard Methodologies (R7). To conform to standard methodologies in RE, the system must be able to adapt itself to contain the artifacts pertinent to the selected underlying methodology, in particular the Unified Process (UP) [10] and Use Case models [4], [5].

B. ReqWiki Architecture

A wide range of tools, ranging from conventional office suites to dedicated commercial tools, are used for requirements engineering purposes. Wikis, as an affordable, lightweight documentation platform have demonstrated their capabilities in distributed requirements elicitation [11] and documentation [12]. We also adopted a wiki engine at the core of ReqWiki to provide a collaborative environment (R1) for SRS development. This brings a number of advantages, including wide acceptance among users, mainly due to its lightweight nature, ease of use, familiarity, and simple markup language [11]. Because a wiki system is essentially a web-based application, it is especially suited for distributed RE environments, as it requires no special tools other than a web browser on the user's side.

However, unadorned wikis can not fulfill the additional requirements, in particular consistency management (R2), semantic support (R3), and natural language processing (R4, R5). For this, we add two more concepts, detailed below: a semantic data model and NLP web services. Fig. 1 shows the complete system architecture, integrating a semantic wiki system with custom plug-ins (middle), as well as the Semantic Assistants NLP web services framework (right). The details of their integration are described in the implementation section below.

C. Semantic Model for SRS

Ontologies provide for formal specifications of a shared conceptualization, consisting of vocabularies and relations: In the case of ReqWiki, our domain of discourse is requirements engineering. In order to ensure consistency (R2) and compliance with a standard methodology (R7), both within and between requirements artifacts, we formally model these artifacts (e.g., use cases in the UP), as well as their entities (e.g., actors, goals) and their inter-relationships using the web ontology language (OWL).[1] This formal model is particularly useful when using *semantic wikis*, which allow users to import formal ontologies to the wiki, and thereby connect unstructured textual descriptions with their corresponding entities. This provides in part for the required quality assurance (R4) through the semantic constraints modeled in the ontology; Additional quality assurance is provided through the NLP analysis services described below.

1) Ontological Formalization of SRS: We discussed in our system-to-be requirements that a consistent understanding (R2) of the RE concepts and terminology is essential in an RE process. In ReqWiki, we formally model our system entities and their relationships by defining an ontology written in OWL. Our ReqWiki ontology (Fig. 2) defines the main concepts in the unified process and use case methodology

[1] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/

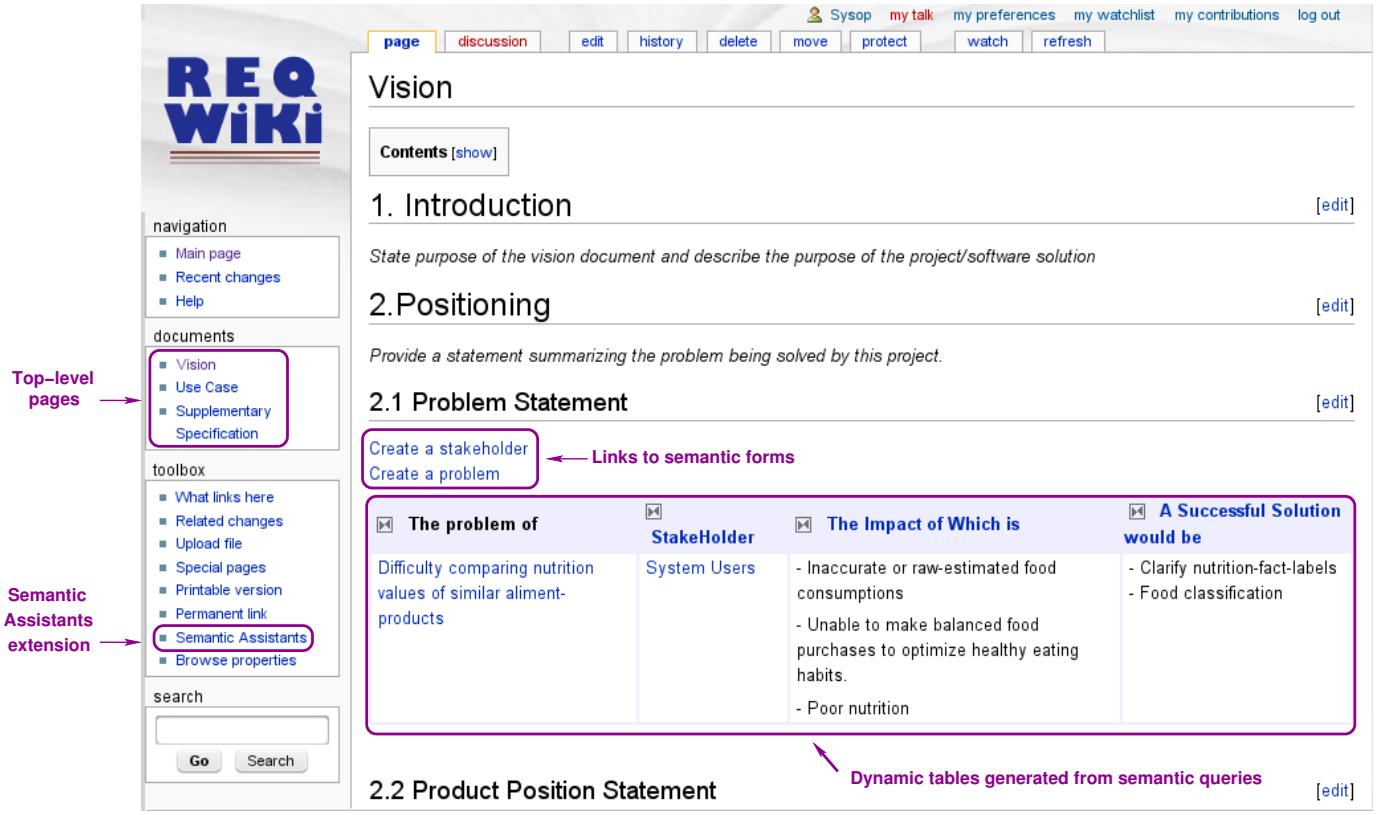

Fig. 3. The ReqWiki main user interface, integrating textual content, semantic knowledge, and access to NLP services

for requirements engineering. This ontology provides a formal, semantic description of concepts stored in the wiki (e.g., actors, goals, use cases, test cases, features) and their relationships with each other. For example, we can now formally establish the relations between a *Use Case* and its *Actors*.

Once the content of a ReqWiki installation is enriched with semantic metadata, we can use this ontology to semantically *query* it, e.g., to retrieve a list of all actors in a wiki (R3): This is an important advantage of semantic approaches over standard wikis or textual documents, where a list of all actors, goals, or use cases in a SRS could only be manually curated, but not automatically generated and updated. In addition, reasoning capabilities can be added to the system to automatically detect inconsistencies and conflicts between entities.

2) Automatic Traceability: Motivated by maintainability and liability project factors, *traceability* is the degree to which various software artifacts in a developing system are interrelated with each other [1]. In a collaborative and volatile environment such as RE, traceability ensures the rationale for all artifacts are properly accounted for towards building "the right product". Manually creating and updating these links, for example, with cross-references between documents, is highly time-consuming and error prone. Hence, we aim to automatically generate and update traceability links as far as possible. Additionally, it must be possible to reference requirements artifacts from those produced in subsequent phases, such as design and implementation. Relying on the populated ontology, ReqWiki supports the automatic creation of three forms of traceability links: semantic links inside the wiki, query-based links, and revision links.

For the first type, semantic concepts defined in wiki pages are presented as hyperlinks, so that users can navigate to them and retrieve their corresponding content. When defining these semantic concepts, the forms themselves already constrain field input to existing (non-empty) ontology instances of the expected field type. Thus, this form of traceability enforces link correctness and simplifies verification and validation phases, since dead links and unlinkable artifacts are easily detectable.

For the second type, using the same semantic metadata in our ontology, we can insert in-line queries in wiki pages to create dynamic tables (e.g., use cases vs. actors) from the ontological metadata stored in a wiki (shown below).

A third form of traceability available in ReqWiki is its revision control capability that records page changes: This not only indicates which author changed what content at which times, but also allows reverting to previous revisions in case of erroneous modifications.

D. NLP Services for Requirements Engineering

The second major component of our approach is the idea of NLP services that support users in developing the requirements specification. In accordance with R5, we aim to offer these analysis capabilities directly within our wiki, to avoid users having to switch to a different text mining application: This

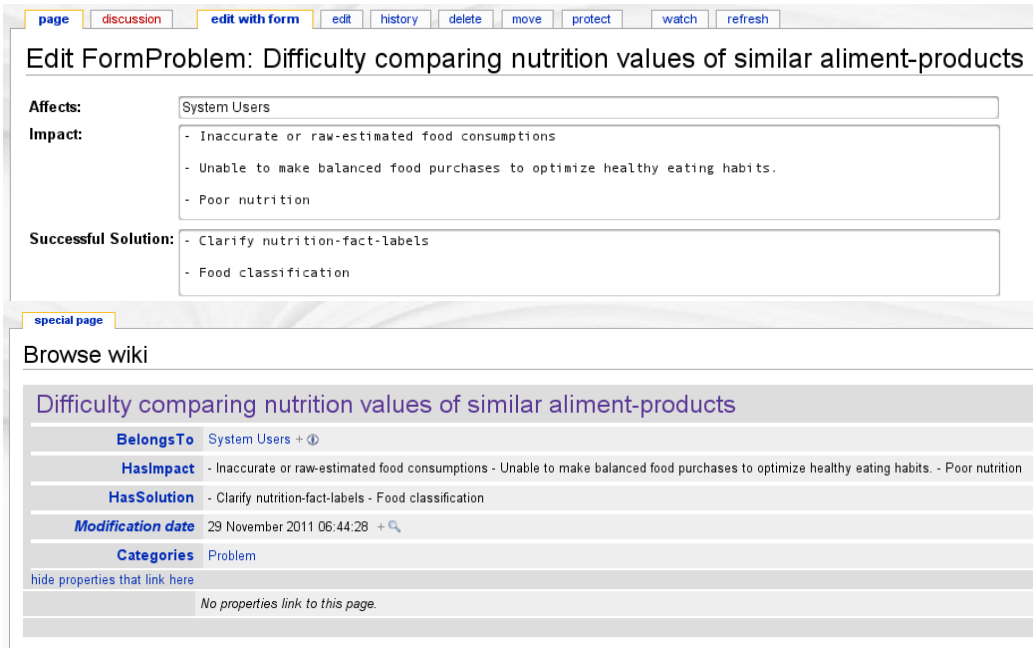

Fig. 4. Example wiki Semantic Form (top) used to generate embedded RDF data (bottom)

integration is achieved through our Wiki-NLP architecture [13].

Furthermore, we want to support setting up an installation with custom NLP services, e.g., for domain-specific analysis pipelines or custom (company or organization-specific) quality rules (R6). By developing a service-oriented architecture, new analysis services can be deployed and then dynamically discovered by the users. For this, we integrated the *Semantic Assistants* framework, which can broker any deployed NLP pipeline as a standard web service [14]. As described below, the wiki can then dynamically discover the available services and offer them to a user, or even execute them pro-actively.

The semantic NLP services also aid users in developing SRS by automatically annotating wiki pages with semantic metadata extracted from their content (R3, R4). Within our initial experiments, we made a number of both general and requirements-specific NLP services available to ReqWiki users:

Writing Quality Assessment performs grammar and spell checking on the content and provides suggestions for improvements, where possible. This service integrates the *After The Deadline* [15] tool and helps ReqWiki users find spelling and grammatical mistakes, as well as passive voice defects, in their requirements specification documents.

Readability Assessment measures the readability of a given text based on standard metrics, like Flesch and Kincaid [16]. This service provides users with an overall readability score of the content they have produced, indicating how hard to read and comprehend their text is for other stakeholders. It also can be used to trace the readability over time [17].

Requirements Quality Assurance is a service developed based on the NASA ARM requirements quality metrics [18]. It detects SRS defects like *Options, Directives* or *Weak Phrases* in a document. By using this service, users have the chance to automatically find these defects in their specifications and correct them, resulting in a higher quality SRS document.

Document Indexer creates a back-of-the-book style index of the SRS content as a new wiki page. This service builds on MuNPEx,[2] an open source tool that groups words into noun phrases, to generate an inverted index, with entries automatically linked to wiki pages. Users can compare the result of this service to the glossary section of the SRS documents to check its completeness.

Information Extractor performs named entity extraction on wiki pages, based on the ANNIE system [6]. This service can aid users by automatically extracting named entities, such as persons, organizations or locations, which is especially useful in analyzing existing domain documents during early requirements analysis phases.

Based on the Semantic Assistants server settings configured in ReqWiki, the list of available NLP services is dynamically generated and provided in a drop-down box (Fig. 5). A user then has the option to view a brief description of what each service does and customize the service execution by providing additional runtime parameters. New services only need to be deployed on the Semantic Assistants server and then become immediately accessible by all connected wiki clients.

IV. Implementation

In this section, we describe the implementation of our ReqWiki system. In particular, we show how the concepts of wikis, semantic markup, and NLP services connect together to make ReqWiki into an intelligent, agile authoring environment for software specifications (Fig. 3).

[2]MuNPEx, http://www.semanticsoftware.info/munpex

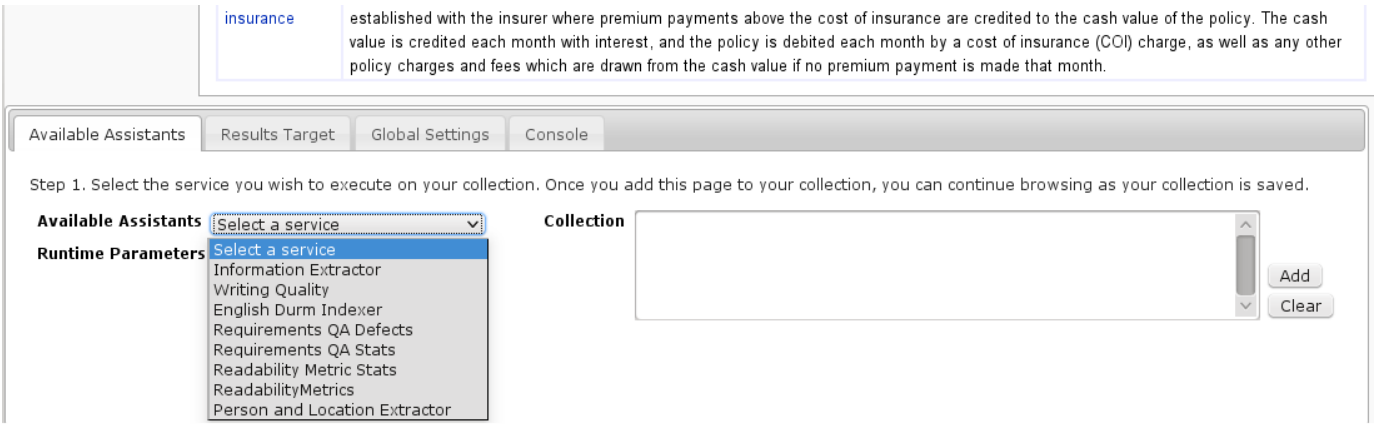

Fig. 5. Selecting one of the available NLP services in ReqWiki

A. Wiki Engine and Extensions

ReqWiki is powered by the well-known MediaWiki[3] engine, which allows additional functionalities to be added to its core engine by installing *extensions* (Fig. 1). In ReqWiki, we use a number of these extensions to add semantic and NLP capabilities to its wiki engine: To provide semantic support, we integrated Semantic MediaWiki (SMW).[4] This extension allows the ReqWiki content to be enriched with semantic metadata using the SMW syntax, queried with in-line queries, and exported in the RDF[5] format. In order to reduce the learning curve for users when using ReqWiki, we also make use of the Semantic Forms[6] extension, which allows them to enter and edit semantic wiki content using web forms, instead of working with raw markup. In addition, semantic forms allow users to create structured data with semantic markup by populating a pre-defined set of templates. NLP capabilities are provided to the MediaWiki engine through installing the Semantic Assistants extension, described in Section IV-D.

B. Semantic Markup and Forms

In order to support the standard use case methodology and other artifacts from the UP [10] (R7), we developed a number of semantic forms that support users in creating the main UP artifacts: *Vision Document, Supplementary Specification,* and the *Use Case* model [5], while at the same time automatically creating the required semantic metadata. Particular care was taken to reflect the original UP templates in ReqWiki, so that software engineers familiar with these standard artifacts can adopt them immediately – see Fig. 6 for a typical part of a word processor template and Fig. 4 for its translation into Semantic Forms.

We associated the semantic forms and wiki templates with their corresponding concepts in our ontology (cf. Section III-C1) using semantic properties defined in ReqWiki. For example, according to our ontology, each *Goal* entity in ReqWiki is related to one or more *Actor* instances by a *belongs to* object property. Therefore, whenever a goal instance is created using its corresponding semantic form, the resulting page contains semantic markup that relates the goal entity to the specified actor instances. Each field in the form is associated with a specific semantic property inside the wiki and once the data is saved, the input content is automatically enriched with semantic metadata using the SMW syntax. Moreover, for the form fields that are associated with a semantic property, the Semantic Forms extension also provides an auto-completion feature. For example, when a user is filling an actor field in the use case form, all the existing actor instances are queried inside ReqWiki and presented in a list format. This feature not only brings further convenience when using ReqWiki, but also prevents defects, like different spellings and punctuations, for the same concept.

C. Templates and Queries

To display the semantic form-generated data, we defined templates that format the content. As an example, the top half of Fig. 8 shows part of a use case (UC) that was entered via a Semantic Form, including links to the semantic entities, like actors and stakeholders.

An important feature of the templates are embedded *queries* that help to keep the specification up-to-date, without any manual effort required by the user. As an example, the UP Supplementary Specification templates contain a number of traceability tables (cf. Section III-C2) that connect various

2.1. Problem Statement

The problem of	*Difficulty comparing nutrition values of similar aliment-products*
Affects	*System Users*
The impact of which is	*- Inaccurate or raw-estimated food consumptions* *- Unable to make balanced food purchases to optimize healthy eating habits.* *- Poor nutrition*
A successful solution would be	*- Clarify nutrition-fact-labels* *- Food classification*

Fig. 6. Original Word Processor Template Example

[3]MediaWiki, https://www.mediawiki.org
[4]SMW, http://semantic-mediawiki.org/
[5]Resource Description Framework (RDF), http://www.w3.org/RDF/
[6]Semantic Forms, http://www.mediawiki.org/wiki/Extension:Semantic_Forms

```
1  {{#ask: [[Category:Features]]
2  | ?BelongsTo=Need
3  | ? = Feature
4  | format= table
5  }}
```

User Needs versus Features

Need	Feature
Modify policy detail information	Alter policy information
Modify policy detail information	Query the status of policy information
Alteration of unit link product	Input conversion to unit investment
Alteration of unit link product	Modify conversion to unit investment
Alteration of unit link product	Query unit investment
Alteration of unit link product	Query unit price

Fig. 7. Automatic traceability links in ReqWiki: Embedded SMW query (top) and resulting Feature-Need Table (bottom)

artifacts, e.g., user *needs* vs. system *features* [5]. We can automatically extract these relationships from the wiki's semantic data through an embedded query, as shown in Fig. 7. Therefore, traceability matrices that would have been manually created in traditional SRS documents can automatically be kept up-to-date, since the queries are processed every time the page is requested.

D. Semantic Assistants

NLP capabilities are provided in ReqWiki through the Semantic Assistants architecture [14] as shown in Fig. 1. The Wiki-SA Connector [13] is the main component that allows ReqWiki to invoke any arbitrary NLP service available in the Semantic Assistants NLP subsystem. Technically, the Wiki-SA Connector is a proxy server that uses a number of J2EE technologies to intercept the communication between ReqWiki and the Semantic Assistants web server. When the proxy is enabled on the wiki via the Semantic Assistants extension, all the service invocation requests are sent from ReqWiki to the Wiki-SA Connector component and the component in turn makes the actual service calls to the Semantic Assistants web server through the Semantic Assistants client-side abstraction layer. Eventually, the results of a service execution are transformed to wiki markup by the connector component and written back to the ReqWiki database.

As a proxy server, the Wiki-SA Connector also intercepts the communication between the ReqWiki engine and the user's browser. This means that once the proxy is requested by the user, the connector component retrieves the demanded wiki page from the ReqWiki database on behalf of the browser. Then, the proxy injects service inquiry and invocation capabilities into the page on-the-fly using JavaScript and sends it back to the user's browser. This approach creates the impression that the user is still using the native interface of the wiki, whereas the page is served by the proxy with the Semantic Assistants user interface depicted in Fig. 5, embedded at the bottom of the page.

The connection to the Wiki-SA Connector is made available in ReqWiki through an additional menu item that triggers a request call to proxy the wiki page for the user. The new menu item is introduced to the underlying MediaWiki engine through a lightweight Semantic Assistants extension. In addition to the new menu in the wiki, the Semantic Assistants extension also introduces a number of predefined MediaWiki templates that are used to present various UP artifacts and the results retrieved from the NLP service invocations. Using templates in the wiki introduces uniformity and consistency for ReqWiki entities, in addition to providing a clear separation between the data and its view. Moreover, it bundles maintenance efforts and changes to the presentation of entities for ReqWiki in one place.

V. Application

In this section, we describe the application of our ReqWiki system in a typical RE process. In order to use ReqWiki, a MediaWiki instance needs to be installed, along with the SMW and Semantic Forms extensions added to its engine. Semantic and NLP capabilities are then provided to the core engine by installing the Semantic Assistants extension. This extension will restructure the wiki navigation menu and import the top-level pages, the semantic forms and the semantic property pages to the wiki repository.

A. Writing a specification with ReqWiki

In the current default installation, ReqWiki features three top-level wiki pages that are considered the entry points to the SRS documentation platform. They have been designed to guide stakeholders through the RE process (Fig. 3). For each top-level page, a corresponding "talk" page – provided by the MediaWiki engine – is also available for users to discuss contradictory ideas and leave notes for other stakeholders. The top-level pages are as follows:

- A *Vision* page to define the product position, stakeholders, assumptions, dependencies, needs, and features;
- a *Use Case* page to define actors, goals, and use cases;
- a *Supplementary Specification* page to define functional and non-functional requirements, standards, legal notes, test cases and traceability links.

Each of these top-level pages is divided into multiple sections. Users can manually edit the static sections content, such as the introduction to the SRS document. The dynamic sections, on the other hand, are populated by the pre-defined queries embedded in the page markup every time the page is requested by a user. This way, it is ensured that the top-level pages are always presenting the latest state of a ReqWiki's content. In addition, a list of user-defined terms is automatically generated at the bottom of all three top-level pages and presented as a glossary to aid users in reflecting about the correct usage of the terms. The presence of this feature in ReqWiki helps to establish a common language among stakeholders and promotes consistency.

Each entity in ReqWiki, like a use case description, is uniquely identified by a URI in the system and can be retrieved via its title or the full-text search feature that the MediaWiki engine offers. At this point, users can start defining entities

UC/Manage Tasks

Description	The manager receives a customer service request. The manager directs the operation for creating, updating, deleting and querying a task. Some operations use either the automatic or manual task assignation functionality that were defined in the Supplementary Specification document.
Level	user-goal
Primary Actor	A / Manager
StakeHolders	Manager, Senior technician, Junior technician
Pre-Conditions	The manager must be identified and authenticated in the application.
Success end condition	The task is created and assigned to the technicians with status Assigned. The tasks is updated and assigned to the technicians with status Assigned. The task is queried. The task is deleted. The system logs.
Failure end condition	The task is created with status Submitted. The task is updated with status Submitted. The task is not deleted.
Features	Manage Task

ReadabilityMetrics on UC/Manage_Tasks (View)

Content	Type	Start	End	Features
The tasks is updated and assigned to the technicians with status Assigned.	Passive Voice	686	760	▪ The sentence has been detected as passive and can be improved by changing the verb phrase

Writing Quality on UC/Manage_Tasks (View)

Content	Type	Start	End	Features
The tasks is	Grammar	686	698	▪ problem: Wrong Auxiliary Verb ▪ suggestion: The task is

Fig. 8. Results from the NLP services 'Readability Metrics' and 'Writing Quality' executed on a use case, stored inside a ReqWiki page

like actors or goals, by clicking on the provided links in the top-level pages. For example, in the *Use Case* page, a user can define a new use case description by clicking on the "Create a use case" link. The link takes the user to the corresponding semantic form to define use case attributes, such as *actors* or *main success scenario*. Once the user saves the form, a new page with the use case description is created in the wiki and the user input data embedded in the pre-defined ReqWiki template for a use case is stored in the generated page. The new page is automatically annotated with semantic metadata, so when the user navigates back to the top-level *Use Case* page, he will see the new use case in a table that is generated by the in-line query embedded in the page.

B. Invoking NLP Services

In order to perform an NLP analysis, like quality assessment, on the SRS content, a user first clicks on the 'Semantic Assistants' link in the wiki menu (Fig. 3, left). This interaction triggers the proxy to inject the NLP interface to the user's browser so that he can select a service to invoke on the ReqWiki content (Fig. 5). Additionally, the user can choose a place for the results to be written. Optionally, using the collection feature in the Semantic Assistants NLP user interface, the user can browse to other pages of the wiki, add them to his collection, and invoke a service on all of them at once. When the service execution is completed, the NLP service results are inserted into their corresponding ReqWiki templates and stored in the user's place of choice.

Fig. 8 shows the results of the *Readability Metrics* and *Writing Quality* services invoked on a use case description page in the wiki. The results show the defects of different types (Grammar and Passive Voice) found by the NLP service and their exact start and end offsets in the text. Also, where available, suggestions are provided to the user on how to correct the defect found by the pipelines.

Fig. 9 is an example of the output from the *Document Indexer* service, which creates a book-of-the-book style index of SRS content. This is especially useful in early phases of RE, e.g., to analyze existing domain-specific documents.

VI. Evaluation

We conducted a case study to evaluate our ReqWiki system along two dimensions, namely, *usability* and *effectiveness*. For both studies, a total of 22 students from one undergraduate and one graduate level software requirements engineering course

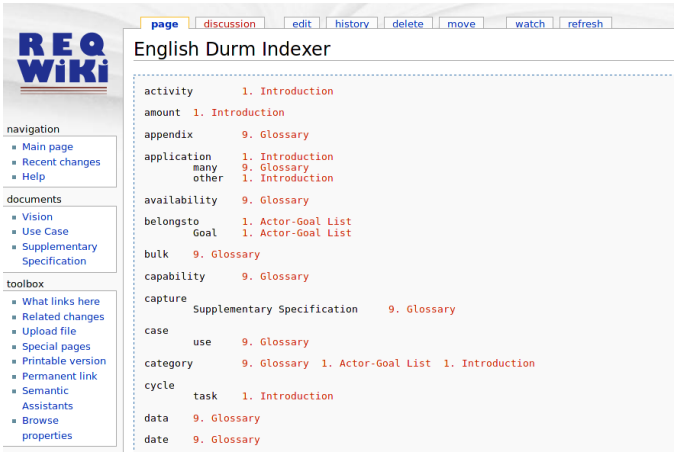

Fig. 9. NLP-generated 'back-of-the-book index'

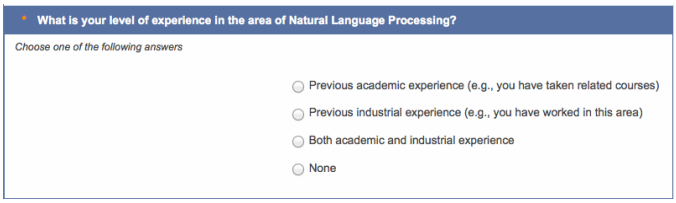

Fig. 10. Sample question from the user study

were teamed up in groups of two to collaboratively create SRS documents of a hypothetical software project using ReqWiki.

The main question in usability evaluation was to see if users with fair or no background knowledge about NLP can leverage ReqWiki's NLP capabilities. After being introduced to ReqWiki and its semantic capabilities, the users developed their first artifacts (Vision document). The participants were then introduced to NLP services, which were deployed throughout the study. At the end of the course, we provided students with a web-based anonymized questionnaire, in which we explicitly asked them about their background knowledge in the field of NLP (see Fig. 10) and how easy to use they found the Semantic Assistants interface in ReqWiki to use, from a Likert-scale of *Very Easy* to *Very Difficult*.

Fig. 11 presents the results gathered from the students' feedback. As can be observed, 72% of the students with no previous background knowledge of NLP, 40% with mere textbook background knowledge, and 50% with professional experience in the field of NLP found the ReqWiki interface *Very Easy* or *Easy* to use, followed by an average of 40% to be *Neutral*. This corroborates our hypothesis that users do not require background knowledge in NLP to make use of sophisticated semantic support, provided that it is offered through an intuitive interface.

For the effectiveness evaluation of ReqWiki, we examined the impact of the integration of NLP capabilities on the quality of the developed SRS. Once the data was entered into the wiki and related pages were cross-linked, students were asked to perform a manual quality assessment of their SRS documents to correct defects, such as weak phrases or passive voice. When the documents were submitted to markers, we counted the number of defects that were missed by students in their documents. For the second revision, we deployed the Semantic Assistants services on their wiki systems and asked them to perform an additional quality assessment, taking the suggestions of the analysis services as provided in the wiki interface into account. Eventually, after the submission of the second revision, the markers again counted the number of defects remaining in

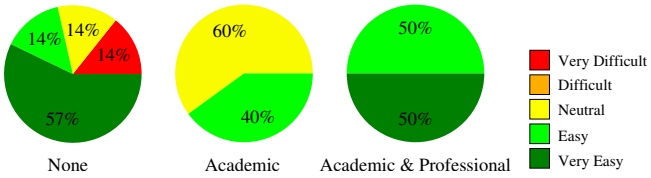

Fig. 11. Level of NLP background knowledge versus ReqWiki usability

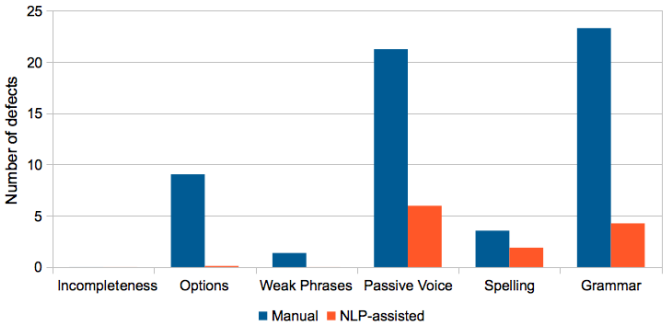

Fig. 12. Comparison of the average number of defects found in SRS documents in two revisions, without (left) and with (right) NLP service support

the students' SRS document. Fig. 12 shows a comparison of the average number of defects found in students' assignments, comparing the manual approach with our NLP services. As can be seen from Fig. 12, using NLP services for quality assessment purposes significantly reduced the average number of defects, thus helping to improve the quality of SRS documents.

A threat to the external validity of our findings is related to conducting an experiment with students and generalizing it to actual software engineers working in industry. To mitigate this effect, we intentionally included graduate students in addition to the undergraduates, since most of our graduate students either have industrial experience or are working in the software industry while studying part-time. In our results, both groups performed similar with respect to defect reduction, which indicates that ReqWiki usage benefits software engineers independent of their background. Another fact posing a threat to our methodology is the lack of a control group in our experiment, i.e., enabling the NLP services for only half of the students and comparing the results to the control group. This threat was inevitable in our experiment, since all the students must have had access to the same resources to ensure fairness in grading of their assignments.

VII. Related Work

Using wikis for RE has been the subject of a number of studies. Decker et al. [11] are among the first to demonstrate wiki capabilities as an asynchronous collaborative tool to support active stakeholder participation in RE, by contrasting it to plain office suites and dedicated RE tools. They introduce the Software Organization Platform (SOP) wiki, which is based on the MediaWiki engine and has an underlying document structure compliant with Cockburn's templates [4].

With the introduction of semantic capabilities to wiki engines, numerous semantic engines have been tailored for RE purposes to promote shared and unambiguous understanding of requirements among stakeholders. Semantic wikis allow annotations to be added to wiki pages and stored in a knowledge base or connected to background ontologies, making the inherent structure of a wiki accessible for machines. The semantic annotations are then used for formalizing and reasoning [19] on the requirements. WikiReq [20] is a semantic wiki that exploits the Semantic MediaWiki engine to manage software

requirements in combination with a goal-oriented language. A noteworthy feature of WikiReq is its ability to partially automate the translation of organizational business processes and system artifacts from the requirements description by directly transforming semantically annotated requirements to Eclipse Modeling Framework (EMF) instances.

SoftWiki [21] aims at 'semantifying' requirements by developing an ontology that defines core concepts of requirements engineering and the way they are interrelated. SoftWiki is essentially an application that adopts a wiki paradigm by providing users with wiki-like features, such as rollback changes or a facility to discuss requirements, rather than an extension to an existing wiki engine. It allows users to express requirements, link them to application parts and their usage context and classify them using a pre-defined topic structure. Finally, SoftWiki enables semantic interoperability with further tools by exporting requirements in RDF format according to its underlying ontology.

The approach most similar to ReqWiki is SmartWiki [22], which also utilizes SMW templates for managing UC requirements. SmartWiki includes some basic support for 'heuristic feedback', like detected passive voice defects. However, it does not provide for the integration of arbitrary NLP services through a service-oriented approach like ReqWiki. The SmartWiki system is also not publicly available.

In contrast to these existing works, we envision with ReqWiki a system that takes a collaborative approach, where humans make use of automated NLP assistants, both working on a shared formal representation in the form of ontologies. In doing so, we leverage the advantages of ontologies for the formalization of requirements, while at the same time benefiting from state-of-the-art NLP techniques to manage unstructured natural language, both tightly integrated into a wiki system for intuitive use. Finally, our work is the first that systematically investigated the potential quality improvement through NLP support and its acceptance by software engineers.

VIII. Conclusion and Future Work

Requirements engineering is one of the most important phases in the software development process, with more than 50% of failed projects attributed to poor RE. Despite these facts, many projects, in particular in small and mid-size companies, still rely on office tools without built-in support for RE. In this research, we showed how modern semantic techniques can be combined with NLP services in a collaborative web-based wiki platform to improve RE for all involved stakeholders. We are currently in the process of refining the user interface with suggestions obtained through the user study. In ongoing work, we investigate the connection of ReqWiki with other software engineering tools, like Eclipse, to improve traceability across different software artifacts.

Acknowledgment

We thank the Concordia software engineering students that participated in the evaluation of ReqWiki and Srinivasan Rajivelu for his contributions to the implementation of ReqWiki.

References

[1] A. van Lamsweerde, *Requirements Engineering*. Wiley, 2009.

[2] M. Luisa, F. Mariangela, and N. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, pp. 40–56, 2004, 10.1007/s00766-003-0179-8.

[3] L. Kof, *Text Analysis for Requirements Engineering: Application of Computational Linguistics*. Saarbruecken, Germany: VDM Verlag Dr. Mueller, 2007, ISBN 978-3-8364-4525-2.

[4] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2000.

[5] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Use Case Approach*, 2nd ed. Pearson Education, 2003.

[6] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters, *Text Processing with GATE (Version 6)*. University of Sheffield, Department of Computer Science, 2011. [Online]. Available: http://tinyurl.com/gatebook

[7] I. Hussain, O. Ormandjieva, and L. Kosseim, "Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier," in *7th International Conference on Quality Software (QSIC 2007)*. IEEE Computer Society, 2007, pp. 209–218.

[8] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Transactions on Software Engineering and Methodology*, vol. 14, no. 3, pp. 277–330, 2005.

[9] P. Sawyer, P. Rayson, and K. Cosh, "Shallow Knowledge as an Aid to Deep Understanding in Early Phase Requirements Engineering," *IEEE Trans. on Software Engineering*, vol. 31, no. 11, pp. 969–981, 2005.

[10] P. Kruchten, *The Rational Unified Process: An Introduction*, 3rd ed. Addison-Wesley, 2003.

[11] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-Based stakeholder participation in requirements engineering," *IEEE Software*, vol. 24, no. 2, pp. 28–35, 2007. [Online]. Available: http://dx.doi.org/10.1109/MS.2007.60

[12] C. Silveira, J. P. Faria, A. Aguiar, and R. Vidal, "Wiki-Based Requirements Documentation of Generic Software Products," in *Proceedings of the 10th Australian Workshop on Requirements Engineering (AWRE05)*, Melbourne, Australia, 2005.

[13] B. Sateli and R. Witte, "Natural Language Processing for MediaWiki: The Semantic Assistants Approach," in *Proceedings of the 8th International Symposium on Wikis and Open Collaboration (WikiSym '12)*. ACM, 2012. [Online]. Available: http://dx.doi.org/10.1145/2462932.2462976

[14] R. Witte and T. Gitzinger, "Semantic Assistants – User-Centric Natural Language Processing Services for Desktop Clients," in *3rd Asian Semantic Web Conference (ASWC 2008)*, ser. LNCS, vol. 5367. Bangkok, Thailand: Springer, 2008, pp. 360–374. [Online]. Available: http://rene-witte.net/semantic-assistants-aswc08

[15] R. Mudge, "The Design of a Proofreading Software Service," in *Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids (CL&W 2010)*, 2010.

[16] W. H. DuBay, *The Principles of Readability*. Impact Information, 2004.

[17] D. Schreck, V. Dallmeier, and T. Zimmermann, "How documentation evolves over time," in *IWPSE '07: Ninth international workshop on Principles of software evolution*. New York, NY, USA: ACM, 2007, pp. 4–10.

[18] P. A. Laplante, *Requirements Engineering for Software and Systems*. Auerbach Publications, 2009.

[19] P. Liang, P. Avgeriou, and V. Clerc, "Requirements reasoning for distributed requirements analysis using semantic wiki," in *Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE '09)*. IEEE Computer Society, 2009, pp. 388–393. [Online]. Available: http://dx.doi.org/10.1109/ICGSE.2009.61

[20] L. Abeti, P. Ciancarini, and R. Moretti, "Wiki-based requirements management for business process reengineering," in *2009 ICSE Workshop on Wikis for Software Engineering (WIKIS4SE '09)*, 2009, pp. 14–24. [Online]. Available: http://dx.doi.org/10.1109/WIKIS4SE.2009.5069993

[21] S. Lohmann, P. Heim, S. Auer, S. Dietzold, and T. Riechert, "Semantifying Requirements Engineering – The SoftWiki Approach," in *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS '08)*, 2008, pp. 182–185.

[22] E. Knauss, O. Brill, I. Kitzmann, and T. Flohr, "SmartWiki: Support for High-Quality Requirements Engineering in a Collaborative Setting," *4th Workshop on Wikis for Software Engineering at ICSE 2009, Vancouver, Canada*, May 2009.