Computers in Industry xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Computers in Industry

journal homepage: www.elsevier.com/locate/compind

An exploratory study on ontology engineering for software architecture documentation

K.A. de Graaf [a,*], P. Liang [a,b], A. Tang [c], W.R. van Hage [d], H. van Vliet [a]

[a] VU University Amsterdam, Amsterdam, The Netherlands
[b] Wuhan University, Wuhan, China
[c] Swinburne University of Technology, Melbourne, Australia
[d] SynerScope B.V., Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 20 June 2013
Received in revised form 11 April 2014
Accepted 16 April 2014
Available online xxx

Keywords:
Ontology engineering
Software architecture
Software ontology
Ontology-based documentation
Knowledge acquisition
Knowledge management

Abbreviations:
SA, software architecture
AK, architectural knowledge
HTML, hypertext markup language
WYSIWYG, what you see is what you get
GUI, graphical user interface
CF, contextual factor

ABSTRACT

The usefulness of Software Architecture (SA) documentation depends on how well its Architectural Knowledge (AK) can be retrieved by the stakeholders in a software project. Recent findings show that the use of ontology-based SA documentation is promising. However, different roles in software development have different needs for AK, and building an ontology to suit these needs is challenging. In this paper we describe an approach to build an ontology for SA documentation. This approach involves the use of typical questions for eliciting and constructing an ontology. We outline eight contextual factors, which influence the successful construction of an ontology, especially in complex software projects with diverse AK users. We tested our 'typical question' approach in a case study and report how it can be used for acquiring and modeling AK needs.

1. Introduction

For a Software Architecture (SA[1]) to be useful, it needs to be understood [1] and effectively communicated between its designers and its users [2]. Documenting the Architectural Knowledge (AK) in a SA facilitates its communication and understanding. AK can be defined as "*the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context/environment*" [3].

It is common for AK to be documented in file-based documents such as text files, diagrams, source code, and meeting notes. This however introduces practical limitations as it is hard to describe AK unambiguously and comprehensively for all of its different uses. Additionally, AK is highly interrelated, making it hard to support the needs of different users by a single document indexing structure.

In order to address these limitations, we study ontology-based SA documentation, in which AK is made explicit and unambiguous by applying a semantic structure. An ontology refers to a formal domain model describing its concepts and relationships among concepts [4]. In [5] de Graaf et al. report on a controlled experiment that was conducted in a large and complex industrial software project. The results provided empirical evidence that ontology-based SA documentation is more effective and efficient for AK retrieval than file-based SA documentation.

* Corresponding author. Tel.: +31 6 18 47 91 32
E-mail addresses: ka.de.graaf@vu.nl (K.A. de Graaf), liangp@cs.vu.nl (P. Liang), atang@swin.edu.au (A. Tang), willem.van.hage@synerscope.com (W.R. van Hage), hans@cs.vu.nl (H. van Vliet).

[1] See 'Abbreviations' under Article Info for a list of abbreviations used in this paper.

http://dx.doi.org/10.1016/j.compind.2014.04.006
0166-3615/

The authors of this paper had to implement ontology-based SA documentation in the software project in which this industrial experiment was executed. To do so, we had to consider what ontology engineering approach would be suitable. This paper is about building the ontology structure for ontology-based SA documentation. It is not about instantiating (or 'populating') this ontology or about the experiment in which the ontology was populated, as described in [5].

Developing an ontology for a software project in industry should be economically feasible, i.e. it should be efficient and accurate, for organization and individual users [6]. If the ontology is inaccurate, documentation users will not retrieve or understand the AK that they need. Users would lose interest and confidence in ontology-based SA documentation, and revert to other means to get the required AK.

In the context of large software projects it can take much time and effort to develop an accurate ontology. Knowledge acquisition from many diverse stakeholders, each having their own needs and views [1] of AK, is required to build an accurate ontology. These users of AK are generally pressed for time and their primary interest is seldom about making documentation. Moreover, the AK needed by users in large software projects is often domain specific and complex.

Developing the ontology is not a one-time effort because the AK needs of SA documentation users shift over time. For example, during development users will be interested in AK that relates requirements to the components implementing those requirements, while during integration testing (other) users might be interested in relations between software releases and requirements coverage. Shifting AK needs necessitates a regular evaluation and revision of the constructed ontology.

In this paper we describe eight contextual factors in software projects. These contextual factors influence ontology engineering for SA documentation, especially in large and complex projects. We devised a 'typical question' approach for ontology construction that takes the contextual factors in large and complex software projects into account. In this approach typical questions about AK are acquired from SA documentation users and used to build an ontology. These typical questions are frequently asked by AK users[2] during their everyday tasks, i.e. questions that represent their everyday AK needs.

We applied the 'typical question' approach in an exploratory industrial case study which provided several insights. In the case study we explored how well our 'typical question' approach was applied to construct a useful ontology by acquiring and modeling AK needs of many diverse users that use SA documentation in different projects and product lines. The 'typical question' approach can continuously refine the AK ontology when AK needs evolve.

This paper is motivated by the lack of applied ontology engineering approaches for constructing an ontology for SA documentation. We make the following contributions:

- Illustrate a 'typical question' approach for constructing the ontology used in ontology-based SA documentation.
- Outline important contextual factors that influence ontology engineering for SA documentation in software projects.
- Demonstrate how the 'typical question' approach can be applied through an exploratory case study in an industry software project.

This paper is organized as follows. Background of ontology-based SA documentation, ontology engineering, knowledge acquisition, and Grounded Theory is given in Section 2. Section 3 describes our 'typical question' approach for ontology construction. Section 4 describes the contextual factors that influence ontology engineering for SA documentation in software projects. Section 5 reports the exploratory case study and the lessons learned from it. Section 6 presents related work and Section 7 concludes this paper.

[2] We consider AK users the same as SA documentation users in this work.

2. Background

2.1. Ontology-based SA Documentation

Many relationships exist between AK in SA documentation, e.g., between requirements, decisions, and components. Consider a single decision recorded in a document. A software engineer needs to know how the decision impacts components and interfaces s/he is working on. When evaluating the decision a software architect is interested in its rationale, alternatives, related decisions, and related requirements. A quality assurance manager might need to know all quality attributes that the decision impacts or vice versa.

The linear organizational nature of file-based documentation makes it hard to provide a structuring of AK that is suitable for every user. This structuring can be done using views [1], perspectives, or some other sectioning in a Table of Contents. However, once written down, the structuring of AK becomes static and linear in file-based documentation causing difficulties for users that want to retrieve AK that is unsupported by the structure. Furthermore, it can be troublesome to describe AK unambiguously in documentation, i.e. clearly, consistently, with explicit notations for AK and AK interrelations, especially when documentation and AK evolves.

On the other hand, an ontology provides a type of AK structuring which facilitates AK retrieval by different types of documentation users. Knowledge in an ontology can be searched by concepts and reasoned with. The type (or 'class') of AK becomes explicit in an ontology and the relationships between AK have explicit semantics, e.g., '*realized by*' and '*results in*'. This improves the efficiency and effectiveness of AK retrieval [5].

Users of ontology-based SA documentation can retrieve AK using a semantic wiki [7] or plug-ins in text-editors [8]. Semantic wikis allow for web-based visualization and management of (ontology and its instances in) knowledge bases and semantic-enhanced search facilities such as graph-like exploration, faceting, and filtering of knowledge instances based on semantic interrelations.

An advantage of ontology-based documentation in a semantic wiki over traditional wiki and hypertext systems is the use of semantic relationships. Hyperlinks provide pointers to information but the pointers do not specify the meaning of relationships. Whereas semantic relationships can be specified in ontology-based documentation, each with an explicit name, type, and meaning. Moreover, one can assign formal properties to semantic relationships, e.g., transitivity and symmetry, which allow for automatic inference and reasoning.

The ontology-based SA documentation that was used in the experiment in [5] consists of a semantic wiki (see http://archimind.nl/archimind/) in which fragments, sections, and diagrams from file-based SA documentation are stored as HTML in wikipages. Basic version control, a WYSIWYG editor to update wikipage content, and change history, support maintenance activities. Using semantic annotation, the AK in wikipage content (e.g., a description of '*GUI*') is highlighted and instantiated as belonging to a class of AK (e.g., component) and having relationships (e.g., 'satisfies') to other AK (e.g., requirement '*login*').

When an AK user views an AK instance, e.g., *GUI*, s/he can clearly see that *GUI* is a component, with relationship '*satisfies*' to requirement *login*, and what SA documentation text (in wikipages)

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

details on *GUI*. A user viewing (a list of) AK instances can see what wikipages describe each of the AK instances. This makes AK traceable across SA documentation content in a project or multiple projects.

The experiment in [5] involved AK users answering questions such as "*what decisions have been made around component XX*". Use of the ontology-based approach (with semantic wiki) to retrieve AK was more efficient and effective than use of a file-based approach. The underlying reason is that experiment participants used more fitting organisation and explicit AK descriptions in the ontology-based approach compared to the file-based approach.

Lopez et al. report evidence in [9] that ontology-based SA documentation improves efficiency and effectiveness when retrieving design rationale from SA documents. Jansen et al. provide evidence in [8] that use of ontology-based documentation leads to increased architectural understanding during architectural reviews. Su et al. propose use of an ontology for visualization and non-linear navigation of SA documentation which reduces the cognitive load on its users [10].

2.2. Ontology engineering for SA documentation

The ontology used in ontology-based SA documentation determines what AK one can retrieve using its structure and semantics. One can use a predefined ontology (as in [7]) or build an ontology specifically for a software project domain. A domain specific ontology can be built, e.g., by letting document authors and architects identify AK concepts in existing SA documentation [8].

The approach proposed by Jansen et al. in [8] is, to our current knowledge, the only approach aimed at deriving a domain specific ontology for ontology-based SA documentation. Their use of existing documentation however has several limitations:

- Existing SA documentation might not be available.
- Existing SA documentation might not convey the AK needed by all SA documentation users.
- Experts that identify AK concepts from SA documentation might not know the AK needs of all users.

Building and maintaining an ontology for a certain domain (such as in this case the domain of AK) is called ontology engineering. An overview of the ontology engineering phases, adapted and simplified from [11], is depicted in Fig. 1. Arrows in this figure represent a transition to another phase.

The process starts with the domain analysis phase which includes identification of the domain scope and the AK needs of users of SA documentation.

AK concepts are derived from AK needs and used to build a domain ontology in the conceptualization phase. The quality of the constructed ontology is then evaluated in the evaluation phase. Domain analysis, conceptualization and evaluation is iterated until a satisfactory ontology is obtained.

After construction the ontology is used to annotate and retrieve AK from SA documentation. The ontology is maintained to cope with evolving AK needs and concepts.

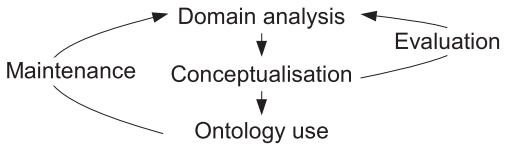


Fig. 1. Ontology engineering phases.

2.3. Knowledge acquisition

Building an AK ontology requires knowledge acquisition during the ontology engineering phases described in the previous section. Knowledge acquisition is part of building knowledge-based systems in general. The suitability of knowledge acquisition approaches differs per domain. A separation can be made between top-down, middle-out and bottom-up approaches.

Top-down approaches start modelling based on a general model, with extension and refinement to suit specific needs. These approaches can be efficient and accurate for well-defined domains in which an ontology engineer knows quite well about the general concepts, questions, tasks, and use-cases. Bottom-up approaches are used to build a model based on specific domain knowledge. These are suitable for domains in which an ontology engineer cannot predetermine all the concepts, questions, tasks, and use-cases. Such domains can be broad, multidisciplinary, or novel. Middle-out approaches combine a top-down and bottom-up approach. These work well for domains that are partially well-defined and partially specific.

2.4. Grounded Theory

Our 'typical question' approach makes use of coding techniques from Grounded Theory in which a domain theory is generated by empirical generalization [12]. This is a bottom-up approach to knowledge acquisition (see previous Section 2.3).

First, data is collected in the domain under investigation, and patterns that indicate concepts are identified from this data. The concepts identified are aggregated into categories. Second, properties of the identified categories are developed by constantly comparing the categories with collected domain data. These properties are developed with respect to a 'core' or 'central' category [13] that is the subject of investigation in the domain. Memos capture thoughts about the possible concepts, categories, and relationships in the domain data. Finally, if no new properties can be identified from domain data (categories are 'saturated'), the domain model is compared to literature.

Urquhart et al. discuss several myths about Grounded Theory in [14] and conclude that Grounded Theory is a rigid and flexible method that is suitable for use in information systems research. Grounded Theory is able to generate theories that are relevant to practitioners [15] and that are 'grounded in data' [12].

Grounded Theory can be used during ontology engineering when analysing a domain and constructing an ontology. An ontology engineer starts with the collection and analysis of domain data in the domain analysis phase (see Fig. 1 and Section 2.2). Similarly, data is collected and analyzed in a domain of interest when one uses Grounded Theory in the domain analysis phase. In the ontology conceptualization phase an ontology engineer can use Grounded Theory to create classes (or 'categories') and relationships between classes.

3. Ontology engineering using the 'typical question' approach

We devised an ontology engineering approach that uses typical questions to create an ontology in the context of a large and complex project at Océ Technologies.

The 'typical question' approach focuses on the elicitation of questions that AK users normally try to answer during their use of SA documentation. We want to understand what AK users typically ask of the SA documentation. Knowing the detailed knowledge needs of AK users allows one to create SA documentation that is optimal for its users and therefore cost-effective [16].

The typical questions convey what concepts and relationships AK users want to retrieve from SA documentation. An ontology

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

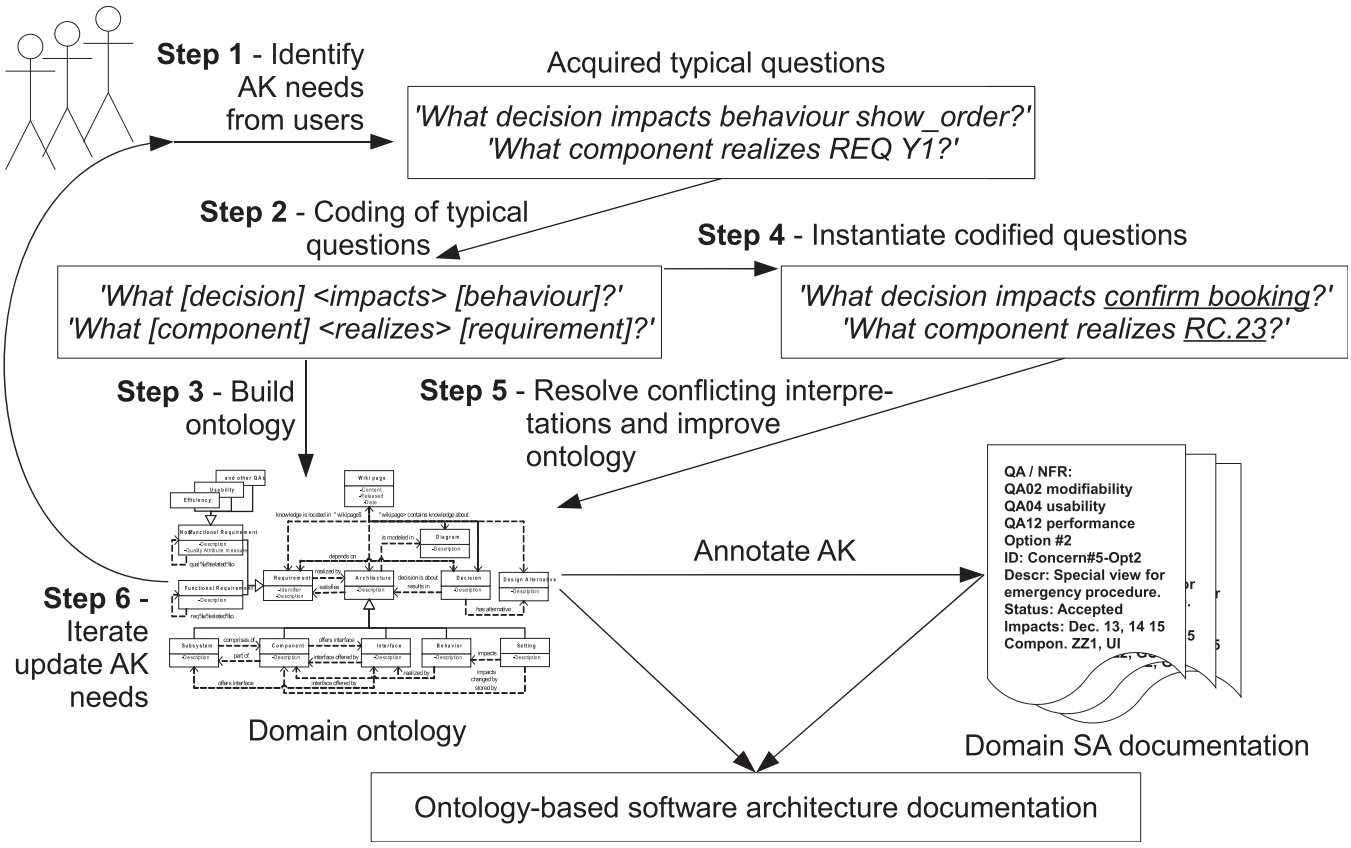


Fig. 2. Overview of the proposed 'typical question' approach for ontology engineering.

engineer identifies these concepts and relationships on basis of 'typical questions' to construct an ontology. This ontology is subsequently used to support retrieval of the concepts and relationships from SA documentation.

Fig. 2 depicts an overview of our approach. In this approach the process of creating an ontology requires one or more steps in each ontology engineering phase.

In the domain analysis phase of ontology engineering, an ontology engineer acquires typical questions from documentation users to capture their AK needs (step 1). In the conceptualization phase these typical questions are codified (step 2) and modeled in an ontology (step 3) using coding mechanisms from Grounded Theory. In the evaluation phase typical questions are instantiated (step 4) to resolve conflicting interpretations of AK concepts and to evaluate and improve the constructed ontology (step 5).

In the ontology use phase, existing file-based documentation is annotated and the semantic wiki is used for AK retrieval. During the maintenance phase the techniques to acquire the typical questions (in step 1) are iterated to detect changes of AK needs and if the ontology needs to be updated (step 6).

Interviews, daily logging of typical questions, and mailing lists are techniques that can be used to acquire typical questions. These are further detailed in Section 4.1.

3.1. Domain analysis phase (step 1)

In the first step of our approach AK needs are identified from all users, or a representative subset of users, by acquiring their typical questions about AK. These are questions that are representative of what AK users ask themselves during their everyday activities, e.g., during design, development, quality assurance, testing, etc. For example, a developer might ask: "*what components and behavior are impacted by my change in this subsystem?*". Snippet 1 gives an example of two typical questions.

> Snippet 1. Example of typical questions:
> - "Which module assures adherence to requirement '24 – admin login'?"
> - "What quality attributes are realized by subsystem 'transaction handler'?"

Normally such typical questions are answered by reading architecture and design documents, inspecting source code, or consulting colleagues. Instead we use these typical questions in our approach to build an ontology which in turn provides users with structure and semantics to answer these typical questions from ontology-based SA documentation.

A core idea behind the use of typical questions is the assumption that the AK needed in any use-case, scenario, or task can be accurately represented as a set of questions that should be answered. As such, acquiring sets of typical questions allows for a fine-grained and detailed specification of the AK needed for tasks, use-cases, and scenarios. Recording or recalling these typical questions can reduce difficulties for users in articulating domain knowledge [17], i.e. increase efficiency by reducing effort. This addresses difficulties in acquiring AK needs of diverse users in a complex and multi-disciplinary domain.

Benefits can be gained by the use of typical questions that are (1) efficiently acquired from many users (2) tangible for users, (3) accurately represent AK needs and daily practice of users, (4) do not require extensive abstract thinking from users when acquired, yet (5) convey much conceptual information, and (6) can be used throughout the ontology engineering phases. These benefits help

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

to minimize the effort required from individual users of SA documentation, as well as the total effort required in complex domains with many diverse users.

3.2. Conceptualization and Evaluation Phase (Steps 2-5)

In the ontology conceptualization phase an ontology engineer identifies AK concepts from typical questions in order to build an ontology. AK concepts are identified from phrases, i.e. one or more words, in the acquired typical questions. Phrases in typical questions are classified as representing a class, relationship, or attribute in the ontology. This is done by applying coding techniques from Grounded Theory, discussed in Section 2.4.

An ontology engineer starts with open coding in which textual data, a typical question in this case, is broken down into discrete parts, e.g., words and phrases. These discrete parts are examined in detail and classified (or 'categorized'[3]) by comparing data for conceptual similarities and differences [13]. Snippet 2 gives examples of typical questions in which open coding, or 'labeling', i.e. assigning conceptual names (in italic between square-brackets), is applied to phrases that refer to AK concepts. Various labels are applied to illustrate possible conceptualizations in open coding. This process corresponds to the first phase of the overall process of Grounded Theory described in Section 2.4.

Snippet 2. Example of initial open coding of typical questions:
- "Which module [*module, component, hardware, subsystem*] assures adherence to [*constraints, satisfies, realizes*] requirement '24 – admin login' [*functional or non-functional requirement, Quality Attribute*]?"
- "What quality attributes [*quality attribute, non-functional requirement*] are realized by [*constraints, satisfies, realizes*] subsystem 'transaction handler' [*subsystem, component, module*]?"

Axial (or 'theoretical') coding is then used to (re)assemble and interrelate the classes, identified during open coding, to their sub or superclasses. Selective coding is then used to integrate, interrelate and refine the classified concepts in an ontology based on the central 'theme', 'idea', or 'category' [13] of the investigation. This central theme is: "*AK that users need to retrieve from SA documentation*".

Axial and open coding are used in the second phase of Grounded Theory described in Section 2.4. The described coding approach is based on the method described by Strauss and Corbin in [13]. Snippet 3 shows an example of coded phrases in typical questions after completion of above coding steps.

Snippet 3. Example of refined codified classes and relationships in typical questions:
- "Which component [*component (class)*] assures adherence to [*satisfies (relationship)*] requirement '24 – admin login' [*functional requirement (class)*]?"
- "What quality attributes [*non-functional requirement (class)*] are realized by [*realized by (relationship)*] subsystem 'transaction handler' [*subsystem (class)*]?"

[3] The term 'category' is used in Grounded Theory literature, however we adopt the term 'class' instead of 'category' for consistency with the rest of the paper.

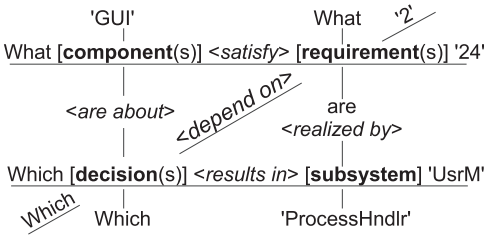


Fig. 3. Ideal mapping of coded typical questions into AK ontology structure.

An ontology engineer may decide to execute one or more iterations of the coding steps, e.g., after evaluation by users. Ideally the AK concepts in typical questions can be directly mapped to an ontology structure. This is illustrated in Fig. 3 in which classes are denoted between square brackets, relationships between angle-brackets, and instances of classes between single-quotes.

In practice refinement of the AK concepts is needed as not all users will use the same phrasing for their AK needs in typical questions. For example, phrases 'setting', 'configuration', 'variability' and 'preset' might all refer to (1) the same AK concept, (2) distinct yet related AK concepts (e.g., subclasses), or (3) different AK concepts in the domain. An example for refinement of relationships between AK concepts is the set of phrases 'constraints', 'required', and 'realizes'. Even if the same name is used for a concept in many typical questions, we cannot assume based on this frequency that all users have the same interpretation of this concept. Users might mean different things with the same phrase.

An ontology engineer evaluates the interpretation of AK concepts by presenting users with different instantiations of typical questions that have been coded and classified. Phrases in these typical questions that have been coded to an ontology class or relationship are replaced with different instances of the same class or relationship, e.g., instances '*requirement X1*', '*R2 - admin login*', and '*Req. 3*' for class requirement. This is akin to validation in Grounded Theory where respondents in a study comment on how well the theory, e.g., represented as a 'story', fits their case [13].

An ontology engineer subsequently asks users feedback about the representativeness, correctness, and accuracy of these typical questions with different instances of AK concepts. This supports identification of conflicting interpretations of AK concepts (step 5) between users working in diverse roles and disciplines. Snippet 4 gives the typical questions from Snippet 3 with different instances of AK concepts.

Presenting typical questions with different instances of AK concepts allows users to evaluate the concepts in different contexts. The interpretation of an AK concept may differ between AK users when they are presented with different instances of the AK concept. Part of the evaluating users may remark that "*allow backup scheduling*" (in Snippet 4) is a non-functional requirement or feature instead of a functional requirement. This identifies incorrect or inaccurate classification of AK concepts in typical questions.

Snippet 4. Example typical questions with instantiated AK concepts for evaluation:
- "Which component satisfies functional requirement '*18 – allow backup scheduling*'?"'
- "What non-functional requirements are realized by subsystem '*Order configuration*'?"

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

Grounded Theory coding allows an ontology engineer to construct an ontology "grounded in data" with relative high efficiency and accuracy and without requiring extensive effort from documentation users. This requires minimal guidance by domain experts which might not be readily available in many software projects.

After the ontology is conceptualized an ontology engineer may decide to compare the ontology to other ontologies, e.g., in literature. This allows identification of useful domain-independent AK concepts that may be missing in the constructed domain specific ontology.

3.3. Ontology maintenance phase (step 6)

When AK needs shift and concepts evolve the ontology has to be maintained to remain accurate. The ontology engineer can identify a shift in AK needs by re-acquiring typical questions (step 1). An ontology engineer then compares the newly acquired typical questions and those previously acquired. If AK needs have significantly shifted, steps 2–5 of the approach can be repeated to update the ontology. Changes in AK concepts are detected in step 4 of the approach when newly acquired typical questions are instantiated and evaluated by AK users.

The necessity of reacquiring typical questions should be estimated or planned at the start of each software project phase. Estimating the necessity of reacquiring typical questions or the required accuracy of this reacquisition is outside the scope of this paper.

4. Contextual factors in ontology engineering

In the introduction section, we briefly described how the development of an SA documentation ontology is affected by its users and the software project context. These Contextual Factors (CF) influence the use of the 'typical question' approach. In agile software development, we learn that contextual factors influence the successful adoption of agile practices [18]. We recognize that these factors are also applicable in terms of producing SA documentation.

Production of SA documentation takes place in the context of software development projects and the context of a project and its products is captured in SA documentation. Moreover, users of SA documentation in a project determine what AK is relevant to capture in SA documentation. Several characteristics of the SA documentation users influence how one can most effectively find out what AK is relevant to these users.

An ontology engineer would need to consider these factors whilst building a suitable ontology for SA documentation:

CF1 Number of AK users – impacts on the extent of AK needs acquisition. For example, interviewing a large number of AK users might be infeasible in a project.
CF2 Accessibility of AK users – communication with AK users can be constrained by their accessibility, e.g., in terms of location and schedule.
CF3 Commitment of AK users – influences how much time and effort users are willing to spend on providing their AK needs and ontology evaluation.
CF4 Diversity of AK users – the role, experience, and education background can impact on the interpretation of AK concepts between users.
CF5 Product domain complexity and specificness – influences the efforts required of an ontology engineer to understand and model AK concepts in a very specific and complex software product domain.
CF6 Product domain multidisciplinarity – impacts on how many different AK concepts, e.g., from the healthcare, embedded systems, and chemistry discipline, are needed by AK users.
CF7 Shifting AK needs – Users provide their AK needs based on their roles and the current tasks. When a project progresses and their tasks change, these AK needs may shift. Additional AK needs may be required to enhance an ontology.
CF8 Volatility of AK concepts – AK concepts can change over time. This affects the accuracy of the AK concepts initially captured in an ontology.

4.1. Contextual factors influencing the acquisition of typical questions

During the acquisition of typical questions, we noticed that contextual factors influenced what acquisition techniques we used. We have used three knowledge acquisition techniques (see Table 1) in our case study. We summarize, based on our experience, how contextual factors 'user accessibility' and 'user commitment' influence the performance of each acquisition technique.

Interviews with AK users allow an ontology engineer to clarify their responses and thoughts, e.g., using examples, and this allows an ontology engineer to acquire AK needs. Interviews however require high commitment, accessibility, and much time and effort from users and ontology engineer. Relatively high accuracy is traded off against lower efficiency.

A daily log, in which users consistently record their typical questions each day, requires a fair amount of accessibility to and commitment of users. Even though the time-efficiency and required effort is better than that of interviews, it is less accurate due to the lack of direct interaction with an ontology engineer. Accuracy becomes even lower when user commitment is low and AK needs are not consistently recorded every day.

The use of a mailing list provides time-efficient and effortless acquisition of typical questions, even with low user commitment and accessibility. The accuracy of AK needs acquisition is low as there is no direct interaction between documentation users and ontology engineer, however, more AK needs can be acquired. The use of a mailing list can prove to be very suitable for projects with many users and distributed development. Bürger et al. provide evidence in [19] that suggests that the use of e-mail is more efficient than face-to-face meetings.

The use of multiple acquisition techniques at the same time may be suitable in some situations. For example, interviews and a mailing list may be used at the same time for acquiring typical questions from a group of practitioners residing in the

Table 1
Evaluation of techniques for acquiring typical questions.

Technique for acquiring typical questions	Suitable for Contextual factors	Results in
Interviews	High user accessibility and commitment	High accuracy and low efficiency
Daily log	Medium user accessibility and commitment	Medium accuracy and efficiency
Mailing list	Low user accessibility and commitment	Low accuracy and high efficiency

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

same location as the ontology engineer and another group of practitioners working elsewhere.

4.2. Contextual factors in the domain analysis phase

During domain analysis an ontology engineer identifies what AK is needed by SA documentation users. If AK needs are overlooked documentation users will have less support to retrieve this AK. The possibilities for acquiring AK needs from users and the amount of AK needs in a software project influence how much time and effort is required from the ontology engineer and AK users for the identification of AK needs.

Large software projects typically have many (CF1) different (CF4) stakeholders that are users of AK, such as software architects, engineers, testers, and product managers. AK in large projects is often multidisciplinary in nature (CF6), conveying views from many diverse stakeholders who are both users of AK and experts in their respective domains (CF4).

We need to talk to a lot of AK users before all the AK needs for their roles (CF4) are clear, especially in complex software projects (CF5). This becomes difficult when AK users are not accessible (CF2) or committed (CF3) to cooperate [17] [6].

In our approach typical questions are used to capture domain-specific AK needs from many diverse users. Formulating questions requires relatively little effort from users and acquiring questions can be scaled up in large projects. The accessibility and commitment of AK users is addressed by selecting one of several techniques for acquiring typical questions listed in Table 1.

4.3. Contextual factors in the conceptualization and evaluation phase

After AK needs are identified, AK concepts can be derived from the AK needs and modeled in an ontology. If the ontology is inaccurate or ambiguous it will not support efficient and effective AK retrieval.

Conflicting interpretations of AK concepts are likely to occur with many users [17] (CF1) with diverse roles (CF4) that work from different disciplines (CF6). An ontology engineer might not have complete and thorough knowledge of all AK concepts and their interpretation in a software product domain that is very specific and complex (CF5). Reuse of a generic ontology is limited in such domains. Domain experts can help with ontology modeling and evaluation, yet these experts may be inaccessible (CF2) or uncommitted (CF3).

In our approach an ontology engineer uses coding techniques from Grounded Theory to model AK concepts from typical questions. This coding process is refined and iterated to improve the accuracy of ontology modeling without heavily relying on domain experts. Typical questions are then evaluated by AK users to verify that AK concepts in the ontology are accurate and interpreted consistently between AK users. The accessibility and commitment of AK users is addressed by selecting one of several techniques listed in Table 1 for acquiring evaluations.

4.4. Contextual factors in the maintenance phase

The AK retrieved by users should remain accurate even when their AK needs evolve (CF7) and AK concepts evolve (CF8) (Table 2). Therefore the ontology should be updated accordingly [20]. Updating the ontology should be efficient (1) for economic feasibility and (2) to prevent of any lag between the moment AK needs and concepts change and this new AK can be retrieved by users [6].

AK needs shift and AK concepts can become deprecated in a software project when architecture, design and development methods change, societal and organizational changes impact the SA [1], new insights and solutions are found, project phases progress, or when concepts from the product domain(s) evolve. This is likely to happen in complex project domains (CF5) involving many (CF1) diverse (CF4) users working in multiple disciplines (CF6) that evolve with time. In our approach typical questions are re-acquired and compared to previously acquired questions to detect shifting AK needs and AK concepts.

5. An exploratory case study of ontology engineering for SA documentation

In this exploratory case study [21] on the use of the 'typical question' approach we explore two questions:

- *How do the contextual factors in this case study influence the application of the 'typical question' approach?*
- *How well does the 'typical question' approach work in this case study to construct a useful ontology for SA documentation?*

We developed an ontology that was applied to SA documentation at Océ Technologies, an international leader in digital document management and a Canon Group company. This SA documentation specifies the software for document printing machines developed at Océ Technologies and is used in multiple projects and product lines.

The documentation for which the ontology was built consists of 7 SA documents with 79 pages in total and is a small yet

Table 2
Influence of contextual factors in ontology engineering phases.

Contextual factors	Influence
All ontology engineering phases	
CF2: Accessibility of AK users	Accurate acquisition of AK needs becomes difficult when AK users are inaccessible or uncommitted.
CF3: Commitment of AK users	Ontology construction and evaluation needs involvement of committed AK users.
Domain analysis phase	
CF1: Number of AK users	These CFs influence how many different AK concepts have to be identified, supporting the AK needs of users in various roles, disciplines, and product domains.
CF4: Diversity of AK users	
CF5: Product domain complexity and specificness	
CF6: domain multidisciplinarity	
Ontology conceptualization and evaluation phase	
CF1: Number of AK users	Conflicting interpretations of AK concepts are likely to occur when there are many diverse AK users, each having their own jargon in their domain and role. An ontology engineer may not have thorough understanding of all disciplines and product domain specific concepts, and require assistance from domain experts.
CF4: Diversity of AK users	
CF5: Product domain complexity and specificness	
CF6: domain multidisciplinarity	
Ontology maintenance phase	
CF7: Shifting AK needs	AK needs of users shift, e.g., between software project phases and when domain concepts change. Maintenance may be needed to keep the ontology up to date with the AK needs of its users.
CF8: Volatility of domain concepts	

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

representative subset of the available types of documents in a project at Océ Technologies. This subset of documents was selected because of timing constraints for the experiment reported in [5].

The products built at Océ Technologies evolve with market needs, which introduces a high rate of change (CF8) in their domain. Their product teams deal with software, firmware and specific hardware (CF5, CF6) and consist of diverse documentation users (CF4) such as domain architects, product testers, workflow architects, etc. Océ applies agile development in a product line environment. This means that the architectural design and software project phases iterate rapidly (CF7).

A software professional at Océ estimated that there are well over 50 users of the SA documentation (CF1). Software projects typically take place in three locations in Europe. However, our study was limited to documentation users in one site (CF2). Even though documentation users were pressed for time, they were committed to this research project (CF3).

5.1. Domain analysis phase (step 1)

In order to work with the schedule and accessibility of the AK users at Océ Technologies, we used a mailing list to acquire their AK needs. We asked SA documentation users to send their typical questions about AK that they had during their work activities. 7 documentation users, among which software engineers, a software project manager and a software architect, provided 17 questions. Snippet 5 gives a subset of the acquired typical questions at Océ. Parts of these questions are obfuscated for confidentiality reasons.

Snippet 5. Subset of typical questions acquired at Océ:
- "What is the rationale behind this requirement? (And whom can we ask?)"
- "Which subsystem is responsible for fixing the XX defaults based on the device configuration?"

AK needs acquisition during domain analysis using a mailing list was time- and cost-efficient. Because many users involved in this case study proved to be accessible and committed the acquisition was, in retrospect, also relatively accurate (see Table 1 for comparison).

5.2. Conceptualization and evaluation phase (steps 2–5)

After acquisition of typical questions in the Océ case study we labeled and classified the AK concepts in these typical questions. Snippet 6 lists three typical questions, two from Snippet 5, that we acquired. Labels (between square brackets) show the classification of phrases and words after we applied the coding mechanisms from Grounded Theory [13].

Snippet 6. Coding of typical questions at Océ:
- "What is the rationale ["*Decision (class)*"] behind ["*depends on (relationship)*"] this requirement ["*requirement (class)*"]? (And who ["*stakeholder (class)*"] can we ask?)"
- "Which subsystem ["*subsystem (class)*"] is responsible for fixing ["*changed by (relationship)*"] the defaults based on the device ["*Device (class)*"] configuration ["*setting (class)*"]?"
- "I changed the behavior ["*behavior(class)*"] of some interface method ["*method(class)*"] after I had convinced myself that the method ["*method (class)*"] was not used at all in other parts of the system and so the change would have no ["*change task (class)*"] impact ["*impacts (relationship)*"]. However, at the same time another team made functional enhancements ["*change task (class)*"] that relied ["*depends on (relationship)*"] exactly on the *old* behavior ["*behavior(class)*"] of that method ["*method(class)*"]. Could I have known that a new subsystem ["*subsystem(class)*"] dependency ["*depends on (relationship)*"] on this method ["*method (class)*"] was upcoming ["*change task (class)*"] or was it just bad luck that these actions ["*change task (class)*"] had crossed each other?"

During coding we found that many words and phrases in the typical questions could almost directly be translated into relationships and classes, e.g., similar to Fig. 3. We recorded the rationale for our actions during the coding steps in a document (named 'field notes' or 'memos' in Grounded Theory). We instantiated the classified AK concepts in typical questions, of which two are shown in Snippet 7.

Snippet 7. Typical questions with instantiated AK concepts at Océ:
- "What decision depends on requirement '*REQ. 23*'?"
- "Which subsystem changes setting '*external indication light color*'?"

Feedback interview sessions were held with a small group of committed and accessible documentation users in diverse roles. These documentation users evaluated the classified and interrelated AK concepts and resolved conflicting interpretations. A software designer and software engineer each evaluated 7 questions with instantiated AK concepts on their accuracy as well as their relevancy and representativeness for the roles in the project. A software architect provided feedback on the interpretations of several AK concepts. These interviewees did not partake in providing the typical questions. Below is a summary of how a conflict between the interpretations of 'behavior' and 'feature' was resolved:

The AK concepts 'behavior' and 'feature' are on different abstraction levels and used by users in different roles: 'Feature' is similar in meaning to 'behavior', but is a term adopted by users that work from a business perspective.
'behavior' is adopted as the primary representation of the AK concepts in the ontology.

Fig. 4 depicts an ontology[4] partially built using the coded and evaluated AK concepts from Snippets 6 and 7. This ontology was used for the industrial experiment reported in [5]. Note that we only included AK concepts in this ontology that could later be annotated in the subset of documents used in the experiment. We did not include the other identified concepts such as 'Change task', 'Method', 'Device', and 'Stakeholder'.

We compared the identified AK concepts and relationships to those in the Lightweight Software Ontology proposed by Tang et al. in [22]. The Lightweight Software Ontology offers a starting point or a template to help ontology engineers capture AK that is commonly used. The AK concepts and relationships we identified largely coincided with those in the Lightweight Software Ontology,

[4] See http://www.vanrijnswou.nl/owl/ontology.owl.xml for source file.

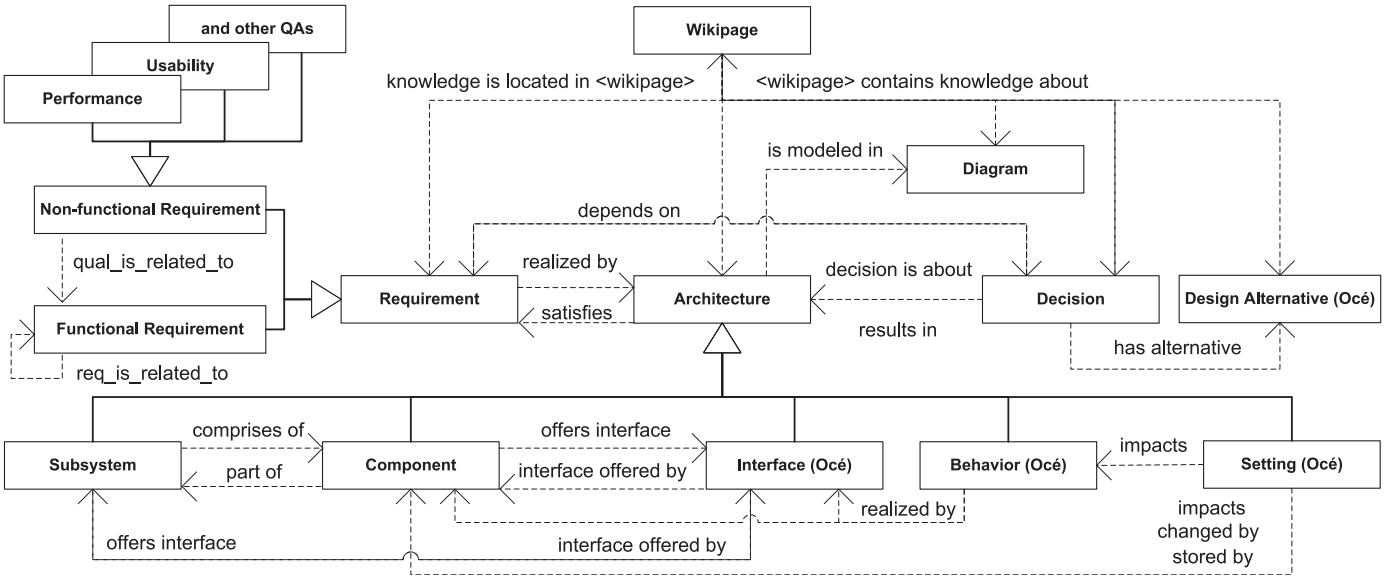


Fig. 4. AK ontology constructed in Océ software project domain.

i.e., the 'typical question' approach was also able to identify commonly used domain-independent AK.

We adopted concepts 'Wikipage' and 'Diagram', derived from concept 'DC' (Dublin Core) in the Lightweight Software Ontology, to enable ontology-based documentation in a semantic wiki tool. The steps taken in the construction of the ontology in Fig. 4 are part of a middle-out approach (see Section 2.3), combining a top-down predefined ontology (i.e. the Lightweight Software Ontology) and bottom-up acquisition of domain specific AK needs using the 'typical question' approach. 11 semantic relationships and 4 classes, appended with "(Océ)", in the ontology are Océ domain specific. Concepts 'Behavior' and 'Setting' in this ontology have specific semantics in the Océ product-domain.

The ontology was used to store the content of the file-based documentation subset in the semantic wiki as wikipages. The ontology classes and relationships were instantiated by semantic annotation of AK in the SA documentation content in wikipages.

The semantic annotation of AK concepts, i.e. the documentation content itself, described above is not part of our 'typical question' approach. We describe this to illustrate how the constructed ontology was used as part of an ontology-based documentation approach. Please see [5,23] for details on how the ontology was populated with AK instances in a semantic wiki and used in the experiment.

5.3. Ontology maintenance phase (step 6)

In the Océ case study we observed that the AK needs of users shifted between software project phases. High-level AK about product lines, changing independent of individual product projects, is present in the documentation and can introduce shifts in AK needs. Moreover, much product research and innovation takes place in the R&D department of Océ, e.g., on hardware and mechanical components. This introduces volatility of AK concepts.

Several months after previous steps were executed and ontology-based SA documentation was implemented, we asked a diverse group of documentation users to record their typical questions about AK in a daily log (technique from Table 1). We decided to use daily logs to acquire typical questions because users were still quite accessible and committed and because it gave a balanced trade-off between accuracy and efficiency of AK needs acquisition. We acquired 39 typical questions from 9 documentation users, including 5 software engineers, 2 product testers, 1 software architect and 1 project manager.

We collected daily logs during the build and integration phases of the project in which ontology-based SA documentation was implemented. New AK needs were acquired compared to the known AK needs acquired at an earlier project phase. The daily logs contained typical questions about software builds, releases, planning, product-lines, control flow, physical products, and automated tests. Considerably fewer typical questions about change impact were acquired as compared to the typical questions initially collected. Four examples of typical questions containing new AK needs are given below in Snippet 8.

Snippet 8. Newly acquired typical questions at Océ:

- "I need to find AK on build XX from a different domain team (but for the same product)."
- "Is setting XX for device YY useful for product ZZ in product line?"
- "How is development tool XX used in offshore location YY?"
- "Is state XX persistent? A test case fails on behavior YY."

5.4. Lessons learned

The case study gave us insight in how contextual factors influenced the application of the 'typical question' approach in the Océ project and whether the approach could be used to construct a useful ontology.

- *How do the contextual factors in this case study influence the application of the 'typical question' approach?*

The 'typical question' approach was used to acquire AK needs for many diverse AK users (CF1 and CF4) in steps 1 and 6 of the approach. The roles of these AK users include software engineers, architects, project managers, and product testers. The time and effort spent by the AK users was acceptable for them as they had

spent around 5–10 min to type an email with typical questions or a few minutes each day to record a typical question in their daily log.

AK users were not only able to phrase their own typical questions in steps 1 and 6, but also evaluate the questions of other AK users during steps 4 and 5 of the approach. Part of the typical questions acquired in the case study were not only about AK but also about detailed design and implementation details. We observed that the amount of AK needed by AK users is influenced by their role (CF4) and the software project phase in which typical questions are acquired.

In the case study the researchers had good access to AK users who were quick to help. AK users provided typical questions, clarified AK needs, and evaluated the ontology. This commitment also impacted the time and effort spent by AK users to phrase and rephrase their typical questions, making sure they were relevant, and consistently record them in a daily log. Accessibility (CF2) and commitment (CF3) of AK users are regarded as preconditions for the 'typical question' approach to work. However, our evaluation (see Table 1) suggests that our approach can address low accessibility by using an acquisition technique involving email or mailing lists. A case study in a project where SA documentation users have low accessibility would give insight in this, e.g., in distributed development.

The diversity of AK users in terms of their roles (CF4) and disciplines (CF6) was relevant and necessary for understanding and aligning diverse AK concepts. Examples of this were the interpretation of 'setting' between the tester and developer role and the interpretation of 'feature' between users from engineering and business disciplines. These conflicting interpretations were detected and resolved using instantiated typical questions in steps 4 and 5 of the approach. As such, it appears advantageous to have stakeholders in different roles to participate in evaluation.

We found that AK concepts that are specific to the product domain (CF5) do not always show up in acquired typical questions. The reason is that we asked the AK users to provide their typical questions about architecture, and consequently the users did not actively ask questions about background knowledge on the product domain. Part of the product domain specific AK was implicit or omitted in the typical questions. Therefore this knowledge was not explicit in the constructed AK ontology.

Several months after the initial ontology was constructed, the approach was used again to detect shifting AK needs in the same project (CF7). We did observe shifting AK needs, but the meaning of the classes and relationships remained the same (CF8). The typical questions that we acquired from users in the initial execution of step 1 in the approach did not cover the AK needed by users for their later tasks. This means that if user tasks change frequently, more ontology maintenance is needed.

- *How well does the 'typical question' approach work in this case study to construct a useful ontology for SA documentation?*

In a questionnaire we asked five Océ professionals that are users of AK to evaluate the ontology. This evaluation took place after they used the ontology to retrieve AK from SA documentation in the experiment reported in [5]. We asked the five AK users whether *the ontology is a correct representation of reality*. Three AK users answered "*yes*" and two AK users answered "*to a certain extent*". These two AK users remarked that the ontology should include more printing machine domain knowledge.

Not all of the printing machine domain knowledge that was identified was included in the evaluated ontology depicted in Fig. 4. This is because the evaluated ontology was built for annotating a subset of SA documentation. AK concepts that were not described in this subset of SA documentation were not included in the evaluated ontology. Moreover, we asked AK users to provide typical questions about the architecture, not about the product domain. As a result we cannot claim that our approach is comprehensive for building a full ontology for all possible AK needs.

All five Océ professionals confirmed in the questionnaire that they found it practical to work with the ontology. Moreover, all five Océ professionals found the ontology helpful to reason about what AK is in SA documentation, and what AK should be in the SA documentation. This gives us confidence that the 'typical question' approach worked well to construct a useful ontology in this case study.

The ontology constructed in the case study was used together with a semantic wiki to construct ontology-based SA documentation for the experiment reported in [5]. 26 Océ professionals used this ontology-based SA documentation to retrieve AK and their efficiency and effectiveness was significantly higher than when using file-based SA documentation to retrieve the same AK.

6. Related work

In this section we discuss how aspects of the 'typical question' approach for ontology engineering relate to aspects of other ontology engineering approaches and how they differ.

Jansen et al. describe how they construct an ontology from existing SA documentation in [8]. Researchers and a software architect identified and annotated classes of AK in documentation text, to elicit the AK needs in a documentation use case about architectural reviews. Existing SA documentation and the use case(s) may however not convey AK needs of all documentation users. Moreover, annotating researchers or architects may not know the interpretation of AK concepts and the AK needs of all documentation users by heart.

Questions can be used to classify design artifacts in the Zachman framework for enterprise architecture [24]. Typical questions gathered in our approach are mostly asked from the owner, designer, and builder perspective and from all abstractions (what, how, where, who, when, and why) described in the Zachman framework. Zachman argues from empirical observation that design artifacts (e.g., product descriptions and engineering documentation) can be classified by the users of these design artifacts. In our approach the users of AK evaluate classifications of AK concepts that are used to describe design artifacts.

In their TOVE enterprise modeling approach [25] Grüninger and Fox make use of 'competency questions' for ontology evaluation. Competency questions are considered ontology requirements and are used to evaluate an ontology based on its ability to answer the competency questions [25]. The notion of typical questions is similar to that of competency questions. Competency questions are however not refined and evaluated by users in order to construct an ontology (introduce ontological commitments), as is done in our approach with typical questions.

The On-To-Knowledge methodology [26] developed by Sure et al. uses competency questions to add relations to an ontology. In [27] Uschold and King however found competency questions to be too specific to guide early ontology development. Moreover, formalizing competency questions in first-order logic requires different skills from an ontology engineer than those required in the 'typical question' approach.

Acquisition of competency questions in these approaches is different to the acquisition of typical questions in our approach. For example, in the UPON methodology, proposed by Nicola et al. in [28], competency questions are gathered using interviews with domain experts, brainstorms and document analysis, whereas direct acquisition of typical questions from all users is proposed in our approach. Moreover, optimization of the efficiency and accuracy of AK needs acquisition and evaluation is proposed in our approach. This optimization is achieved by selecting different

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

techniques for acquiring typical questions based on the commitment and accessibility of users.

The DILIGENT methodology, described by Pinto et al. in [29], focuses on distributed ontology development involving different stakeholders, possibly in separate locations, with varying needs and purposes. Ontology users can change an initial shared domain ontology in their local environment according to their needs. These users can then provide arguments for change requests to a central board which makes decisions and judgments on modeling of user needs and conflicting requirements. Similarly users give feedback on ontology concepts in our approach. Our approach however uses typical questions in the evaluation process. Moreover, AK users in our approach not only evaluate AK concepts needed by themselves but also those needed by other AK users.

Kotis et al. describe their HCOME methodology in [30] in which users collaborate to solve conflicting interpretations of concepts in an ontology. HCOME allows ontology users to propose an updated ontology by themselves. This relies on different skills of ontology users as compared to our approach in which SA documentation users need the skill to sharply phrase their typical questions about AK.

7. Conclusions and future work

The use of ontology-based SA documentation can improve AK descriptions and retrieval. The implementation of ontology-based SA documentation requires acquisition and ontology modeling of the AK needed by SA documentation users. In software projects in industry it is important for organizations and individual users that this is done efficiently and accurately. For industry domains that are large, complex, and have many diverse users, this becomes challenging.

We devised a 'typical question' approach to ontology construction for SA documentation in the context of a large and complex software project. In this approach typical questions are used to acquire a tangible representation of AK needs from SA documentation users. AK concepts are identified from the typical questions and modeled in an ontology using coding techniques from Grounded Theory. Typical questions are also used for ontology evaluation and to identify conflicting interpretations of AK concepts between AK users working from different roles and disciplines.

We described contextual factors that influence the construction of ontologies for SA documentation in software projects, e.g., the specificness of the product domain and the accessibility of AK users. We found that the 'typical question' approach could be applied to build an ontology for many diverse AK users in the context of a large and complex software project. This is reasonable since: (1) the 'typical question' approach is based on the involvement of different roles (2) resolves conflicting interpretations between roles and (3) supports different forms of acquisition, e.g., interviews and emails, to handle different levels of commitment and availability. AK users evaluating the ontology all confirmed that the ontology was practical to work with and most AK users confirmed it was a correct representation of reality.

Our approach caters for changes in AK needs, however it cannot prevent ontology maintenance as AK users mostly provide AK needs for their current tasks. The frequency of such maintenance depends on the frequency with which user tasks change.

These findings are based on a single case study of SA documentation for software-intensive products. Our exploratory case study was a first step to test if our 'typical question' approach is suitable for gathering enough explicit product-specific domain knowledge to produce an accurate AK ontology. We plan to do additional case studies, to generalizing our findings beyond one company and investigate if the 'typical question' approach can be applied by industry professionals. A comparative study of the use of other ontology engineering approaches, possibly combined with ours, to gather AK needed for both current and future tasks of AK users will be future work.

Acknowledgements

The authors wish to thank René Laan, Wim Couwenberg, Pieter Verduin, Amar Kalloe, and the other good folks at Océ R&D for their support, interest, participation, and excellent insights. This research has been partially sponsored by the Dutch "Regeling Kenniswerkers", project KWR09164, "Stephenson: Architecture knowledge sharing practices in software product lines for print systems" and by the Natural Science Foundation of China (NSFC) project No. 61170025 "KeSRAD: Knowledge-enabled Software Requirements to Architecture Documentation".

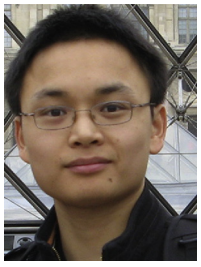
References

[1] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, 3rd ed., Addison-Wesley Professional, 2012.
[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2002.
[3] P. Lago, P. Avgeriou, First workshop on sharing and reusing architectural knowledge, ACM SIGSOFT Softw. Eng. Notes 31 (5) (2006) 32–36.
[4] C. López, P. Inostroza, L.M. Cysneiros, H. Astudillo, Visualization and comparison of architecture rationale with semantic web technologies, J. Syst. Softw. 82 (8) (2009) 1198–1210.
[5] K.A. de Graaf, A. Tang, P. Liang, H. van Vliet, Ontology-based software architecture documentation, in: Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), IEEE, 2012, 121–130.
[6] M. Hepp, Possible ontologies: How reality constrains the development of relevant ontologies, IEEE Internet Comput. 1 (1) (2007) 90–96.
[7] H.-J. Happel, S. Seedorf, Ontobrowse: a semantic wiki for sharing knowledge about software architectures, in: Nineteenth International Conference on Software Engineering & Knowledge Engineering (SEKE), 2007, 506–512.
[8] A. Jansen, P. Avgeriou, J.S. van der Ven, Enriching software architecture documentation, J. Syst. Softw. 82 (8) (2009) 1232–1248.
[9] C. López, V. Codocedo, H. Astudillo, L.M. Cysneiros, Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach, Sci. Comput. Prog. 77 (1) (2012) 66–80.
[10] M.T. Su, C. Hirsch, J. Hosking, Kaitorobase: visual exploration of software architecture documents, in: International Conference on Automated Software Engineering (ASE), IEEE, 2009, 657–659.
[11] E.P.B. Simperl, C. Tempich, Ontology engineering: a reality check, in: On the move to meaningful internet systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, Springer LNCS, 2006, pp. 836–854.
[12] B.G. Glaser, A.L. Strauss, The Discovery of Grounded Theory, Weidenfeld and Nicolson, 1967.
[13] A. Strauss, J. Corbin, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 2nd ed., Sage, 1998.
[14] C. Urquhart, W. Fernandez, Grounded theory method: the researcher as blank slate and other myths, in: International Conference on Information Systems (ICIS), 2006, 457–464.
[15] S. Adolph, W. Hall, P. Kruchten, Using grounded theory to study the experience of software development, Emp. Softw. Eng. 16 (4) (2011) 487–513.
[16] J.A. Díaz-Pace, M. Nicoletti, S.N. Schiaffino, C. Villavicencio, L.E. Sanchez, A stakeholder-centric optimization strategy for architectural documentation, in: Proceedings of the 3rd International Conference on Model & Data Engineering (MEDI), Springer LNCS, 2013, pp. 104–117.
[17] E. Turban, J.E. Aronson, Decision Support Systems and Intelligent Systems, 6th ed., Prentice Hall, 2000.
[18] P. Kruchten, Contextualizing agile software development, J. Softw. Evol. Proc. 25 (4) (2013) 351–361.
[19] T. Bürger, E. Simperl, S. Wölger, S. Hangl, Using cost-benefit information in ontology engineering projects, in: Context and Semantics for Knowledge Management, Springer, 2011, pp. 61–90.
[20] Y. Sure, C. Tempich, D. Vrandecic, Ontology engineering methodologies, in: Semantic Web Technologies: Trends and Research in Ontology-based Systems, Wiley, UK, (2006), pp. 171–190.
[21] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Emp. Softw. Eng. 14 (2) (2009) 131–164.
[22] A. Tang, P. Liang, H. van Vliet, Software architecture documentation: The road ahead, in: Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE, 2011, 252–255.

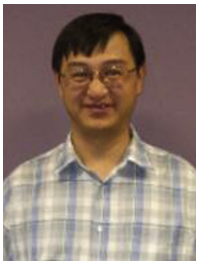
Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006

[23] K.A. de Graaf, Annotating software documentation in semantic wikis, in: Fourth workshop on Exploiting Semantic Annotations in Information Retrieval (ESAIR), ACM, 2011, pp. 5–6.
[24] J. Zachman, The Zachman Framework for Enterprise Architecture, Zachman International, 2002.
[25] M. Grüninger, M.S. Fox, Methodology for the Design and Evaluation of Ontologies, International Joint Conference on Artificial Intelligence (IJCAI), Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.
[26] Y. Sure, S. Staab, R. Studer, On-to-knowledge methodology (OTKM), in: Handbook on Ontologies, Springer, 2004, pp. 117–132.
[27] M. Uschold, M. King, Towards a methodology for building ontologies, International Joint Conference on Artificial Intelligence (IJCAI), Workshop on Basic Ontological Issues in Knowledge Sharing, 1995.
[28] A.D. Nicola, M. Missikoff, R. Navigli, A proposal for a unified process for ontology building: Upon, in: Database and Expert Systems Applications – DEXA, Springer LNCS, 2005, pp. 655–664.
[29] H. Pinto, S. Staab, C. Tempich, Y. Sure, Distributed engineering of ontologies (diligent), in: Semantic Web and Peer-to-Peer, Springer LNCS, Springer LNCS, 2006, pp. 303–322.
[30] K. Kotis, G. Vouros, Human-centered ontology engineering: the HCOME methodology, Knowl. Inform. Syst. 10 (1) (2006) 109–131.

K.A. de Graaf is a Ph.D. student in the Software Engineering research group, Department of Computer Sciences, at VU University Amsterdam in The Netherlands. He received his master degree in Computer Science from VU University. His research interests include software architecture, software documentation, knowledge engineering, and reliability prediction.

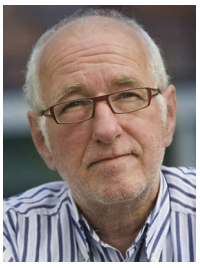


P. Liang is a professor of Software Engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China. He is currently a visiting researcher at VU University Amsterdam, the Netherlands. Between 2007 and 2009, he was a postdoctoral researcher at the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He is a young associate editor of the journal Frontiers of Computer Science. His research interests concern the area of software architecture and requirements engineering. He has published more than 60 articles in peer-reviewed international journals, conference and workshop proceedings, and books.



A. Tang received the Ph.D. degree in information technology from the Swinburne University of Technology. He is an associate professor in Swinburne University of Technology's Faculty of Science, Engineering and Technology. Prior to being a researcher, he had spent many years designing and developing software systems. His research interests include software architecture design reasoning, software development processes, software architecture and knowledge engineering. He is a member of the ACM and the IEEE.

Dr. W.R. van Hage (Ph.D. TNO/VU University Amsterdam, 2009) is Chief Data Scientist at SynerScope B.V. and guest researcher at the VU University Amsterdam in the field of interdisciplinary e-Science and Web Science. His main research topics in the past 10 years are augmented sense making, visual analytics, information integration, and semantics. He is principal investigator in the US ONRG funded SAGAN and COMBINE projects and work package leader in the EU FP7 project NewsReader and the Dutch BSIK COMMIT Metis and Data2Semantics projects, all dealing with knowledge extraction and Linked Data visualization. He is a co-organizer of the LISC and DeRiVE workshop series; the Ontology Alignment Evaluation Initiative (OAEI); and the Linked Science Tutorial series about improving the speed, efficiency and transparency of Web research. He has developed the Simple Event Model (SEM), an OWL ontology for the description of event data; spatiotemporal indexing for SWI-Prolog (awarded with a best paper award at the EKAW 2010 conference), and the SPARQL package for the R statistical programming language.



H. van Vliet is Professor in Software Engineering at the VU University Amsterdam, The Netherlands, since 1986. He got his PhD from the University of Amsterdam. His research interests include software architecture, knowledge management in software development, global software development, and empirical software engineering. Before joining the VU University, he worked as a researcher at the Centrum voor Wiskunde en Informatica (CWI, Amsterdam). He spent a year as a visiting researcher at the IBM Almaden Research Center in San Jose, CA. He is the author of "Software Engineering: Principles and Practice", published by Wiley (3rd edition, 2008). He is a member of IFIP Working Group 2.10 on software architecture, and the Editor in Chief of the Journal of Systems and Software.

Please cite this article in press as: K.A. de Graaf, et al., An exploratory study on ontology engineering for software architecture documentation, Comput. Industry (2014), http://dx.doi.org/10.1016/j.compind.2014.04.006