CAKE: Codifying Architecture Knowledge Effectively

Danny Weyns and Jesper Andersson
Linnaeus University, Campus Växjö, Sweden
Department of Computer Science: https://cs.lnu.se/index.php
Information Engineering Center: http://lnu.se/iec
{danny.weyns,jesper.andersson}@lnu.se

ABSTRACT
A company’s architecture knowledge is often personalized across specific people that share experience and knowledge in the field. Companies are often skeptical about codifying architecture knowledge because they associate it with creating documents. However, omitting the codification of architecture knowledge may result in ad-hoc practices, with all the adverse consequences. In our research, we study how architecture knowledge can be codified effectively to improve industrial practice. Instead of creating passive documents that have to be kept in synchrony with the actual system, we propose an active approach in which architecture knowledge is harvested from the actual system and represented in a way that suits the target stakeholders. We call this approach CAKE, short for Codifying Architecture Knowledge Effectively. Central in CAKE are: (1) an architectural viewpoint that codifies the kind of architecture knowledge required to support particular stakeholder concerns, (2) a supporting tool to harvest the necessary data from the system, analyze the data, and represents the knowledge in a proper way to the stakeholders. We have applied CAKE in a collaborative research project with Egemin Automation to improve updates of deployed logistic systems with high availability requirements. An empirical evaluation of the approach demonstrates a significant improvement of quality of system updates with respect to correct execution of updates and the availability of services during the updates.¹

1. INTRODUCTION
According to Kruchten et al. [5] architectural knowledge consists of architecture design as well as the design decisions, assumptions, context, and other factors that together determine why a particular solution is the way it is. A company’s knowledge management strategy captures which architecture knowledge is codified in explicit accessible models, and which knowledge is personalized across specific people that share experience and knowledge in the field [1]. Except for the architecture design part, most of the architectural knowledge usually remains implicit as tacit knowledge in the heads of the architects. [5] suggests that an explicit representation of architectural knowledge is helpful for building and evolving quality systems. Our experience are even stronger and indicate that omitting the codification of architecture knowledge may lead to ad-hoc practices, which result in inefficient or error-prone solutions, increasing costs, etc. Nevertheless, companies are often skeptical about codifying architecture knowledge because they associate it with creating documents. Besides the inherent overhead and complexity of capturing and representing architecture knowledge, maintaining the knowledge in the face of system evolution is also a difficult task [2].

In our research, we study how architecture knowledge can be codified effectively to improve industrial practice. Our particular focus is on representing architecture design, rather than design decisions as for example studied in [5]. Instead of creating passive documents that have to be kept in synchrony with the actual system, we propose an active approach in which architecture knowledge is harvested from the actual system and represented in a way that suits the target stakeholders. We call this approach CAKE, short for Codifying Architecture Knowledge Effectively. CAKE consists of two complementary parts: (1) an architectural viewpoint that codifies the kind of architecture knowledge required to support particular stakeholder concerns, (2) a supporting tool to harvest the necessary data from the system, analyze the data, and represents the architecture knowledge in a proper way to the stakeholders.

We have applied CAKE in a collaborative research project with Egemin Automation¹, an industrial manufacturer of logistic systems, to improve updates of deployed logistic systems with high availability requirements. Logistic systems are long-living systems that include a warehouse management system and control software for one or more transportation subsystems, such as automated guide vehicles, cranes, and conveyors. During their lifespan, logistic systems obviously require updates, e.g., a part of the system has to be replaced to improve performance, or new control software system needs to be integrated with the existing logistic system. Most of the architecture knowledge of Egemin’s products is personalized across a small group of people. In addition, the logistic systems comprise a lot of legacy software components, whose documentation is often incomplete or outdated. As a result, integrators lack the architecture knowledge that is needed to perform the update tasks correctly with minimal interruption of service of the deployed systems. Faulty updates increase costs, harm the company’s reputation, or even worse, they may cause serious damage to industrial installations.

In the next section, we give a general overview of CAKE. Then, we explain how we have applied CAKE to Egemin’s problem context. The paper ends with conclusions and some reflections.

2. GENERAL OVERVIEW OF CAKE
The CAKE approach consists of three main phases as shown in Fig. 1. The Viewpoint Definition phase defines the viewpoint for the problem under consideration. The ISO/IEC 42010 standard [4] defines an architecture viewpoint as "a work product establishing

¹http://www.egeminautomation.com/en/
the conventions for the construction, interpretation and use of architecture views to frame specific system concerns. We document viewpoints using the template for architecture viewpoints proposed by ISO/IEC 42010. The template consists of: viewpoint name, summary, concerns, stakeholders, model kinds (types of models that deal with the stakeholder concerns), analysis (computations required to synthesize knowledge). For the project with Egemin, we defined an update viewpoint that deals with concerns related to updates of deployed logistic systems. The primary stakeholders involved are the architect, integrators and system admins of the deployed systems. Their main concerns are correctness of updates and minimal interruption of service during the updates.

The definition of the viewpoint consists of three steps as shown in Fig. 1. This process is typically lead by the software architect. First the key stakeholders and there concerns are identified. We describe the concerns in the form of questions. For example, correctness is a concern of an integrator (stakeholder) and questions that help to clarify this concern may be “Which components have to be replaced?" or “Which processes have to be shut down and restarted to replace a particular component?" Second, the stakeholders are interviewed and the necessary information is collected about the stakeholder concerns. Third, the material derived from the interviews is translated into different types of models that allow the representation of the architecture knowledge for the different concerns of the stakeholders. Some of the models may require analyses, e.g. to find inconsistencies between two versions of a system. We represent these types of models using regular UML.

Developing Tool Support supports the automation of the knowledge gathering and representation. We have developed a reusable tool that supports this automation process, illustrated in Fig. 2.

At a general level, stakeholders interact through a GUI (Graphical User Interface) to harvest architecture knowledge, build models, and browse the models. The workflow controller triggers the architecture knowledge collector, architecture knowledge repository, and the model builder to execute the actions. We explain the different parts in more detail following the four steps of the developing tool support phase as shown in Fig. 1.

First, the model kinds defined in the first phase are instantiated in the architecture knowledge repository. We use the Eclipse Modeling Framework (EMF) as a basis for defining the repository. EMF supports specifying the model and generating a Java implementation.

Second, harvester components have to be built that perform the actual knowledge gathering and populate the architecture repository. Depending on the concrete setting, knowledge can be extracted from run-time system components, resource files, system configurations, etc. An example of a harvester that we have used in the project for Egemin is an Assembly Harvester. This harvester gathers knowledge about the the assemblies of the deployed system per location, including assemblies’ version and compile time dependencies. This harvester includes a C# program that supports inspection of programs and libraries.

Third, analyzers have to be built, i.e. algorithms that take knowledge provided by the harvesters and synthesize this knowledge to deal with the stakeholders concerns. For the Egemin project we defined an algorithm to check the consistency of a deployed product under update, and an algorithm that determines the sequence of update steps to migrate the system from the as-is to the to-be version.

Fourth, a graphical user interface has to be built that allows the visualization of the models in a way suitable for the stakeholders. To show the models, the model builder queries the architecture repository to generate the architecture models. For the Egemin project, the integrators required models of the as-is and to-be version of the system, as well as a model that describes the update procedure.

In the third phase Applying CAKE, the work products of the previous phases are actually used by the stakeholders. Applying CAKE typically starts with an initialization phase where the use case scenario is specified. For example, to perform an update in the Egemin project, the physical setting of the deployed system and the location of the files of the new version of the system have to be specified for the tool. Next, stakeholders can use the tool to harvest data and analyze it to generate the required models. The tool will then represent the models to the stakeholders that can use them, e.g. to perform the sequence of steps to perform an update.

3. CAKE IN PRACTICE

In this section, we illustrate some key aspects of CAKE in the Egemin project. Subsequently, we give a brief overview of the integrated model we developed to harvest knowledge from the different sources. Then, we show one of the resulting models derived from the harvested knowledge. We conclude with a brief summary of the empirical evaluation we performed. For a detailed explanation of the case study, we refer the interested to [7, 6].

3.1 Integrated Model

Figure 4 shows how the integrated model that we developed for Egemin. The family of logistic systems deployed by Egemin make up a software product line (SPL) [3]. A SPL contains a set of products, i.e. logistic systems. A product has a version. Products consist of assets that are maintained in the SPL asset base. Assets are different logistic subsystems. An asset has a version and an owner. Each logistic subsystem in owned by a team. The assets may have asset constraints that constrain the compositions of assets supported by the SPL. E.g., a constraint may be that a particular version of one subsystem requires a version of another subsystem. A product can already be deployed on a set of locations (i.e., the hosts of a deployed as-is product), or it can be ready for deployment with a set of installation bundles that have to be installed on locations (i.e., the MSI files of a deployable to-be product). Each location on which a
product is deployed contains a set of resources that correspond to particular assets. A resource has a version. Example resources are an assembly–DLL file (Dynamic Link Library) or an EXE file (Executable), a database, and a config file. A resource may have resource dependencies to other resources. A deployed assembly that is executable can be started as a process that runs on a location, i.e., a windows process. A process uses one or more resources.

3.2 Example Model

Figure 3 shows a snapshot of a update procedure model for one of the products of Egemin’s SPL. The box top left shows the different locations on which the product is deployed. The box on the right hand side shows the installation bundles (product installers) that have to be deployed on the selected location. The box at the bottom shows the update script that resulted from the analysis. The update script shows the subsequent update steps the integrator has to perform to realize the update.

3.3 Empirical Evaluation

We have evaluated CAKE applied for Egemin’s SPL in an empirical experiment. We evaluated a total of 68 updates of industrial logistic systems performed by 17 professionals, half of them with Egemin’s traditional update approach, the other half with the tool that automatically generates update scripts. We formulated hypotheses with respect to the correct execution of product updates and the availability of services during the updates. Statistical analysis demonstrate that 44% of the updates with the traditional approach contain errors, while all the updates with the tool were performed correctly. Furthermore, the results show that with the traditional approach 58% of the process shutdowns were unnecessary, while there were only 7% redundant process shutdowns with the tool.

4. CONCLUSIONS

We started this paper with arguing that omitting the codification of architecture knowledge may lead to ad-hoc practices, resulting in inefficient or error-prone solutions, and ultimately increasing costs. Instead of following a passive style of creating and synchronizing documents with architecture knowledge, we propose an active approach in which architecture knowledge is harvested from the actual system and represented in a way that suits the target stakeholders. We gave an overview of CAKE, a practical approach to codify architecture knowledge effectively. We described the 3 phases to put CAKE in practice: viewpoint definition, developing tool support, and applying CAKE, and we illustrated how we have applied the steps of each phase in a project with Egemin.

We conclude with some final remarks. Architecture knowledge is often considered from a document-oriented point of view. In CASE, we take a different perspective by directly linking architecture knowledge to the system of interest. By automating harvesting and analysis, CAKE guarantees consistent architecture knowledge at any time. Finally, the effort to apply CAKE is considerable. For Egemin the effort was worthwhile as the results can be reused for many of the 200 deployed products of their SPL. The tradeoff may be different for other settings. Nevertheless, we believe that CAKE can be of interest to industry and contribute to improved engineering practice. One interesting direction is how CAKE may be used to support search-based software engineering techniques that aim at improving test-case selection in heterogeneous systems.

5. REFERENCES