Improving the software architecture design process by reusing technology-specific experience

Glib Kutepov

Information Systems Development
Fraunhofer ISE
Fraunhofer-Platz 1
67663 Kaiserslautern
glib.kutepov@iese.fraunhofer.de

Abstract: Experience with particular technologies such as SOA, Cloud Computing, or Mobile Apps plays a crucial role when designing the architecture of a software system. Being aware of the challenges usually encountered when using a technology and knowing in advance how to resolve these challenges can dramatically increase the quality of the software system architecture and decrease the design effort. However, it is not always a straightforward process to collect the necessary architectural experience, persist it on the organizational level, and reuse it in the right way, especially if the technology is new. This paper describes how architecture design processes can be improved by supplementing them with architectural experience related to a particular technology. The way architectural experience can be described using architectural scenarios and solution patterns is explained and persisted in the architecture design process. The efficiency of the approach is validated with the help of a case study.

1. Introduction

Consider a modern software development organization that develops its products with a strong focus on the architecture. During the software architecture design process, the architect identifies key challenges that influence the current system architecture and finds appropriate solutions. This task is carried out successfully because the architect bases his system on familiar, wide-spread IT technologies whose caveats are well known to him or her. Fortunately, the world of technology does not stand still: Every day, some new technology appears on the IT scene, which opens up new opportunities: maybe an alternative or an improvement to an existing technology, or a completely new technological paradigm, such as SOA, Cloud Computing, or Mobile Applications. The organization may have to incorporate the new technology in its architectural practices in order to stay competitive. Initially, the identification of challenges and solutions will be less effective than before due to a lack of knowledge on the architect’s part regarding the new technology. This may increase the time needed for architecture design and decrease quality. However, with time the process will become more effective and the quality of the product will improve. This will be mostly due to the tacit experience of the software
architect. Having one person as the only source of this specific knowledge may jeopardize all the benefits of adopting the new technology. Thus, a way to systematically collect and persist the architect’s experience in an organization has to be found.

Hofmeister et al. [HN99] suggest performing similar activities before designing every architectural view and refer to these activities as “Global Analysis”. The core of “Global Analysis” is the identification of factor-strategy pairs, which are then applied in the actual view design phase. “Factors” represent the challenges or problems that may influence architecture design and “strategies” represent solutions to them. We see such pairs as an effective tool for describing and persisting technology-specific experience.

In this paper, we propose an approach for systematically collecting, describing, and reusing such pairs in the context of the architecture design process. The pairs are collected with the help of project post-mortems, while the factors are described in the form of architectural scenarios and strategies with solution patterns. Furthermore, several scenarios are provided for using such pairs in the architecture design phase, thus operationalizing the experience.

The paper is structured as follows: After an introductory part, the pros and cons of well-known techniques for persisting architectural experience are described in section 2. In section 3, the details of the approach are given. Our explanation is based on Fraunhofer ACES - the architecture design process [KK11] used for software architecture design at Fraunhofer IESE. Section 4 shows how we instantiated our idea for the actively evolving technology of mobile “apps” and particularly its application in the business domain. The “Initial Validation” chapter features a case study that was performed in order to take first steps towards the validation of the approach. The section “Conclusion” concludes the paper.

2. Related Work

The aim of this paper is to show the reader how new technology can be efficiently adopted in the architectural practices of software engineering companies. The main challenge in adopting a new technology is the lack of structured organization-wide architectural experience that can be reused in an operational manner. Thus, our goal is to find a way to collect and persist this experience inside the architecture design process.

Well-known processes such as ADD [BK01] or BAPO/CAFCR [Sm03] provide solid guidance for architecture design but, unfortunately, do not offer any facilities for collecting and persisting architectural experience for further reuse. This means that such facilities have to be either invented or created by combining prior works. We examined related work on existing solutions to the following challenges: ways to collect architectural experience and ways to persist it within the architecture design process. One of the prominent techniques for collecting experience is “Project Postmortems” [Ti90]. How this technique was used for our purpose will be explained in section 3.2. A popular approach for persisting architectural experience are domain-specific software architectures [Tr95]. Researchers have developed a body of various methodologies for
approaching such architectures, ranging from guidelines and recommendations for creating domain-specific architectures [AA07] to proposals of ready-to-use reference architectures [ZL00], [DN08]. Unfortunately, the applicability of these approaches is limited to particular well-scoped business domains, which is not our case. Our target is a technology that can have multiple application areas; therefore it would be too effort-consuming and inflexible to model it with a set of software components.

Alternative approach for technology-specific experience persistence are pattern languages, such as [BM00], [GH94], or the factor-strategy pairs of [HN99]. Although pattern languages are a very effective approach, we feel that they lack precision in describing architectural problems: In [HN99], the description of the factor (problem) is rather unstructured, and in the “Gang of Four” work [GH94], the problems are described in terms of object-oriented programming rather than architecture. In this paper, these drawbacks are mitigated with the help of architectural scenarios [KB94], [DF99].

It is always a challenge to find the right detail level for describing the solution part of a pattern. For example, the detail level of [GH94] patterns differs from ours: Unlike the implementation level used by [GH94], we describe patterns on the architectural level, which is more abstract and can lead to multiple implementations; hence, it is a complex task to describe a pattern in such a way that it gives the architect solid guidance without prescribing any particular implementation. Therefore we propose our own generic template for solution descriptions. The next section describes how these approaches were combined in order to mitigate the drawbacks of each one and develop the needed technique.

3. Approach

This section contains a step-by-step description of our approach for persisting technology-specific architectural experience in the architecture design process.

3.1 Baseline Architecture Method

As a basis for the technology-specific architecture design process, we take Fraunhofer ACES [KK11] – the architecture design process actively used by Fraunhofer ISEE. The input to the process are system requirements obtained in previous phases of the software system engineering process, and the output is, among others, a set of documented architectural views of the system. ACES is a comprehensive process that guides architects in all their activities, starting from the identification of stakeholders and the understanding of architectural drivers via architecture realization all the way to the documentation and validation of the system architecture produced. The approach consists of two parts: core competence – the main part, which includes domain-independent sub-processes for architecture design, and domain competencies – an optional part, which contains domain-specific static experience artifacts related to the architecture design. The structure of Fraunhofer ACES is shown in Figure 1.
The “domain competencies” (in our case rather “technology competencies”) part of Fraunhofer ACES foresees three crucial experience collection areas - challenges, solutions, and technologies, which represent a similar concept as the factors and strategy pairs of the “Global Analysis” of [HN99]. Challenges correspond to factors and solutions correspond to strategies. The technologies area is used for persisting information about COTS solutions that are relevant for the considered domain. Solutions can address technologies in their description. This paper focuses on challenges and solutions in architecture design; therefore the technology part of ACES is not addressed here. ACES provides experience areas, but gives no concrete guidelines on how to collect this experience or how to represent it.

In order to reuse architectural experience during the architecture design process, we examined two sub-processes of the Fraunhofer ACES core competencies: ASR [KK11] (Architecturally Significant Requirements) – the process analyzing stakeholders and eliciting requirements specifically related to the architecture, and DMM [KK11] (Design, Modeling, Migration) – the process for designing the actual system architecture based on the elicited requirements. In sections 3.2 – 3.5, we describe how one can fill out the “domain competencies” part of ACES with the necessary experience and then reuse it in ASR and DMM processes.

3.2 Eliciting Challenges and Solutions

The idea of collecting experiences by identifying challenges of a particular area is not new and has already been proposed by [DF99]. However, the problem is that [DF99] did not specify where to get these challenges from and how to describe them. Knowledge about challenges regarding a particular technology comes from experience; therefore, an
experience source is required. We consider the best experience source to be pilot projects implemented with the current technology. Therefore, several initial projects have to be performed during which the experience base is filled. After each project, a project postmortem [DD05], [Ti90] has to be performed in order to elicit tacit architectural experience residing in the mind of the architect, or in the artifacts of the project, such as documentation, code etc. Postmortems are techniques that allow for systematically examining the completed project for the purpose of eliciting pitfalls to avoid in the future, and best practices to be reused. We use postmortems for eliciting challenges that recur when applying a specific technology. We perform project postmortems in the form of interviews with architects. These interviews consist of the following steps:

1. **Find out challenging architectural scenarios.** Our experience in designing software architecture shows that most of the challenges are usually related to the quality (non-functional) requirements of the architecture. In Fraunhofer ACES [KK11], quality requirements are described in the form of architectural scenarios [DF99]. Thus, all that architects need to do during the post-mortem meeting is to point out architectural scenarios that in their minds do not only represent a unique challenge met in one project, but that reflect the crosscutting challenge of the current technology. More details on describing challenges with architectural scenarios can be found in section 3.3.

2. **Check for duplicates.** Check if this architectural scenario already exists in the experience base. If not, add it; then check if the related solution has been refined or improved in the current project. If yes, supplement the existing solution with the new details.

3. **Elicit and describe the solution.** An architect needs to elicit the solution for the identified challenge from the project documentation and describe it according to the solution pattern template given in section 3.4. A solution described with the solution pattern template is then added to the experience base.

4. **Retain traceability.** We trace relationship between architectural scenarios and solution patterns with the help of a traceability matrix; therefore this matrix has to be updated every time a change is made to the experience base.

The artifacts elicited in this phase are: a set of architectural challenges described using the architectural scenario template presented in section 3.3, a set of solution patterns described using the template presented in section 3.4, and a traceability matrix, which establishes the relationship between both.

### 3.3 Describing the Challenges with Architectural Scenarios

Quality requirements for software architectures are described in ACES with architectural scenarios [KB94]. According to [RW05] - “An architectural scenario is a crisp, concise description of a situation that the system is likely to face, along with a definition of the response required of the system”. Initially, architectural scenarios were used as a tool for software architecture evaluation [KB94], but later this approach has also turned out to be
suitable for describing quality requirements for software architectures [DF99]. The template shown in Figure 2 can be used for describing architectural scenarios. In order to clarify this issue better, an example of an architectural scenario is given in section 4.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Name of scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>Related quality attribute</td>
</tr>
<tr>
<td>Environment</td>
<td>Context applying to this scenario</td>
</tr>
<tr>
<td>Stimulus</td>
<td>The event or condition arising from this scenario</td>
</tr>
<tr>
<td>Response</td>
<td>The expected reaction of the system to the scenario event</td>
</tr>
<tr>
<td>Response Measure</td>
<td>The measurable effects showing if the scenario is fulfilled by the architecture</td>
</tr>
</tbody>
</table>

Figure 2: Architectural scenario template

3.4 Describing Solutions

Following the idea of pattern languages [BM00], [GH94], we use “Solution Patterns” for the description of our solutions. The crucial point in describing solution patterns is the level of detail. On the one hand, the more details the solution contains, the better. On the other hand, each additional detail induces a certain assumption regarding the context in which the system is developed, which makes the solution less applicable. Developers of pattern languages use templates for their descriptions [GH94]. It is good to use strict templates when the context of an application is well known, as in the case of pattern languages for specific domains. In our case, we consider a whole technology that can be applied in various ways. The template for describing the solution must therefore be sufficiently detailed to be understandable and implementable, and at the same time remain applicable when used in different contexts. Therefore we keep our solution pattern template simple and rather base it on examples than on principles, resulting in more freedom during description and subsequent application. The template consists of the following parts:

1. **Textual description.** A clear description of the way the current pattern resolves the challenge described in the architectural scenario. The description must include pros, cons, restrictions, and known uses of the solution. Should the challenge be resolved by employing COTS components (frameworks, platforms etc.), a link to this component has to be specified.

2. **Structural Diagram.** A structural view of the system that supports the current architectural scenario.

3. **Behavioral Diagram.** An instance(s) of interaction among the components of the system, which supports the current architectural scenario(s).

3.5 Enriching the Process

Once the static experience artifacts (challenges and solutions) have been collected and described appropriately, they must be included in the architecture design process and
provide the users of the process with usage guide. How artifacts are included into Fraunhofer ACES and then reused is sketched in Figure 3.

Figure 3: Technology-specific Fraunhofer ACES

We guide users with the help of reuse scenarios that describe how experience artifacts shall be applied. Three key reuse scenarios (the numbers of reuse scenarios can be matched to the numbers in Figure 3) that may occur when designing a software architecture with the technology-specific Fraunhofer ACES are:

1. **Using architectural scenarios for finding missing requirements.** There are certain quality requirements that are very likely to appear in the context of a particular technology. The customer might first overlook these requirements and come back to them only in a later phase of the project when any change is costly. Therefore, the architect will find it handy to use typical architectural scenarios of the technology as a checklist for finding missing requirements.

2. **Using architectural scenarios as input for the architecture design process.** Architectural scenarios accurately describe the challenge that needs to be resolved by the architecture of a particular software product. Ideally, the architecture scenario will be linked directly to the solution pattern (see point 3) that resolves it; otherwise, its precise description will help the architect to significantly narrow the solution search domain and will later on serve as a basis for assessing the selected solution.

3. **Applying solution patterns for quality driven design.** Based on the architectural scenarios to be satisfied, an architect selects solution patterns to be implemented. Using proven solution patterns will make the architecture design process more efficient and will increase the quality of the resulting product.
4. Application Example

In order to collect initial evidence regarding the effectiveness of our approach, we apply it to one of the technologies currently under research at Fraunhofer IESE. Due to the current trend towards ubiquitous computing and the growing need for mobile support for enterprises, we decided to choose mobile apps as our target technology and scope it to the business domain of the application (excluding, e.g., the gaming domain).

The challenges encountered here are mostly related to the fact that business-oriented mobile applications have the same quality requirements as common desktop applications, but run in a totally different environment. These applications suffer from Internet connectivity problems, high power dependency, and other challenges brought on, for example, by the specifics of the operating system or by a particular application store. Therefore it is absolutely necessary to be aware of the typical challenges of this technology and make sure they are covered with the system architecture.

4.1 Identified Challenges

After conducting several pilot projects at Fraunhofer IESE and performing their post-mortems (section 3.2), we identified a set of challenges related to mobile business applications. Some of these challenges are presented in the table in Figure 4. The second and third columns of the table represent the name of the challenge and architectural scenario that describes it. The first column classifies the challenges into challenge areas.

<table>
<thead>
<tr>
<th>Challenge Area</th>
<th>Challenge (Quality Requirement)</th>
<th>Architectural Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unreliable Connectivity</td>
<td>Application must operate with bad internet connection</td>
<td>Seamless Connectivity</td>
</tr>
<tr>
<td></td>
<td>Application must operate when there is no internet connection</td>
<td>Connection Loss Tolerance</td>
</tr>
<tr>
<td></td>
<td>Remote communication must be fast</td>
<td>Reduced Network Latency</td>
</tr>
<tr>
<td></td>
<td>Remote communication must be reliable</td>
<td>Consistent Communication</td>
</tr>
<tr>
<td>Limited Energy Supply</td>
<td>Application must be energy efficient</td>
<td>Reduced Power Consumption</td>
</tr>
<tr>
<td>Deployment</td>
<td>It must be possible to deploy application update within one hour</td>
<td>Rapid Application Deployment</td>
</tr>
</tbody>
</table>

Figure 4: Challenges of Mobile Business Applications

One prominent challenge for mobile apps is deployment. In contrast to desktop or web applications, the standard facility for deploying apps is an “appstore”. An “appstore” is completely proprietary entity and lies beyond the developers’ control: There is no guarantee that the application will pass the internal approval process and that deployment will be allowed. Furthermore, the duration of the approval process cannot be foreseen and deployment time can therefore not be guaranteed to the customer. Controlled deployment is clearly a crucial requirement for the mobile application development organization, which is legally obliged to guarantee software defect removal within a certain timeframe. The architectural scenario “Rapid Application Deployment” in Figure 5 describes a concrete instance of a controlled deployment challenge. Having such precise description allows an architect to easily evaluate the suitability of the proposed solution by simply playing the scenario.
4.2 Identified Solutions

Based on the experience obtained during the pilot projects, we were also able to find appropriate solution patterns for the identified challenges. In Figure 6, the reader can find a traceability matrix that establishes the relationship between solution patterns and the architectural scenarios they resolve. It can be clearly seen that one pattern can resolve various scenarios and one scenario can be resolved by multiple patterns.

In Figure 7, Figure 8, and Figure 9, an example of a solution pattern described according to the template given in section 3.4 is given. This pattern is named “Deployment bypassing appstore” and resolves the architectural scenario given in Figure 5.
4.3 Initial Validation

This section features an initial validation of our assumption regarding the improved efficiency of the architecture design process and better quality of the end product due to the reuse of technology-specific experience during the architecture design process. Both points are hard to assess without a real project setting. However, a coarse validation of the efficiency aspect can be carried out with the help of a case study.

The case study was designed as follows: We took an architecture document of one of the mobile business applications developed at our institute, designed a change request for it, and asked three Fraunhofer IESE employees with different levels of knowledge in software engineering and only general knowledge in mobile systems to perform several tasks related to this change request with the help of the mobile technology experience persisted in Fraunhofer ACES. The experience base included a catalog with 21 architectural scenarios and 22 solution patterns (partially shown in Figure 6) identified at Fraunhofer IESE using the approach described in section 3. The participants had to
imagine having to implement the change request and thus had to redesign some of the system structures accordingly. They were asked to record the time they spent on:

1. Finding the relevant architectural scenario and solution pattern in the catalog;

2. Reading the pattern, understanding how they would apply it for the given system architecture, and sketching modified structural and behavioral views of the system.

According to the results, the participants reusing experience artifacts needed five minutes on average to find the matching architectural scenario and solution pattern in the catalog. In order to understand the application of the chosen pattern to the current system architecture and sketch modified views, the participants needed 18 minutes on average.

The participants were asked to compare their experiences using the extended ACES approach with using the original approach. Their assessment was that given Fraunhofer ACES without technology-specific experience artifacts, they would require 2.5 hours on average to come up with their own solution for the given problem. The use of the extended approach thus reduced the time spent to less than 20%.

However, these results are based only on one case and on the subjective judgment of the case study participants. Obviously, a more thorough validation needs to be done regarding the proposed approach in order to enable drawing more concrete conclusions about its efficiency. It will be necessary to perform similar case studies or full-size experiments while varying factors such as number of participants, level of participants’ experience, size of the catalog, experiment context, and type of tasks for the participants. Also, ways to validate the quality aspect of the resulting product have to be found.

5. Conclusion

In this paper, we have shown a way to collect and persist technology-specific architectural experience in an organization. Reuse of this experience during follow-up projects is supposed to increase the efficiency of the software architecture design process and the quality of the resulting software product. Furthermore, stored experience will lower an organization’s dependence on human resources.

We described how technology-specific experience can be persisted and reused beneficially within the architecture design process. For experience persistence we used challenge-solution pairs. We described a method for collecting these pairs using project postmortems and gave templates for their precise description with architectural scenarios and solution patterns. Finally, we gave an example of a typical challenge-solution pair for the technology of mobile apps and described it using given templates. A case study served as coarse validation and allowed us to draw first conclusions regarding the efficiency of a software architecture design process supplemented with technology-specific experience artifacts. Although this case study showed a noticeable increase in
efficiency, it is too early to draw final conclusions in this regard. A more thorough validation of the approach has to be performed.

References


[GH94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Massachusetts, 1994.


