Improving quality attributes of software systems through software architecture patterns
Harrison, Neil Bruce

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2011

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
8 Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues

This chapter consists of material from the following publications:

Harrison, N. and Avgeriou, P. “Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues – an Exploratory Study”, accepted to Transactions on Pattern Languages of Programming

Harrison, N. and Avgeriou, P. “Pattern-Based Architecture Reviews”, accepted to IEEE Software

Abstract

Architecture reviews are effective for identifying potential problems in architectures, particularly concerning the quality attributes of the system, but are generally expensive. We propose that architecture reviews based on the architecture patterns and their interactions with quality attributes can be done with small effort. We performed an exploratory study to investigate how much time and effort is required to perform such a review, and how many related issues it uncovers. We performed nine architecture reviews on small systems, and recorded the time and effort spent, and the number of issues identified. On average, a pattern-based review took less than two person-days of effort and less than a day of calendar time. The median number of issues identified was three, one of which was major. We recommend that where extensive architecture reviews are too expensive, a pattern-based review can be done with small effort and time.

8.1 Introduction

A particular challenge of quality attributes is that because they tend to be system-wide characteristics, system-wide approaches are needed to satisfy them; these approaches are defined at the system architecture level and not the component level. However, they cannot be fully tested until the software is undergoing system test. This creates a large gap between when approaches to satisfy quality attributes are designed and when they are completely validated.

Because quality attributes are largely constrained by the systems’ software architecture (see [22]), the architecture is often reviewed to determine how well the system will meet its quality attribute requirements. Numerous architecture review methods have been developed, and general guidelines have been established [1,
Several of the most prominent architecture review methods are classified and compared in [45], and further classifications are made in [8]. (Note that in this chapter a review is an independent examination of the architecture for the purpose of finding potential architectural issues; this is sometimes called an evaluation [38].)

Architecture reviews have been shown to be effective tools in uncovering architectural issues [38, 97], many of which are related to quality attributes. Maranzano, et al. report that the most common design issues found are in performance engineering, error handling and recovery, reliability and availability, operations administration and maintenance, and system evolvability [97]. Bass, et al. report common risk themes in projects using the Architecture Tradeoff Analysis Method (ATAM) include availability, performance, security, modifiability, and security [17]. Architecture reviews are, of course, most useful when done early in the development cycle, before too much code has been written [1]. ATAM reviews are well-known, and are similar to other architecture reviews (such as those discussed by Maranzano, et al.) For this reason, we have compared pattern-based architecture reviews, described below, to ATAM reviews.

Unfortunately, software architecture reviews have several important limitations. The most significant is that they require significant effort and time to complete. Abowd, et al. report that the average cost of an architecture review in AT&T was 70 staff days [1]. The SEI Software Architecture Analysis Method (SAAM) requires an average of 14 staff days [1]. Clements, et al. report that the approximate cost of an ATAM-based architecture evaluation ranges from 32 to 70 person-days (split roughly evenly between the evaluation team and the stakeholders) [38].

Closely related to effort is time. Maranzano, et al. report that the review preparation time lasts between two and six weeks [97], and the review itself lasts from one to five days. Clements, et al. state that ATAM-based evaluations should result in about three actual project days being consumed by the process [38].

In spite of the proven benefits of architecture reviews, these limitations make projects less willing to use them. This is particularly true of small projects, where time is severely limited. Many small projects also follow the Agile development philosophy [21], in which documentation such as architecture documentation is eschewed.

In response to this problem, we have developed a lightweight method for reviewing architectures called Pattern-Based Architecture Reviews (PBAR). It is a shift from a comprehensive architecture review to one that focuses on issues related to the satisfaction of the system’s important quality attributes. It uses the patterns found in the architecture to identify quality attribute issues.
8.1.1 Problem Statement
The overall problem can be stated as follows: is PBAR a viable architecture review method for small projects? By viable, we mean that the costs are low enough that small projects might consider using it, and that its benefits outweigh the costs of the review. As stated above we compare the costs of PBAR with those of ATAM.

It is important to learn this because if PBAR is a viable review method for lightweight software projects, it can help their developers find architectural issues related to quality attributes while the software is still in early development, and such issues are relatively easy to fix.

8.1.2 Research Objectives
Our research objectives are to empirically analyze the use of PBAR in small projects for the purpose of learning the following:

1. How much effort does a PBAR review take compared to an ATAM-based evaluation?
2. How much calendar time does a PBAR review require compared to an ATAM-based evaluation?
3. How many major and minor architectural issues can a PBAR review uncover?

This paper presents an empirical validation of PBAR with respect to these three questions. For the first question, we examined the effort in terms of person-days required to perform the review. For the second question, we looked at the duration of the review meetings. For the third question, we examined the number and type of issues found. We note that these research objectives are a part of the overall problem as stated above, and do not aim to quantify the effectiveness of PBAR as a whole. They are intended to give evidence that can be used to help reason about the effectiveness of such reviews.

This empirical study was exploratory in nature (see [88]). The type of exploratory study was an observational study done in the field: we collected data on actual reviews done on live projects. We began 9 reviews and completed and analysed data from 7 reviews. Details of the type of empirical study, as well as the setup and execution of the study, are provided in this chapter.

In the rest of this paper is as follows: Section 8.2 describes the PBAR process. It also describes related work. Section 8.3 presents the empirical study method used and explains how the study was planned and reviews set up. It then describes the execution of the reviews and the data collection. Section 8.4 gives a summary of the results, and section 8.5 gives our interpretation of the results. The final section gives conclusions.
8.2 The Pattern-Based Architecture Review (PBAR) Process
The goal of Pattern-Based Architecture Reviews is to provide an inexpensive architecture review process that can be used where traditional architecture review methods would not because of their high cost. The premise is that time and effort can be reduced through focusing review on the most important quality attributes, keeping the number of reviewers to a minimum, limiting the advance preparation, and limiting the meetings to the review meeting itself.

Figure 8.1: The PBAR Architecture Review Process

The structure of Pattern-Based Architecture Reviews (PBAR) is based on architecture review processes, namely the AT&T review process [97], which one of the authors used while an employee there, as well as ATAM [15]. It is simpler than these processes chiefly in that the up-front work is reduced, and that the focus is narrowed to the quality attributes and patterns used. While architecture patterns may be identified during ATAM and other reviews, PBAR’s pattern focus also comes from reviews of organizations, where patterns were used to identify organizational issues [43]. The key differences between PBAR and traditional architecture reviews are summarized in this table:

Table 8.1: Key Focus Differences in Review Methods

<table>
<thead>
<tr>
<th>Traditional Architecture Review</th>
<th>Pattern-Based Architecture Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on all requirements</td>
<td>Focus on key quality attribute requirements</td>
</tr>
<tr>
<td>Discovery through analysis and other study methods for satisfaction of requirements</td>
<td>Discovery through pattern identification and association with quality attributes</td>
</tr>
</tbody>
</table>
We emphasize that this process came from our experience in conducting architecture reviews and our desire to make them simple and practical for small projects. Thus the process is heuristic rather than formal. Its audience is practitioners who need simple solutions, hence the description of the process is also simplified and oriented towards this audience. It is aimed at producing pragmatic results rather than rigorous verification and validation of an architecture. The flow of the reviews is as follows:

**Resources and Planning:** At least one reviewer is needed. On the project side, all developers are encouraged to take part in the review; other stakeholders are not required.

**Preparation:** A few days before the review, the reviewer studies any available documentation of the system architecture and the system requirements, especially the quality attribute requirements. The team should not be asked to produce any documentation especially for the review.

**The Review Meeting:** The review is a face-to-face meeting. The architect and key developers must be present, but the entire development team is encouraged to participate. During the review, do the following:

1. Review the major requirements of the system, and identify the most important quality attributes.
   a. If no requirements were provided, the team provides them during the review. The reviewer usually asks what the major capabilities of the system are.
   b. With the team, identify the most important quality attributes.
   c. Elicit more detail about the quality attributes by discussing scenarios of the functional requirements. For example, when walking through a purchase scenario in an e-commerce system one might ask, “What should
happen if the link to the customer drops just before a purchase is finalized?"

2. Discuss the architecture of the system. If no architecture diagram was provided, ask (or help) the team to draw one.
   a. It is useful to draw the architecture on a white board, even if it was provided beforehand.

3. Identify the architecture patterns used in the architecture. This can be done in various ways, such as the following:
   a. Look for pattern names in architecture documentation, module names, or annotations on modules. For example, a module named “Shared Database Layer” would indicate the presence of the Layers pattern as well as a member of the Repository family (Shared Repository, Active Repository, or Blackboard; for descriptions, see [11, 33].)
   b. Match structural decomposition (components and connectors) to the structures of the patterns. For example, data flow through sequential modules indicates the use of Pipes and Filters [33]. This method is a common way of identifying patterns, particularly where architects are unfamiliar with patterns. Note that it has been shown that architecture patterns can be found through architecture diagrams [62].
   c. Identify the tactics used [15], and map them to potential patterns that may implement them. For example, the security tactics of Limit Exposure and Limit Access [15] suggest the use of the Layers [33] architecture pattern.

4. Examine the architecture and quality attributes together to determine the effects of each pattern on the system’s quality attributes.
   a. Identify the tactics (to be) used to implement the quality attributes. It is useful to examine how the tactics might be implemented in the patterns, and how easy such implementations are. For example, see [65] for information on reliability tactics and patterns.
   b. Review the scenarios previously used, and discuss what tactics will be used to implement the quality attribute measures discussed, and where in the architecture they belong. This highlights both missing tactics and tactics that may not be fit well in the patterns used.
c. It is useful to annotate the architecture diagram on the white board with notes showing where the various tactics are (to be) implemented.

5. Identify issues; including the following:

a. Quality attributes not addressed or not adequately satisfied. If feasible, discuss possible solutions (e.g., certain tactics that might be used.)

b. Patterns that are not used that might be useful

c. Potential conflicts between patterns used and quality attributes. For example, a layered architecture is often incompatible with a high performance requirement.

**Follow-up:** After the review, the reviewer may perform further analysis of the data. In any case, create a short summary report for the project. It is useful to meet with the entire team for a feedback session.

### 8.2.4 Related Work

Related work consists of surveys and descriptions of architecture review methods, and empirical studies related to architectural review. We begin with surveys and descriptions of review methods.

Numerous architecture review methods have been developed. Virtually all the well-known methods are based on using scenarios to evaluate the architecture; these include Scenario-Based Architecture Analysis Method (SAAM) [86] and some variants, Architecture Tradeoff Analysis Method (ATAM) [87], Active Reviews for Intermediate Design (ARID) [36], Architecture-Level Modifiability Analysis (ALMA) [25], Architecture-Level Prediction of Software Maintenance (ALPSM) [23], and Scenario-Based Architecture Reengineering (SBAR) [24]. Dobrica and Niemelä have surveyed these architecture analysis methods, and compared eight elements of them [45]. Ali-Babar, et al. performed a subsequent study, and included additional elements [8]. The following table shows five elements selected from these two studies where PBAR has significant differences from many other methods (the other elements have insignificant differences). These differences reflect tradeoffs made in order to accommodate the low cost needs and characteristics of small projects, such as minimal architectural documentation. This table shows how specific elements of architecture reviews are done in PBAR. It also shows how PBAR relates in these respects to specific architecture review methods.
Table 8.2: SA Review Method Classification Applied to PBAR

<table>
<thead>
<tr>
<th>Method Element</th>
<th>PBAR’s Focus; Key Differences From Other Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method’s goals [8]</td>
<td>Evaluate the ability of SA to achieve quality attributes. Focus only on QAs and not functional requirements (Similar to SBAR [24], but SBAR is for analysis of existing systems.) PBAR trades off comprehensive coverage (such as ATAM [87]), for lower cost, focusing on a high payoff area.</td>
</tr>
<tr>
<td>Quality attributes [45]</td>
<td>Multiple quality attributes (similar to SBAR [24] and ATAM [87].) This trades off depth in a single QA for breadth across many. Some methods focus on a single QA (e.g., ALMA [25] focuses on maintainability.)</td>
</tr>
<tr>
<td>Architectural description [8]</td>
<td>Uses whatever the project has; if none is available, use verbal descriptions and whiteboard sketches. All other methods require some architectural description; PBAR attempts to work with inferior or nonexistent documentation, because if extensive documentation is required, projects may not afford the review at all.</td>
</tr>
<tr>
<td>Reusability of existing knowledge [45]</td>
<td>Use existing (external) architecture pattern and tactic knowledge [15, 65]. Other methods use none, or focus on use and maintenance of internal experience repositories [87, 23]. PBAR uses general knowledge, which is inferior to internal experience repositories, but small projects usually have no such repositories.</td>
</tr>
<tr>
<td>Resources Required [8]</td>
<td>Very few resources required; see later section. All projects that specified resource requirements were much higher; see [87] for example.</td>
</tr>
</tbody>
</table>

We note that we are not comparing PBAR to any existing method, but rather propose that PBAR provides a way for projects to get some of the quality attribute-related benefits of reviews in cases where the costs of such reviews are prohibitive. PBAR can also be an inexpensive way to identify issues, prior to following up with a more rigorous review method.

Bachmann, et al. [12] propose a method for designing software architectures to achieve quality attributes that is based on a reasoning framework. The reasoning
framework helps architects create an architecture that meets the needs of quality attribute scenarios, which may include the selection of patterns and tactics. PBAR is similar in that it focuses on quality attributes, as well as uses patterns and tactics. However, PBAR is a review approach rather than an architectural design method, and uses pattern and tactic documentation rather than a reasoning framework.

Zhu, et al. describe an approach to extract scenario and tactic information from patterns, for the purpose of selecting and calibrating a quality attribute reasoning framework [137]. The reasoning framework is then used, as noted above, in architecture reviews. Our approach is somewhat the other way around: quality attribute scenarios are derived from user stories, and compared against the architecture patterns in use.

An empirical analysis of architecture reviews was performed by Bass, et al., who extensively analysed data from 18 final reports of architecture reviews [17]. They categorize the most common risk themes identified. They also note that about twice as many risk themes are risks of “omission” rather than risks of “commission.” We did not classify the issues we found into their risk theme categories, nor did we identify risks of omission vs. commission; although we saw issues of both types. We note that all their reviews were of large projects, while ours were almost exclusively small projects. We do not know if this has an effect on the number and type issues identified.

8.3 Case Study Design

8.3.1 Study Type
This study is exploratory in nature (see [88]), meaning that we set out to learn about the nature of the treatment (specifically the effort and calendar time required by PBAR, and number of issues it identifies), rather than to formulate a hypothesis to test the treatment against. We were also not attempting to compare PBAR to established architecture review methods. It is also exploratory because there have been no other empirical observations of pattern-based architectural reviews such as PBAR, and thus the underlying theory has not been established yet. This study should provide a basis for future formal studies testing hypotheses about the effectiveness of these reviews.

The empirical method we used was an observational study, which is an in situ study of industrial practice [88]. Alavi and Carlson call it a field study, and state that it entails no manipulation of independent variables and is carried out in the natural settings of the phenomenon of interest [3]. In our studies, independent variables included project size, domain (in particular, which quality attributes were important), and background of participants.
Other methods of empirical studies might be considered for this study [74]; the following are three possible alternatives, along with reasons we didn’t use them.

**Controlled Method:** “… provides for multiple instances of an observation in order to provide for statistical validity of the results.” [136] The difficulties of finding real world projects that allowed for controlling the variables of domain, personnel, etc., rendered this approach impossible.

**Laboratory Study:** Similar to a controlled method; “… examination of computer-organization problems within a research goal setting of acquiring knowledge that is separate and distinct from the normal operational goals of the organization under study” [131]. Kitchenham, et al. refer to these as formal experiments [88]. In comparing observational studies to formal studies, Kitchenham, et al. note that, “Observational studies give a better overview of the full range of benefits accruing from inspections [than do formal studies].”[88] As an architecture review is very similar to an inspection (quote refers to Fagan inspections, studied in [46]), we prefer an observational study to a formal study.

**Grounded Theory:** Focuses on observations made in the real world: “… to develop a theory from data rather than gather data in order to test a theory of hypothesis. This means that qualitative methods are used to obtain data about a phenomenon and that a theory emerges from the data” [58]. Because we had specific research goals, we did not choose a grounded theory study.

We report the details of the study and its results in the next sections, following the model given by Wohlin, et al. [133], which Jedlitschka and Ciolkowski note is for generic empirical research [83]. (As their proposal describes guidelines for reporting controlled experiments, it was not a good fit for this study.) The introduction and problem statement are given in section 1. The following section, case and subject selection, describes the projects selected. The next section, Experiment Operation describes how the reviews were done. Section 4 presents the data, and sections 5 and 6 discuss its interpretation, including threats to validity and limitations of application. We finish with conclusions and future work.

**8.3.2 Case and Subject Selection**

We selected nine projects for the experiment. The projects were selected for their availability and ability to participate in the study.

We classified the projects based on number of developers involved. Large projects had more than 30 developers, medium had 10 to 30, developers, and small were less than 10. The projects were as follows:
All the projects studied except one were medium or small, with 30 or fewer people on the project. Most of the projects had fewer than 10 people working on them. The reason we use number of people on the project as a size metric is twofold. First, most of the reviews were done before the code was complete, so the size in terms of implementation (e.g. lines of code) was not known. Second, a functionality metric such as function points, feature points, or even user stories could not be used because few if any of the projects used them. We did not observe any notable differences in the results of the review due to the sizes of the projects.

The methodology the projects used was not specified, as the review is independent of the project methodology.

Architecture reviews are generally done in the early phases of projects [1]. We performed most reviews during the development phase, although two were done in later phases. The reviews that were performed in later phases ended up serving as validity checks for issues identified (see interpretation of results.) The only difference we observed between early and late phases was that the projects in late phases did not make changes based on the issues found. One would expect this; indeed, architecture reviews are to be done as early as is practical.

There were no constraints put on the application domains. Of course, the size of the project plays a significant role in the type of application; for example one would not expect life-critical systems to be done in the small teams such as we studied. Our domains were concentrated more in web-based applications, but the driver was size, rather than application domain.

Five of the projects were student capstone projects, the rest were industry projects. Note that the capstone projects were all projects done for real external customers; none was a canned project for the students. In addition, nearly all the students who participated had part-time or full-time jobs in industry, and had several years of professional experience. Each team had at least one team member with over 3 years professional experience. The reviews were scheduled at the convenience of the project.

The following table identifies each project studied.
Table 8.3: PBAR Projects Studied

<table>
<thead>
<tr>
<th>System</th>
<th>Size</th>
<th>Project Phase</th>
<th>Project Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Large</td>
<td>Implementation</td>
<td>Streaming data manipulation and analysis</td>
</tr>
<tr>
<td>B</td>
<td>Medium</td>
<td>Architecture</td>
<td>Computer-controlled process control (similar to an automated assembly line)</td>
</tr>
<tr>
<td>C</td>
<td>Small</td>
<td>Post release</td>
<td>Embedded GPS platform application</td>
</tr>
<tr>
<td>D</td>
<td>Small</td>
<td>Early Implementation</td>
<td>Web-based time tracking system</td>
</tr>
<tr>
<td>E</td>
<td>Small</td>
<td>Early Implementation</td>
<td>Distributed data entry and subscription management system</td>
</tr>
<tr>
<td>F</td>
<td>Small</td>
<td>Early Implementation</td>
<td>E-commerce inventory management system</td>
</tr>
<tr>
<td>G</td>
<td>Small</td>
<td>Early Implementation</td>
<td>Android™1 phone application</td>
</tr>
<tr>
<td>H</td>
<td>Small</td>
<td>Early Implementation</td>
<td>Web-based game platform</td>
</tr>
<tr>
<td>I</td>
<td>Small</td>
<td>Architecture</td>
<td>Web-based business process support system</td>
</tr>
</tbody>
</table>

8.3.3 Experiment Operation
We performed each review according to the process described in Section 2.3. The independent variable in the study was the review approach, namely PBAR, as compared to ATAM. We collected data for the following dependant variables:

- Number and importance of issues uncovered, as well as which quality attributes were affected.
- Time required for the review
- Effort required for the review

---

1 Android is a trademark of Google Inc. Use of this trademark is subject to Google Permissions.
We also measured the following extraneous variables, in order to calibrate the results as necessary:

- Project size, as measured by number of people on the project.
- Phase in development
- Quality attributes important to the project
- Number of patterns identified

In order to avoid “false positives,” identifying issues that weren’t really there, we did the following to verify the validity of the issues we found:

- Asked the participants whether each issue was indeed a real problem.
- Followed up at a later time to see whether the team acted on the issues. Lack of action does not mean that an issue is not real, but action does indicate that an issue is real.
- Ascertained whether the issues had already been considered, and possibly corrective action taken.

For each project, we filled out a summary report template that contained a description of the project and the data collected, and returned it to the project. A sample is found in appendix C.

Two of the reviews were not completed. In project I, the review was not completed at all. In project A, the review itself was completed but feedback from the architects was not obtained. Both these reviews involved deviations from the process and thus provide possible insights into the process, as we will discuss later.

### 8.4 Data Analysis

Our research objectives stated in section 1.1 were to determine how much effort is required for a PBAR review, how much calendar time is required, and how many issues are typically found. As this is an exploratory study, we did not test a hypothesis, but rather gathered information so that hypotheses about the effectiveness of PBAR can be made.

#### 8.4.1 Effort Required

The effort required depends on the number of reviewers and the number of stakeholders involved in the review. This is important not only for determining the effort required, but also when comparing the efforts of different types of reviews.

In all our studies, a single reviewer was involved, so the reviewer effort can easily be calculated. The number of stakeholders participating varied, so we calculated the effort per person. Because there is no preparation required for stakeholders (unlike other review types), the total stakeholder effort was in the review meeting and the
follow-up session. So we were able to average the meeting time to come up with the per stakeholder effort. Table 8.4 shows the breakdown of effort for the reviewers, compared to various versions of the ATAM method. We note that Clements, et al. [38] do not precisely define the difference between ATAM-Medium and ATAM-Small; it appears that ATAM-Small is a scaling down of the medium review.

Table 8.4: Reviewer Effort in person-days, by Phase (review team size: 1)

<table>
<thead>
<tr>
<th></th>
<th>PBAR</th>
<th>ATAM-Medium</th>
<th>ATAM-Medium -- Checklists</th>
<th>ATAM-Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>0.5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Review Meeting(s)</td>
<td>0.25</td>
<td>20</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>Follow-up</td>
<td>0.75</td>
<td>15</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>Total Effort</td>
<td>&lt; 2</td>
<td>36</td>
<td>25</td>
<td>15</td>
</tr>
</tbody>
</table>

Using a second reviewer would be desirable, and we surmise that adding a second reviewer would not double the effort of a single reviewer. The reason is that much of the follow-up effort is devoted to writing a summary of recommendations; the effort for this task would not double with the doubling of the reviewers.

Table 8.5 summarizes the effort of the stakeholders. The descriptions of ATAM split the stakeholders into project decision makers and other stakeholders. Project decision makers include architect, project manager, and in the case of the medium reviews, the customer. In these tables, the ATAM entries show only the effort of the project decision makers.

Table 8.5: Stakeholder Effort in person-days, by Phase

<table>
<thead>
<tr>
<th></th>
<th>PBAR (5 stakeholders)</th>
<th>ATAM-Medium (3 stakeholders)</th>
<th>ATAM-Medium -- Checklists (3 stakeholders)</th>
<th>ATAM-Small (2 stakeholders)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Review Meeting(s)</td>
<td>1.25</td>
<td>12</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Follow-up</td>
<td>&lt; 1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>2</td>
<td>18</td>
<td>13</td>
<td>11</td>
</tr>
</tbody>
</table>

In PBAR, as many as all developers may attend, so the total effort will vary. We found that the effort required for each person averaged 2-2.5 hours for the review meeting and half an hour or less for the follow-up meeting, for a total of 3 hours per
person. The table above shows a team of 5; larger organizations will have a proportionally larger effort.

### 8.4.2 Calendar Time Required

In the following table, we see that PBAR requires somewhat less calendar time than ATAM-Small reviews. Clements, et al. do not summarize calendar time, but the time given below is based on times given for various parts of the review.

<table>
<thead>
<tr>
<th>Calendar Time</th>
<th>PBAR</th>
<th>ATAM-Small</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; 1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8.6: Calendar Time for Review Types, in days

Clements, et al. state that for ATAM reviews, the calendar time added to a project is minimal. Preparation and follow-up are done behind the scenes, and thus have no schedule impact. The review meetings usually consume about three days [38]. In PBAR reviews, preparation and follow-up were also done behind the scenes. The difference then is that PBAR review meetings consumed less than one day, while ATAM meetings generally last two or more days. Note that because the entire development team is encouraged to attend PBAR meetings, the schedule impact is equivalent to meeting length: a PBAR that consumes half a day causes half a day impact to the schedule. ATAM reviews appear not to require the entire time, so the calendar time is estimated; Bass, et al. do not give an exact figure.

### 8.4.3 Number of Issues Identified

The following table summarizes the issues found in the reviews (as noted above, reviews A and I were omitted because they were not completed):
Table 8.7: Issues Found in Reviews

<table>
<thead>
<tr>
<th>System</th>
<th>Development Phase</th>
<th>Patterns Identified</th>
<th>Total Issues Identified</th>
<th>Major Issues</th>
<th>Major Issues Implemented</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Architecture</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>Released, Legacy</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>Early Impl.</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>Early Impl.</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>Early Impl.</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>Early Impl</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>Early Impl</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Median</td>
<td></td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The number of patterns identified is consistent with patterns found in other systems (see [62].) Since patterns provide ready-made documentation, they are useful during the review. However, the number of patterns is not expected to be correlated with the number of issues identified; indeed, a simple inspection of the data indicates it is not.

Issues were defined as potential problems with satisfying a quality attribute. Major issues were those issues that had the potential to compromise the satisfaction of a quality attribute, as agreed by the team. Examples of major and minor issues are:

- **Major issue:** In the time tracking system (project D), if a user logs in, and the link between the user’s client (which tracks their work session) and the server goes down, and later comes up, an inaccurate time record may be generated. (Quality attribute: Reliability.)
- **Minor issue:** In the e-commerce support system (project F), the system allowed file transfers, but the verification of the file transfer was left to the user to do manually. Automated transfer verification would be nicer, but users were billed for per message, so automated verification would cost them. Perhaps a future change would be to allow the verification to be free. (Quality Attribute: Usability.)
The table above also shows that all major issues except two were fixed. However, the major issue identified and not fixed in project B had already been discussed by the team, and they had decided that the benefits outweighed the liability of the issue. When that is taken into account, only one major issue was not fixed. This tends to support the notion that the issues detected by PBAR are valid issues.

An important consideration in any review is whether important issues are missed by the review. We do not know of any information about how many issues are missed by any architecture review method; PBAR or published methods such as ATAM. However, one might consider comparing the issues found by each method on a review of the same architecture, to determine whether one method finds issues missed by the other. We have not done this yet. Such a comparison will be most effective if the nature of the issues found and missed is explored; it may show, for example, that PBAR is well-suited or ill-suited for identifying issues related to certain quality attributes.

8.5 Interpretation of Results

8.5.1 Effort Required

It is striking to see the large difference in effort between PBAR and review methods such as ATAM. We describe the probable reasons that the effort for PBAR is so low, based on observations of these experiences. This validates the intent of PBAR given above.

1. Much of the time difference is in the review meetings. PBAR has a narrow focus. Rather than trying to discover every potential issue, PBAR focuses on the aspects of the most important quality attributes of a project. This means that the review does not have to exercise a complete set of scenarios, but rather focus only on a rather limited set of scenarios associated with the most important quality attributes.

2. Another reason for the differences in review meeting effort is that PBAR has a simple, single meeting format. ATAM has a two-phase evaluation activity, which accounts for one fourth to one half of the meeting effort. Some of the effort in the first phase might be considered preparation for the review; PBAR has few preparation activities for the stakeholders.

3. Because the review has a narrow focus, fewer issues will be found. Thus, the follow-up work will be less.

4. In our study PBAR employed only one reviewer; this reduces the overall effort.
5. The projects we reviewed with PBAR were small, so the architectures will surely be simpler than large projects. It should take considerably less time to go through a small architecture than a large one.

Is the cost low enough that small projects are likely to use PBAR? Clearly, small projects are more likely to use a low-cost review method than a high cost one. Not only is the effort small, but the calendar time required for the review meeting is short; under a day. This is important, as many small projects have short release cycles (3-4 weeks [40]), and may have daily units of work [114]. We recommend that this validation be done.

8.5.2 Time Required
PBAR requires somewhat less calendar time than reviews such as ATAM, although the difference is not as striking as the difference in effort. The main reason for the difference is that the review meeting, being focused rather than comprehensive, is considerably shorter. This begins to suggest that PBAR may present an opportunity for a project to make tradeoffs: a project where development schedule is critical may choose the focus of a PBAR-style review, where a project that needs comprehensive examination will wish a review similar to ATAM. We speculate that there may be approaches to reviews between PBAR and ATAM that may be more comprehensive than PBAR yet lighter weight than ATAM.

8.5.3 Number and Types of Issues Identified
We see that all the reviews found relevant issues, some of which were major. This indicates that PBAR is successful at finding issues. This begs the question of whether PBAR finds all the issues, or even a significant portion of important issues. Naturally, we cannot know if PBAR – or any architecture review method – detects all the issues. It is logical to assume that PBAR does not find all the issues that a comprehensive review does, but we have no data that quantifies the difference. In order to quantify the difference, one would need to compare PBAR with another method in a formal experiment. This may be an area of future research.

We note that the projects studied had a wide range of important quality attributes. We also note that issues were found among nearly all the quality attributes that were identified. This shows some diversity among the projects, and indicates that PBAR is likely to be useful across several different domains.

Bass, et al. [15] list several benefits that architecture reviews have beyond identification of errors. It is useful to see whether PBAR provides the same benefits. Based on our observations of the reviews, but not based on any rigorous study, we suggest that PBAR may have the benefits that Bass, et al. present:
1. Puts Stakeholders in the Same Room: Because PBAR employs a review meeting, it does this. In six of the nine reviews, all developers were present, but non-developer stakeholders were not present. Thus PBAR has some of this benefit.

2. Forces an Articulation of Specific Quality Goals: During each of the reviews, the teams were asked to identify the most important quality attributes, and their acceptable levels. Goals not associated with quality attributes were not explored.

3. Results in the Prioritization of Conflicting Goals: Teams were asked to identify the most important quality attribute goals, although it was informal.

4. Forces a Clear Explication of the Architecture: Each team was required to explain their architecture; some teams without architecture documentation produced architecture diagrams during the review. In fact, people from two different projects mentioned this benefit in their feedback. One of these was a team of only four people and there was different architectural understanding even within such a small team.

5. Improves the Quality of Architectural Documentation: Where no architectural documentation existed, teams produced architecture diagrams during the review. Improvements to existing architecture documentation were not assessed.

6. Uncovers Opportunities for Cross-Project Reuse: We observed no evidence of this benefit.

7. Results in Improved Architecture Practices: Bass, et al. explain that organizations that use architectural reviews as part of their development process have improved architectures over time. We observed some evidence of this benefit in that teams increased their knowledge and use of architecture patterns. At the beginning of the reviews, none of the participants had extensive knowledge of architecture patterns, and some were completely unfamiliar with them. In the review, as patterns came up, they were discussed, along with their advantages and disadvantages.

Our experiences with the two reviews that were not completed gave us some insights into the process. In project A we were unable to meet with the team. Because the project had extensive architecture documentation, we performed the review offline, in the spirit of Johnson and Tjahjono [84]. While we found several issues including two we considered major, the project did not give us feedback about the nature of the issues. This indicates the value of a face-to-face meeting, although
one could certainly consider a distributed meeting [7]. In project I, we attempted to use a different reviewer, but the reviewer did not have architectural expertise, and the review fell flat. It seems obvious that the reviewer should be highly experienced with architecture; we found it to be true.

We also considered types of issues identified, where types are based on applicability to different quality attributes. The data in table 8 show that reliability issues were prominent, and may indicate that PBAR reviews are well suited to find such issues. However, we note that the data are not conclusive.

Table 8.8: Quality Attributes in the Systems

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Number of Systems where Important</th>
<th>Total Issues Found</th>
<th>Number of major issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Reliability, (inc. Fault Tolerance)</td>
<td>7</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Usability</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Security</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Maintainability, Extensibility</td>
<td>5</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Portability</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Configurability</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8.6. Limitations of the Study

The validity of the study is limited by several factors. We note the following limitations:

1. Kitchenham, et al. [88] note that evaluating one’s own work, as we have done here, can lead to positive bias. In this case, there could be bias to overstate the effectiveness of the reviews; e.g., find more issues than were really there. In order to avoid this bias, we draw no conclusions about the effectiveness of the reviews, and instead have focused on effort and issues. We validated the issues identified by asking participants during the review whether the issues were legitimate or not.

2. Some of the participants were students of the reviewer, and may have felt pressure to conform. This would not affect the number of issues identified, but might have affected whether they fixed the issues. To counter this, the students were instructed that the review was for research purposes, and the results of the review, and their response to it, in no way affected their grade (i.e., there was no requirement at all to fix any of the issues that came up.)
3. The number and quality of issues found by a review depends not only on the review process, but on the quality of the architecture (including the experience of the architects involved as well as the quality and quantity of architectural documentation) and the maturity of the architecture. Therefore, we do not make claims about how many issues one might find in a typical PBAR. We also note the phase of the project (see table 7), although we did not have enough projects, particularly in phases other than early implementation, to make any distinctions based on project phase.

4. The study does not identify important architectural issues that PBAR may miss. Therefore, we do not recommend that PBAR substitute for more extensive architecture review methods, but rather as a tool where expensive architecture methods are not practical.

In addition, Falessi, et al. [47] describe lessons learned from applying empirical software engineering to software architecture, and note several challenges that may threaten the validity of such studies. We note the following challenges are particularly relevant to this study:

1. Describing the Desired Return on Investment: In our case, this relates to cost versus the benefit of PBAR reviews as compared to ATAM-style reviews. While the costs of each can be quantified, the benefits are difficult to characterize. A major reason for this is that PBAR reviews have a narrow focus, which makes the benefit difficult to compare against ATAM. But a more basic issue is that benefit itself is difficult to measure; number of issues identified, for example, is at best a gross indicator of benefit. Therefore, we do not draw any conclusions about the benefit of the reviews.

2. Describing Social Factors: In particular, the development team size can influence the architecture process, and the subsequent architecture. Of course, the results of an architecture review are dependent on the quality of the architecture reviewed. Nearly all of the projects studied had uniform (small) team sizes, so there was consistency across projects. We note that the larger projects we studied had similar results, but we do not claim similar results for larger teams.

3. Evaluating the Software Architecture Without Analyzing the Resulting System: The fact that we did not evaluate the resulting systems weakens the validation of the issues identified: we did not verify that the issues were indeed important to have identified. One mitigation action we undertook was to determine whether the issues were resolved – a resolution of an issue
indicates that the development team felt it was important enough to rectify. This adds some credibility to the issues.

4. Subjects: Falessi, et al. note that software architecture decision making requires a high level of experience. Inexperienced architects can be expected to make architectural errors, including errors of omission: not considering important issues such as quality attribute issues. Reviews of architectures produced by inexperienced architects would therefore logically be expected to find a greater number of issues. Many of these projects’ developers were students, although most students did have professional experience as well. In addition, every project studied did include at least one highly experienced person. Nonetheless, the number of issues identified could be somewhat high. We expect that this would have no impact on the time and effort required.

5. Training: The particular issue with this study is that the effectiveness of a pattern-based review requires that the reviewer have significant knowledge of architecture patterns and their interactions with quality attributes. Our reviews all used the same reviewer, which ensured consistency across reviews, but exposes a significant weakness in that the technique may not have the same results with a reviewer who has a different background. We recognize this limitation in generalizing the results, and recommend this be addressed in future work.

6. Complexity: Falessi, et al. note that software architecture is really useful only for large software systems, but empirical studies frequently use small of simple systems. This is true of this study. We note that complex systems may indeed have more issues than we saw in the small reviews, but the time and effort require may well increase. However, our focus was on small projects as the main target group for PBAR.

8.7. Applications to Production-Focused Projects

In spite of the demonstrated benefits of architecture reviews, many projects are unable or unwilling to use them. These projects tend to have the following characteristics:

- Short schedules, possibly including repeated development episodes with very short cycles, such as 3 weeks
- Tight deadlines, leaving little or no time for activities not focused on producing the product.
- Neglected documentation, especially internal documentation such as architecture documents.
• Frequently changing technological and/or user requirements
• Typically small teams

This leads to a focus on producing working software, or “getting the product out the door.” All activities that do not directly and immediately contribute to this goal are lower priority. For lack of a better term, we will call this class of projects, “production-focused” projects. Many such projects follow practices found in Agile and Lean software development methodologies [20, 41,109], though not all do.

Typical software architecture review practices as described in [45] examine quality attributes in architectures in depth. However, they have key incompatibilities with production-focused projects, including:

• For production-focused projects, a review must not require much effort and calendar time. These typically small projects generally have no resources beyond those needed to just write the software. But published architecture review methods generally are expensive; for example, Clements, et al. report that the approximate cost of an ATAM-based architecture evaluation for even a small project is 32 person-days [38]. Abowd, et al. report that the average cost of an architecture review in AT&T was 70 staff days.

• Production-focused projects have little architectural documentation, but many architecture review methods base much of their analysis on the architecture documentation [38]. One of the most frequently cited problems in conducting architecture reviews is the inadequacy of architecture documentation [85].

• These projects must deal with changing requirements. However, architecture reviews methods delve into the requirements in considerable detail, requiring detailed requirements specification and corresponding stability of the requirements. They also require significant preparation time (Maranzano reports that preparation time lasts between two and six weeks [97].) Extensive preparation hinders a review from being held in response to changing requirements.

These incompatibilities lead to projects choosing to not review their architecture, and forgoing the benefits of such reviews. However, we have found that a lightweight review process that addresses these incompatibilities allows projects to have some of the benefits of an architecture review.

Pattern-Based Architecture Reviews (PBAR) leverage the patterns in the architecture and their relationships with quality attributes to create a review that is compatible
with production-focused projects. It addresses the key incompatibilities between these projects and traditional architecture reviews:

- PBAR requires only a small amount of time and effort. This makes it more compatible with projects that are typically small and are focused on writing production code.
- PBAR does not require architecture documentation. Instead, it finds the architecture patterns in use, and leverages the existing pattern documentation to make inferences about how quality attributes are to be implemented within the context of those patterns.
- Production-focused projects accommodate changing requirements. PBAR has a short preparation time, a short review, and can return feedback to the project within one or two days. This allows it to be used on short notice in response to changing requirements.

In addition to these characteristics, PBAR is also compatible with common practices in production-focused projects. Table 8.9 shows typical practices of these projects, and shows how PBAR accommodates these practices versus traditional heavyweight reviews. These practices are also found in many Agile and Lean methodologies, and references to such are given so that the reader can get more information about the practices. Note that not all production-focused projects follow agile methodologies and conversely not all agile projects are production-focused.

Table 8.9: Common Practices of Production-focused Projects and Architecture Review Practices

<table>
<thead>
<tr>
<th>Production-focused Practice</th>
<th>PBAR</th>
<th>Traditional Reviews</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent Releases [20, 41,109]</td>
<td>Short review-feedback cycle fits well in small release windows. PBAR can also be scheduled between early releases.</td>
<td>Long planning-review-feedback time can cut across releases; would not be practical between releases.</td>
</tr>
<tr>
<td>Accommodate changes to user needs [20,109]</td>
<td>Focus on quality attributes, which are more stable than functional requirements. Allows features to change.</td>
<td>Comprehensive architectural examination requires stability of requirements, including functional requirements.</td>
</tr>
<tr>
<td>Lightweight architecture and requirements documentation [20,41]</td>
<td>No special documentation required. Leverages knowledge in patterns about architecture-QA issues.</td>
<td>Based on architecture documents; encourages extensive architecture documentation; may require some to be written for the review.</td>
</tr>
<tr>
<td>Walking Skeleton [41]</td>
<td>Implementation of a walking skeleton is natural time to hold PBAR; PBAR can be scheduled in response to walking skeleton being implemented.</td>
<td>Would be a natural time for a review, but the extensive planning required forces calendar-based scheduling, not scheduling based on an event like implementation of the walking skeleton.</td>
</tr>
</tbody>
</table>

Let us examine each practice:

**Frequent Releases**: In order to increase flexibility, projects may have frequent internal or external releases. An architecture review should fit into this time: the planning should be short, and the review itself must be of short duration. The fact that PBAR requires no participant advance preparation allows PBAR to be flexibly scheduled, and its short duration is only a minor disruption to even a very short release cycle.

**Accommodate Changes to User Needs**: Comprehensive architecture reviews are based on requirements specifications, generally written. But because requirements often change the utility of the review is reduced. PBAR focuses on quality attributes, which are likely to be more stable than functionality requirements.

**Lightweight documentation**: Traditional reviews tend to be based on comprehensive architecture documentation, but it can simply be too much work for a project to produce it. PBAR is a lighter weight alternative in these cases.

**Walking Skeleton**: A Walking Skeleton is an early end-to-end implementation of the architecture, often used like prototyping to help prove architectural concepts. The completion of a walking skeleton is an ideal time for an architecture review. Because of the small preparation time and effort needed, PBAR can be held as soon as a walking skeleton is implemented. On the other hand, a traditional review needs considerable planning and up-front work; thus it cannot be scheduled on a moment’s notice.

**8.7.1. Case Study**
Here we summarize the PBAR review of project G, as a production-focused project. This project was a student capstone project, and the students had no extra time for a lengthy review. The project was very small, with only 3 developers. The team followed no particular methodology, had few written requirements, and no written architecture documentation. An additional challenge was that the Android software development kit (SDK) was very new at the time, and was under constant change.
This affected how features were to be developed, and had some impact on what features could be implemented in the time given.

We began the review by discussing the functional and quality attribute requirements. There were four important quality attributes; usability, security, reliability (fault tolerance), and extensibility. We walked through scenarios to help us understand the quality attributes. This was especially helpful for exploring fault tolerance.

We then discussed the architecture, and the team drew the architecture on the board, using boxes and lines to represent components and connectors. A team member took notes, so at the end of the review they did have some architecture documentation. We identified two architecture patterns: Peer-to-Peer, and Shared Repository (see [11] for descriptions.)

We identified three issues with the quality attributes, one of which was significant. We discussed ways to resolve the issues, and identified three measures they could implement to do so (Bass, et al. call these “Tactics” [15].) We annotated the architecture diagram with notes about where these tactics would be implemented, thus giving the team a “map” of how to implement them. This concluded the review, which took under 2 hours.

The team noted these specific benefits to the review:

- They now had some architecture documentation
- They understood their architecture better
- They understood the quality attribute requirements better
- They had some issues with proposed solutions

This experience demonstrated that PBAR was useful even when the architecture documentation is entirely nonexistent and the requirements are only sparsely documented.

### 8.8. Considerations and Limitations for Production-Focused Projects

Based on these experiences, we list some considerations for conducting PBAR reviews, especially in production-focused environments.

First, the architecture reviewer must come from outside the project. This is the case with all types of reviews, and similar to the rationale for pair programming – a separate set of eyes can detect problems that project members are blind to. Note that having a team of two reviewers is better, if possible.

When should a review be done? The principle is that the review should be done as early as possible, once enough of the architecture is in place to hold a meaningful
review. It is important to note that because of the lightweight nature of PBAR, it can be done very early, even before the architecture has solidified. However, if the quality attribute requirements are not yet solidified, the review will likely fail. As noted above, the low preparation time and overall cost of PBAR allows it to be held in response to changing requirements; significant changes to quality attribute requirements can trigger a review.

What if the architects did not use patterns in their architecture? This was the case in most of the reviews we conducted. Because architecture patterns are almost always present [62], the review can proceed normally and patterns will be identified.

PBAR is necessarily limited in its utility, including the following:

The most important limitation is that PBAR will not find all the issues that a traditional architecture review will. Instead, a tradeoff was made; a review process that requires little time and effort and can work even with little architectural documentation can provide more benefit to an agile project than a heavyweight review process that is not used. PBAR finds incompatibilities between architecture patterns used and important quality attributes (e.g., performance vs. Layers, or fault tolerance vs. Pipes and Filters); it will not find obscure problems such as performance issues from complex interactions of components.

The second important limitation is that the reviewers must be well versed in architecture, architecture patterns, quality attributes, and tactics. This is similar to traditional reviews: traditional reviewers need similar expertise, although architecture pattern knowledge is not as critical. The key challenge for many organizations will be finding reviewers with sufficient expertise.

One limitation is that nearly all the projects that have used PBAR so far are very small, which may not have the demands of larger industrial projects. We expect that PBAR would continue to be successful, but have little experience so far.

8.9. PBAR Conclusions
The problem driving this work is to find out whether a pattern-based architecture review method is an effective tool for small projects. In this study, we learned that in reviews of small projects, architecture patterns can be identified and used to identify potential problems related to satisfying the projects’ important quality attributes. We learned that these issues may be significant enough that the projects fixed them. We stress that we did not determine that all the important issues were identified. Such reviews require little effort, about two person-days each for reviewers and stakeholders. The calendar time required for such reviews is less than a day.
Our findings suggest that these reviews can be beneficial for small projects early in the development cycle that are unable or unwilling to undergo a comprehensive architecture review. A review requires a very small amount of effort as well as calendar time, and can be expected to uncover about three issues, one of which is major. The reviews may be well suited for systems where reliability is important. We would not recommend them for projects late in the development cycle. We also do not recommend them for projects that have many highly critical quality attributes; comprehensive analysis of the architecture is likely required in such situations. The study does not draw any conclusions about the suitability of PBAR for large projects, although many such projects will have many critical quality attributes. We propose, though, that the interaction of patterns with quality attributes may be a fruitful area of study as part of a large comprehensive architecture review.