Using Pattern-Based Architecture Reviews to Detect Quality Attribute Issues – an Exploratory Study

Neil B. Harrison [1] [2], Paris Avgeriou [1]

[1] Department of Mathematics and Computing Science, University of Groningen, Groningen, the Netherlands
[2]Department Computer Science and Engineering, Utah Valley University, Orem, Utah, USA

harrisne@uvsc.edu, paris@cs.rug.nl

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.

Keywords: Software Architecture Patterns, Architecture Reviews, Quality Attributes

1 Introduction

Some of the most important requirements concern the system's quality attributes. These requirements are characteristics that the system has, as opposed to what the system does, such as usability, maintainability, performance, and reliability. Sommerville notes that functional requirements are services the system should provide, while non-functional requirements (quality attributes) are constraints on the functions offered by the system [1]. Bachmann et al and Bass et al [2, 3] use quality attribute scenarios to describe quality-attribute-specific requirements; such a scenario includes a stimulus acting on an artefact (a system or part of a system) within an environment, and a response and its measure. The set of quality attributes is rich and varied, and various classification schemes are used (see for example [3] and [4]).

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.

Clements et al state the following [1]: "Modifiability, performance, security, availability, reliability – all of these are precast once the architecture is laid down. No amount of tuning or clever implementation tricks will wring any of these qualities out of a poorly architected system." 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 also [5]), 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 [6, 7]. Several of the most prominent architecture review methods are classified and compared in [8], and further classifications are made in [9]. (Note that in this paper a review is an independent examination of the architecture for the purpose of finding potential architectural issues; this is sometimes called an evaluation [10].)

Architecture reviews have been shown to be effective tools in uncovering architectural issues [10, 11], 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 [11]. Bass et al report common risk themes in projects using the Architecture Tradeoff Analysis Method (ATAM) include availability, performance, security, modifiability, and security [12]. Architecture reviews are, of course, most useful when done early in the development cycle, before too much code has been written [6]. 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 [6]. The SEI Software Architecture Analysis Method (SAAM) requires an average of 14 staff days [6]. 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) [10].

Closely related to effort is time. Maranzano et al report that the review preparation time lasts between two and six weeks [11], 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 [10].

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 [13], 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.

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 a point of clarification, we are not comparing the effectiveness of PBAR to other architecture review methods; indeed it is obvious that a lightweight review method such as PBAR cannot go into the depth that comprehensive reviews can. Yet even a lightweight review is better than no review at all. To illustrate how its low cost might appeal to small projects, 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 small 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.

We note that there are different ways that "small projects" may be defined. For the purposes of this study, we looked at projects with few developers; nearly all were under 10. The rationale is that projects with so few people are unlikely to have any extra time for documenting or reviewing architectures. We do not mean to exclude other measures of size, such as number of requirements, function points, or components, but we haven't examined them in this particular study.

1.2 Research Objectives and Approach

Our research objectives are to empirically analyse 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 [14]). 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 paper.

The rest of this paper is as follows: Section 2 gives background about key concepts, namely architecture patterns and quality attribute tactics, and describes the PBAR process. It also describes related work. Section 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 4 gives a summary of the results, and section 5 gives our interpretation of the results. The final section gives conclusions and describes future work.

2 Research Background

In order to understand the background behind using patterns in architecture reviews, one must understand the role of architecture patterns and tactics, and their relationship with quality attributes. These terms and their relationships are described below. A short description of the PBAR process follows thereafter.

2.1 Architecture Patterns

Software patterns are generalized solutions to recurring design problems in software development. They provide a proven approach and guidance on using the pattern solution. The solutions are generic, and are customized to the situation in which a pattern is used. Patterns include information on the consequences of applying the solution; this is important information. The most well-known software patterns are object-oriented design patterns [15]. Architecture patterns give approaches to the design of the entire software system. Architecture patterns dictate a particular high-level modular decomposition of the system [16, 17, 18]. While there are many types of patterns, in this paper we are concerned only with software architecture patterns.

For example, the Layers pattern divides the system into distinct layers so that each layer provides a set of services to the layer above and uses the services of the layer below. Within each layer, all constituent components work at the same level of abstraction and can interact through connectors. Between two adjacent layers a clearly defined interface is provided. In the pure or strict form of the pattern, layers should not be by-passed: higher-level layers access lower-level layers only through the layer beneath [19].

One of the key benefits and unique characteristics of architecture patterns is that they capture the general architectural structure of the system and couple it with the consequences of using that architectural structure. These consequences are the impact of using the pattern on the quality attributes of the system. The impact makes it easier or harder to satisfy a certain quality attribute. In nearly every case, architecture patterns impact multiple quality attributes.

The Layers pattern supports fulfilling many security requirements, because authentication services can be encapsulated into a layer surrounding the main application. On the other hand, the Layers pattern requires that requests pass through multiple layers in order to be fulfilled; this hurts performance. If performance is an important quality requirement, Layers may not be the best choice.

Because the patterns are based on long experience, their impact on quality attributes is usually documented as part of the pattern description (see [18].) This documentation is a valuable resource for the architect. This information is beginning to be organized and summarized for handy use [20].

2.2 Quality Attributes and Tactics

Certain specific measures are often implemented to improve quality attributes. Bass et al call these measures tactics [3]. For example, a duplication scheme to improve the reliability (nonstop operation) of the system is a tactic. The implementation of a tactic may require significant changes to structure and behavior of the architecture patterns in the system. Because of this, future maintenance of the system may be considerably more difficult and error-prone. Tactics may be "design time," or overall approaches to design and implementation, such as "hide information" to improve modifiability, or may be "runtime tactics", which are features directed at a particular aspect of a quality attribute, such as "authenticate users" to improve security. In this paper, we concern ourselves with the runtime tactics, and when we refer to "tactics", we mean runtime tactics.

We find that architects often consider which tactics to use at the time they design the architecture [21, 22]. However, unless the relationship between the architecture patterns and the tactics being used is well understood, architects risk making architectural decisions (either concerning which patterns to use or which tactics to use) that could be difficult to implement correctly and maintain. The relationship between the tactics selected and the patterns used is therefore an important consideration in an architecture review.

2.3 The Pattern-Based Architecture Review (PBAR) Process

The goal of Pattern-Based Architecture Reviews is to provide a ligheweight 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.

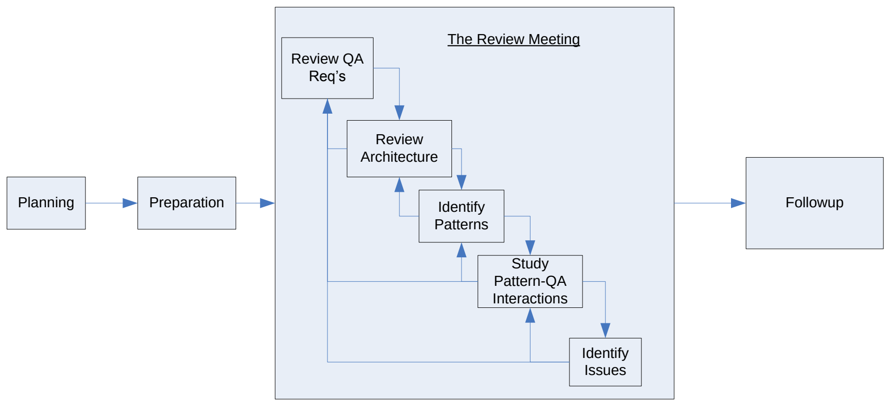

Figure 1. The PBAR Architecture Review Process

The structure of Pattern-Based Architecture Reviews (PBAR) is based on architecture review processes, namely the AT&T review process [11], which one of the authors used while an employee there, as well as ATAM [3]. PBAR 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 [23]. The key differences between PBAR and comprehensive architecture reviews are summarized in this table:

Table 1: Key Focus Differences in Review Methods

Comprehensive Architecture Review	**Pattern-Based Architecture Review**
Focus on all requirements	Focus on key quality attribute requirements
Discovery through analysis and other study methods for satisfaction of requirements	Discovery through pattern identification and association with quality attributes
Extensive preparation by reviewers and stakeholders	No preparation for stakeholders
Documentation often prepared just for the review	Only existing documentation used
Architects and key stakeholders attend review	Entire team encouraged to attend review

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 desired.

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. If architecture documentation is available, the reviewer can begin to collect information that is discussed in steps 1-4 of the review meeting.

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. Ideally, stakeholders provide input to determine 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. Review the architecture of the system. If no architecture diagram was provided, ask (or help) the team to draw one. In general, a components and connectors view is the most useful diagram for PBAR.
    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. Because published patterns have rich documentation about their consequences; in particular their impact on quality attributes (see [20]), one should look for established architecture patterns. A comprehensive list of architecture patterns can be found in [19].One can identify the patterns by doing the following:
    a. Look for common pattern names (see [19]) 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 [18, 19].) Of course, the use of a pattern name does not necessarily mean that the pattern is actually present; the reviewer should question the architects about it and delve into the structure to verify whether the pattern is present; see below.

   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 [18]. 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 [24].

4. Study the architecture and quality attributes together to determine the effects of each pattern on the system's quality attributes.
   a. Identify the tactics used and map them to potential patterns that may implement them. Several important tactics have been identified in [3], and extensive reliability tactics are given in [25].For example, the security tactics of Limit Exposure and Limit Access [3] suggest the use of the Layers [18] architecture pattern. Note that tactics may not be spelled out in any architecture documentation; an effective way to identify them is to ask the architects how they address each of the quality attributes. This will reveal the use of tactics, and may also reveal important omissions. Examine how the tactics might be implemented in the patterns, and how easy such implementations are. For example, see [26] 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. They may be specific to this architecture, but may also include general characteristics the patterns, such as the fact that 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.

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) [27] and some variants, Architecture Tradeoff Analysis Method (ATAM) [28], Active Reviews for Intermediate Design (ARID) [31], Architecture-Level Modifiability Analysis (ALMA) [32], Architecture-Level Prediction of Software Maintenance (ALPSM) [33], and Scenario-Based Architecture Reeingineering (SBAR) [34]. Dobrica and Niemelä have surveyed these architecture analysis methods, and compared eight elements of them [8]. Ali-Babar et al included additional elements [9] in a subsequent study. The following table shows five elements selected from these two studies where PBAR has significant differences from many other methods, and shows how they are done in PBAR. These differences reflect tradeoffs made in order to accommodate the low cost needs and characteristics of small projects, such as minimal architectural documentation.

Table 2. SA Review Method Classification Applied to PBAR

Method Element	**PBAR's Focus; Key Differences From Other Methods**
Method's goals [9]	Evaluate the ability of system architecture to achieve quality attributes (QA). Focus only on QAs and not functional requirements (Similar to SBAR [34], but SBAR is for analysis of existing systems.) PBAR trades off comprehensive coverage such as ATAM [28], for lower cost, focusing on a high payoff area.
Quality attributes [8]	Multiple quality attributes, similar to SBAR [34] and ATAM [28]. This trades off depth in a single QA for breadth across many. Some methods focus on a single QA (e.g., ALMA [32] focuses on maintainability.)
Architectural description [9]	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.
Reusability of existing knowledge [8]	PBAR uses existing (external) architecture pattern and tactic knowledge [3, 26]. Other methods tend to focus on internal experience repositories [28, 32]. PBAR may also use internal experience repositories, but can also work for small projects which have no such repositories.
Resources Required [9]	Very few resources required; see later section. All projects that specified resource requirements were much higher; see [28] for example.

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 [2] 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 examine patterns to find tactics that may be applied, for the purpose of selecting and calibrating a quality attribute reasoning framework [29]. 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 [30]. 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.

3 Case Study Design

In order to fully understand the implications of any empirical study, one must understand how the study was designed. This includes the type of study, as well as the selection of subjects and how the study was carried out. These topics are explored in more detail in this section.

3.1 Study Type

This study is exploratory in nature (see [14]), meaning that we set out to learn about the nature of the effort and calendar time required by PBAR, and number of issues it identifies (treatment), 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 [14]. 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 [35]. In our studies, independent variables included project size, domain (in particular, which quality attributes were important), and background of participants.

We report the details of the study and its results in the next sections, following the model given by Wohlin et al [37], which Jedlitschka and Ciolkowski note is for generic empirical research [38]. (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.

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 the 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 (e.g., Agile, waterfall) was not specified.

Architecture reviews are generally done in the early phases of projects [6]. 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 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 2: PBAR Projects Studied

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

We note that other classifications of projects, such as number and complexity of requirements, etc., may also be useful for understanding the applicability of PBAR, and recommend such studies be done.

[1] Android is a trademark of Google Inc. Use of this trademark is subject to Google Permissions.

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

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 the appendix.

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.

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.

4.1 Case and Subject Selection

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 4 shows the breakdown of effort for the reviewers, compared to various versions of the ATAM method as reported in [10]. We note that Clements et al [10] 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 4: Reviewer Effort in person-days, by Phase

	PBAR *(review team size: 1)*	**ATAM-Medium** *(review team size: 1)*	**ATAM-Medium (using Checklists)** *(review team size: 1)*	**ATAM-Small** *(review team size: 1)*
Preparation	1/2	1	1	1
Review Meeting(s)	1/4	20	12	6
Follow-up	3/4	15	12	8
Total Effort	< 2	36	25	15

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 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 5: Stakeholder Effort in person-days, by Phase

	PBAR *(5 stakeholders)*	**ATAM-Medium** *(3 stakeholders)*	**ATAM-Medium (with Checklists)** *(3 stakeholders)*	**ATAM-Small** *(2 stakeholders)*
Preparation	0	3	1	3
Review Meeting(s)	1.25	12	9	6
Follow-up	< 1	3	3	2
Total	2	18	13	11

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 shows a team of 5; larger organizations will have a proportionally larger effort.

4.2 Calendar Time Required

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

Table 6: Calendar Time for Review Types, in days

	PBAR	ATAM-Small
Calendar Time	< 1	3

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 [10]. 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.

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):

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 7: Issues Found in Reviews

System	**Development Phase**	**Patterns Identified**	**Total Issues Identified**	**Major Issues**	**Major Issues Implemented**
B	Architecture	6	4	1	0
C	Released, Legacy	7	2	0	0
D	Early Impl.	4	7	1	1
E	Early Impl.	3	3	2	1
F	Early Impl.	3	3	1	1
G	Early Impl	2	3	1	1
H	Early Impl	3	5	0	0
Median		3	3	1	1

The number of patterns identified is consistent with patterns found in other systems (see [24]). Since patterns provide ready-made documentation (at the very least the structure and behaviour of the pattern, and its generic advantages and liabilities), 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.)

- 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.

5 Interpretation of Results

In analyzing the results, we consider three things, the effort required to perform the reviews, the time it takes, and the number and type of issues uncovered by the reviews.

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 characteristics 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 [39]), and may have daily units of work [40]. We recommend that this validation be done.

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.

5.3 Number and Type 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 [3] 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:

Bass et al [3] 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 et al [41]. 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. 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: Quality Attributes in the Systems

Quality Attribute	**Number of Systems where Important**	**Total Issues Found**	**Number of major issues**
Performance	5	3	0
Reliability, (inc. Fault Tolerance)	7	12	1
Usability	3	3	1
Security	5	4	1

Maintainability, Extensibility	5	5	2
Portability	3	0	0
Configurability	1	0	0

6 Limitations of the Study

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

1. Kitchenham et al [14] 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 [42] 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.

7 Conclusions and Future Work

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.

We plan to perform other studies with different reviewers to strengthen the results and to give greater insights into the requirements of the qualifications of the reviewers. We recommend studying PBAR as part of a traditional architecture review in large projects. In other words, use the identification of the architecture patterns and their impact on quality attributes as one of the investigative tools within, for example ATAM. PBAR might complement such existing approaches.

References

1. Sommerville, I.: Software Engineering. Addison Wesley Longman, 8th ed. (2007)
2. Bachmann, F., Bass, L., Klein, M., Shelton, C.: Designing Software Architectures to Achieve Quality Attribute Requirements. IEE Proceedings 152(4), pp. 153-165. (2005)
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 2nd edn. (2003)
4. International Standards Organization, Information Technology – Software Product Quality – Part 1: Quality Model, ISO/IEC FDIS 9126-1 (2000)
5. Bengtsson, P.: Towards Maintainability Metrics on Software Architecture: An Adaptation of Object-Oriented Metrics. In: First Nordic Workshop on Software Architecture, Ronneby, (1998)
6. Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L., Zaremski, A.: 'Recommended Best Industrial Practice for Software Architecture Evaluation', Technical Report CMU/SEI-96-TR-025, CMU Software Engineering Institute, (1997)
7. Obbink, H., Kruchten, P., Kozaczynski, P., Hilliard, R., Ran, A., Postema, H., Lutz, D., Kazman, R., Tracz, W., Kahane, E.: Report on Software Architecture Review and Assessment (SARA). http://philippe.kruchten.com/architecture/SARAv1.pdf, (2002)
8. Dobrica, L., Niemelä, E.: 'A Survey on Software Architecture Analysis Methods', IEEE Trans. Softw. Eng. 28, 7. pp. 638-653 (2002)

9. Ali-Babar, M., Zhu, L., Jeffery, R.: A Framework for Classifying and Comparing Software Architecture Evaluation Methods. In: 15th Australian Software Engineering Conference, Melbourne, pp. 309-318 (2004)
10. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley (2002)
11. Maranzano, J., Rozsypal, S., Zimmerman G., Warnken, G., Wirth, P., Weiss, D.: Architecture Reviews: Practice and Experience. IEEE Software, vol. 22(2), pp. 34-43. (2002)
12. Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Through Architecture Evaluations. SEI Report CMU/SEI-2006-TR-012 (2006)
13. Beck, K., et al.: 'Manifesto for Software Development,' http://agilemanifesto.org, accessed October 2008
14. Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. 28(8), pp. 721-734. (2002)
15. Gamma, E., et al.: Design Patterns: Elements of Reusable Object-Oriented Software Addison-Wesley (1995)
16. Harrison, N., Avgeriou, P., Zdun, U.: Architecture Patterns as Mechanisms for Capturing Architectural Decisions. IEEE Software, 24(4) pp. 38-45 (2007)
17. Shaw, M.: 'Toward Higher-Level Abstractions for Software Systems'. Proc. Tercer Simposio Internacional del Conocimiento y su Ingerieria, October 1988, pp. 55-61. Reprinted in Data and Knowledge Engineering vol. 5 pp. 19-28 (1990)
18. Buschmann, F., et al.: 'Pattern-Oriented Software Architecture: A System of Patterns' Wiley (1996)
19. Avgeriou, P., Zdun, U.: Architectural Patterns Revisited – a Pattern Language. In: 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005), Irsee, Germany (2005)
20. Harrison, N., Avgeriou, P.: Leveraging Architecture Patterns to Satisfy Quality Attributes. In: First European Conference on Software Architecture, Madrid, pp. 263-270, Springer LNCS (2007)
21. Harrison, N., Avgeriou, P.: Pattern-Driven Architectural Partitioning - Balancing Functional and Non-functional Requirements. In: First International Workshop on Software Architecture Research and Practice (SARP'07), IEEE Computer Society Press (2007)
22. Harrison, N., Avgeriou, P., Zdun, U.: Focus Group Report: Capturing Architectural Knowledge with Architectural Patterns, 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006), Irsee, Germany (2006)
23. Coplien, J., Harrison, N.: Organizational Patterns of Agile Software Development, Prentice-Hall (2004)
24. Harrison, N., Avgeriou, P.: Analysis of Architecture Pattern Usage in Legacy System Architecture Documentation. In: 7th Working IEEE/IFIP Conference on Software Architecture, Vancouver, pp. 147-156. (2008)
25. Utas, G.: Robust Communications Software: Extreme Availability, Reliability and Scalability for Carrier-Grade Systems, Wiley (2005)
26. Harrison, N., and Avgeriou, P.: Incorporating Fault Tolerance Techniques in Software Architecture Patterns. In: International Workshop on Software Engineering for Resilient Systems (SERENE '08), Newcastle upon Tyne (UK), ACM Press (2008)
27. Kazman, R., Bass, L., Abword, G., Webb, M.: SAAM: A Method for Analyzing the Properties of Software Architectures, In: 16th International Conference on Software Engineering, (1994)
28. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J.: The Architecture Tradeoff Analysis Method. In: Proceedings of IEEE, ICECCS (1998)

29. Zhu, L., Babar, M. A., Jeffrey, R.: Mining Patterns to Support Software Architecture Evaluation. In: Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE Computer Society, Washington, DC (2004).
30. Bass, L., Nord, R., Wood, W., Zubrow, D., Ozkaya, I.: Analysis of architecture evaluation data. J. Syst. Softw. 81, 9. pp. 1443-1455. (2008)
31. Clements, P.: Active Reviews for Intermediate Designs, SEI Carnegie Mellon University CUM/SEI-2000-TN-009, (2000)
32. Bengtsson, P.-O., Lassing, N., Bosch, J., Van Vliet, H.: Architecture-Level Modifiability Analysis, Journal of Systems and Software vol. 69. (2004)
33. Bengtsson, P.O., Bosch, J.: Architectural Level Prediction of Software Maintenance. In: 3rd European Conf. on Software Engineering Maintenance and Reengineering. (1999)
34. Bengtsson, P.O., Bosch, J.: Scenario-based Architecture Reengineering. In:. 5th International Conference on Software Reuse *(*1998)
35. Alavi, M., and Carlson, P.: A Review of MIS Research and Disciplinary Development. Journal of Management Information Systems*,* 3(4), pp. 45-62. (1992)
36. Holz, H., Applin, A., Haberman, B., Joyce, D., Purchase, H., Reed, C.: Research methods in computing: what are they, and how should we teach them?. SIGCSE Bull. 38, 4, pp. 96-114. (2006)
37. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: an Introduction. Kluwer Academic Publishers (2000)
38. Jedlitschka, A.,Pfal, D.: Reporting Guidelines for Controlled Experiments in Software Engineering, In: Intl. Symp. on Empirical Software Engineering (2005)
39. Cockburn, A.: Agile Software Development: The Cooperative Game. 2nd ed. Addison-Wesley, (2007)
40. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press (2004)
41. Johnson, P. M., Tjahjono, D.: Does Every Inspection Really Need a Meeting? Empirical Softw. Eng. 3, 1 pp. 9-35. (1998)
42. Falessi, D., Ali Babar, M., Cantone, G. Kruchten, P.: Applying empirical software engineering to software architecture: challenges and lessons learned, Empirical Swre. Eng., to appear, (2010)

Appendix A: Sample PBAR Report

The following is a report from an actual PBAR session:

7.1 System Description and Requirements

A local company provides discount magazine subscriptions to its customers, and has found a rather lucrative niche market for their business. Inmates in prison have a lot of time on their hands, and appreciate magazines. So this business mainly targets prisoners.

The main part of the business consists of receiving orders, and then placing the orders with the magazine companies. The service they provide is one-stop shopping for magazine subscriptions at a very good price. (I assume they get good volume prices from the magazines themselves.)

At this time, all their orders are received on paper through the mail. This system allows employees to enter the customer information and the order information from the paper orders into a database, where it can be used later.

There are several key requirements:

1. Customers may have multiple addresses, and the addresses change often. It is important to keep former addresses around (at least one former address), because customers sometimes move, and their magazines don't catch up to them. So they complain to this company.
2. In the future, customers will be able to be entered directly from the customer website.
3. In the future, orders might be taken via the web.
4. One order may contain orders for several different magazines.
5. Orders arrive with payments, which can be cash, check, or credit card. Payments may be of two or more types (e.g. two different checks).
6. Customer complaints are common, and the company spends considerable time and money resolving them. Besides the forwarding address problem noted above, a common complaint is that an order is placed, and the subscription never starts. So tracking order information and status is desirable. (Note that the original vision of the system included support for handling customer problems, but that has been deferred to a later release.)
7. Data entry errors are a fact of life. In the future, the system should support a double-entry system, where two different employees enter the same information, and it is compared for accuracy.
8. Data entry employees may work remotely.

7.2 Key Quality Attribute Requirements

1. Reliability (accuracy and fault tolerance): The consequences of inaccurate data are serious: Customers get angry because they don't get the magazines they paid for. And it costs time and trouble to hunt down the error, even to the point of finding the original paper order and verifying it. This is a key quality attribute.
2. Capacity: The amount of data being handled is quite large for a system of this size. There are currently around 90 thousand customers, and roughly one million orders. There are around 1000 different magazine titles. However, because each magazine can come from multiple suppliers, the real number of magazines is three to four times that number. The system must handle not only this load, but larger amounts of data as the business expands. This is a key quality attribute.
3. Extensibility: There are numerous important capabilities that will be needed in the future, so the system must be easily extended. This is a key quality attribute.
4. Security (authentication and authorization): It is important to keep the system safe from unauthorized use. In particular, data entry employees must

be authenticated, and may have different privileges. This quality attribute is important.

5. Performance: The system has no hard or soft real time needs, but operations on this large database should be quick.

7.3 System Architecture

This system is a student team project, and as such is rather small, and has work left to be done. The architecture relies heavily on existing software packages. These packages, and the way they are designed, play a major role in shaping the architecture. Prominent patterns include the following:

1. Repository: The database is central to this application.
2. MVC: MVC is really central to the human interface of the application. The framework they used for implementing the UI is based on MVC, and even identifies its components as such. However, see the note in Layers, below
3. Layers: The system is also highly layered. The Views, Controllers, and Model form (almost) the top three layers, which is really not a true use of the MVC pattern. Underneath that, there is another layer associated with the Model layer, and then the database. There is another layer that is sometimes present on top of (or in) some of the Views. It is called "widgets." Its purpose is to go nearly directly to the database for certain actions (e.g., large queries) to improve performance. It bypasses intermediate layers.
4. Broker: There is some possibility for a Broker to manage efficient and reliable access to the database. However, at this time, it is simply potential.

7.4 Patterns and Quality Attributes

In general, the patterns are compatible with the important quality attributes of the system. Of note:

1. (Performance) The most notable liability of the Layers pattern is its performance. Indeed, this architecture uses the common "wormhole" variation of Layers to circumvent some layers in order to improve performance. However, this variant has some issues:
    a. It's a bit harder to understand.
    b. It's a bit more problematic for extensibility, in that as new capabilities are added, one must always consider whether or not to use the 'wormhole" in the new features. This will almost certainly be an issue in this application. That said, the "wormhole" appears to be worth the tradeoff.
    c. In this case, it required a bit of data to be replicated.
2. (Security) The Layers pattern is so compatible with authentication and authorization that they are normally a slam-dunk in a Layered architecture. Unfortunately, it isn't quite so easy here. The "wormhole" causes security checks to be done in two different places. So it isn't quite as clean as we

might like. It doesn't appear to be a big problem, and is likely not worth doing differently.

3. (Extensibility) The MVC architecture will make it easy to add future web capabilities, such as adding customers or orders. This is a key strength of this architecture.
4. (Capacity) The use of a commercial database package takes care of any capacity worries. In addition, one of the commercial layers can easily become a Broker, thus adding further support for capacity, if needed. No worries here.
5. (Reliability) The commercial database package also takes care of most reliability concerns. It provides the ability to group actions into transactions which can be committed as a unit. We discussed the possibility of race conditions that might allow a record to be lost and appear to the employee as if it had been entered. Upon further reflection, I suspect that the easiest way to deal with this (if it even it possible) is simply to implement the double-entry system that will be put in place for catching erroneous data entry. That should take care of it. We also discussed running audits on the data to ensure that customer and order records are not missing anything. Given the importance of correct data, it still might be a good idea.

7.5 Summary Notes

There was no architecture documentation on the project, yet the review still worked. The architect was very knowledgeable.

Overall assessment: Successful: Few issues were raised, but that was because the system appears to be well-designed.

Statistical assessment:

1. Time: 1 hour, 15 minutes
2. Size: small
3. Team size: 4 people (students)
4. Domain experience: 1 person – high, 3 people – moderate by this time
5. Project phase: mid implementation, first release.
6. Patterns found: 3 (Broker doesn't count, because it is potential, not present)
7. Issues found: 3
    a. Reliability; in particular issues about potential loss of data in rare conditions. (Discussed during first part)
    b. Extensibility, with respect to the Layers "wormhole" (discussed during the second part – strong pattern discussion) (Major Issue)
    c. Security, with respect to the Layers "wormhole" (discussed during the second part)
    d. Pattern awareness: The team knew all the patterns we discussed.