9. Thesis Conclusions

This chapter presents the conclusions of this thesis. It begins with the answers to the questions presented at the beginning of the thesis. This is followed by a discussion of the chief contributions of this work to the field of software engineering. The chapter ends with a short discussion of future research directions and open questions.

9.1 Research Questions and Answers

In section 1, the research questions of this thesis were presented. Each of these questions is reiterated here, along with their answers. We begin with the specific research questions. The answers to these contribute to the answer to the overall research question, which is given afterwards.

The first specific question is as follows:

*RQ-1: How common are architecture patterns in software architectures? In particular, which patterns are commonly found individually and in pairs in certain application domains?*

Most software systems use between 1 and 4 architecture patterns. The most commonly used architecture patterns are, in descending order of frequency, Layers, Shared Repository, Pipes and Filters, Client-Server, Broker, Model View Controller, and Presentation Abstraction Control.

The most common pair of architecture patterns used together was Layers-Broker, followed by the following pairs: Layers-Shared Repository, Pipes and Filters-Blackboard, Client Server-Presentation Abstraction Control, Layers-Presentation Abstraction Control, and Layers-Model View Controller.

Among domains studied, the following patterns were the most common:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Most Common Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded Systems</td>
<td>Pipes and Filters</td>
</tr>
<tr>
<td>Dataflow and Production</td>
<td>Layers</td>
</tr>
<tr>
<td>Information and Enterprise</td>
<td>Shared Repository</td>
</tr>
<tr>
<td>Web-Based Systems</td>
<td>Broker</td>
</tr>
<tr>
<td>CASE and Related Tools</td>
<td>Layers</td>
</tr>
<tr>
<td>Games</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>Scientific Applications</td>
<td>Pipes and Filters</td>
</tr>
</tbody>
</table>
The second question concerned how patterns fit in the big picture of architectural decisions:

**RQ-2: What is the relationship between architecture patterns and architectural decisions, particularly those concerned with quality attributes?**

Architecture patterns embody major architectural decisions about which architectural structure and behavior to employ. In other words, key decisions about the high level structure (and to a lesser extent, the behavior) of the system are very often decisions to use particular architecture patterns. With respect to quality attributes, these decisions are made to put architectural structure and associated behavior in place that the architects believe will satisfy the quality attributes. Architects often base these decisions on characteristics of candidate architecture patterns. Because architecture patterns are well understood and documented, use of architecture patterns helps solve the difficult problem of documenting architectural decisions, the rationale behind them, and their consequences.

A particularly important class of consequences is the impact of the pattern on quality attributes. The use of a particular pattern may have positive or negative impact on a given quality attribute, based on the characteristics of the pattern, thus certain patterns are good or bad fits for certain quality attributes. Among the most common patterns and quality attributes, the following patterns are particularly good fits for these quality attributes:

- **Usability**: Model View Controller; also Presentation Abstraction Control and Broker
- **Security**: Layers; also Broker
- **Maintainability**: Layers; also Broker and Pipes and Filters
- **Efficiency**: Pipes and Filters
- **Reliability**: Layers
- **Portability**: Pipes and Filters and Broker; also Layers and Presentation Abstraction Control
- **Implementability**: Broker

The next question explores the types of impact they can have on each other.

**RQ-3: What model describes the interaction between patterns and the tactics one uses to implement quality attributes?**

A brief summary of the model is as follows: An architecture pattern affects architectural concerns of a system architecture, among which are the quality attributes. The way this happens is that quality attributes are satisfied through the implementation of specific measures called tactics. The implementation of the
tactics must be done within the context of the architecture patterns. In particular, the tactic must be implemented within components and behavior of the pattern (designated as “pattern participants.”)

Implementation of tactics within a pattern entails some changes to the pattern participants. The types of changes to components and connectors are as follows:

- Implemented in: little or no change is required in order to implement the tactic.
- Replicates: components and/or connectors are replicated, but their structure remains the same.
- Add, in the pattern: new components and/or connectors are added, keeping the structure of the pattern the same (e.g., an additional layer is added in the Layers pattern.)
- Add, out of the pattern: new components and/or connectors are added which change the shape of the pattern.
- Modify: the structure of components and/or connectors must change.
- Delete: a component and/or connector is removed (postulated; not observed.)

It is possible to use a simple annotation scheme to show these changes on a typical component-and-connector architecture diagram.

For a given quality attribute, there are certain tactics implemented to satisfy the quality attribute. A tactic’s impact on a pattern is characterized by the types of changes, as outlined above, and the overall expected difficulty of implementing a given tactic in a given pattern can be gauged; I used a five-level scale.

The model leads to a sub-question:

*RQ-3a: What do we learn about patterns and satisfying quality attributes through the application of this model?*

A study of tactics associated with software reliability showed how the implementation of tactics in various architecture patterns fits in the various types of changes. The impact of the changes was assessed on the aforementioned five-level scale. A controlled experiment showed that this information can be useful in the assessment of work required to implement tactics.

The next question concerns how this model extends to complex systems where multiple patterns and quality attributes are in play:
RQ-4: In a complex system requiring the use of multiple patterns and multiple tactics, what characteristics of the patterns, the tactics, and even the system itself influence where and how tactics are most effectively implemented?

Three important factors influence the implementation of tactics. They are as follows:

1. The nature of the tactic itself indicates whether the tactic influences all components of the architecture, or just a subset of them.

2. Previous decisions about the system become constraints to which the system must conform. In particular, the selection of architecture patterns constrains how tactics are implemented, making it easier or harder to implement a tactic.

3. The requirements of the system with respect to the quality attributes constrain where a tactic is to be implemented. These requirements may cause a tactic to be implemented in the structure of a pattern that is a good fit for it, or in one that is a poor fit for it.

These factors combine to facilitate or hinder the accomplishment of desired quality attributes.

These complex relationships have been explored for a set of the most common tactics associated with reliability.

We now wish to find practical application for this understanding. In the next question, we consider how to use it to help form the architecture.

RQ-5: How does one incorporate patterns into the architectural analysis and synthesis phases of architectural design in order to help the architect consider how the structure impacts the satisfaction of quality attributes?

The use of patterns, or a pattern-driven approach to architecture is entirely compatible with common architectural analysis and synthesis methods. The key steps to using patterns are as follows:

1. Identify the most prominent architectural drivers of the system. These include both functional requirements and quality attributes.

2. Select candidate architecture patterns that address the needs of the architectural drivers.

3. Partition the system by applying a combination of the candidate patterns.
4. Evaluate whether the partitioning satisfies the architectural drivers. This may include:
   a. Examine the forces of the pattern
   b. Examine the consequences of the pattern
   c. Examine the interaction among the patterns selected

5. Perform tradeoffs with respect to the different architecture drivers. These tradeoffs include exploring candidate tactics, and considering their interaction with the candidate architecture patterns.

We continue exploring the practical use of knowledge of the interaction of patterns and quality attributes in the following, where we concern ourselves with how we can learn whether the patterns selected are a good fit for the quality attributes, at the time the architecture is being formed.

*RQ-6: How can one gain insight into the impact of the architecture patterns used on quality attributes early in the development cycle – while the architecture can still be readily modified?*

One can use an architectural evaluation method that is based on traditional architectural evaluations, but is tailored to focus on the critical quality attributes and their interaction with architecture patterns. The evaluation process is designed to be very lightweight so that projects that cannot afford lengthy and expensive architecture reviews can obtain some of the important benefits of architecture reviews.

The main part of a pattern-based architecture process is the review meeting. The main activities of the review meeting are as follows:

1. Review the quality attribute requirements
2. Present and discuss the architecture
3. Identify the patterns in the architecture
4. Examine the interactions between the patterns and quality attributes; consider tactics which are being used or considered, as well as tactics which might be important to use.
5. Identify issues

These activities may be performed iteratively.
We have used this review process and found that it can identify potentially significant architectural issues with respect to the systems’ quality attributes. We have found that the process requires little time and effort, and can be done for small projects.

The answers to the above research sub-questions lead to the answer to the main research question. The main research question is as follows:

*How can architects leverage patterns to create architectures that meet quality attribute requirements, during analysis, synthesis, and evaluation?*

The interaction among architecture patterns and quality attributes is indeed rich. Thus, there are numerous ways in which an architect can take advantage of this interrelationship to create improved software architectures. I have identified the following ways:

1. At the most basic level, an architect can examine the available architecture patterns to understand existing solutions to architectural problems. The architect can focus on the patterns that are most commonly used, especially in particular problem domains. As most systems employ more than one architecture pattern, this also includes being able to focus on the pairs of patterns most commonly used in a domain.

2. A second use is that an architect can use patterns to help capture architectural decisions. That is, if an architect uses a pattern, documenting the use of the pattern leverages the existing pattern documentation; the architect does not need to rewrite it. The documentation of the consequences of the pattern is especially helpful, particularly as many of the consequences impact quality attributes.

3. With the additional insight about architecture patterns’ impact on quality attributes, an architect can make well informed choices about the architecture of the system under design. In particular, the architect can understand the impact of an architecture pattern on the system’s quality attributes, and can use this understanding to make tradeoff decisions concerning the system’s architecture.

4. The detailed information about the interaction of the patterns and tactics can help the architect outline how the quality attribute’s tactics are to be implemented. This includes the very common case where more than one architecture pattern is used; it can help the architect understand in which pattern(s) a tactic will be implemented, and its impact on those patterns. In
addition, the detailed interaction information helps the architect make more accurate estimates of the work needed to implement the system.

5. Through a simple method of annotating architecture diagrams, architects can show where architectural changes are needed to implement quality attributes tactics.

6. By following a simple process, the architect can explicitly use patterns during the design of the architecture, thus gaining the benefits listed here. In particular, an architect can seek out and select patterns that fit the needs of the system being designed. Common architectural design methods can easily be adapted to include the pattern-based approach to architectural design.

7. Architects can learn about potential issues concerning the architecture of the system and the important quality attributes by employing a pattern-based architecture review process.

Some of these pattern-based activities that an architect can do are based on others; some are required, while others are helpful. The most notable relationships are illustrated in the following diagram.
9.2 Contributions

The answers to the questions presented in the previous section lead to several contributions that this thesis makes to the field of software engineering. In general, this thesis helps software architects design software architectures that better satisfy quality attributes. This is achieved through additional insight and through advanced architectural design and review techniques.

Specific contributions are as follows:

- **Understanding of the use of Architecture Patterns in Practice**: Chapter 2 stated that virtually all software systems employ one or more architecture
patterns, and listed the most commonly used architecture patterns, based on study of 45 software architectures.

- **Understanding of the relationship between Architecture Patterns and Architectural Decisions:** Chapter 3 describes the special role that architecture patterns play in architectural decisions. This highlights the importance of architecture patterns in software architecture. It also shows the common architecture patterns and their impact on common quality attributes, and the reasons for this impact.

- **A Model of How Architecture Patterns and Tactics Interact:** Chapter 4 gives insight into how patterns and tactics interact, and models this interaction and gives architects an easy way to capture where tactics are implemented in the architecture, and show it in architecture diagrams.

- **A Method for Annotating Architecture Diagrams with Tactic Information:** Chapter 5 shows how information about the interaction of patterns and tactics can be used in a practical setting to improve the ability of the software to meet quality attribute requirements. This becomes a reference for designers working on high reliability systems.

- **Categories of impact of tactics on complex architectures:** The material in chapter 6 describes that tactics fall into three categories of impact on the components of a system. These categories help one understand how much of an impact the implementation of a tactic will have on the system components. This helps architects decide among alternate tactics, as well as understand which components in an architecture are likely to be affected.

- **PDAP:** The Pattern-Directed Architecture Process described in chapter 7 gives architects a way to guide software architecture. It helps architects take advantage of the characteristics of different architecture patterns. It is a lightweight process, but can be easily incorporated into other more heavyweight software architecture processes.

- **PBAR:** Pattern-Based Architecture Reviews described in chapter 8 gives the architect a way to use patterns and their interactions with quality attributes to evaluate architectures for potential difficulties in satisfying quality attributes.

### 9.3 Limitations

These contributions are necessarily limited in their application. For each of the contributions, I highlight the key limitations.
• **Understanding of the use of Architecture Patterns in Practice:** The state of software products continues to advance, particularly as new software applications, technologies and platforms are developed. Software architectures are evolving as software becomes ever more distributed. The architectures that were studied to learn the dominant architecture patterns may not reflect the common architectures of the future. In a few years, different architecture patterns may be the ones in vogue.

• **Understanding of the relationship between Architecture Patterns and Architectural Decisions:** The key limitation of this contribution is that it depends on the architect – architects are constrained by their knowledge of the patterns. The best informed decisions rest upon the foundation of in-depth knowledge of architecture patterns.

• **A Model of How Architecture Patterns and Tactics Interact:** This model, like any other model, is an approximate representation of some real phenomenon. In other words, there may be real-world cases where the patterns and tactics interact in ways not captured in this model. The model should be taken as a guide, and not used blindly in every situation.

• **A Method for Annotating Architecture Diagrams with Tactic Information:** This notation system, like all notations, has two significant constraints. First, one must learn the notation sufficiently well that the message imparted by the notations is clear at a glance. This takes experience, and is a limiter to those who wish to adopt it. Second, a notation like this needs wide exposure before it becomes widely used, and there is little mechanism for it to be shared with developers. Frankly, it stands a good chance of being used only rarely.

• **Categories of impact of tactics on complex architectures:** The categories of impact are well defined, but a tactic may be used in ways that are different from what was envisioned when the tactic was defined or analyzed. The fluid nature of software development allows wide flexibility of use. This means that the impact depends in part on the particular architectural decisions made in the context of the domain. This means that information on the impact of tactics on multiple patterns must be taken as general guidelines only.

• **PDAP:** The key limitation of PDAP is the amount of knowledge of patterns that the architect has. The less knowledge the architect has about patterns, the less PDAP can be used effectively.
• **PBAR**: The key limitations of PBAR are that it may have difficulty scaling, and that it is a high-level architecture evaluation approach, trading detailed analysis for low cost. Thus it is not appropriate in all situations.

### 9.4 Open Research Questions and Future Work

As with any research, the answering of research questions leads to other open questions, and opportunities for future work. In particular, the following questions are fertile areas for further research:

*How well does the model apply to tactics other than those associated with reliability?*

Reliability tactics appear to have been researched more comprehensively than other tactics. There are some tactics associated with security, which can be readily studied. Studies of the model with tactics beyond those will require identification and characterization of the tactics themselves.

*How can the interaction of patterns and quality attributes be most likely to be of practical use?*

The key is that the pattern-tactic interaction information must be readily available to architects. To this end, each of the well-known architecture patterns must be analyzed, and a complete catalog of the patterns with their non-functional impacts must be created. This work will be most effective when it is complete and available in one place. Cross-referencing capabilities, e.g., searching the catalog by quality attribute would be extremely useful.

Some patterns and reliability tactics fit particularly well together, and may indeed be commonly used. We would like to investigate architectures to see whether some combinations of patterns and reliability tactics are common. These may form a set of “reliable architecture patterns;” variants of architecture patterns especially for highly reliable systems. Such information would an essential part of such a catalog.

*How well does the PBAR process scale up to large industrial software projects?*

Additional studies with different reviewers should be done 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.

*The model concerns only the so-called runtime tactics, yet there are also design time tactics. Do they fit into a model as well?*
We find that the other tactics they describe, design time tactics, tend to cut across all design partitions, and are implemented implicitly in the code. Therefore, they are not as good a fit for the model as are the run time tactics, nor do they have as well-defined effects on patterns. The model and the design time tactics should be studied in more detail to determine how the design time tactics can be represented in the model and annotation. Besides refinement of the model and annotation, it will lead to greater insights about the nature of design time tactics versus run time tactics.

*What other relationships among architecture patterns are there?*

One very interesting consequence of implementing the tactics is that since it involves changing the architecture, some changes may actually change the pattern composition of the architecture. An architecture pattern may be added. In certain cases, an existing pattern may even change to a different pattern. Obviously, the transformation of one pattern to another can happen only where patterns are similar. We have seen two examples of this type of transformation in architectures we have evaluated. We intend to study this further in order to understand its architectural implications.

In addition, the following are topics for further research:

Existing pattern variants should be studied both at a structural level and for their impact on quality attributes. In addition, undocumented pattern variants should be studied and common undocumented variants should be documented.

We noted that the architecture diagrams represented different views of the systems, and most incorporated elements of more than one of the 4+1 views. It is possible that architecture patterns are more readily apparent in different views. We intend to study which views match certain patterns better in the sense that the patterns become more visible and explicit. Ideally one specialized view per pattern would solve this problem, but the combination of patterns can complicate things.

Our studies have shown that architects consider the annotation of architecture diagrams with tactic implementation information useful. But longer term, how is it used? How useful is it for maintainers who are learning the system architecture? This is an area for long-term study.

The interaction of tactics with each other should be studied. In particular, it appears that the really interesting interactions of tactics come between tactics from different quality attributes; for example tactics of fault tolerance and tactics of performance. Studies of reliability tactic interaction can provide specific information as input to such tradeoff analyses.
Each of the tactics introduces some additional behavior into the system. In this respect, one can consider a tactic to be a fault tolerance feature: behavior of the system with the goal of improving fault tolerance.

We noted that timing of actions is particularly important for certain fault tolerance tactics; for example:

- In Ping/Echo and Heartbeat, messages must be sent within a certain time period, or else the component is considered to be in failure.
- With Active and Passive Redundancy, messages must be sent to the redundant components within a certain timeframe; otherwise synchronization can be lost.

We did not study timing in detail, but it appears that timing may be an important issue with Pipes and Filters, where processing of data can be in large units. In addition, sequencing of actions is part of behavior. Both these aspects of behavior should be studied in more detail to understand how they affect the patterns.

One very interesting consequence of implementing the tactics is that since it involves changing the architecture, some changes may actually change the pattern composition of the architecture. An architecture pattern may be added. In certain cases, an existing pattern may even change to a different pattern. Obviously, the transformation of one pattern to another can happen only where patterns are similar. We have seen two examples of this type of transformation in architectures we have evaluated. We intend to study this further in order to understand its architectural implications.

The use of architecture patterns in conjunction with common architectural methods can be further explored. In particular, we are interested to explore the interaction of patterns with quality attribute reasoning frameworks. This should include analysis of the interaction of multiple quality attributes within each pattern, in order to understand how to focus on a single critical quality attribute. After the impact of patterns on quality attributes is cataloged, it might be used as some of the knowledge in the reasoning frameworks.

Additional studies of PBAR should be done 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


27. Booch, G. “Handbook of Software Architecture: Gallery”


52. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.


66. Harrison, N. and Avgeriou, P. “Implementing Reliability: the Interaction of
Requirements, Tactics, and Architecture Patterns,” Architecting Dependable

67. 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.

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

Architectural Knowledge with Architectural Patterns”, 11th European Conference
on Pattern Languages of Programs (EuroPLoP 2006), Irsee, Germany.

70. Harrison, N., Avgeriou, P. and Zdun, U. “Architecture Patterns as Mechanisms for

on Architecture Patterns”, International Workshop on Software Engineering for

72. Hofmeister, C.; Kruchten, P.; Nord, R.L.; Obbink, H.; Ran, A. & America, P.
“Generalizing a Model of Software Architecture Design from Five Industrial

73. Hofmeister, C., Nord, R., and Soni, D. Applied Software Architecture, Addison-
Wesley, 2000, 7-8.

74. Holz, H., Applin, A., Haberman, B., Joyce, D., Purchase, H., and Reed, C.: ‘Research methods in computing: what are they, and how should we teach

75. IEEE 1471: Recommended Practice for Architectural Description for Software-

76. International Standards Organization. Information Technology – Software

77. Im, T. and McGregor, J. D. “Toward a reasoning framework for dependability”,

Proceedings of the 34th Annual Hawaii international Conference on System
Sciences ( Hicss-34)-Volume 9 - Volume 9 (January 03 - 06, 2001).


122. SEI: “Resources for Educators in Software Architecture”

123. SIGCSE: The ACM Special Interest Group on Computer Science Education, [www.sigcse.org](http://www.sigcse.org)


