Software Architecture Pattern Morphology in Open-Source Systems

Neil B. Harrison, Erich Gubler, and Danielle Skinner
Department of Computer Science and Engineering
Utah Valley University
Orem, Utah, USA
neil.harrison@uvu.edu, erichdongubler@gmail.com, dskinner462@gmail.com

Abstract— **Architecture patterns are commonly used to specify, understand, and document software architectures. As a system evolves, its architecture patterns are affected. In some cases, the patterns themselves may change. We studied the architectural documentation of a large number of open-source systems to learn how the architecture patterns change as the systems evolve. In some cases, the existing patterns accommodate the system evolution without changing; such as adding layers to a layered architecture. In other cases, new patterns are added where no patterns existed. In some cases, new patterns were added to existing architecture patterns. And in a few cases, a pattern changed to a different architecture pattern. We observed instances of each type of change. In most cases, there was a logical structural evolutionary path to the new pattern. Typically, the architecture documentation also explained the important quality attributes that motivated the changes. We propose this work as a foundation for the study of how architecture patterns change as system architectures evolve.**

Keywords— ***Architecture Patterns; Evolution; Open-Source Systems***

I. Introduction

Architecture patterns are architectural structures which are commonly used, understood, and documented [1] [2]. A pattern describes the high-level structure and behavior in general terms, and is applied as part of the architectural design of a software system. It contains the major components and connectors of the system or part of the system. Most modern software architectures employ one or more architecture patterns [3]. Architecture patterns are similar to architectural styles [4] [5] [6], though the styles focus on sets of constraints. Architecture patterns are chosen in response to early design decisions, including decisions about how to satisfy functional requirements, non-functional requirements (quality attributes), and physical constraints (such as physical distance between user and service provider) [7]. Avgeriou and Zdun have summarized and organized many of the most commonly used architecture patterns [8].

As a system evolves, its architecture may change in order to accommodate new structures and behavior of the system. The changes may or may not affect the architecture patterns in the system. Such changes may be made in order to satisfy quality attributes (see [9] for example associated with reliability.) For example, a common pattern is Layers. If the system needs to be modified to increase security, a layer concerned with authentication might be added at the "top" of the existing layers. In this case, the architecture pattern is extended. Systems which use the Pipes and Filters pattern often evolve through the creation of new filter components, which can be tied together with existing filters.

On the other hand, some changes to systems significantly change the structure and/or behavior of the patterns used. This may involve adding components and connections which do not fit within the pattern, or modifying behavior [7]. The greater the changes, the less recognizable the pattern is.

Some architecture patterns are structurally similar, and changes to one may result in the other pattern. Harrison, Avgeriou, and Zdun proposed that during the evolution of a system, some architecture patterns may easily change to other related patterns [10]. For example, a system built on the Client-Server pattern may need to evolve to handle greater load by increasing servers, balance requests, or increase availability through replication of servers. A broker component may be introduced to manage the communication between clients and servers; thus the Client-Server pattern changes to the Broker pattern. (Note that in this case, while the Client-Server pattern still exists, it is subsumed by the Broker pattern.)

While it was postulated that architecture patterns can morph to a different pattern as part of the normal course of system evolution, it was necessary to determine whether such morphs actually happen in practice. If they do, there may be morphs that occur frequently. If we can understand which pattern morphs are common, and what the motivations for the morphs are, this information can help architects understand common evolutionary paths of software systems. It can also help guide architects in selecting architecture patterns during initial design and during evolutionary design.

II. Background

Software architects have always been concerned with system evolution: how well the architecture supports evolution, and how the evolution of the system affects the architecture. Taylor et al state that an architecture-centric approach to software development provides a solid basis for effective evolution [11]. Conversely, Cockburn notes that changes in requirements foster incremental re-architecture [12].

One of the important motivators of some architecture patterns has been that they support system evolution. The pattern descriptions given in [1] include discussions of consequences, many of which include a statement about whether a pattern supports system evolution. For example, part of the consequences of the Broker pattern are as follows: "Modifying the internal implementation of the broker, but not the APIs it provides, has no effect on clients and servers other than performance changes. Changes in the communication mechanisms used for the interaction between servers and the broker, between clients and the broker, and between brokers may require you to recompile clients, servers, or brokers. However, you will not need to change their source code."

Of course, many changes to a system involve changes to the structure of the system, which can change the architecture patterns used in the system. Harrison and Avgeriou explored the nature of changes to architecture patterns in the context of supporting quality attributes, and specifically, implementation of various tactics within architecture patterns [9] [13] [14]. They identified categories of changes to architecture patterns. The categories, in order of increasing impact on the pattern, are: implementing a tactic within the pattern with no changes to the pattern, duplicating components of the pattern, adding new components but keeping the pattern intact, adding new components that change the pattern, and modify key components of the pattern. Major impact might change the pattern enough that the pattern itself could change.

Harrison [14] also postulated that changes could result in the removal of a pattern from the system, though he had no evidence of its occurrence. Architectural drift and erosion could cause the loss of a pattern.

Harrison, Avgeriou, and Zdun [10] proposed that some such changes could lead to other patterns, thus changing one architecture pattern to another. This was based on structural similarity of some patterns to others. It also proposed that there should be a logical motivation for such morphs beyond structural affinity of the patterns. This work continues this proposition: investigating whether evidence of pattern morphs exists in the evolution of real systems. It also begins to investigate that if pattern morphs exist, what structural paths from one pattern to another exist. It also begins to consider the motivations for such evolution.

Of course, not all architectural evolution must be from one pattern to another. We considered five different types of pattern evolutions:

1. A pattern evolves without changing the pattern itself. For example, the Layers pattern adds new layers. This is generally within the intent of the use of the pattern. Since there are no pattern changes, this was not a focus of our study.

2. A pattern changes to another pattern. The new architecture contains the new pattern, but not the old pattern (unless it is part of the new pattern, such as Client-Server is part of Broker.)

3. A new pattern is added to an architecture, interacting with an existing pattern.

4. A new pattern is added to the architecture, where there was no pattern before.

5. A pattern is removed from the architecture.

III. Related Work

Bosch [6] identifies the imposition of an architecture pattern as an architectural transformation to add functionality. He gives a hypothetical example of introducing the Blackboard pattern into an architecture with no previous pattern. We considered the addition of a pattern to be a type of morphology, and thus looked for them. The work does not discuss the morph of one pattern to another.

Barnes and Garlan [15] [16] discuss architectural evolution and introduce architectural evolution styles. These are evolution paths specified as lists of properties of transitions. For example, an evolution style may stipulate that each node ("node" was used but not defined by the authors) must specify whether it is intended to be a public release. Although they do not mention architecture patterns, they do mention architectural styles. The pattern morphs might be included in the model of evolution styles.

Tamzalit and Mens [17] also discuss evolution paths, and identify some architectural styles. However, the styles are not the same as those described by Bass et al. [3], which are similar to architecture patterns.

Aoyama [18] discusses that architectural evolution can be continuous or discontinuous. They state that discontinuous software evolution changes some essential aspects, such as software architecture and major features. By this definition, a morph from a pattern to a closely related pattern would probably be considered continuous evolution. If any morph happens between structurally unrelated patterns, it would probably be considered a discontinuous evolution. Patterns are not mentioned.

Bhattacharya and Perry [19] describe a model for assessing system evolution. Bengtsson et al [20] describe an architecture-level modifiability analysis model. Neither work considers patterns, but regular pattern morphs, if identified, could become components in these models.

IV. Research Questions

Our research focused on the following questions:

1. Existence: Is there evidence of architecture pattern evolution during system evolution? We are looking for three particular types of evolution: are there cases where one pattern changes to different pattern (a "morph"), new patterns added to existing patterns, and patterns added where no patterns existed? If so, how commonly do these pattern evolutions occur?

2. Structural path: Where morphs occur (if they do), are there structural similarities between the two patterns that constitute a logical path for morphing?

3. Quality attribute influence: What is the role of quality attributes on pattern evolution?

V. Study Methodology

In order to study how software architecture patterns evolve, it is necessary to have a substantial body of architectural documentation. This documentation must contain two things for each system studied. First, the documentation must be sufficiently structural in nature that one can identify patterns, where present. For example, the 4+1 Physical view (see [21]), and the Components and Connectors view in AADL [22] and other similar descriptions would be useful. Second, the architectural descriptions must include some information about the evolution of the architecture of the system. And of course, the more architectures to study, the better.

The challenge is to find enough available architectural descriptions to have a meaningful sample. We therefore explored architectural descriptions of open-source software projects. Two volumes of such descriptions have been published in [23] [24]. These books are compilations of a total of 49 architectural descriptions of open-source applications. This provided a substantial body of literature for our investigation.

We performed a study of software architecture documentation to identify the types of architectural changes that occurred with regard to the architecture patterns used. We chose a cross section of open-source software projects. There were several reasons for using the published patterns as the cross section. First, we wished to study successful projects, since successful projects are more likely to show effective use of patterns – we don't want to emulate failure. Second, successful projects are more likely to have an evolutionary story. The fact that the projects were selected for inclusion in the aforementioned volumes suggests that they are successful.

Two other major motivators for studying these projects were that the architectural documentation was available for study, and there was some consistency in the motivation behind the documentation (they were to be published together in a series of books.)

An important benefit of these descriptions is that they had a common theme: "the authors … explain how their software is structured, and why. What are each program's major components? How do they interact? And what did their builders learn during their development" [23]? We found that nearly all of the descriptions were sufficiently common and had enough structural information to study. Of the 49 descriptions, we were able to find meaningful structural architectural information in 44 of them.

A third volume is devoted to exploring the performance of OSS systems [25]. While the focus was clearly on performance, we did find that eight of the twelve descriptions contained enough structural architectural information that it they could be added to our study. Of particular note was one chapter that described the evolution of a system described in the first volume to a new version.

Of the 56 chapters with structural architectural information, there were 23 that discussed the evolution of the systems. From these we were able to find information of how architecture patterns evolved.

The systems spanned a wide variety of domains. We identified sixteen different general domain categories, although there is still considerable diversity within most categories. We didn't differentiate the results by domain for this study, as it is a general study.

For this study, we were interested in how quality attributes influenced pattern morphing. So we also identified quality attributes mentioned in the descriptions, and noted any that were motivators of the architectures.

Our study methodology was as follows:

1. Each chapter was studied by one researcher, who identified which patterns were used in the architecture. The researcher also identified which of the patterns, if any, were introduced during system evolution, and if it had morphed from an existing pattern, had been added to an existing pattern, or added from no pattern. The patterns we looked for were the architecture patterns from Buschmann et al. [1] and those summarized by Avgeriou and Zdun [8] (which includes the Buschmann patterns.)

2. We noted the strength of the evidence of the patterns, either strong, medium, or weak. In only a few cases, the authors named a pattern directly. That, of course, was strong evidence. In other cases, components either had names related to patterns (e.g., "client"), or were clear from component structure (e.g., structures that showed filter components connected by pipes.) These were also considered strong. Medium and weak patterns required more digging to identify. Therefore, we eliminated many of the medium and all of weak cases – see below, leaving most of the patterns identified as strong.

3. Each chapter was checked by another researcher, who also identified the patterns. We compared the results, and where there was disagreement, we discussed it among all three of us. Where a pattern was identified by one researcher and not by the other, we eliminated it from consideration, unless the researcher could show clearly where the pattern was used. We also discussed and resolved cases where a pattern was identified by both, but with different strength. The result of these comparisons was that most patterns with only weak evidence were eliminated.

4. We also identified the important quality attributes of each system. We did this by noting quality attributes that were explicitly named, or were clear from the context. For example, some papers stated that ease of use (usability) was an important design driver, while other papers simply mentioned in passing that some changes were made to make a particular feature easier to use. We did not perform cross-checking on the quality attributes, partly because they were not the central focus of the study, and partly because they were obvious enough that cross-checking was deemed unnecessary. In 90% of the cases where we identified quality attributes, they were explicitly mentioned (194 occurrences out of 218 total).

   We differentiated the quality attributes as relevant to the initial architecture or the evolution of the system. In many cases, a quality attribute influenced both. For the purposes of this study, we were primarily concerned with the quality attributes that influenced system evolution.

5. Where we found evidence of pattern morphing, we explored in more detail. We attempted to ascertain the motivation for the change, as evidenced by the important quality attributes. (We tried to find which quality attributes, if any, were motivators for the change.) We also examined the structure of previous patterns, or the entire system, if there was no previous pattern, to understand the structural changes needed to adopt the new pattern.

We summarized the data and analyzed the results.

VI. Results

In the 45 systems architectural descriptions we studied, we found evidence of architecture patterns in all the systems we studied, with an average pattern density somewhat higher than in [3]. We also found a high density of quality attributes. The most common patterns were Layers, Client-Server, Plugin, Pipes and Filters, and Shared Repository. Further details of these results are beyond the scope of this paper.

There were 23 descriptions that specifically discussed the evolution of the system. Of these, 16 different architectures described evolution that involved architecture patterns. We found four ways in which architecture patterns were involved in the evolution of the systems, plus weak evidence of a fifth way:

1. Existing patterns were enhanced or extended without changing the pattern. For example, an additional layer might have been added to an existing Layers pattern.
2. A new pattern was added to the architecture independently of any other patterns in the architecture.
3. A new pattern was added to the architecture based on one or more existing patterns in the architecture.
4. A pattern in the architecture was redesigned, and became a different pattern. (This is a “morph” from one pattern to another.)
5. A pattern was removed from the architecture. This is the change where we found only weak evidence in a single case.

We will discuss each of these in turn. Note that several systems had evolutions that involved multiple architecture patterns, so the number of pattern evolutions described below is considerably greater than 16.

A. *Expanded use of existing patterns*

In some cases, an architecture used a pattern, and as the architecture evolved, the pattern remained intact, and more components were added within the framework of the pattern. We found two different patterns where this occurred. First, one or more systems added one or more layers to the existing Layers architecture. This occurred at least once. Second, additional filter components were added into a Pipes and Filters architecture. This occurred at least twice.

This is not very noteworthy, and therefore we did not look for it, or note it every time we saw it. Thus there are likely more cases of such evolution. It is simply verification that patterns that have strong support for extensibility are extended as designed.

B. *New Patterns Added to the Architecture*

We found several cases where patterns were added to the architecture where there were no patterns previously. In other words, while the systems may have used patterns in their architectures, additional patterns were added independently of the existing patterns. Adding patterns in this manner occurred nine times. Table 1 summarizes the patterns added and the number of occurrences of each.

TABLE I. New Patterns Added to an Architecture

Pattern Added	Number of Occurrences
Layers	3
Model-View-Controller	2
Event-Based System	1
Client-Server	1
Broker	1
Reflection	1
Rule-Based System	1

It is unsurprising to see several instances of the addition of Layers, as the addition of functionality is often accomplished by creating a separate module to handle the new capability. In one case, a database was added to the architecture, and layering was added to support it. In the second case, layers were introduced to help organize code that was becoming difficult to maintain. In the third case, layers were added to introduce a reflection capability while minimizing the changes to existing code.

The two cases of adding Model-View-Controller were notable. In both cases, the motivation for doing so appeared to be mainly to follow a standard architecture pattern for user interaction. Extensibility and maintainability were also cited as motivations. Neither paper discussed how difficult adding MVC to their existing system was.

There was a logical path to the Event-Based pattern. Implementing traditional multi-threading resulted in worse performance than single-threading. To take advantage of multi-core boxes to increase performance, they implemented a model of an actor passing messages which eliminated a lot of the traditional multi-threading problems. This model required an event system because any blocking operations would result in blocking other objects sharing the same thread.

One system changed to a Client-Server pattern in order to allow multiple processes of execution, and to allow different user experiences. It also evidently helped avoid rework in recovering from errors. The paper says that the client-server functionality was factored into the design “over a period of time.” The architecture also describes a component that may be a broker component, though the evidence was not compelling enough for us to call it a Broker rather than a Client-Server pattern.

One system added a broker to dispatch requests to appropriate handlers, depending on the type of request.

One system was a dynamic runtime environment. It added an object reflection protocol implemented as a DLL to allow any modern dynamic scripting language runtime model to be used for their implementations on the runtime environment. There was insufficient information to ascertain how easily the original software architecture changed to add the reflection pattern.

One system evolved to a rule-based architecture pattern. The original system had some configuration options, and as more were added, the formality of handling them increased, eventually resulting in a rule-based architecture to manage them. It appeared that the transition was quite smooth.

TABLE II. PATTERNS ADDED TO OTHER PATTERNS

Existing Pattern	Pattern Added	Number of Occurrences
Pipes and Filters	*Plugin*	2
Pipes and Filters	*Rule-Based System*	1
Pipes and Filters	*Interpreter*	1
Plugin	Broker	1
Plugin	Reflection	1
Model-View-Controller	Rule-Based System	1

Finally, we note that the ability to add an architecture pattern also depends largely on the existing structure of the architecture. Thus the selection of a pattern to add was determined not only by the needs of the project, but by the existing architecture. The Layers pattern lends itself well to addition: it is often easy to wrap all or part of a system to create a layered architecture. This notion is supported by the fact that Layers was the most common pattern to be added.

For the other patterns added, however, there is a less clear general structural path for adding the pattern. We did not find any general architectural structures that support the easy addition of the other six patterns that were added. These patterns might have been added without natural architectural structure support because of other motivators, such as quality attributes.

C. *Patterns added to other patterns*

We found several patterns that were added to work together with existing patterns in system architectures. They are summarized in Table II.

We can first look at which patterns are commonly built upon; which patterns might lend themselves well to the addition of other patterns. We see that Pipes and Filters appears to be a common base for adding other patterns. However, the evidence is actually weaker than it appears, because one architecture accounted for three of the pattern additions: adding Plugin, Rule-Based, and Interpreter to Pipes and Filters. (See italicized rows.) On the other hand, it does indicate that the filter components can easily be enhanced to provide additional capabilities, such as plugins. Likewise, the Rule-Based System pattern was added to allow dynamic modification of filters, as well as dynamic configuration of the filter topology. The Interpreter pattern was added to wrap the Pipes and Filters system and control it.

The other instance of addition of Plugin to Pipes and Filters was apparently done for ease of extensibility. Little else was mentioned about it in the architectural description document.

The two cases where patterns were added to a Plugin architecture were from a single system. The management of interdependent plugins was becoming increasingly complicated. As the heart of the system was extensibility, it was important to provide a way to manage packages of plugins. Thus the designers added Reflection and Broker together to manage the packages. So in this case, the growth of packages created the need for the additional patterns.

TABLE III. PATTERNS MORPH TO A NEW PATTERNS

Original Pattern	Pattern that it Morphed to	Number of Occurrences
Client-Server	Broker	3
Simple Repository	Shared Repository	2-3
Client-Server	Broker + Event-Based System	1
Broker	Peer-to-Peer	1
Rule-Based System	Indirection layer	1
Shared Repository	Event-Driven System	1

Adding the rule-based pattern to an MVC architecture was done to automatically catch input bugs and security flaws through a type system. Thus rules were added to the input processing part of the system. It appears that it was facilitated by the separation of input processing from model processing that is inherent to MVC, but not necessarily unique to it. (See, for example, the Presentation-Abstraction-Control pattern [1].)

D. *Morph from one pattern to another*

We found several instances of architecture patterns that were changed to another or replaced by another pattern. They are summarized in Table III.

The most common morph of one pattern to another was from Client-Server to Broker. Structurally, this is a very natural morph, as all that is required to accomplish this change is to add a broker component to manage the interaction between clients and servers. This was the primary pattern morph proposed by Harrison et al. [10]. The Broker pattern supports many different quality attributes, and this was reflected in the motivations for the broker described in the papers. In one case, the Broker was used to improve performance. Similarly, in another system, the Broker was incorporated to enable load balancing. In another case, a Broker was used to support different types of clients easily.

In a special case of changing Client-Server to Broker, an Event System was added at the same time. There was a logical path to adding a broker, which was implemented as a reverse-proxying server, which allowed them to put an upper limit on their API calls. The architects realized that with the other improvements to the system, the number of API calls would increase dramatically, which would likely be a problem. Because they identified this concern during their evolutionary design process they were able to implement this pattern concurrently with the Event System pattern implementation. So as a stand-alone evolution, it may not make as much logical sense, but with the implementation of the Event System it makes more sense.

Shared Repository is a common architecture pattern. The pattern implies a degenerate, or simpler instance, where the database is not shared, but still figures prominently in the architecture. One might call this a Simple Repository. We found two cases of strong evidence of an original Simple Repository which morphed to a Shared Repository, and one case with weak evidence. The evolution path from Simple Repository to Shared Repository is strong, both in terms of structure and in quality attribute motivation. Structurally, a Simple Repository could change to a Shared Repository by the addition of software to handle multiple and possibly concurrent accesses. Such capabilities may be already built into the database software package used. The quality attribute motivations behind the morph would likely include capacity and/or flexibility.

We saw one case of a morph from Broker to Peer-to-Peer. Structurally, this is not a difficult morph, but not necessarily trivial. In this case, while the Peer-to-Peer pattern was clear, the evidence of a previous Broker pattern was fairly weak: it could be that the Peer-to-Peer pattern was added to the architecture replacing or augmenting no other pattern. The motivation for the pattern was to support completely non-blocking messaging.

The evolution from a Rule-Based system to an Indirection Layer was a significant change for the system where this occurred. We could see no natural structural evolution path. It was motivated by years of trying to maintain an ever-growing set of rules on a system that was more enduringly popular than was originally envisioned. Thus the change was motivated by a need for improved maintainability and flexibility of the system.

In the last case, the Event System was added to the system as it evolved, replacing a Shared Repository. There was a logical path to the Event System pattern. The designers goals with implementing an Event System was to allow the server to work as quickly as the client-side computations while not creating negative networking side effects. These issues included being restricted to a small number of connections and slowing communication to the server by taxing the upload bandwidth. They also worried about reliability and security problems because the data was stored as a snapshot and therefore had the potential to be modified erroneously or maliciously. Having the server do its own processing of commands logically solves these problems, which then drove the use of the Event System pattern.

E. Pattern removal

One other possibility of architecture pattern morphology is the removal of a pattern. We expect that pattern removal would be very rare. We found very weak evidence that one system had removed the Client-Server pattern during evolution, but would need much more solid evidence to declare that it actually happened. We suspect that if removals happen, they are rare enough that study of them would not be interesting.

VII. Limitations of the Study

A limitation to this study is that the architectures studies are all of open-source systems. Therefore, the results might not generalize to software systems in general. However, we make no claims that the results are representative of all of software. A large part of the study was to learn whether architecture pattern morphs occur in practice, which does not require that the sample studied be representative of the general population. Furthermore, we note that the projects studied were a wide variety of applications, giving a more general sample anyway. We also note that companies consider the architecture of systems to be intellectual property, and thus proprietary. It is likely that OSS systems are the only possible large source of generally available software architecture information.

A concern is how the OSS projects were selected for inclusion, and whether that invalidates the results in any way. It is likely that a major criterion for selection is that the projects were well-known, and by inference, successful. This restricts the conclusions that can be drawn from this study. One cannot conclude that the pattern morphs we found are typical of all software systems, or even typical of OSS systems in general. However, they do demonstrate existence of the three types of pattern morphs we were investigating.

A possible threat to validity concerns possible researcher bias in identifying the patterns in the documentation. In order to eliminate this bias, two researchers examined each architecture independently; each identified patterns. In each case of disagreement (one researcher identified a pattern that the other did not), all three of us discussed it and came to a conclusion together. We erred on the side of caution, and eliminated several patterns where the evidence was weak. The end result was that all three of us agreed on the identification of all of the patterns, as well as the pattern morphs.

One consideration is that we didn't consult the authors themselves to verify the patterns and the pattern morphs we identified. However, a primary purpose of architecture documentation is to inform others without the need for clarification from the author: the work must stand on its own. We took this approach in our study; we did not want additional information from the authors not contained in the documentation. Furthermore, we were able to find evidence of pattern morphing in the documentation even without consulting the authors. In fact, it is possible that communication with the authors will reveal additional morphs not described in the documentation. We recommend that follow-up research include consultations with the authors. Note, however, that it was apparent from the papers that the authors' knowledge of patterns varied. This will affect the validity of questionnaires asking authors to identify the patterns used, or to validate the presence of patterns identified.

VIII. Conclusions

We see that pattern morphs do occur in industrial OSS systems. We found evidence of the four types of pattern evolution described above, and slight evidence of pattern removal. In the interesting cases, we found the three kinds of architecture pattern morphs: those from no pattern, those that add additional structures to an existing pattern (such as adding layers), and those that changed one pattern to another. While there was not sufficient data to draw any conclusions about which type of morph is the most common, we observed that each happened several times.

Overall, we found pattern evolution to be common in the sample projects we analyzed. Over two thirds of the systems with evolution information had one or more pattern morphs. This gives some evidence that pattern evolution is not only possible, but a common way of evolving a system architecture.

Several morphs involved the addition of a pattern, which validates a premise from Bosch [6]: he describes the imposition of an architecture pattern as one possible transformation of an architecture. We see that architecture patterns are useful tools not only during the initial architecture, but also during system evolution.

It was clear from the descriptions that quality attributes often strongly influenced the patterns used. While this is hardly new, we saw evidence of how they influence the patterns used in evolution of systems.

Certain morphs were common, and in each case, there was a logical structural path for the morph. This may indicate natural evolutionary paths for systems and/or architecture patterns. On the other hand, there were some morphs that happened although there was not a logical structural path from one pattern to another, or to build one pattern on another. This shows that while the existing patterns in an architecture may influence its evolution, they don't necessarily constrain it. The functional and quality attribute needs of the system may trump the existing structure. This is expected.

While it is not conclusive, there is burgeoning evidence that some morphs may be common enough to become typical or standard pattern morphs. If further research establishes such standard morphs, they will become important tools for architects. Architects who not only know the patterns but also the standard morphs will be better equipped to design systems for change. In addition, standard morphs would provide deeper understanding into consequences of certain patterns; namely, that they provide natural evolutionary paths for systems.

IX. Future Research

There are many opportunities for continued research in this area. The following are only some of the research that can be done.

1. We plan to follow this work up with discussions with the authors of the documentation. This can further validate the research and potentially identify new pattern morphs. It may also provide a basis for quantitative research into pattern morphology. It would be well, where possible, to interview the original architects of the systems as well.

2. We also recommend studying other architectures for evidence of morphs from one pattern to another. This also may identify other morphs that are used in practice, and find further evidence of the morphs. However, we note the difficulty of finding architectural documentation that includes evolutionary information.

3. If possible, it would be desirable to explore architecture of non-OSS systems in order to generalize the results to software systems in general. Of course, documentation of architectures of non-OSS systems is considerably more difficult to find.

4. The systems we studied came from several different domains. An earlier study of patterns in architectures identified the prevalence of patterns in certain domains [3]. This could also be done with these systems, and expanded to consider morphs. It may be that certain morphs are characteristic of certain domains. We expect to explore this further, but because the sample covered many domains, there are only a few in each category. Thus it will probably require a larger set of data.

5. We postulated that some pattern morphs were more or less natural, while some were structurally unnatural, but driven by specific needs of the application. As more evidence of pattern morphs is accumulated, it should be possible to identify the natural morphs, and the application-specific unusual or "one-off" morphs.

6. It could be enlightening to study the motivation for architectural changes that led to the pattern evolution. For example, technical motivations for changes may result in different changes than non-technical motivators.

7. Our investigation into the logical structural path of morphs is preliminary. We hope to explore these paths further and specify them in terms of the architecture pattern changes outlined by Harrison and Avgeriou [7] [14]. This may create the foundation of a model and theory of architecture pattern morphology.

8. It was clear that various quality attributes were the motivators behind most, if not all of the pattern morphs. It would be useful to study the morphs together with their motivating quality attributes. If a model of quality attributes and morphs can be created, it might be used to predict what morphs are likely to occur, based on the important quality attributes in a system, together with its current architecture.

9. Another approach to understanding architecture pattern morphs is to study the patterns themselves. Identify other potential morphs from one pattern to another through structural affinity. It is then necessary to identify a plausible motivating story for such a morph, and then find existence proofs. We suggest that this might yield interesting possibilities.

Because of the many possibilities for further research, we welcome others who wish to partner with us.

Acknowledgment

We thank Paris Avgeriou at the University of Groningen (the Netherlands) and Uwe Zdun at the University of Vienna for their discussions and ideas about pattern morphology, along with validation of early concepts. We also thank Olaf Zimmerman at the Institute for Software, Hochschule für Technik,(Rapperswil, Switzerland) for his ideas on the topic, and for pointing us to the architecture documents.

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1996.

[2] N. Harrison, P. Avgeriou, U. and Zdun, "Using architecture patterns to capture architectural decisions", IEEE Software, vol. 24, no. 4 Jul-Aug 2007 pp. 38-45.

[3] N. Harrison, and P. Avgeriou "Analysis of architecture pattern usage in legacy system architecture documentation", Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, Canada, Feb 18-22, 2008, pp.147-156.

[4] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall, 1996.

[5] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 2nd ed., Addison-Wesley, 2003.

[6] J. Bosch, Design & Use of Software Architectures: Adopting and evolving a product-line approach, Addison-Wesley, 2000.

[7] N. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A Model and Annotation", Journal of Systems and Software, 83, 10 (October 2010), pp. 1735-1758.

[8] P. Avgeriou and U. Zdun, "Architectural patterns revisited—a pattern language," Proc. 10th European Conf. Pattern Languages of Programs (EuroPLoP), UVK Konstanz, 2005, pp. 431–470.

[9] N. Harrison, P. Avgeriou, and U. Zdun, “On the impact of fault tolerance tactics on architecture patterns”, International Workshop on Software Engineering for Resilient Systems (SERENE ’10) April 15-17, 2010, pp. 12-21.

[10] N. Harrison, et al., “Software architecture patterns: reflection and advances” [summary of the MiniPLoP Writers' Workshop at ECSA'14]. SIGSOFT Softw. Eng. Notes 40, 1 (February 2015), pp. 30-34.

[11] R. Taylor, N. Medvidovic, and E. Dashofy, Software Architecture: Foundations, Theory, and Practice, Wiley, 2010.

[12] A. Cockburn, Agile Software Development: 2nd ed., Addison-Wesley, 2007.

[13] N. Harrison, and P. Avgeriou, “Leveraging architecture patterns to satisfy quality attributes”, First European Conference on Software Architecture (ECSA), Madrid, Sept 24-26, 2007, pp. 263-270.

[14] Harrison, N. “Improving Quality Attributes of Software Systems Through Software Architecture Patterns”, University of Groningen, the Netherlands, 2011.

[15] J. Barnes, Software architecture evolution, (CMU-ISR-13-118). Pittsburgh, PA: Software Engineering Institute, 2013.

[16] D. Garlan, J. M. Barnes, B. Schmerl, and O. Celiku, "Evolution styles: foundations and tool support for software architecture evolution," Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA), Sept. 2009, pp. 131-140.

[17] Tamzalit, D.; Mens, T., "Guiding architectural restructuring through architectural styles," 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS), 22-26 March 2010, pp.69-78.

[18] M. Aoyama, “Continuous and discontinuous software evolution: aspects of software evolution across multiple product lines”, 4th International Workshop on Principles of Software Evolution (IWPSE '01). New York, NY, pp. 87-90.

[19] S. Bhattacharya and D. E. Perry, "Architecture assessment model for system evolution," Working IEEE/IFIP Conference on Software Architecture, 2007 (WICSA '07), 6-9 Jan. 2007, pp. 8-8.

[20] PerOlof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. 2004. Architecture-level modifiability analysis (ALMA). J. Syst. Softw. 69, 1-2 (January 2004), pp. 129-147.

[21] Kruchten, P.: “The 4+1 view model of architecture”, IEEE Software 12(6) (1995).

[22] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R. & Stafford, J. Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2002.

[23] A. Brown and G. Wilson, eds., The Architecture of Open Source Applications, vol. 1, lulu.com publishing, 2008. Also available at: http://aosabook.org/en/index.html.

[24] A. Brown and G. Wilson, eds., The Architecture of Open Source Applications, vol. 2, lulu.com publishing, 2012. Also available at: http://aosabook.org/en/index.html.

[25] T. Armstrong, ed., The Performance of Open Source Applications, (vol. 3 of The Architecture of Open Source Applications), lulu.com publishing, 2015. Also available at: http://aosabook.org/en/index.html.