An Interview-Based Case Study in Software Architecture Evolution

Jeffrey M. Barnes
*Institute for Software Research, Carnegie Mellon University**
Email: jmbarnes@cs.cmu.edu

Abstract

In recent years, researchers have worked to develop approaches and models to support software architects in planning and carrying out major evolutions of software systems. To date, this line of work has been largely theoretical. These new approaches and models are seldom accompanied by empirical evidence to support them, let alone subjected to rigorous empirical evaluation.

This paper describes a formal case study examining architecture evolution in a real-world organization. Based on content analysis of interview data and architectural documentation, the study examines how practicing architects plan and reason about evolution, what challenges they face, and whether the modeling approach developed in our previous work can capture the concerns that arise in a real-world evolution.

1. Introduction

Software architecture—the discipline of designing the high-level structure of a software system—is today widely recognized as an essential element of software engineering. However, one topic that current approaches do not adequately address is software architecture evolution. Architectural change is commonplace; as systems age, they often require redesign to accommodate new requirements, support new technologies, or respond to changing market conditions. Today, architects lack tools to help them in developing plans for such evolution.

In recent years, we and other researchers have worked to develop approaches and models to support architects in reasoning about evolution [1], [2], [3], [4], [5]. To date, this research has been largely theoretical, with only tenuous empirical support.

This paper seeks to establish a sounder empirical basis for this line of research by presenting a new case study examining the needs of real-world architects and the applicability of the approach that we have developed in previous work. This case study distinguishes itself from the few existing empirical studies by its careful data collection procedures, its rigorous analytical method, and a formal case study design that permits sound analytic generalization of the results.

The study was designed to answer three questions:

RQ1. How do practicing architects in a real-world organization plan and reason about evolution?

RQ2. What difficulties do practicing architects face in planning and carrying out evolution?

RQ3. How well can the modeling framework developed in our previous work capture the concerns that arise in a real-world architecture evolution?

In addition, this research makes a methodological contribution, demonstrating how research methods from the social sciences can be applied to the field of software architecture in a novel and useful way. A key goal was to provide ample information to allow other researchers to reproduce this work or adapt its methods to other purposes. To that end, this conference paper is accompanied by a 104-page technical report [6], available online, which exhaustively documents the study's methods.

This paper is organized as follows. Section 2 discusses existing research in this area. Section 3 outlines the case study design. Section 4 documents the collection of data. Section 5 describes the analysis procedures. Section 6 summarizes the findings, and Section 7 concludes.

2. Background

Though software evolution has been a vibrant research area for decades, relatively little attention was devoted specifically to *architectural* issues in software evolution until recently. Within the last decade or so, this has been changing. Detailed surveys of the burgeoning literature on architecture evolution now exist [7], [8], so I do not give a full survey here. Instead, Section 2.1 discusses how this work has been supported *empirically*. Then, Section 2.2 provides a summary of the approach that we have developed in our own research.

** The author carried out this research while a PhD student at Carnegie Mellon University. He recently graduated and is now a software engineer at Google, Inc.*

2.1. Related Work

Existing architecture evolution research has tended to rely heavily on artificial examples. Tamzalit, Le Goaer, et al., for example, have published a large body of work proposing formalisms and techniques for reasoning about architecture evolution (e.g., [2], [9], [10]), but they seem to have relied mainly on fictitious examples rather than observing real evolutions. Several of their papers (e.g., [2]) use an example of a client–server architecture based on an example from Cheng et al. [11]. Another paper [9] uses examples of a banking application and a chat application. Yet another [10] describes an online store evolving to a client–server architecture. All these examples seem to be artificial; there is no suggestion that they were based on observation of real evolutions.

Cuesta et al. [5] present what they call "a real-world case study" to illustrate their approach. However, it seems to be a fictitious example rather than an actual evolution that was carried out.

Wermelinger & Fiadeiro [3], Grunske [4], and Fahmy & Holt [12] present various approaches for architecture reconfiguration based on graph transformations, but none provide a substantial case study. Barais et al. [13], in their work on specifying architectural transformations, use a small, artificial example of a banking application. Spitznagel & Garlan [14], in their work on connector transformation, carried out a small case study involving enhancing a Java RMI connector with Kerberos authentication. However, this was a small example in laboratory conditions, not an observation of real-world practice.

Erder & Pureur [15] present a case study drawn from their experience in the banking industry: a loan-servicing company migrating to a service-oriented architecture. However, the case study is brief and gives few specifics.

This is not a complete survey, and it is possible to find solid empirical work that touches on architecture evolution by looking further afield (see the "Related Work" section of my dissertation [16]). But the point holds: most work in this area has not been well supported empirically. Even when a paper purports to present a "case study," it more often turns out to be something closer to a worked example, or at best an informal reflection on practice, than to a true case study.

Our own early work [17], in which we proposed our approach, also relied on artificial examples. However, in 2012, I published a case study examining the evolution of a telemetry processing system at NASA [18]. This was a first step in the direction of establishing empirical support for our work. The present study distinguishes itself from this earlier work in several ways:

- Data in the present study was collected through formal semistructured research interviews of practitioners. In the previous case study, data collection was conducted informally during an internship.
- The results are based on qualitative content analysis of the data. The NASA study lacked formal analysis.
- The study's findings are linked to the data by a rigorous case study design, with explicit consideration of validity and reliability. The conclusions of the previous case study were informal and impressionistic.

2.2. Our Approach

In previous work [1], we developed an approach to help architects to reason about and plan evolutions of software systems. Here I provide a very brief summary.

Our approach is based on considering possible *evolution paths*—ways of evolving from an initial architecture to a desired target architecture. An evolution path can be understood in terms of the evolutionary steps necessary to carry it out, which can in turn be specified as compositions of *evolution operators*—reusable architectural transformations like *add adapter* or *migrate database*.

Once the evolution paths are defined, the next step is to apply analyses to select the optimal path. To support the architect in selecting a path, we provide two kinds of analysis: *evolution path constraints*, which define which paths are legal or permissible, and *path evaluation functions*, which provide quantitative assessments of qualities such as duration and cost.

These concepts are simple, but there is a substantial formal framework supporting them. To formalize path constraints, for example, we have developed a language based on linear temporal logic.

3. Case Study Design

The organization that served as the subject of this case study was Costco Wholesale Corporation, a major global retailer. A few years ago, the company embarked on an extensive modernization effort to revamp its aging software systems. As a result, architectural-level evolution of software systems is now pervasive there. The contemporaneous overhaul of so many core systems has posed significant integration challenges. These factors made the company an appealing candidate for a case study on architecture evolution.

The case study comprised the following main phases:

Design. I first developed a formal case study design and secured IRB approval for the study.

Data Collection. I spent two weeks at the organization under study, conducting semistructured research interviews with architects and collecting architectural documentation (see Section 4).

Analysis. Analysis proceeded in two phases (see Section 5). First, I subjected the interview data and collected architectural documentation to content analysis. Then, to evaluate the modeling approach developed in our previous work, I used the output of the content analysis to develop an evolution model using our approach.

Synthesis of Results. Once the analysis was concluded, I synthesized the findings and wrote up the study results.

4. Data Collection

I visited Costco headquarters for two weeks to collect data. The primary data source was interviews with architects. I adhered to a *semistructured* interviewing discipline, which means that although the interviews were guided by a protocol that defined the topics to be examined, I was free to devise new questions on the fly to explore interviewees' knowledge. This is in contrast to a *structured* interview, in which every question is scripted (useful when the goal is to compare responses across participants), or an *unstructured* interview, in which the interviewer is unconstrained and the interview has no set format (useful in early exploratory work). I interviewed six participants in eight sessions with a mean duration of 41 minutes. I recorded and transcribed all interviews. For details on the transcription method, see the companion technical report [6].

I was also permitted to access architectural documents pertaining to the company's software systems. These were a useful complement to the interview data.

5. Analysis

This section describes the case study's analysis procedures. Section 5.1 explains the content analysis, which was the main phase of analysis. Section 5.2 describes the construction of an evolution model to evaluate the applicability of our modeling approach.

5.1. Content Analysis

The main analytical method used in this case study was content analysis, a research method for extracting meaning from text. First systematized during the Second World War, content analysis has become a major research method in many disciplines. A standard definition is that given by Krippendorff [19]: "a research technique for making replicable and valid inferences from texts (or other meaningful matter) to the contexts of their use."

A number of variants of content analysis exist. One of the most important and divisive distinctions is between quantitative and qualitative content analysis. Content analysis was first introduced as a quantitative method, and some methodologists regard qualitative content analysis as an inferior variant [19]. But although early qualitative content analyses sometimes lacked the rigor of the quantitative method, qualitative content analysis has undergone increasing systematization and formalization since the 1980s and is now widely recognized as capable of producing strong, reliable findings [20].

I chose qualitative content analysis for use in this study. Qualitative content analysis excels in cases where interpretation is needed to analyze data. Here, we are explicitly concerned with interpretation of the language architects use. Architectural jargon is not sufficiently standardized to justify a classical approach in which we apply our own unexamined interpretations to the data; even basic terms like *architecture* and *evolution* mean different things to different people. Qualitative content analysis provides means to deal with such differences and helps us to avoid imposing our own biases.

Schreier [20] describes a qualitative content analysis as proceeding in steps: (1) formulation of research questions, (2) construction of a coding frame, (3) segmentation of the material, (4) piloting the coding frame, (5) evaluating and revising the coding frame, (6) the main analysis, and (7) interpretation of the findings. The construction of the coding frame and the segmentation of the material are discussed below. Findings are discussed in Sections 5.2 and 6. For discussion of the other phases, see the companion technical report [6].

5.1.1. Coding Frame. Coding is an analysis method in which segments of text are annotated with descriptive codes or categories. Coding is done in accordance with a coding frame, which defines the categories to be applied: *main categories* that define the dimensions of the analysis and *subcategories* that refine them.

It is useful to observe here that the research questions in Section 1 are of two different characters. RQ1 and RQ2 are questions about how evolution happens in the real world. They are *descriptive* of the practice of architecture. They can be answered directly through content analysis of the interview data.

RQ3 asks whether our approach to architecture evolution is suitable for representing the concerns of a real evolution. This question is *evaluative* of our approach. RQ3 was addressed via a two-step process in which I first applied content analysis to data pertaining to a specific evolution, then used the output of that content analysis to construct an evolution model using our approach.

It is thus useful to describe the analysis as actually comprising two separate content analyses: one (**CA1**) targeted at the descriptive research questions and one (**CA2**) targeted at the evaluative question.

CA1

Evolution motives
- Add features
- [Seven other categories]

Causes of problems
- Lack of experience
- [Six other categories]

Consequences
- Lost sales
- [Four other categories]

Challenges
- Cultural challenges
- [Thirteen other categories]

Approaches
- Experience and intuition
- [Fifteen other categories]

CA2

Classification
- Component
- Connector
- Port or role
- System
- Grouping
- Containment relation
- Evolution operation
- [Three other categories]

Presence in initial architecture
- Present
- Absent

Presence in target architecture
- Present
- Absent

Figure 1: Overview of categories in the content analyses.

The differences between CA1 and CA2 necessitated that their coding frames also be constructed differently. It is helpful to consider a distinction drawn in the content analysis literature: the distinction between *concept-driven* (deductive) and *data-driven* (inductive) coding frame development [20]. With a concept-driven strategy, categories are defined a priori, based on preexisting theory. With a data-driven strategy, categories are derived through progressive summarization or other bottom-up strategies. In general, a concept-driven strategy is most appropriate for testing hypotheses or drawing comparisons with prior work, while a data-driven strategy is most useful for rich description of material.

I adopted a principally data-driven strategy for CA1 and a principally concept-driven strategy for CA2. The purpose of CA1 was to describe architects' perceptions and experiences regarding evolution in detail, so a data-driven approach was most appropriate. To construct the coding frame for CA1, I went through the interview data, marking passages relevant to the research questions, then consolidated similar passages into categories.

For CA2, a concept-driven approach was appropriate. The goal of CA2 was not open-ended description, but rather evaluation of an existing theoretical framework. This framework formed the basis for the coding frame.

Since its output would be used to produce an evolution model, CA2 had to identify the elements to appear in the evolution model. Thus, the coding units in CA2 were descriptions of architectural elements. The first output that the content analysis had to yield was the identification of these elements. For each element described by a coding unit, CA2 had to determine how that element should be modeled—whether it should be characterized as a component, connector, constraint, operator, or some other type of element. Thus, the first part of the coding frame was a classification scheme for elements of the architecture evolution, with subcategories such as

Challenges: Managing scope and cost (abbreviation: cha-cst)

Description: This category should be applied to challenges in scoping a project or keeping costs low.

Challenges in reconciling large expenses with a cultural of frugality should instead be coded "Challenges: Cultural challenges."

Example: "The biggest concern retailers have about [alternative payment methods] is the fees they have to pay as part of them"

Figure 2: A category definition from the coding guide.

"Component," "Constraint," and so on.

The second output that had to be produced was what phases of evolution the elements appeared in. For each element, we wanted to know whether it was present or absent in the initial architecture, and likewise for the target architecture. The coding frame for CA2 thus included "Presence in initial architecture" and "Presence in target architecture" categories, with subcategories "Present" and "Absent." (One could extend this, adding categories for intermediate states in addition to the initial and final states, but I opted to keep things simple.)

The point of using content analysis to guide model construction (rather than building a model based on informal impressions, as is often done in architecture research) is that it reliably ties the model to the research data, giving us confidence that our conclusions are supported by the data. Because segmentation (i.e., identification of model elements) and coding (i.e., classification of elements) are conducted according to well-defined methods, we can be surer that, for example, an element that we identify as a connector truly represents a connector in the system as described in the data.

The category hierarchies appear in Figure 1. The full coding guide, which appears in the companion technical report [6], runs 18 pages and provides detailed inclusion criteria for each category. Figure 2 shows one of the simpler category definitions from CA1 as an example.

5.1.2. Segmentation. In content analysis, codes are not applied at will to arbitrary passages of text, as in some coding methods. Rather, the text is first divided into *coding units*, then the researcher analyzes the material segment by segment. This avoids the risk that the researcher will ignore portions of the text or devote too much attention to the passages that are most striking.

Segmentation was handled separately for CA1 and CA2. For CA1, the interview material was segmented thematically—divided into coding units so that each unit pertained to one topic or category. Figure 3 shows an interview excerpt as it was segmented and later coded.

Segmentation was more complex for CA2. The goal of CA2 was to soundly and reliably identify the architectural elements described in the data—to identify and distinguish among components, connectors, evolution operators, and so on, so that a model could be built.

Researcher: When you say you generally have an idea of where you want to go and how you want to get there—you just can't necessarily execute on it as easily as you'd like to—how do you develop that plan? [. . .] Do you have processes to help you, or is it mostly just intuition and experience?

Participant: A9⟨Mostly intuition and experience, yeah.⟩ A10⟨You look to the industry to see what's going on⟩, A11⟨but ultimately, [. . .] in this business, you tend to lean toward simplicity. The next system you build is going to last twenty years, and it needs to be maintainable [. . .], so you really don't try to get too crazy with it. We're not launching shuttles here, we're selling mayonnaise and toilet paper, so let's keep it in perspective.⟩

A9: Approaches: Experience and intuition

A10: Approaches: Industry practices

A11: Approaches: Rules of thumb and informal strategies

Figure 3: A segmented and coded passage from an interview transcript.

One challenge that this posed was how to deal with multiple mentions of the same element. Correctly identifying and coding an element would require consideration of *all* its mentions throughout the material. This stands in contrast to the piecemeal interpretative process that is more typical of content analysis, in which each segment is coded in isolation. This required the use of noncontiguous coding units; I treated multiple mentions of a single referent (e.g., a particular component) as together constituting a coding unit.

Another challenge was that not all of the "text" was textual; in addition to the interview material and written prose, there were a number of diagrams in the architectural documentation. Fortunately, this is not a real problem as far as content analysis is concerned. Content analysis has been used for decades to analyze images, multimedia, and other nontextual material. In principle, we can segment and code diagrammatic elements—boxes, lines, clouds—in much the same way that we can segment and code phrases and paragraphs.

Even so, the heterogeneity of the data created some challenges. A coding unit in CA2 is much more complex and multifaceted than a coding unit in a typical content analysis. While in a typical content analysis a coding unit is just a phrase or passage, here a coding unit is an aggregation of words, phrases, and passages (occurring in both transcribed speech and written documentation) as well as diagrammatic elements.

5.2. Evolution Model Construction

After the content analysis was finished, I constructed of an evolution model based on the output of CA2, to evaluate the applicability of our modeling approach. This involved the definition of initial and target architectures, evolution operators, evolution constraints, and evaluation functions, all directly based on the content analysis.

The subject of CA2 was the evolution of Costco's point-of-sale (POS) system. The goal of the evolution is to replace the off-the-shelf POS package at the core of the system with a more modern package. At the same time, many changes are occurring in systems with which the POS system integrates. See the companion technical report [6] for further background.

5.2.1. Initial and Target Architectures. I modeled the initial and target architectures in the Acme architecture description language [21]. Since the content analysis had already identified and classified the system elements and determined which appeared in the initial and target architectures, modeling initial and target architectures was straightforward.

Of course, the content analysis was not sufficient, on its own, to specify the initial and target architectures fully. The interviews and documentation seldom got into the specifics of attachment points and element types. Nonetheless, the big decisions about system structure had already been made. The content analysis thus succeeded in systematizing and formalizing the major decisions involved in architecture representation.

Space constraints prevent inclusion of the initial and target architectures; see instead the technical report [6].

5.2.2. Evolution Operators. Because the POS evolution is large and complex, architects mostly spoke in terms of its major stages rather than individual operations. Even so, the content analysis did identify some operations at a finer granularity—13 in total. Examples included removing a legacy element and upgrading the operating system of warehouse controllers.

Obviously, a paltry 13 operators are not alone sufficient to describe all the many changes in the POS evolution—to transform the initial architecture into the target architecture. But the operators mentioned in the data are a good basis on which to evaluate our approach. These operators are likely to be no easier to model than typical evolution operators. On the contrary, operators specifically mentioned by architects in a high-level discussion of an evolution might be more complex on average than the typical evolution operator, if anything.

I attempted to model each of the 13 operators using the specification notation introduced in our prior work [1]. The operator "removing a legacy element" was the easiest; it could be modeled as a simple deletion operator:

```
operator removeElement(e) { transformations { delete e; }}
```

Modeling other operators was more complex. For details of all 13, see the technical report [6]. However, none posed great challenges. Even those that were conceptually complex could be expressed with a specification of at most 20 lines. Though we cannot draw too broad a

conclusion from this small set, this suggests that real-world operators may often be fairly simple to specify.

5.2.3. Constraints. CA2 identified 16 constraints. As with operators, this is certainly not a complete list of constraints relevant to the evolution, but it is a good basis on which to validate the applicability of our method.

In the content analysis, I took an inclusive view of what constitutes a "constraint." I included anything amounting to a constraint on the evolution, without considering whether it was an architectural constraint or lower-level, or whether it was amenable to formalization.

Of the 16 constraints, eight were easily representable as evolution path constraints or operator preconditions. For example, a constraint that all components attached to the integration architecture should eventually use the integration architecture for communication could be represented in our temporal constraint language by

$$\Diamond\Box \mathit{intArchUsedWhenAvailable}(\mathit{system}),$$

where *intArchUsedWhenAvailable* is an architectural predicate that checks whether all participating systems communicate through the integration architecture. The companion technical report [6] shows how this predicate can be defined in the Armani constraint language.

Four were high-level constraints, not specific enough to model directly, such as a constraint that a system should have 99.999% availability.

Three were too low-level to model architecturally. For example, one architect described certain constraints on the user interface, such as how receipts are printed.

One would be representable only with significant modifications to the model, because it requires a notion of multiplicity not well supported by the model I built.

5.2.4. Evaluation Functions. CA2 identified seven dimensions of concern relevant to the evolution. Five could be modeled as evaluation functions with little trouble. For example, one dimension identified as important to the evolution was cost. In our approach, cost can be modeled as a property of operators, and then an operator analysis can simply add up all the operators within an evolution path to get an estimate of the path's total cost:

```
function analyzeCost(states, transitions) {
  var total = 0;
  transitions.forEach(function (transition) {
    transition.operators.forEach(function (operator) {
      total += operator.analysis.costInDollars; }); });
  return total;
}
```

Of the two remaining dimensions, one referred to low-level details not relevant at an architectural level.

The other was flexibility. The architects I spoke with emphasized that a key goal of the evolution was making the system flexible and open to future changes. System flexibility is difficult to estimate based on an architectural model. Most existing methods for architectural analysis of flexibility are not model-based, but instead rely on procedures such as definition of change scenarios [22] or interviews with stakeholders [23]. Such methods lie beyond the scope of our approach.

6. Findings

Section 5.2 discussed the findings of the modeling phase following CA2. We now turn to CA1. CA1 had five top-level categories (Figure 1). For space reasons, I discuss only two here. The others are covered in the companion technical report [6].

6.1. Motives for Evolution

There is a great deal of research on reasons for software changes. One of the first influential taxonomies of software change was that proposed by Swanson [24] in 1976, which identified three kinds of software maintenance: corrective, adaptive, and perfective. Swanson's taxonomy was later incorporated into ISO/IEC 14764, with the addition of a fourth type: preventive maintenance. More modern ontologies of software maintenance have been developed that refine this early work, such as that of Chapin et al. [25].

But little research has been done on motivations for *architecture* evolution specifically. Of course, many motivations for low-level software maintenance can also be motivations for architecture evolution, but there are likely important differences. Williams & Carver [26] present a taxonomy categorizing architectural changes along a number of dimensions, including a change's motivation. However, this taxonomy is based on a review of software maintenance research generally rather than work focusing on architectural change. Thus, while it purports to be a taxonomy of architectural change, it is not based on architecture research. Similarly, Jamshidi et al. [8], in their classification framework for "architecture-centric" evolution, borrow the ISO/IEC 14764 typology for their "Need for Evolution" dimension.

There is at least one empirical study that has examined causes of architecture evolution specifically. Ozkaya et al. [27], in an interview-based survey of nine software architecture projects, collected data on the causes of the evolutions that the survey participants described. Responses included new features, market change, technology obsolescence, and scalability.

Clearly, more research investigating motivations for architectural change is needed. Of course, our objective here is not to develop a general taxonomy of motivations

	Freq.
Add features	12
Improve interoperability	9
Modernize technology	5
Keep apace of business needs	3
Improve reliability	3
Improve flexibility	3
Improve performance	2

Table 1: Stated motivations for architecture evolution.

for architecture evolution. Data from a single case study would be insufficient to support such a taxonomy even were it our wish to construct one. But what this data can do is help inform us as to how architects in a real software organization think about causes of evolution.

The interviewees mentioned various motivations for evolution, which I coded in seven categories; see Table 1. Each frequency in the table is the number of coding units to which a category was applied. For other frequency measures, see the companion technical report [6].

The most frequently mentioned motive was to add a new feature. A wide range of specific features were mentioned, such as member personalization features, legal compliance features, and improved sales forecasting.

Several categories—improving flexibility, improving performance, improving reliability, improving interoperability—fall under the general heading of improving architectural qualities. By far the most prevalent was improving interoperability. One architect explained:

> We're going from a very proprietary, inflexible point-of-sale to a much more open, easily adapted point-of-sale that can talk to most systems, so that fits in with what we're trying to do overall in modernization.

In summary, most reasons for architecture evolution fell into three broad classes: evolutions motivated by a need to improve architectural qualities, evolutions motivated by new feature requirements, and evolutions driven by a desire for technology modernization.

6.2. Challenges

An interview-based study provides a unique opportunity to learn about the challenges that practitioners face. Software architecture research is often driven by researchers' beliefs about what will help architects. These beliefs are often founded on subjective impressions, generalizations from our own experiences, and informal conversations with practitioners. However, it is important to ground these beliefs with empirical data whenever possible, and one of the best ways to do so is to interview architects about the challenges they face.

"Challenges" should be understood broadly. I collected data on all kinds of challenges, not just those addressable through our approach. Encouraging architects to speak freely about the challenges they face, rather than just asking them about those which we aim to address, gives us the broadest and least biased picture of architects' needs and helps us to understand the role that approaches like ours can play in addressing a subset.

	Freq.
Communication, coordination, and integration challenges	22
Dealing with rapid change and anticipating the future	13
Business justification	9
Dealing with legacy systems	9
Scalability, reliability, and performance	8
Managing expectations and experiences of users, stakeholders	7
Managing people	7
[Six categories with freq. < 7]	26

Table 2: Stated challenges of architecture evolution.

Frequencies appear in Table 2. The commonest code was "Communication, coordination, and integration challenges." This suggests that participants view communication and coordination challenges as a critical factor in architecture evolution.

One theme that emerged from multiple participants was integration issues among simultaneously evolving systems. One architect explained:

> It's not enough to say I'm going to upgrade my point-of-sale system, which is a huge and daunting task in and of itself here at Costco. When you talk about all of the dependencies and all of the different components that are going to be interacting with point-of-sale, now you've got a unfathomable challenge in front of you, and it really takes a lot of focus to keep from getting into trouble.

After communication and coordination issues, the next most frequently occurring category was "Dealing with rapid change and anticipating the future," which captures various challenges involved in making decisions in a rapidly changing environment. The high frequency of this category suggests that this is viewed as a particularly important challenge. As one participant explained:

> You really have to have that ability to shift gears and change directions quickly, because the technology landscape changes so fast, and when you're on a project that's going to run three to five years, changes are inevitable.

The third most frequent category was "Business justification," which captures challenges in relating architectural efforts to business goals. One participant characterized executive support for architecture as the biggest issue that the industry faces with respect to architecture evolution:

> The biggest challenges are executive buy-in on the architecture function in general, because when you throw architects in at the beginning (goodness, I hope it's at the beginning) of a project, you're adding time to it [...]. We've been fortunate that we've got CIO buy-in of what we're doing. But [...] when I meet with other people who run EA and solutions architectures groups from

other companies, they always say that that's the largest challenge they face.

7. Conclusion

7.1. Answers to the Research Questions

We now return to the research questions of Section 1.

RQ1. How do practicing architects in a real-world software organization plan and reason about evolution? This was addressed by CA1, particularly the "Approaches" category, which was not covered in detail in this paper. See instead the companion technical report [6]. The most prominent category was "Communication and coordination practices." Other frequently mentioned approaches included organizational techniques, consultation of expert sources, and various informal rules of thumb. We also found that architects often reason about evolution in terms of phases, with explicit consideration of evolution alternatives. This suggests that our approach, which is based on modeling potential evolution paths in terms of discrete steps, is compatible with the way architects already think about evolution.

RQ2. What difficulties do practicing architects face in planning and carrying out evolution? This research question was also addressed by CA1, particularly the "Challenges" category. The most prominent class of challenges was "Communication, coordination, and integration challenges." This aligns neatly with the result that "Communication and coordination practices" was the most frequent subcategory of "Approaches" and reinforces the point that communication and coordination issues are extraordinarily important in managing architecture evolution. Other challenges appear in Table 2.

RQ3. How well can our modeling framework capture the concerns that arise in a real-world architecture evolution? This research question was addressed by CA2 and the model constructed based on it. As detailed in Section 5.2, the great majority of the operators, constraints, and evaluation functions identified by the content analysis could be modeled using our approach, although some were too low-level (pertaining to implementation details rather than architecture) or too high-level (encompassing a broad range of considerations that were not understood in sufficient detail to model).

7.2. Reliability and Validity

Although these results are encouraging, we must also consider issues of reliability and validity.

	% agreement	α
CA1	91.5%	0.912
CA2: "Classification"	94.7%	0.936
CA2: "Presence in initial architecture"	91.8%	0.874
CA2: "Presence in target architecture"	93.8%	0.900

Table 3: Intrarater reliability measures.

7.2.1. Reliability. The best-known treatment of reliability in content analysis is that of Krippendorff [19], who distinguishes among three kinds of reliability: *stability*, the degree to which repeated applications of the method will produce the same result; *reproducibility*, the degree to which application of the method by other analysts working under different conditions would yield the same result; and *accuracy*, the degree to which a method conforms to a standard. Accuracy is rarely relevant in content analysis, because it requires some preexisting gold standard, such as judgments by an expert panel. Such standards are rarely available. Thus, only stability and reproducibility are practically relevant.

We begin with stability. The most direct way of assessing the stability of an instrument is with a test-retest procedure, in which the instrument is reapplied to the same data and the results compared. I incorporated a test-retest procedure into my analysis. For each content analysis, I conducted a second round of coding more than two weeks after the first. (Schreier [20] recommends that at least 10–14 days elapse between successive codings by one researcher.) With two rounds of coding completed, I compared the results to evaluate intrarater agreement.

Intrarater agreement can be quantified using any of the standard metrics that are used to measure interrater agreement in studies with multiple coders. The simplest is percent agreement, but this metric is problematic because it does not account for agreement that would occur merely by chance. Various coefficients have been developed to address this defect, the most popular of which are Scott's π, Cohen's κ, and Krippendorff's α. The differences among them are irrelevant here, since all three are nearly equal for our data.

Our intrarater reliability figures appear in Table 3. For CA1, only one set of measures is shown, since each coding unit could be assigned exactly one category from the coding frame. For CA2, there is one row for each main category, because in CA2 each coding unit could be assigned one subcategory of each main category.

The reliability coefficients in Table 3 range from 0.87 to 0.94. Though there is no universal threshold for what constitutes "enough" reliability, these numbers are high by any standard (see the technical report [6] for further discussion). Stability is thus adequately demonstrated.

This leaves the other main reliability criterion: reproducibility, usually measured through interrater agree-

ment. In a single-coder study, interrater reliability cannot be assessed. Schreier [20] recommends that in a content analysis with one coder, a second round of coding may be used as a substitute for assessing interrater reliability. Similarly, Ritsert [28] writes, "In an analysis by an individual, the important possibility of intersubjective agreement as to the coding process is precluded, but the methods of 'split half' or 'test retest' can still provide an individual with information on the consistency and reliability of his judgments" (translation mine).[1]

But even in the absence of interrater agreement, reproducibility remains an important goal. Fortunately, there are other ways of getting at this quality. Steinke [29] suggests that in qualitative studies where strong notions of intersubject verifiability are not applicable, researchers should instead strive "to produce an intersubjective *comprehensibility* of the research process on the basis of which an evaluation of results can take place." She suggests that this intersubjective comprehensibility can be demonstrated in three ways. First, the research process should be thoroughly documented so that "an external public is given the opportunity to follow the investigation step by step and to evaluate the research process." Second, interpretations can be cross-checked in groups. Steinke writes that a strong form of this is peer debriefing, "where a project is discussed with colleagues who are not working on the same project." Third, the use of codified procedures contributes to intersubjectivity. These methods were used in abundance in this study.

7.2.2. Validity. The situation with validity is more muddled than that with reliability. There is a bewildering array of flavors of validity: internal validity, external validity, construct validity, content validity, face validity, social validity, criterion validity, ecological validity, and many others. In the companion technical report [6], I untangle this knot of concepts and examine the validity of this research from many angles.

For space reasons, I focus here on one type of validity that is likely to be of particular interest: external validity, or generalizability. Generalization is one of the most important aspects of the validation of a case study that seeks to have implications beyond the case it examines.

It is important to bear in mind that a case study is not generalizable in the same way that a study based on statistical sampling is generalizable. In a case study, the goal is not statistical generalizability, but instead *transferability* or *analytic generalizability*.[2] The case studied is unique, but the findings of the case study can still be applied to other contexts.

Another key point is that generalizability is not all-or-nothing. That is, the question is not *whether* a case study is generalizable, but in what respects and to which contexts it is generalizable.

In evaluating case study generalizability, we must consider what properties of the case may have influenced our findings. For example, in this study, "Dealing with legacy systems" was one of the most frequently mentioned evolution challenges. However, legacy challenges have a great deal to do with the history of a company. At companies whose software systems have a similar history—companies with complex, decades-old software systems built using now-archaic technologies—this result would be transferable. And in fact, there are many companies with such a history. But at a company with a very different history, the result might be different.

On the other hand, the top challenge that emerged was "Communication, coordination, and integration challenges." There was no a priori reason to expect such challenges to be particularly acute in this case. On the contrary, the organization studied has taken significant measures to ameliorate these difficulties, but significant challenges remain. Moreover, on a theoretical basis, we might expect that challenges in integrating systems and communicating with personnel are highly relevant to architecture evolution in general. Thus, our result on the prominence of communication and integration challenges has good analytic generalizability.

What about the results on the applicability of our modeling approach? Again, there are some aspects of the case studied that limit generalization. For example, our analysis took advantage of the fact that the POS system can be understood well from a component-and-connector architectural perspective. In a system where different architectural view types are important, the coding frame and the modeling procedures would require revision.

But the overall result—that our approach can capture the main elements of a real-world evolution—seems to be one that has a good deal of generalizability. There was no a priori theoretical reason to believe that the POS evolution would be especially easy to model with our approach. We picked that evolution because of the availability of personnel familiar with it, not because of any special properties that would render

1. There is significant disagreement on this. Krippendorff [19], for example, argues that stability "is too weak to serve as a reliability measure in content analysis."

2. There is disagreement on the handling of generalization in qualitative research [30]. Some methodologists hold that *transferability* is the proper generalizability criterion. The goal of a case study, they say, is not to discover some kind of law that is of general applicability, but rather to achieve rich understanding of a single case through thick, naturalistic description, so that findings can be transferred on a case-by-case basis to other contexts.

Others espouse some notion of theoretical generalization. Yin [31], for example, argues that a case study can serve to validate a previously developed theory, an approach he calls *analytic generalization*.

it more amenable to our approach. Thus, the main evaluative result seems to have fairly strong external validity. But of course this generalizability has limits. For example, it would be questionable to transfer this result to evolutions with special properties that we have theoretical reasons to believe might be hard to model (e.g., significant uncertainty). Ultimately, case study generalization involves a clear understanding of relevant theory, careful attention to the specifics of the case being generalized, and a good deal of judgment.

Acknowledgments. This study was carried out with financial support from the Software Engineering Institute. The views expressed herein are mine alone and do not represent the opinions of Costco Wholesale Corporation, Carnegie Mellon University, or the Software Engineering Institute. Where individuals are quoted, their comments should not be interpreted as representative of the policies or positions of their employer.

I would like to thank my thesis committee for their guidance: David Garlan, Travis Breaux, Ipek Ozkaya, and Kevin Sullivan. I am deeply grateful to Shrikant Palkar at Costco for making this study possible.

ReCal2 [32] was used to calculate reliability coefficients.

References

[1] J. M. Barnes, D. Garlan, and B. Schmerl, "Evolution styles: Foundations and models for software architecture evolution," *Softw. & Sys. Model.*, vol. 13, pp. 649–678, 2014.

[2] O. Le Goaer, D. Tamzalit *et al.*, "Evolution styles to the rescue of architectural evolution knowledge," in *Proc. SHARK'08*, pp. 31–36.

[3] M. Wermelinger and J. L. Fiadeiro, "A graph transformation approach to software architecture reconfiguration," *Sci. Comput. Program.*, vol. 44, pp. 133–155, 2002.

[4] L. Grunske, "Formalizing architectural refactorings as graph transformation systems," in *Proc. SNPD/SAWN'05*, pp. 324–329.

[5] C. E. Cuesta, E. Navarro *et al.*, "Evolution styles: Using architectural knowledge as an evolution driver," *J. Softw.: Evol. & Proc.*, vol. 25, pp. 957–980, 2013.

[6] J. M. Barnes, "Case study report: Architecture evolution at Costco," Carnegie Mellon Univ., Tech. Rep. CMU-ISR-13-116, 2013. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/isr2013/abstracts/13-116.html

[7] H. P. Breivold, I. Crnkovic, and M. Larsson, "A systematic review of software architecture evolution research," *Inform. & Software Tech.*, vol. 54, pp. 16–40, 2012.

[8] P. Jamshidi, M. Ghafari *et al.*, "A framework for classifying and comparing architecture-centric software evolution research," in *Proc. CSMR'13*, pp. 305–314.

[9] O. Le Goaer and P. Ebraert, "Evolution styles: Change patterns for software evolution," in *Proc. EVOL'07*, pp. 252–261.

[10] D. Tamzalit and T. Mens, "Guiding architectural restructuring through architectural styles," in *Proc. ECBS'10*, pp. 69–78.

[11] S.-W. Cheng, D. Garlan *et al.*, "Using architectural style as a basis for system self-repair," in *Proc. WICSA'02*, pp. 45–59.

[12] H. Fahmy and R. C. Holt, "Using graph rewriting to specify software architectural transformations," in *Proc. ASE'00*, pp. 187–196.

[13] O. Barais, E. Cariou *et al.*, "TranSAT: A framework for the specification of software architecture evolution," in *Proc. WCAT'04*, pp. 31–38.

[14] B. Spitznagel and D. Garlan, "A compositional approach for constructing connectors," in *Proc. WICSA'01*, pp. 148–157.

[15] M. Erder and P. Pureur, "Transitional architectures for enterprise evolution," *IT Pro*, vol. 8, no. 3, pp. 10–17, 2006.

[16] J. M. Barnes, "Software architecture evolution," Ph.D. dissertation, Carnegie Mellon Univ., 2013.

[17] D. Garlan, J. M. Barnes *et al.*, "Evolution styles: Foundations and tool support for software architecture evolution," in *Proc. WICSA/ECSA'09*, pp. 131–140.

[18] J. M. Barnes, "NASA's Advanced Multimission Operations System: A case study in software architecture evolution," in *Proc. QoSA'12*, pp. 3–12.

[19] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*, 3rd ed. Sage, 2013.

[20] M. Schreier, *Qualitative Content Analysis in Practice*. Sage, 2012.

[21] D. Garlan, R. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *Proc. CASCON'97*, pp. 169–183.

[22] P. Bengtsson, N. Lassing *et al.*, "Architecture-level modifiability analysis (ALMA)," *J. Syst. & Software*, vol. 69, pp. 129–147, 2004.

[23] C. Del Rosso and A. Maccari, "Assessing the architectonics of large, software-intensive systems using a knowledge-based approach," in *Proc. WICSA'07*, paper 2.

[24] E. B. Swanson, "The dimensions of maintenance," in *Proc. ICSE'76*, pp. 492–497.

[25] N. Chapin, J. E. Hale *et al.*, "Types of software evolution and software maintenance," *J. Softw. Maint. & Evol.: R. & P.*, vol. 13, pp. 3–30, 2001.

[26] B. J. Williams and J. C. Carver, "Characterizing software architecture changes: A systematic review," *Inform. & Software Tech.*, vol. 52, pp. 31–51, 2010.

[27] I. Ozkaya, P. Wallin, and J. Axelsson, "Architecture knowledge management during system evolution—observations from practitioners," in *Proc. SHARK'10*, pp. 52–59.

[28] J. Ritsert, *Inhaltsanalyse und Ideologiekritik: Ein Versuch über kritische Sozialforschung*. Athenäum, 1972.

[29] I. Steinke, "Quality criteria in qualitative research," in *A Companion to Qualitative Research*, U. Flick, E. von Kardorff, and I. Steinke, Eds. Sage, 2004, pp. 184–190.

[30] C. Seale, *The Quality of Qualitative Research*. Sage, 1999.

[31] R. K. Yin, *Case Study Research: Design and Methods*, 4th ed. Sage, 2009.

[32] D. G. Freelon, "ReCal: Intercoder reliability calculation as a web service," *Int. J. Internet Sci.*, vol. 5, pp. 20–33, 2010.