First International Workshop on Architectures for Software Systems, Seattle, April 1995

Understanding Architectural Influences and Decisions in Large-System Projects

P. C. Clements
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract: Why does a system exhibit the architecture that it does? What influences affected the architectural decisions made by its designer(s)? We hypothesize that these influences flow from the system's requirements, the organization's culture and goals, and the designers' experience and background. We posit that the influences are at least partially enumerable, as are the architectural decisions that they precipitate. This paper discusses the approach taken in a pilot study to uncover the correlation, if any, between architectural influences and architectural decisions in large-scale, software-intensive development projects. We discuss the information extracted in the context of a set of architectural case studies of systems including air traffic control, shipboard fire control, military command centers, machine controllers, database management systems, and flight simulators.

1. Introduction

The cat only grinned as Alice approached... "Cheshire-Puss," she began, rather timidly... "Would you tell me, please, which way I ought to go from here?"

"That depends a good deal on what you want to go to," said the Cat.

-- *Alice's Adventures in Wonderland, by Lewis Carroll*

1.1 Influences and decisions

Why does a system have the architecture that it has? Why do designers choose distributed object-oriented architectures instead of centralized blackboard architectures? Why use event-broadcast-based implicit invocation, rather than pipe-and-filter or direct invocation?

Clearly, an architecture is the summary result of a set of decisions. Architectural decisions include the motivation and rationale for component selection, interconnection mechanisms, coordination mechanisms, and the selection of architectural styles or idioms. Why was each decision made as it was? Why wasn't a new style, a different modularization, another interconnection mechanism chosen?

There are influences at work. An architect designing a hard-real-time system for which he believes the deadlines will be very tight will make a particular set of choices. That same architect, designing for the same system in which he believes the deadlines will be easily satisfiable, might make different choices. And the same architect, designing a non-real-time system, is likely to make quite different choices still.

This paper describes ongoing work at the Software Engineering Institute (SEI) that attempts to understand the influences at work in the selection of an architecture, the component decisions that are made to realize a particular architecture, and the relation between the influences and the decisions they precipitate. Our work is highly influenced by the vision of Garlan and Shaw, in which software architecture is at the folklore stage of developing science, and is in the process of emerging into an engineering discipline in which the experience of others is put to practical use in new applications. If we can discover correlations between architectural influences and architectural decisions, practitioners can reuse the successful decisions of others with more hope of building satisfactory software systems.

We hypothesize the following:

- The architecture that a system exhibits is a function of a set of influencing factors.
- This set of influences is at least partially enumerable.
- The architecture that a system exhibits is the summary result of a set of component decisions made by its architect(s).
- This set of decisions is at least partially enumerable.
- For an arbitrary system, a correlation exists between the enumerated influencing factors and architectural decisions.

The last hypothesis is nontrivial. Certainly any one architectural decision (e.g., how a system is decomposed into components) is influenced by many factors, including previously-made architectural decisions. We hypothesize that we can discover a correlation from influences to decisions, independent of the order in which architectural decisions are made and the influences they have on each other.

This paper describes a pilot project to extract influence and decision data from a set of large software-intensive systems, as the first step to test whether the correlation hypothesis has any validity or is in fact even testable.

1.2 The pilot project

The SEI is launching a set of case studies on large-system architectures in an effort to document and codify the engineering practices that led to successful architectures. These case studies represent real-world development efforts, typically consuming years of work, and cover a spectrum of applications, types of development organizations, and customers.

The case studies include the following:

- Initial Sector Suite System (ISSS), a million-line-of-code air-traffic control system for the Federal Aviation Administration (FAA) to process radar and flight plan data and display real-time situational displays to controllers at en-route air-traffic control centers.
- CelsiusTech’s shipboard fire-control systems, deployed on warships in several European navies, and developed as a product line with a common architecture and common reusable components.
- PRISM, a generic architecture for military command centers developed by the U. S. Army.

- GenVoca, a method of building application generators developed at the University of Texas at Austin. GenVoca has been used to produce Genesis, a system that generates product-quality, high-performance database management systems [1].
- Structural modeling, a method for developing architectures by exploiting common patterns and structures in an application domain; developed at the Software Engineering Institute, and used to build sophisticated flight simulation systems [9].

The plan is to use these case studies to understand what data of architectural significance can be extracted from development legacies. This population of case studies and their widely varying application domains, project goals, and development circumstances will certainly not be adequate or appropriate to provide evidence of any correlation from influences to decisions. However, we hope that they will provide a rich enough basis to help us understand what kind of information to extract in order to measure or assess both. If successful, the pilot studies may serve to launch a wider data-gathering effort, perhaps along the lines of [6].

2. Architectural influences

"Oh, I don't much care where ..." said Alice.

"Then it doesn't matter which way you go," said the Cat.

-- *Alice's Adventures in Wonderland, by Lewis Carroll*

Architectural decision makers are influenced by many factors. Properties of the system being built rank among the most important, certainly, but there are also factors dealing with the culture of the organization, as well as the technical expertise and experience of the architects.

2.1 Project-related influences

Architectures serve primarily as a means towards an end, that being the development of a successful system within the resource constraints provided to the development team. What does "successful" mean? In the context of the projects we are examining, it means the satisfaction of the requirements. We categorize requirements as follows:

- time-independent functional requirements. Under the four-variable model of requirements developed by Parnas and refined by van Schouwen [13], these are the requirements that can be stated as relations between input and system state on the one hand, and output value on the other, where system state does not include time. Intuitively, these are requirements that can be verified by watching the system perform in all of its states and comparing its outputs against some standard of correctness.
- performance requirements. Informally, these are the requirements that can be checked by observing the behavior of the system while holding a stop-watch. More formally, they are relations in the (state x input, output x time) relation in which state is a function of time.

- afunctional requirements. These are properties of the system that cannot be measured or inferred by watching the system execute; rather, they are measurements against a development process of some sort. These include maintainability, openness, portability, and the like.

Note that under this taxonomy, safety, security, fault-tolerance, reliability, and availability are all functional requirements, in that they are formally described by requirements for the system to produce correct output in each state of the system's operating repertoire or "competence set" [8]. Fault-tolerance, for instance, can be viewed as a matter of requiring specific output (perhaps degraded in some way) in the presence of a state representing the failure of some of the system's components. These are often referred to as *quality attributes* and lumped together with the afunctional requirements. We separate them and call them *functional quality attributes*, because although both attract the designers' attention during architecture selection, the methods used to achieve, say, maintainability are quite different from those used to achieve, say, safety.

What role does each type of requirement play in selecting an architecture? Before addressing this question, we draw an intuitive distinction between *requirements* and *driving requirements*. Driving requirements are those that the architects believe they will have difficulty satisfying;[1] we have observed that these are the ones that motivate the most interesting or far-reaching architectural decisions.

For example, a system might have very strict performance requirements, so strict that violation of any of them may result in the loss of life or property. However, if the performance requirements are easy to meet, then the architects may well choose any of several structures for their system, basing their choice on achieving other requirements that are less prone to satisfaction.

These observations lead to the following two axioms about project-related influences:

P1: The driving afunctional quality requirements are a major influence in the architecture chosen.

P2: The driving functional quality requirements are a major influence in the architecture chosen.

Most design tradeoffs pit performance against some other attribute; there is ample evidence (largely anecdotal) that when performance is a driving requirement, it precipitates many major architectural design decisions. Some performance requirements result from interacting with a system's environment and reflect the embedded nature of the system. For instance, a sensor may provide a new reading every 40 milliseconds and the polling software must be synchronized with it to avoid accuracy fall-off. Therefore, we add the following axiom:

P3: The driving performance requirements are a major influence in the architecture chosen.

1. As one wag has put it, driving requirements are the ones that drive designers up a tree.

Table 1 lists the driving requirements for each system in the case-study set, as well as requirements of somewhat less importance. We would expect to see the driving attributes have a first-order effect in the architectural decisions, and the secondary ones to show second-order effects, if any.

Table 1: The driving requirements of the case-study set.

Case study	Primary requirement	Secondary requirements
ISSS (FAA)	Ultra-high availability	Performance, safety, usability
RCS (NIST)	Safety	Performance
CelsiusTech	Product line development	Performance
GenVoca	Short time to market	Performance
PRISM	Reuse	Information security, performance
Structural modeling	Scalability, integrability	Performance

While functional requirements--i.e., achieving time-independent correctness--play a crucial role in system development (claiming sizable testing or verification resources, for example), we posit that they do not, in fact, play a major architectural role. What does it mean to have a driving functional requirement? It means that the attainment of correctness is expected to be difficult. This could be caused by any of the following:

- difficulty in specifying what outputs are required. Solving this is a specification problem, not an architectural one.
- difficulty in implementing software to achieve the desired results. For example, software that simulates the physics of sound waves traveling through an ocean during a thunderstorm and across thermal layers is difficult to get right. However, this is a problem solved by physics and mathematics, not software architecture. A prudent architectural decision may be to encapsulate problematic software in a module of its own, so that inevitable changes are isolated; however, this is not what we would consider a major or far-reaching architectural decision that offers much insight.
- difficulty in meeting accuracy constraints. Except in unusual cases where data transmission media may affect the precision of transmitted values, accuracy is a function of the accuracy of the input values, and the number and types of computations taken to arrive at the final value. While the distribution of the computations among components is an architectural issue, the structure of the end-to-end computation is more of an algorithmic issue than an architectural one. Put another way, the accuracy of a computation would be the same whether the computing steps were all in the same module or scattered across components.

Therefore:

P4: Driving functional requirements, other than those relating to functional quality attributes, are not usually a major influence in a system's architecture.

There are exceptions to this proposition. Suppose a system is distributed and is required to support a large (but varying) number of users. Suppose that each user's station can consist of from one to three consoles, each displaying different data, and that there is a computer in each console that drives that console's display. The requirement for the system to be flexibly configured in this manner is a functional requirement, and meeting it may well require architectural decisions to be made to ensure that the right programs are running in the right processors at the right time.

A more common exception is in the realm of data-intensive or shared-information applications. Fundamental architectural decisions are often made to assure timely and uniform access to consistent data in shared-information systems [11]. Therefore, we modify P4 as follows:

P4: Driving functional requirements, other than those relating to functional quality attributes, are <u>not</u> usually a major influence in a system's architecture, except in data-intensive shared-information systems.

2.2 Organization-related influences

The goals and background of the developing organization play an important role in the choice of architecture. Does the organization have "banked" components that can be (or are mandated to be) reused? Does the organization plan to develop similar systems, and thus wish to build an architecture for the entire product line? Is there technology onboard for, say, automatic testing of client-server protocols?

In this section, we enumerate a set of organizational influences that bear on architectural decision making, as evidenced by at least some of the projects in our case studies. We posit that the influences arise from the presence of any core assets on which the organization is anxious to realize a return on investment, from any organization-wide policies about system development, or from previous projects in which particular architectural decisions were (or were not) successful. The organization-related axioms are as follows:

O1: The existence of tools and/or capital infrastructure tailored to particular architectures will exert a bias towards those architectures.

O2: Organizational goals, such as a mandate to reuse existing products or a desire to evolve the developing system into a product line, will exert a major influence on the chosen architecture.

O3: An organization's development history, as evidenced by architectures of systems developed previously by that organization, will exert a secondary influence on the chosen architecture.

2.3 Architecture-related influences

It is an axiom of this work that if an architect has solved a problem successfully in the past, then he or she is likely to use the same approach to solving the next problem, especially if the two problems are similar. Similarly, if an approach failed in the past, it is less likely to be given a second chance. Thus:

A1: Architectures previously used by the project's architect(s) will exert a major influence on the chosen architecture. The influence (positive or negative) will be directly proportional to the perceived success or failure of the prior effort(s).

2.4 Enumerating the influences to extract data from projects

After hypothesizing about the nature of architectural influences, it is still necessary to determine how to extract the relevant information from a project. Supposing, for example, that a company's previous development efforts will play a role is not immediately helpful in formulating an appropriate elicitation that will help to unmask a correlation. Asking "What architectural styles have your previous development efforts tended to exhibit?" simply won't do. Fortunately, we can appeal to recent efforts to taxonomize problem and design spaces.

To test P1, P2, and P3, we can enumerate quality and performance attributes of interest by appealing to sources such as the International Organization for Standardization (ISO) standard 9126 for quality attributes [4].

To test P4, we can appeal to Shaw's inventory of shared-information architectures [11] to capture such aspects as whether the system requirements indicate batch sequential, distributed problem solving of a statically structured nature, or dynamic integration across distributed systems.

To capture organizational and individual development histories for O3 and A1, we hypothesize that ways of viewing a problem capture varieties of insight into problem solving. Shaw has catalogued a set of viewpoints drawn from published and well-known design strategies [10]; the list includes the following:

Table 2: Problem viewpoints from Shaw [10]

Functional decomposition	Data flow	Object-oriented
State machine	Event-oriented	Process control
Decision table	Data structure	

We can also appeal to Blum's taxonomy of software development methods [2] in this regard, which is reproduced in Table 3.

One of the goals of the pilot study will be to assess the adequacy of these and other catalogs and to modify and enhance them where appropriate.

Table 4 summarizes our approach to cataloging influences.

Table 3: Blum’s taxonomy of software development methods [2]

	Problem-oriented	Product-oriented
Conceptual	**I** Structured analysis Entity-relationship model Logical construction of systems Modern structured analysis Object-oriented analysis	**II** Structured design Object-oriented design
Formal	**III** PSL/PSA JSD[a] VDM[b]	**IV** Levels of abstraction Step-wise refinement Proof of correctness Data abstraction JSP[c] Object-oriented programming

a. JSD: Jackson System Development Method
b. VDM: Vienna Development Method
c. JSP: Jackson System Programming

Table 4: Extracting influences to test hypotheses

Axiom	Pertains to	Criteria for data extraction
P1 P2	Afunctional requirements Functional quality requirements	ISO Standard 9126 [4]; Driving or nondriving
P3	Performance requirements	Source: communication, computation; Driving or nondriving
P4	Functional requirements	Shaw on shared-information system architectures [11]
O1	Tools and infrastructure	Presence of tools that support a particular architectural style
O2	Organizational goals	Presence of reuse repository; desire to engineer product line
O3	Organizational history	Shaw [10] and Blum [2]
A1	Architect’s history	Shaw [10] and Blum [2]

3. Architectural decisions

"... so long as I get *somewhere" said Alice.*

"Oh, you're sure to do that," said the Cat, "if you only walk long enough."

-- Alice's Adventures in Wonderland, by Lewis Carroll

As for architectural influences, we are in a position to take advantage of the work of others in cataloging and developing taxonomies for architectural decisions. We propose that architectural decisions of interest lie primarily in the realms of component selection and component interaction, but that there are several ways to view and catalog these choices. However, there seems to be an overriding architectural decision that in many ways preempts others: Is the system to be built as one process or several?

3.1 Parallel or sequential?

An architectural decision that drives a host of others is whether or not the system is to be fielded on a uniprocessor, or whether the software will be built as a set of cooperating and communicating processes. If the latter, then a subsidiary decision is whether those processes reside on a single machine (with perhaps different processors) or on a distributed network on which interprocess communication costs will play a much more significant role.

We will ask whether the system was built as a single process or multiple processes. If the latter, we will ask whether the processes reside in the same or different processors, and whether or not a communications network is involved.

3.2 Component selection

While the nature of components to realize a system is certainly varied and domain specific, it is possible to categorize each according to a taxonomy such as one proposed by Garlan and Shaw in [12]:

Table 5: Sample component taxonomy, from [12]

Component type	Description
Pure computation	Simple input/output relations, no retained state Examples: math functions, filters, transforms
Memory	Shared collection of persistent structured data Examples: database, file system, symbol table, hypertext
Controller	Governs time sequences of others' events Examples: scheduler, synchronizer
Link	Transmits information between entities Examples: communications link, user interface

Table 5: Sample component taxonomy, from [12]

Component type	Description
Manager	State and closely related operations Examples: abstract data type, many servers

For each component in the architecture being evaluated, we can assign it a type based on this or a related taxonomy.

3.3 Interconnection/coordination mechanisms

To catalog the interconnection and coordination mechanisms, we appeal to previously published taxonomies. For multiprocess (including distributed) systems, Fernandez has produced the following catalog of coordination mechanisms [3]:

Table 6: Coordination mechanisms from [3]

Barrier	Blackboard	Bounded buffer
Broadcast	Event	General semaphore
Lock	Mailbox	Pulse
Relay	Rendezvous	Signal
Timed buffer	Transporter	

Some of these are finer grained classifications of more traditional coordination mechanisms such as shared data, message-passing, and events.

We will ask how and to what extent each of the mechanisms has been used in the project, whether or not others not listed here have played an important role, and how these choices have helped to satisfy the project's driving requirements.

3.4 Connectedness of interaction

After classifying components and how they interact, we will measure how much they interact. Is the component communication graph fully connected? Or are the interactions carefully managed and restricted? We would expect, for example, that for a system in which the quick extraction of functional subsets was an important quality attribute, the interactions would be limited and well structured.

For each component, we will classify its interactions with other components as *unrestricted*, *managed*, or *forbidden*. For the managed and forbidden ones, we will ask how and to what extent the designers felt the management of the interaction helped achieve the driving requirements.

3.5 Volume of interaction

The volume of data that flows between components is a significant architectural decision. For each component-to-component communication path, we will categorize the data flowing between them as

- none
- scalar parameters
- aggregates of data in simple structures (e.g., arrays)
- aggregates of data in complicated structures (e.g., databases)

3.6 Choice of architectural styles

Garlan and Shaw's work on architectural styles teaches us that patterns of architecture-level interaction recur across wide varieties of systems [5]. Although some styles, such as pipe and filter, may be recognizable from the style and layout of component interconnection, others, such as layered organizations, may not. Therefore, we will poll explicitly for examples of styles enumerated in [5]. For each one found, we will catalog where it occurs: across the system as a whole, or within a component or set of components identified earlier.

For each architectural style in [5], we will ask where and to what extent it was used, and the degree to which the designers felt it contributed to meeting a driving requirement.

4. Correlating influences and decisions

Although our investigation of correlations between influences and decisions is largely exploratory, there are some relationships that we expect to see between groups of influences and groups of decisions.

An architecture is usually defined to consist by and large of a system's components and the relationships and interconnections among them. However, there are many different kinds of components, interconnections, and relationships. Components may be objects, object classes, modules, programs, functions, subsystems, processes, or agents. Relationships and interconnections may be "is a sub-module of," "is an instance of," "invokes," "invokes a program on the interface of," "sends data to," "causes to execute," "communicates with," "blocks from executing," or many others. Many relations have their own diagram style associated with them (such as data-flow diagrams, call graphs, etc.), each of which shows quite a different view, possibly of the same system.

Sometimes a diagram will represent more than one view. For example, a pipe-and-filter architectural style [5] is a description of at least two relations, namely "invokes" and "sends data to." A layered architecture says nothing about which way data flows, nor about who invokes whom; rather, it is more concerned with a relation that might be best referred to as "is allowed to invoke." Programs in the same layer are free to call each other, for instance, but you can't tell if they do or not from looking at a layered diagram. Programs at one layer are allowed to invoke programs at their own or the next inner layer, but no other. Neither pipe-and-filter nor layered organization diagrams make any statement about, for example, how the programs represented by the components are gathered into modules.

The moral is that architecture diagrams tend to show quite different components and relations, and that there are a large variety of both. Each diagram represents a particular point of view and, more importantly, represents a specific set of embedded architectural decisions.

A high-level taxonomy of these decisions can be obtained by noting that some of the relations disappear in the running system, and are used only during development activities. The others are observable in the execution. For example, the notion of a particular process executing on a particular processor, and communicating with another process on perhaps a different processor, has meaning and can be checked at run-time. However, the concept of a module (a collection of programs related by their functionality or the information they share) disappears; you may be able to observe individual programs executing at run-time, but you cannot tell from what module they hail. All secrets (which modules and objects encapsulate) disappear at run-time. We call decisions whose effects are observable at run-time *dynamic* and all others *static*. This leads to our first axiom about correlation:

> ***C1: Static architectural decisions tend to affect afunctional properties; hence, driving afunctional requirements tend to motivate the static architectural decisions. There will be an observable correlation between driving afunctional requirements and static architectural decisions.***

Module structure, for instance, affects maintainability [7]. The amount of code that can be reused affects time-to-market, and so forth. We would expect to find a correlation between the driving afunctional requirements and the static architectural decisions, primarily the decomposition into components and the discipline with which components are (and are not) allowed to communicate with each other.

On the other hand, there is a correspondence between process structures and performance, between communication paths and security, and between interconnection strategies and reliability. We would expect to see a correlation between the functional- or performance-related influences and the dynamic architectural decisions. Thus:

> ***C2: Dynamic architectural decisions tend to affect performance properties; hence, driving performance requirements tend to motivate the dynamic architectural decisions. There will be an observable correlation between driving performance requirements and dynamic architectural decisions.***

5. Conclusions

We have posited that there are enumerable influences at work behind architectural decisions and have suggested high-level taxonomies for both. A pilot study is under way to see if sufficient data can be extracted from legacy developments to test the hypotheses. If so, then a larger data extraction effort will be attempted, so that sufficient data are gathered to observe correlations, if they exist. It is our hope that we can discover such correlations, and do our part to bring software architecture one step further out of the realm of folklore and into the world of disciplined engineering.

Acknowledgments

Thanks to Len Bass and Kurt Wallnau for their thoughtful reviews. The Software Engineering Institute is sponsored by the U. S. Department of Defense.

References

1. Batory, D; Singhal,V.; Thomas, J.; Dasari, S.; Geraci, B.; and Sirkin, M. “The GenVoca Model of Software-System Generators,” *IEEE Software* 11, 5 (Sept. 1994):89-94.
2. Blum, Bruce I. “A Taxonomy of Software Development Methods.” *CACM* 37,11 (Nov. 1994):82-90.
3. Fernandez, Jose. *A Taxonomy of Coordination Mechanisms Used in Real-Time Software Based on Domain Analysis* (CMU/SEI-93-TR-34). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, December 1993.
4. *ISO/IEC Standard 9126: Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use,* 1991.
5. Garlan, David and Shaw, Mary. *An Introduction to Software Architecture* (CMU/SEI-94-TR-21). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1994.
6. Kontogiannis, Kostas A. and Tilley, Scott R.“Reverse Engineering Questionnaire,” *ACM SIGSOFT Software Engineering Notes*, 19, 1 (Jan. 1994):31-38.
7. Parnas, David.“On the Criteria for Decomposing Systems into Modules,” *CACM* 15, 12 (Dec. 1972):1053-1058.
8. Parnas, David. *An Alternative Control Structure and Its Formal Definition* (IBM technical report TR FSD-81-0012). Bethesda, MD: IBM Corp., 1981.
9. Pollak, William and Rissman, Michael. “Structural models and patterned architectures,” *IEEE Computer* (August 1994): 67-68.
10. Shaw, Mary. “Making Choices: A Comparison of Styles for Software Architecture,” *Proceeding of the Second ACM SIGSOFT Symposium on the Foundations of Software Engineering,* May 1994.
11. Shaw, Mary. *Software Architectures for Shared Information Systems (*CMU/SEI-93-TR-3). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.
12. Shaw, Mary and Garlan, David. *Software Architecture: Perspectives on an Emerging Discipline,* Prentice-Hall, in publication.
13. van Schouwen, A.J. *The A-7 Requirements Model: Ex-examination for Real-Time Systems and an Application to Monitoring Systems* (technical report 90-276). Kingston, ON: Department of Computing and Information Science, Queen’s University, July 1990.