# UNDERSTANDING SOFTWARE ARCHITECTURE

**BY PANKAJ KAMTHAN**

## 1. INTRODUCTION

[…] software engineering and software design meet in software architecture.
— Rick Kazman

The software architecture is developed as the **initial step** toward designing a software system that has a collection of certain desired properties.

The architecture of a system, including a software system, is concerned with (1) how a system is organized into its constituent elements, and (2) how those elements relate to each other to achieve a given **purpose** [Sangwan, 2015, Chapter 1].

It is important to understand that **every system has an architecture**, but a **suitable architecture** is one that enables a system to achieve the **purpose** for which it was created.

### "COME FLY WITH ME"

(Source: Google Images.)

For example, consider an airplane [Avery, 2011]: simply putting together the wings, fuselage, engines, and the landing gear of an airplane is not useful unless it can make the airplane fly. Furthermore, a system is not limited to the technology; it can also include facilities, policies, and people.

For example, a **successful flight of a commercial airliner** is as much a result of the airplane and its crew, as it is of the ground crew and air traffic control.

This document explores the notion of software architecture and related concepts. In doing so, it points out to the historical significance of software architecture leading to the state-of-the-art.

## 2. UNDERSTANDING SOFTWARE ARCHITECTURE AS A DISCIPLINE

I keep six honest serving men
(They taught me all I knew);
Their names are What and Why and When
And How and Where and Who.
— Rudyard Kipling

In a number of areas, including **investigative journalism**, "the Five Ws (also known as the Five Ws (and one H)) are regarded as basics in information-gathering" [Wikipedia].

In [Vogel, Arnold, Chughtai, Kehrer, 2011], a **'pedagogical' framework** for the discipline of software architecture is given. This framework is motivated by the following **basic questions (or dimensions)**:

- What
- Why
- Who
- Where
- How
- With What

The list of basic questions **does not** include:

- When

The answers to these questions have led to a **partitioning** of the software architecture domain, as well as an **'ontology'** of the software architecture domain. This phenomenon is illustrated in Figure 1.

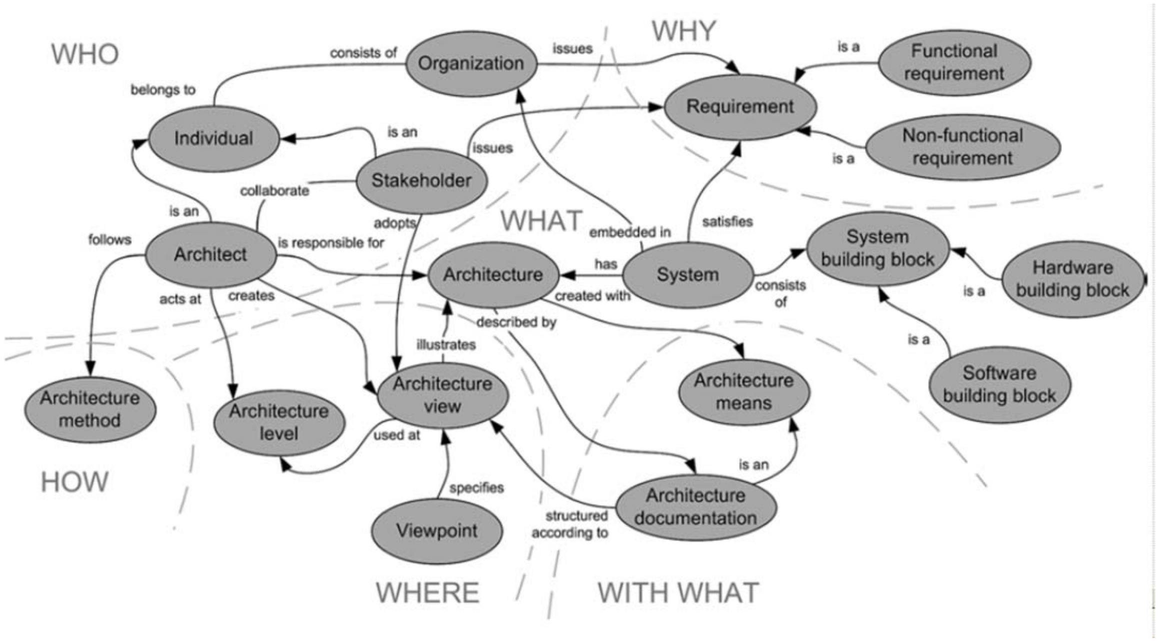


**Figure 1.** An 'ontology' of the software architecture domain. (Source: [Vogel, Arnold, Chughtai, Kehrer, 2011].)

**WHAT**

The WHAT dimension is concerned with the **definition** of software architecture, and **classification** (or types) of software architecture.

**WHY**

The WHY dimension is concerned with the requirements a software system must satisfy, in general, and the software architecture must satisfy, in particular.

The interest here is in the **identifying** requirements that are **architecturally significant**, and **designing** the software architecture that meets these requirements.

This may call for a **revisitation** and/or **reorganization** of given software requirements. Therefore, evidently, a **software architecture process** needs to be **iterative**.

**WHO**

The WHO dimension is concerned with the **roles** of the individuals and the organization, and their **interaction**, on software architecture.

The interest is here is in **human and social aspects** of software architecture. This includes identification and understanding of the **stakeholders** for software architecture.

**WHERE**

The WHERE dimension is concerned with the different levels at which architecting takes place, and the different **perspectives (or views)** from which architecture can be described.

The use of different perspectives enables one to **concentrate on one problem at a time**, a direct consequence of one of the **principles of software engineering** [Ghezzi, Jazayeri, Mandrioli, 2003], namely, **separation of concerns**.

**HOW**

The HOW dimension is concerned with the **approaches** (or methods) that can be adopted, and the **activities** that can be conducted in formulating the desired software architecture.

**WITH WHAT**

The WITH WHAT dimension is concerned with the **resources** that can be used in formulating the desired software architecture. These means can include **experiential knowledge, languages, tools, and software architecture description**.

**WHEN**

The WHEN dimension is concerned with the **software architecture 'workflow'**, that is, **time** of commencement and conclusion of work on software architecture.

The progression of work evidently depends on the **software process** being followed, and the place of design (of software architecture) in it. For example, it can vary across agile and rigid software development methodologies.

## 3. DEFINITION OF DESIGN

The notion of design goes **beyond** software development. There are a number of definitions of design, including the following.

**DEFINITION 1**

**Definition [Design] [Oxford English Dictionary].** To form a plan or scheme of, to arrange or conceive in the mind […] for **later** execution.

The following conclusions can be drawn from the previous definition of design:

- The definition is general. (It applies to any human invention that has design.)
- The definition has a notion of time.

**DEFINITION 2**

**Definition [Design] [IEEE, 1990; ISO/IEC/IEEE, 2010].** The
(1) **process** of defining the architecture, components, interfaces, and other characteristics of a system or component and
(2) **result** of [that] process.

The following conclusions can be drawn from the previous definition of design:

- The definition is specific.
- The definition views design as both noun and as verb.
- The definition suggests that design could be of only a **part** of a (software) system, not necessarily of the entire (software) system.

The separation, in space and in time, of design from implementation is emphasized by: "design (noun) is a created object, **preliminary to** and **related to** the **thing** being designed, but **distinct** from it" [Brooks, 2010].

## 4. DEFINITION OF SOFTWARE DESIGN

There are a number of definitions of software design, including the following, listed **chronologically**.

**DEFINITION 1**

**Definition [Software Design] [IEEE, 2004].** The software engineering **life cycle activity** in which **software requirements** are analyzed in order to produce a description of the software's [external and] internal structure that will serve as the basis for its **construction**.

The following conclusions can be drawn from the previous definition of software design:

- The definition is from a **process viewpoint**.
- The definition views software design as a **noun**.
- The definition focuses on structure (not behavior).
- The definition suggests that software design is an **intermediate activity** that lies between two workflows, namely **software requirements and software implementation**.

## DEFINITION 2

**Definition [Software Design] [ISO/IEC/IEEE, 2010].** The use of scientific principles, technical information, and imagination in the **definition** of a software system to perform pre-specified functions with **maximum economy and efficiency**.

The following conclusions can be drawn from the previous definition of software design:

- The definition is from a **resource viewpoint**.
- The definition **does not distinguish** between software requirements and software design.
- The definition **abstracts** the notion of software quality (specifically, optimization).

## 5. SIGNIFICANCE OF SOFTWARE DESIGN

In design [...] most of the time and effort is spent in **generating the alternatives**, which **aren't given at the outset**. [...] The idea that we start out with all the alternatives and then choose among them is wholly unrealistic.
— Herbert Simon

The following are some of the major reasons for pursuing software design:

- **Early Artifacts.** The design leads to different models that form **'blueprints'** of the solution to be implemented [Goodliffe, 2007, Chapter 14]. For example, a **skyscraper** is (cognitively and structurally) a complex structure. For a building, different types of **blueprints** are used to represent different aspects of the architecture: floor plans, elevations, electrical cabling, water pipes, central heating, ventilation, and so on.

- **Early Analysis.** The design models can be analyzed and evaluated by the project team to determine whether or not they fulfill the requirements.

- **Early Alternates.** The design models can be examined by the project team for alternative solutions and trade-offs.

The use of the common term **‘early’** above is intentional, not accidental.

## 6. TYPES OF SOFTWARE DESIGN

There are two aspects of software design:

1. **Macro-Architecture Design.** The aspect of software design that describes how software is decomposed and organized into components, and the interfaces between those components. This is **independent** of any paradigm. The focus is on components and connectors.

2. **Micro-Architecture Design.** The aspect of software design that describes the components at a level of detail that enable their construction. This is usually **dependent** on a paradigm, such as functional, object-oriented, or procedure-oriented. The focus is on algorithms and data structures.

It can be noted that software architecture is design; however, **not** all software design is software architecture [Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford, 2010, Page 6].

## 7. DEFINITION OF SOFTWARE ARCHITECTURE

**Definition [Architecture] [Lankhorst, 2013].** A collection of fundamental concepts or properties of a system in its environment, embodied in its elements, relationships, and in the principles of its design and evolution.

In other words, architecture is **‘structure with a vision’** [Lankhorst, 2013, Page 2].

There is currently **no unanimously-accepted** definition of software architecture [Garlan, Shaw, 1994]. The definitions of software architecture are **influenced** by the **history and evolution** of software architecture as a discipline.

There are a number of definitions of software architecture, including the following, collated from a number of noteworthy sources and listed chronologically. The positive aspects of a definition are denoted by [+] and positive aspects of a definition are denoted by [−]. (There are **other definitions**[1] of software architecture provided by the Software Engineering Institute (SEI) of the Carnegie Mellon University (CMU).)

## DEFINITION 1

**Definition [Software Architecture] [Buschmann, Meunier, Rohnert, Sommerlad, Stal, 1996; IEEE, 2004].** The description of the subsystems and components of a software system and the relationships between them.

## OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition is general. [+]
- The definition **does not** distinguish between software architecture and its description. [−]
- In the definition, the scope of **'them'** is unclear. [−]

## DEFINITION 2

**Definition [Software Architecture] [Booch, Rumbaugh, Jacobson, 1999].** The **set of significant decisions** about the **organization** of a software system, the selection of **structural elements** and their interfaces by which the system is composed, together with their **behavior** as specified in the **collaborations** among those elements, the **composition** of these elements into progressively larger subsystems, and the **architectural style** that guides this organization — these elements and their interfaces, their collaborations, and their composition.

## OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition is **verbose, even repetitive**. [−]
- The definition views software architecture as a collection of **decisions**. [+]
- The definition suggests that software architecture is about **organization** of a software system. [+]

[1] URL: http://www.sei.cmu.edu/architecture/start/definitions.cfm .

- The definition views software architecture as something related to **structure and, with reservations, behavior**. [+]
- The definition acknowledges the role of **experiential knowledge** (styles) and its reuse. [+]

## DEFINITION 3

**Definition [Software Architecture] [IEEE, 2000].** The fundamental organization of a system [as] embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

### OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition suggests that software architecture is about **organization** of a software system. [+]
- The definition acknowledges the relationship of the elements of software architecture to its **external environment** (namely, the **computing environment**). In other words, software architecture is **not isolated** (from its environment). [+]
- The definition acknowledges the role of experiential knowledge (principles) and its reuse. [+]

## DEFINITION 4

**Definition [Software Architecture] [Bass, Clements, Kazman, 2003].** The **structure or structures** of the [program or computing system], which comprise software elements, the **externally visible properties** of those elements, and the **relationships** among them.

### OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition is mainly concerned with the **structural aspects** of a software system. [+]
- The **'externally visible' properties** are those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. [+]

- The **‘relationships’** among elements include both **runtime and non-runtime** relationships. For example, a connector, such as a UNIX pipe, is a runtime relationship. [+]

## DEFINITION 5

**Definition [Software Architecture] [Medvidović, Dashofy, Taylor, 2007].** The set of principal design decisions about the [software] system.

### OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition assumes that there exists **at least one** design decision. [+]
- The term ‘principal’ in the definition implies that **certain** design decisions are architectural, while others are not. [+]

The **goal** of the system, and the **stakeholders**, determine whether a design decision is architectural.

For example, consider the following **design decision** [Medvidović, Dashofy, Taylor, 2007]:

**The log viewer component will check for new log entries once every second.**

There are two possibilities:

- For software systems where real-time log viewing is needed, the log refresh interval might be specified as part of the architecture.
- For other systems, it may merely be an implementation detail, and therefore not included in the description of the software system’s architecture.

## DEFINITION 6

**Definition [Software Architecture] [Chen, Li, Yi, Liu, Tang, Yang, 2010].** [The software architecture] can be described as the “blueprint” of a system at the **highest level of abstraction**.

## OBSERVATIONS

The following conclusions can be drawn from the previous definition of software architecture:

- The definition is general. [−]
- The definition **does not** distinguish between software architecture and its description. [−]
- The definition acknowledges the role of **abstraction** in software architecture. [+]

## 8. CHARACTERISTICS OF SOFTWARE ARCHITECTURE

(Source: Google Images.)

In an analogy to petals of a flower, for each definition, there some aspects that are **unique**, some aspects that **overlap** with one or more of the other definitions, and some aspects that are **common** with all the other definitions.

These definitions collectively reveal more about the following characteristics of software architecture:

- Existence
- Abstraction
- Behavior

### EXISTENCE

The major problem with intellectual capital is that it has legs and walks home every day.
— Ioana Rus and Mikael Lindvall

The previous definitions imply that **every** software system has software architecture. This is because every software system can be shown to consist of elements and the relations among those elements.

In the most **trivial case**, a software system is itself a **single element** (albeit uninteresting, but an architecture nevertheless).

The existence of software architecture for every system **does not necessarily imply** that the **architecture is known to anybody**. In other words, software architecture can be intentional or accidental [Booch, 2006].

There are several possibilities:

1. The people who designed the system are not stakeholders anymore (for example, have left the organization).
2. The documentation does not exist (or was never produced).
3. The source code has been lost (or was never delivered), and there is only executable binary code.

The previous remarks highlight the **difference** between the **architecture of a system** and the **description of** that **architecture**:

**Software Architecture ≠ Software Architecture Description**

There are **different types of software systems** [Forward, Lethbridge, 2008]. It is evident that software architecture will **vary** with respect to the **nature of software**.

For example, it should be expected that the software architecture of a distributed software system is different than that of a non-distributed software system. Indeed, there are several **classifications** of software architectures, called **reference architectures**[2].

## ABSTRACTION

There are number of **software engineering principles**, one of which is **abstraction** [Ghezzi, Jazayeri, Mandrioli, 2003].

The software architecture is an **abstraction** of a system that **suppresses** details of elements that **do not affect** how they use, are used by, relate to, or interact with other elements.

[2] A **reference architecture** in the discipline of software architecture or enterprise architecture provides a template solution for an architecture for a particular domain. For example, **Java Platform, Enterprise Edition** is a reference architecture.

The elements interact with each other by means of **interfaces**. An interface partitions details of an element into **public and private** parts. The software architecture is concerned with the **public aspect**; the private details, namely those related to internal implementation, are not architectural.

### BEHAVIOR

The behavior of each element is part of the architecture **only to the extent** that behavior can be observed or discerned from the point of view of another element.

## 9. SOFTWARE ARCHITECTURE, ANALOGY, AND METAPHOR

One of the **differences between building architecture and software architecture** is that a lot of **decisions** about a building are hard to change. It is hard to go back and change your basement, though it is possible.
— Ralph Johnson

It is generally accepted in theories of education that **understanding is necessary for learning**. It has been shown in studies on cognitive psychology that **familiarity assists understanding** of a new concept.

The use of analogies and metaphors in software engineering, as in many other disciplines, is to use **familiarity** (or similarity) as a means to facilitate **understanding**.

The discipline of software engineering is relatively new, and is **influenced** by other engineering disciplines, including **civil engineering and industrial engineering**. The influence manifests itself in form of **analogies and metaphors**.

Let X and Y be two things (either nouns or verbs).

### 9.1. ANALOGY

**Etymology:**

The word ‘analogy’ has its origins in the Greek work ‘analogia’ that means ‘proportion’.

**Explanation:**

There is analogy if it is said that **X resembles Y** in some aspect. The idea is that if X resembles Y in some way, then there is a chance that X may resemble Y in other ways as well.

**Example:**

An orange is like a sphere.

(Source: Google Images.)

It is expected that an orange has certain properties of a sphere (existence of a center, smoothness of surface, spherical symmetry, and so on).

## 9.2. METAPHOR

**Etymology:**

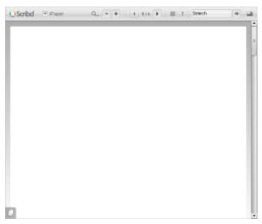
The word ‘metaphor’ has its origins in the Greek work ‘metapherein’ that means ‘to transfer’.

**Explanation:**

There is metaphor if it is said that **X is Y**. Y is used to enhance the 'meaning' associated with X, and X is expressed in terms of Y.

**Example:**

A rendering of a window[3].



[3] A window here is meant in the sense of a graphical user interface (GUI). A window is defined as an independently controllable region on the display screen, used to present objects and/or conduct a dialogue with a user.

## 9.3. ANALOGY VERSUS METAPHOR

In analogy, there is **no replacement**, only aspectual comparison. An analogy is **aspectual**, while a metaphor is **wholistic**.

## 9.4. METAPHORS FOR SOFTWARE ARCHITECTURE

There is one constant, throughout the whole (short) history of software architecture, and regardless of the formality of the approach: it is the **systematic use of metaphors** to describe architectural elements and architectures.
— Philippe Kruchten

### 9.4.1. SOFTWARE ARCHITECTURE AS A METAPHOR

In [Smolander, 2002], based on an empirical study, four metaphors of architecture in software organizations are reported:

- **Architecture as Blueprint:** [The] architecture is the structure of the system to be implemented.

- **Architecture as Literature:** [The] architecture resides in the documentation and reference architectures for future readers.

- **Architecture as Language:** [The] architecture is the language for achieving common conception about the system.

- **Architecture as Decision:** [The] architecture is the decision and basis for decisions about the system to be implemented. (This is related to one of the definitions of software architecture.)

### 9.4.2. A CLASSIFICATION OF METAPHORS FOR SOFTWARE ARCHITECTURE

**Ontological:** For example, “client”, “filter”, “layer”, “pipe”, “server”, “shopping cart”, and so on.

**Structural:** For example, “aligned with”, “on top of”, “parallel to”, and so on.

## 9.5. LIMITATIONS

There are inevitable **limits** to analogic and metaphoric perspectives of software architecture [Monson-Haefel, 2009, Pages 112 and 182].

For example, unlike building architecture, software architecture is **invisible** and **intangible**. Therefore, for essential purposes, software architecture is not affected by **physical laws** (or **laws of nature**).

For another example, building architecture and software architecture differ with respect to **legal implications**. The **national laws** may, and in some cases do, apply to software development[4]. However, unlike building architecture, municipal and provincial laws currently do **not** apply to software architecture.

In certain cases, a name, if viewed as a metaphor, may even be **misleading**. For example, a **physical table (as in furniture)** and an **electronic table (such as in publishing or relational database)** are quite different.

## 10. ARCHITECTURE EVERYWHERE

The examples of inspiring architectures to learn from and to draw upon are abound [Spinellis, Gousios, 2009].

### 10.1. BUILDING ARCHITECTURE

[A]lmost all structural failures may be attributed to **human errors**. [...] [I]n the field of structure, as in any other field of human endeavor, technological improvements alone cannot guarantee a decrease of failures and may even increase it. [It is] only a deeper **consciousness** of our human and social responsibilities can lead to the construction of **safer buildings**.
— Matthys Levy and Mario Salvadori

The field of architecture (as in civil engineering) has a long and interesting history [Hahn, 2012]. There are buildings with **spectacular architectures**[5]. The architecture of buildings has been classified into categories (or **architectural styles**) such as Ancient Egyptian, Baroque, Byzantine, Gothic, Victorian, and so on.

The architecture of buildings **evolved** based on the **needs of the inhabitants and visitors**, and the desire to make it more aesthetically pleasing.

---

[4] For example, in Canada, software is protected as a literary work under the **Copyright Act of Canada**.
[5] URL: http://www.greatbuildings.com/ .

## EXAMPLES

Software and cathedrals are much the same—first we build them, then we pray.
— Earl Everett

Figures 2-7 present examples from architecture of buildings, real or otherwise.

### Real or Possibly Real

**Figure 2.** The Towel of Babel. (A 16th Century Painting by Pieter Brueghel the Elder.) (Source: Google Images.)

**Figure 3.** The Great Pyramid of Giza. (Source: Google Images.)

The Hagia Sophia in Istanbul was built in the **sixth century**, and is an example of the **Byzantine architectural style**. It pioneered the use of structures called **pendentives** to support its enormous dome, as shown in Figure 4(a).

This design was **reused** in the **eighteenth century** to build the dome of St. Paul’s Cathedral in London, as shown in Figure 4(b).

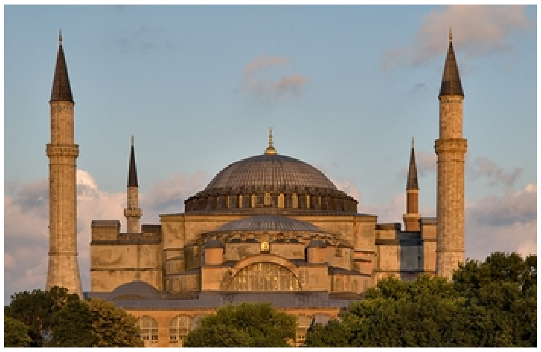


**Figure 4(a).** The Hagia Sophia Cathedral. (Source: Google Images.)

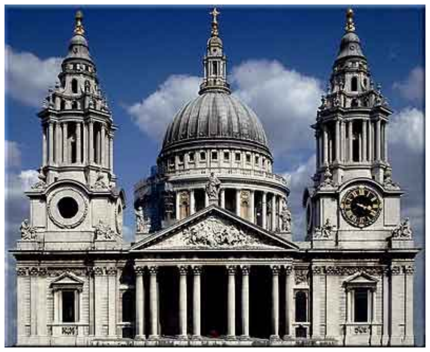


**Figure 4(b).** The St. Paul's Cathedral. (Source: Google Images.)

**Imaginary**

It does not appear that the progress in architecture has not slowed down. The spirit of invention and **imagination** lives on.

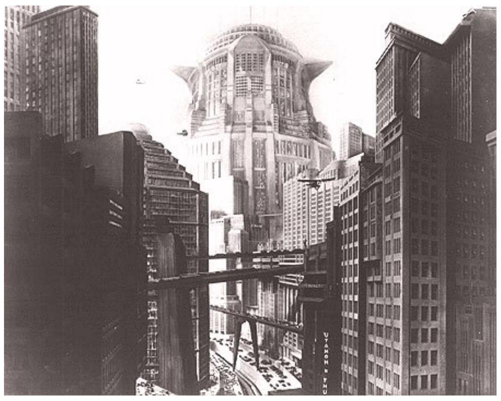


**Figure 5.** The Towel of Babel, as depicted in the year 2026, in the movie Metropolis (1927). (Source: Google Images.)

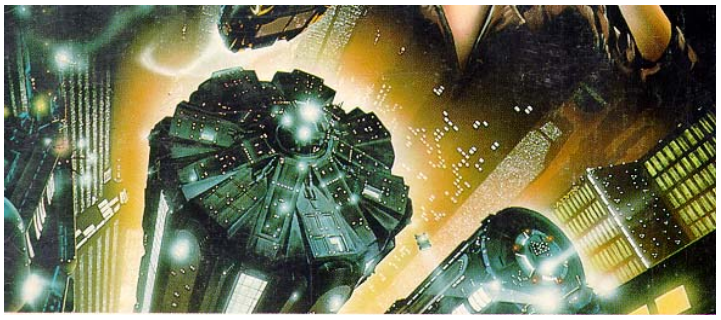


**Figure 6.** The City of Los Angeles, as depicted in the year 2019, in the movie Blade Runner (1982). (Source: Google Images.)

Figure 7 shows the **architecture** and corresponding **architectural description** of the Manchester Cathedral.

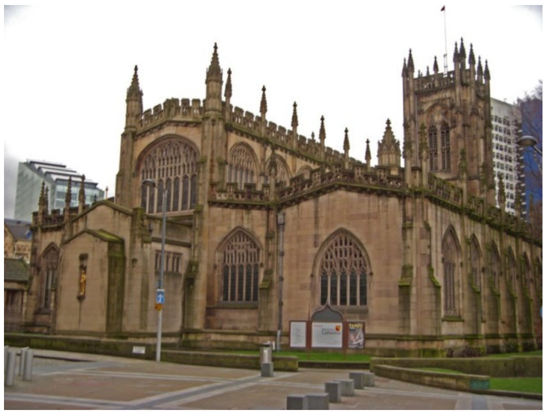


**Figure 7(a).** The Manchester Cathedral. (Source: Google Images.)

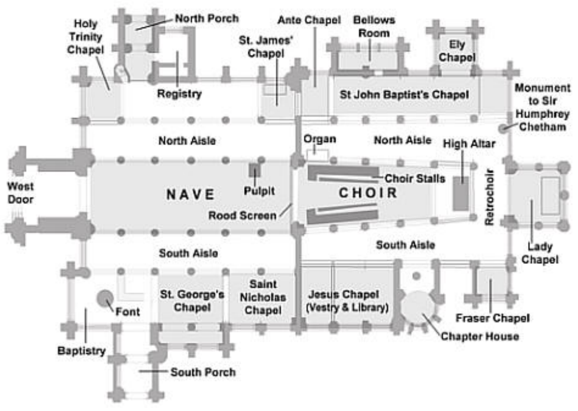


**Figure 7(b).** Plan of the Manchester Cathedral. (Source: Google Images.)

## 10.2. COMPUTER ARCHITECTURE

The discipline of **computer architecture** is influenced by building architecture.

For example, the idea of **layering** (such as hardware, firmware, assembler, kernel, operating system, and applications) in computer architecture is derived from building architecture.

## 10.3. THE IMPLICATION OF THE NOTION OF ARCHITECTURE IN OTHER REALMS ON SOFTWARE ARCHITECTURE

The discipline of software architecture is relatively new, and is influenced and inspired by the notion of architecture in other realms, especially **building architecture** and **computer architecture** [Perry, Wolf, 1992; Maier, Emery, Hilliard, 2001; Goodliffe, 2007, Chapter 14; Babar, Dingsøyr, Lago, Vliet, 2009, Section 3.6].

For example, the architecture in building engineering provides a number of **analogies** and **metaphors** for software architecture:

| Building Architecture | Software Architecture |
|---|---|
| Building Rooms and Furnishings | Software Components |
| Building Halls, Stairways, and Elevators | Software Connectors |
| Building Site | Operating System |

For example, the term **façade** as in the FAÇADE pattern [Gamma, Helm, Johnson, Vlissides, 1995] is inspired by building architecture. The notion of **pattern** has its origins in urban architecture [Alexander, Ishikawa, Silverstein, 1977; Alexander, 1979].

For another example, the terms **macro- and micro-architecture design** in software architecture have been derived from computer architecture.

## 11. SIGNIFICANCE OF SOFTWARE ARCHITECTURE

The Vasa, a formidable warship built by the Swedish Royal Navy from 1626 to 1628, sank on August 10, 1628 after sailing **less than a nautical mile**. Figure 8 shows a rendering of a portrait of Vasa.

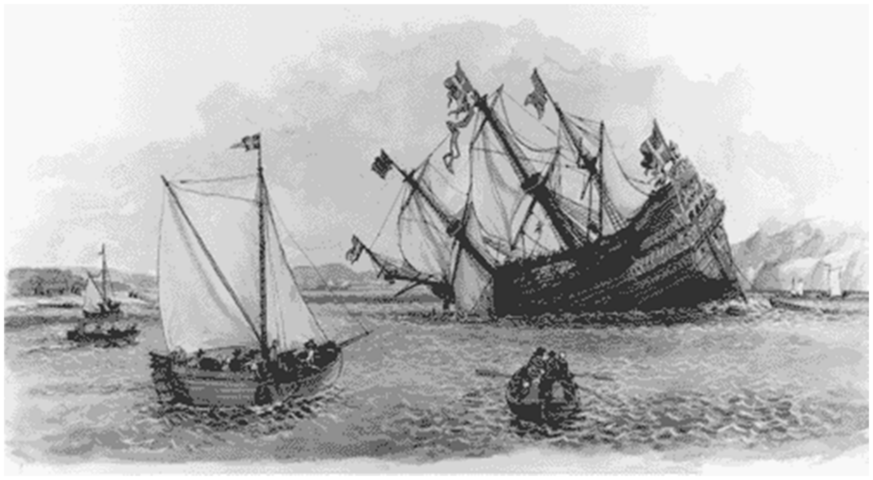


**Figure 8.** The sinking of Vasa. (Source: [Bass, Clements, Kazman, 2003].)

The reason for sinking of Vasa is believed to be flawed architectural design that followed from **poor customer management** and **poor risk management** [Bass, Clements, Kazman, 2003; Ruhe, Wohlin, 2014, Chapter 2].

The study and development of software architecture is significant for a number of reasons [Garlan, 2000; Bass, Clements, Kazman, 2003; Fairley, Willshire, 2003; Goodliffe, 2007, Chapter 14; **Qin, Xing, Zheng, 2009, Section 1.2.4**; Knodel, Naab, 2016, Section 1.1], including the following:

- Allocation of Work
- Communication among Stakeholders
- Early Design Decisions
- Constraints on Implementation
- Conceptual Reuse as Transferable Abstraction of a System
- Reduction of Risk of Software Project Failures

## 11.1. ALLOCATION OF WORK

The software architecture dictates **organizational structure**.

The highest-level decomposition of the system is typically used as the basis for the **work breakdown structure (WBS)** [Wysocki, 2009, Chapter 4].

## 11.2. COMMUNICATION AMONG STAKEHOLDERS

The software architecture represents a **common** abstraction of a system that the stakeholders can use as a **basis** for mutual understanding, as a **common ground** for negotiation and consensus, and as a vehicle for communication.

The software architecture description can be used as a **means** for **training new members** of the team. In the absence of software architecture description, it is difficult to communicate large systems.

## 11.3. EARLY DESIGN DECISIONS

The software architecture exercises considerable **managerial and intellectual control** [Kruchten, Obbink, Stafford, 2006], and therefore **responsibility**.

The software architecture manifests the **earliest design decisions** about a system [Knodel, Naab, 2016, Section 1.1]. These decisions will impact the system's remaining development, its deployment, and its maintenance life.

It is also the **earliest point** in the software process at which design decisions **governing** the system [Booch, 2007b] to be built can be analyzed.

## 11.4. CONSTRAINTS ON IMPLEMENTATION

If the software requirements form the **boundary of the problem**, then the software architecture forms the **boundary of the solution**. The software architecture is initial step in mapping the problem domain to the solution domain [Goodliffe, 2007, Chapter 14].

The software architecture imposes constraints at high-level without sacrificing the freedom of programming at low-level. For example, such constraints could be about **multiplicity, memory usage, and processing time**. (It is common for software architecture-related constraints to manifest themselves in software architecture patterns.)

These constraints permit certain things and prohibit other things. In doing so, they limit what should or should not be done during the implementation of the design. This is important for **validation.**

The **discrepancy** between software architecture description (design) and software architecture (implementation) is called **architectural erosion** [Streekmann, 2012, Section 2.5].

## 11.5. CONCEPTUAL REUSE AS TRANSFERABLE ABSTRACTION OF A SYSTEM

The software architecture constitutes a relatively small, intelligible conceptual model of the system [Albin, 2003; Fairbanks, 2010, Chapter 7]. This model is transferable across systems exhibiting **similar properties**.

It is common to consider **source code reuse**. The software architecture **increases the level of reuse**, namely from source code reuse to design reuse.

For example, **Software Product Line (SPL)** is based on conceptual reuse.

**Definition [Software Product Line[6]].** A set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

[6] URL: http://www.sei.cmu.edu/productlines/ .

A software product line consists of a family of software systems that have some **common functionality** and some **variable functionality**. To take advantage of the common functionality, reusable assets (including requirements and designs) are developed in a manner that they can be reused by different members of the family [Gomaa, 2004].

## 11.6. REDUCTION OF RISK OF SOFTWARE PROJECT FAILURES

For the sake of this document, a **risk** is a **potentially adverse**, and usually **uncertain**, event with a **positive probability** of occurrence.

There are a number of risks to a software project [Ewusi-Mensah, 2003], some of which, if not ameliorated or managed, can lead to a failure of that project. For example, **frequent, unexpected, change in technology** is such a risk.

The software architecture **contributes to reducing software project failures** emanating from such risks.

## 12. SOFTWARE ARCHITECTURE AND ITS ENVIRONMENT

The software architecture is an **outcome of influences**. The influences on software architecture come from a variety of sources. These sources may be express or implied. In particular, the software architecture is a result of influences from **social, business, and technical** environments. The existence of the software architecture, in turn, affects the social, business, and technical environments, which can subsequently influence future software architectures. This **cycle of influences**, from the environment to the architecture and back to the environment, is called the **Architecture Business Cycle (ABC)** [Bass, Clements, Kazman, 2003, Section 1.1], and is shown in Figure 9.

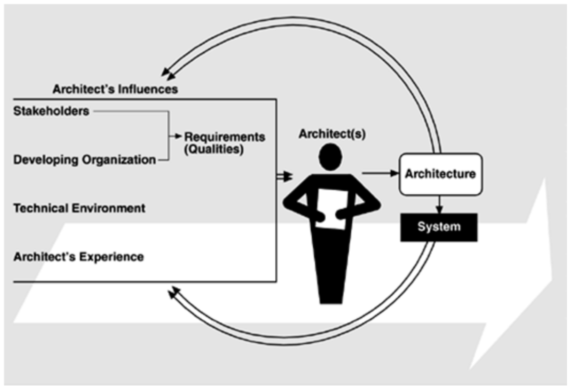


**Figure 9.** The Architecture Business Cycle. (Source: [Bass, Clements, Kazman, 2003, Section 1.1].)

The most significant source of influence on software architecture is the **organization** that is responsible for the corresponding system.

## 13. SOFTWARE ARCHITECTURE AND ORGANIZATION

There is a **symbiotic relationship** between the software architecture of a system and the organizational environment in which the system is developed and exists.

### 13.1. INFLUENCE OF THE ORGANIZATIONAL ENVIRONMENT ON THE SOFTWARE ARCHITECTURE OF A SYSTEM

There are a number of influences including the following:

- Stakeholders
- Organization
- Enterprise Architecture
- Technical Environment
- Software Architect's Knowledge

#### INFLUENCE OF STAKEHOLDERS

The software architecture is influenced by the stakeholders. This influence can manifest itself through one or more **concerns**.

For example, the **project manager** may be concerned with the low cost of the system; a **marketing manager** may be concerned with the parity of the system to competing products; and a **user** may be concerned with the security of the system.

There are different ways of eliciting concerns of the stakeholders. For instance, user concerns can be determined through interviews or by devising **personas** that express user concerns as non-functional (especially, quality) requirements [Babar, Brown, Mistrik, 2014, Chapter 4].

The **challenge** to a software architect is dealing with concerns that are **contradictory**. For example, selecting a commercial-off-the-shelf (COTS) component may be relatively easy (technically), but also relatively expensive (economically).

## INFLUENCE OF ORGANIZATION

The software architecture is influenced by the organization.

For example, the software architecture may be influenced by the development organization's **current assets**, including the software architecture of current products, which may lead to high degree of reuse.

The software architecture may also be influenced by the development organization's **structure**, including the **distribution of its employees** and its **capability to subcontract or outsource**.

Finally, the software architecture may be influenced by the development organization's **long-term business goals** [Sangwan, 2015, Chapter 2].

## CONWAY'S LAW

The **Conway's Law**[7], originally an adage circa 1968, and now considered a time-invariant, macro-level, **sociological observation**, states:

> Any organization that designs a system […] will inevitably produce a design whose structure is a copy of the organization's communication structure.

## ENTERPRISE ARCHITECTURE

**Definition [Enterprise] [Lankhorst, 2013].** A **collection of organizations** that has a common set of goals and/or a single bottom line.

**Definition [Enterprise Architecture] [Lankhorst, 2013].** A coherent whole of principles, methods, and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure.

The relationship of software architecture to other kinds of 'architecture', including **enterprise architecture** is shown in Table 1. The higher-level architectures **constrain** lower-level architectures.

---

[7] URL: http://www.melconway.com/research/committees.html .

| Abstraction Granularity | Key Design Concerns |
|---|---|
| Enterprise Architecture | • Business Processes and Models<br>• Business Data<br>• Organizational Structure and Relationships<br>• Enterprise Stakeholders<br>• IT Infrastructure |
| System Architecture | • Identification of System Context<br>• Partitioning (Hardware/Infrastructure Focus)<br>• Identification of Software Requirements<br>• Overall Systemic Functional Requirements<br>• Systemic Integration and Testing |
| Software Architecture | • Identification of Crosscutting Design Concerns (Quality Attributes)<br>• Software Functional Requirements<br>• Partitioning of Software Application(s)<br>• Software and Systemic Integration and Testing |
| Detailed Software Design | • Language Features<br>• Algorithmic Efficiencies<br>• Data Structure Design<br>• Software Application Testing<br>• Implementation of Functionality |

**Table 1.** The relative positioning of software architecture in other kinds of ‘architecture’. (Source: [Lattanze, 2008, Page 32].)

## INFLUENCE OF THE TECHNICAL ENVIRONMENT

The software architecture is influenced by the technical environment.

For example, the technical environment can include **standard industry practices** prevalent in the software engineering community.

## INFLUENCE OF SOFTWARE ARCHITECT’S KNOWLEDGE

The architecture flows from the architect’s experience and the technical environment of the day.
— Len Bass, Paul Clements, Rick Kazman

The life of a software architect is a long (and sometimes painful) **succession** of **suboptimal decisions** made partly in the dark.
— Grady Booch

The software architecture is influenced by software architect's knowledge, including education and experience.

For example, **multiple positive experiences** may lead a software architect to repeat the same method and, conversely, **multiple negative experiences** may lead a software architect to decide otherwise.

The implication of software architect's knowledge can be seen in the following. Let a set R of software requirements be given.

- To satisfy R, a software architect is likely to **design different systems for different budgets or schedules**.

- To satisfy R, a software architect is likely to **design different systems in different times** (say, 10 years apart).

## 13.2. INFLUENCE OF THE SOFTWARE ARCHITECTURE OF A SYSTEM ON THE ORGANIZATIONAL ENVIRONMENT

There are a number of influences, including the following:

- Structure
- Goals
- Future Commitments

### INFLUENCE ON STRUCTURE

The software architecture affects the structure of the development organization.

In particular, if an organization becomes adept at developing 'families' of certain systems, it tends to invest in **sustaining corresponding areas of expertise**.

For example, in a **Software Product Line (SPL)** [Pohl, Böckle, Linden, 2005], separate groups can be given responsibility for developing and maintaining individual portions of the organization's architecture for a **family of products**.

### INFLUENCE ON GOALS

The software architecture can affect the goals of the development organization [Sangwan, 2015, Chapter 2].

For example, a successful system (built on a particular software architecture style) can **enable the development organization to establish itself** in a particular market.

The **economic impact** of software architecture, especially the **value** it provides to the development organization, has been highlighted in [Booch, 2007c]. In [Hohmann, 2003], the **business perspective** of a software system's architecture is called as **marketecture**.

### INFLUENCE ON FUTURE COMMITMENTS

The software architecture can affect the development process of future, similar systems by the development organization.

For example, a development process that led to **unsuccessful software architecture** is **less likely to be chosen** for future software projects.

The software architecture can affect the requirements for the **next system** from the same customer.

For example, a customer may expect relatively **better quality** (based on learning from experience) and **lower cost** (based on reuse).

## 14. HISTORY AND EVOLUTION OF SOFTWARE ARCHITECTURE AS A DISCIPLINE

The present is the past rolled up for action, and the **past** is the present unrolled for **understanding**.
— Will Durant

It could be noted that **History** $\neq$ **{ (Event i, Date j) }**$_{i, j \geq 1}$. This is inspired by the previous quotation.

The **purpose** of studying history is to appreciate and learn from the past to build a better future.

It is said that **ideas often precede the formulation of a concept**, and the same is true for software architecture. The discipline of software architecture has a long and interesting history. Indeed, it has taken a number of decades for software architecture to be established as a discipline, **trials and tribulations** notwithstanding. The origins of software architecture can be traced back to software development in the **1960s**.

It seems that the initial use of the concept of architecture in the context of software engineering was in **'The Structure of the "THE" Multiprogramming System'** [Dijkstra, 1968][8]. It discusses the use of layers in constructing a large-scale system so that the design has clearer structure and better maintainability.

## 14.1. MATURATION OF SOFTWARE ARCHITECTURE AS A DISCIPLINE

In [Redwine, Riddle, 1985], a model for **software 'technology' maturation** is given. This model has following elements:

1. Basic Research
2. Concept Formulation
3. Development and Extension
4. Internal Enhancement and Exploration
5. External Enhancement and Exploration
6. Popularization

In a retrospective [Shaw, Clements, 2006], the **'golden age'** of software architecture is traced. It is based on the above **software technology maturation model** [Redwine, Riddle, 1985].

Figure 10 shows a time line of how software architecture matured as a discipline.

[8] THE is an early multitasking (but not multi-user) operating system. THE is the acronym for **Technische Hogeschool Eindhoven**, the then-name (in Dutch) of the Eindhoven University of Technology, where the system was developed.

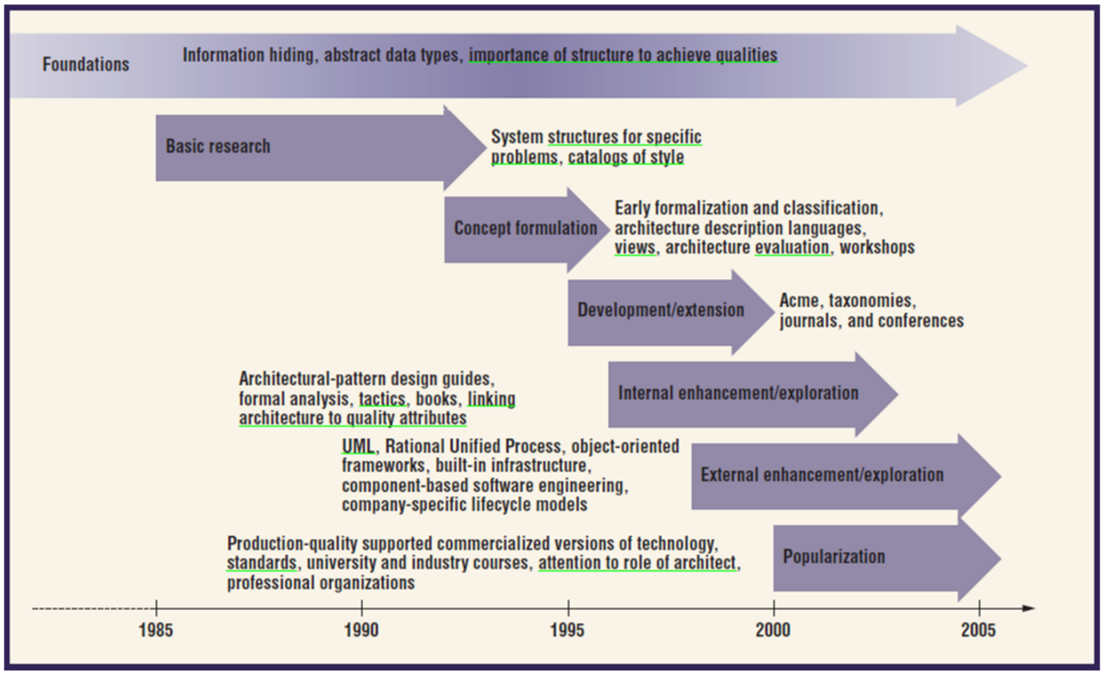

Foundations
Information hiding, abstract data types, importance of structure to achieve qualities
Basic research
System structures for specific problems, catalogs of style
Concept formulation
Early formalization and classification, architecture description languages, views, architecture evaluation, workshops
Development/extension
Acme, taxonomies, journals, and conferences
Architectural-pattern design guides, formal analysis, tactics, books, linking architecture to quality attributes
Internal enhancement/exploration
UML, Rational Unified Process, object-oriented frameworks, built-in infrastructure, component-based software engineering, company-specific lifecycle models
External enhancement/exploration
Production-quality supported commercialized versions of technology, standards, university and industry courses, attention to role of architect, professional organizations
Popularization
1985
1990
1995
2000
2005

**Figure 10.** The maturation of software architecture as a discipline. (Source: [Shaw, Clements, 2006].)

**OBSERVATIONS**

The time line **does not** indicate the work carried out in the **1970s**, that is, before the use of the label ‘software architecture’ (but after the use of the label ‘software engineering’).

**REMARKS**

The **chronology** of software architecture as a discipline has appeared in literature [Perry, Wolf, 1992; **Garlan, Shaw, 1994**; Shaw, Garlan, 1996; Garlan, 2000; Kruchten, Obbink, Stafford, 2006; Medvidović, Krikhaar, Nord, Stafford, 2006; Booch, 2007a; **Qin, Xing, Zheng, 2009, Section 1.1.3**; Chen, Li, Yi, Liu, Tang, Yang, 2010; **Babar, Brown, Mistrik, 2014, Foreword**].

The time of publication of these studies is **relevant**. For example, evidently, older literature is not expected to report newer developments in software architecture.

## 14.2. NEXT GENERATION OF SOFTWARE ARCHITECTURE AS A DISCIPLINE

The discipline of software architecture has evolved from being a **technical discipline** (concerned with structure (and to certain extent behavior), components and connectors, views, patterns, and so on) to a **socio-technical discipline** (concerned with stakeholders, decision-making, management of architectural knowledge, and so on) [Capilla, Jansen, Tang, Avgeriou, Babar, 2016].

## ACKNOWLEDGEMENT

The inclusion of images from external sources is only for non-commercial educational purposes, and their use is hereby acknowledged.

## REFERENCES

[Albin, 2003] The Art of Software Architecture: Design Methods and Techniques. By S. T. Albin. John Wiley and Sons. 2003.

[Avery, 2011] The Evolution of Flight Management Systems. By D. Avery. IEEE Software. Volume 28. Number 1. 2011. Pages 11-13.

[Babar, Brown, Mistrik, 2014] Agile Software Architecture: Aligning Agile Processes and Software Architectures. By M. A. Babar, A. W. Brown, I. Mistrik (Editors). Morgan Kaufmann. 2014.

[Babar, Dingsøyr, Lago, Vliet, 2009] Software Architecture Knowledge Management: Theory and Practice. By M. A. Babar, T. Dingsøyr, P. Lago, H. van Vliet. Springer-Verlag. 2009.

[Bass, Clements, Kazman, 2003] Software Architecture in Practice. By L. Bass, P. Clements, R. Kazman. Second Edition. Addison-Wesley. 2003.

[Booch, 2007a] It Is What It Is Because It Was What It Was. By G. Booch. IEEE Software. Volume 24. Issue 1. 2007. Pages 14-15.

[Booch, 2007b] The Irrelevance of Architecture. By G. Booch. IEEE Software. Volume 24. Issue 3. 2007. Pages 10-11.

[Booch, 2007c] The Economics of Architecture-First. By G. Booch. IEEE Software. Volume 24. Issue 5. 2007. Pages 18-20.

[Booch, Rumbaugh, Jacobson, 1999] The Unified Modeling Language User Guide. By G. Booch, J. Rumbaugh, I. Jacobson. Addison-Wesley. 1999.

[Capilla, Jansen, Tang, Avgeriou, Babar, 2016] 10 Years of Software Architecture Knowledge Management: Practice and Future. By R. Capilla, A. Jansen, A. Tang, P. Avgeriou, M. A. Babar. The Journal of Systems and Software. Volume 116. 2016. Pages 191-205.

[Chen, Li, Yi, Liu, Tang, Yang, 2010] A Ten-Year Survey of Software Architecture. By Y. Chen, X. Li, L. Yi, D. Liu, L. Tang, H. Yang. The 2010 International Conference on Software Engineering and Service Sciences (ICSESS). Beijing, China. July 16-18, 2010.

[Clements, Bachmann, Bass, Garlan, Ivers, Little, Merson, Nord, Stafford, 2010] Documenting Software Architectures: Views and Beyond. By P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford. Second Edition. Addison-Wesley. 2010.

[Dijkstra, 1968] The Structure of "THE"-Multiprogramming System. By E. W. Dijkstra. Communications of the ACM. Volume 11. Issue 5. 1968. Pages 341-346.
[Ewusi-Mensah, 2003] Software Development Failures: Anatomy of Abandoned Projects. By K. Ewusi-Mensah. The MIT Press. 2003.

[Fairley, Willshire, 2003] Why the Vasa Sank: 10 Problems and Some Antidotes for Software Projects. By R. E. Fairley, M. J. Willshire. IEEE Software. Volume 20. Issue 2. 2003. Page 18-25.

[Garlan, 2000] Software Architecture: A Roadmap. By D. Garlan. The Twenty Second International Conference on Software Engineering (ICSE 2000). Limerick, Ireland. June 4-11, 2000.

[Garlan, Shaw, 1994] An Introduction to Software Architecture. By D. Garlan, M. Shaw. Technical Report CMU-CS-94-166. Carnegie Mellon University. Pittsburgh, U.S.A. 1994.

[Ghezzi, Jazayeri, Mandrioli, 2003] Fundamentals of Software Engineering. By C. Ghezzi, M. Jazayeri, D. Mandrioli. Second Edition. Prentice-Hall. 2003.

[Gomaa, 2004] Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. By H. Gomaa. Addison-Wesley. 2004.

[Goodliffe, 2007] Code Craft: The Practice of Writing Excellent Code. By P. Goodliffe. No Starch Press. 2007.

[Hahn, 2012] Mathematical Excursions to the World’s Great Buildings. By A. J. Hahn. Princeton University Press. 2012.

[Hohmann, 2003] Beyond Software Architecture: Creating and Sustaining Winning Solutions. By L. Hohmann. Addison-Wesley. 2003.

[IEEE, 1990] IEEE Standard 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society. 1990.

[IEEE, 2000] IEEE Standard 1471-2000. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Computer Society. 2000.

[ISO/IEC/IEEE, 2010] ISO/IEC/IEEE 24765:2010. Systems and Software Engineering -- Vocabulary. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC)/IEEE Computer Society. 2010.

[Knodel, Naab, 2016] Pragmatic Evaluation of Software Architectures. By J. Knodel, M. Naab. Springer International Publishing. 2016.

[Kruchten, Obbink, Stafford, 2006] The Past, Present, and Future for Software Architecture. By P. Kruchten, H. Obbink, J. Stafford. IEEE Software. Volume 23. Issue 2. 2006. Pages 22-30.

[Lattanze, 2008] Architecting Software Intensive Systems: A Practitioners Guide. By A. J. Lattanze. Auerbach Publications. 2008.

[Lankhorst, 2013] Enterprise Architecture at Work: Modelling, Communication and Analysis. By M. Lankhorst. Third Edition. Springer-Verlag. 2013.

[Maier, Emery, Hilliard, 2001] Software Architecture: Introducing IEEE Standard 1471. By M. W. Maier, D. Emery, R. Hilliard. Computer. Volume 34. Issue 4. 2001. Pages 107-109.

[Medvidović, Dashofy, Taylor, 2007] Moving Architectural Description from Under the Technology Lamppost. By N. Medvidović, E. M. Dashofy, R. N. Taylor. Information and Software Technology. Volume 49. Issue 1. 2007. Pages 12-31.

[Medvidović, Krikhaar, Nord, Stafford, 2006] Understanding the Past, Improving the Present, and Mapping Out the Future of Software Architecture. By N. Medvidović, R. L. Krikhaar, R. L. Nord, J. A. Stafford. The Journal of Systems and Software. Volume 79. Number 1. 2006. Pages 1790-1791.

[Monson-Haefel, 2009] 97 Things Every Software Architect Should Know: Collective Wisdom from the Experts. By R. Monson-Haefel (Editor). O'Reilly Media. 2009.

[Perry, Wolf, 1992] Foundations for the Study of Software Architecture. By D. E. Perry, A. L. Wolf. ACM SIGSOFT Software Engineering Notes. Volume 17. Number 4. 1992. Page 40-52.

[Pohl, Böckle, Linden, 2005] Software Product Line Engineering: Foundations, Principles, and Techniques. By K. Pohl, G. Böckle, F. van der Linden. Springer-Verlag. 2005.

[Qin, Xing, Zheng, 2009] Software Architecture. By Z. Qin, J. Xing, X. Zheng. Springer-Verlag. 2009.

[Redwine, Riddle, 1985] Software Technology Maturation. By S. T. Redwine, Jr., W. E. Riddle. The Eighth International Conference on Software Engineering (ICSE 1985). London, U.K. August 28-30, 1985.

[Ruhe, Wohlin, 2014] Software Project Management in a Changing World. By G. Ruhe, C. Wohlin (Editors). Springer-Verlag. 2014.

[Sangwan, 2015] Software and Systems Architecture in Action. By R. S. Sangwan. CRC Press. 2015.

[Shaw, Clements, 2006] The Golden Age of Software Architecture. By M. Shaw, P. Clements. IEEE Software. Volume 23. Issue 2. 2006. Pages 31-39.

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

[Smolander, 2002] Four Metaphors of Architecture in Software Organizations: Finding out The Meaning of Architecture in Practice. By K. Smolander. The 2002 International Symposium on Empirical Software Engineering (ISESE 2002). Nara, Japan. October 3-4, 2002.

[Spinellis, Gousios, 2009] Beautiful Architecture. By D. Spinellis, G. Gousios. O'Reilly Media. 2009.

[Streekmann, 2012] Clustering-Based Support for Software Architecture Restructuring. By N. Streekmann. Springer Fachmedien. 2012.

[Vogel, Arnold, Chughtai, Kehrer, 2011] Software Architecture: A Comprehensive Framework and Guide for Practitioners. By O. Vogel, I. Arnold, A. Chughtai, T. Kehrer. Springer-Verlag. 2011.

[Wysocki, 2009] Effective Project Management: Traditional, Agile, Extreme. By R. K. Wysocki. Fifth Edition. John Wiley and Sons. 2009.