University of St Andrews

CS5033 Software Architecture

Revision notes

May 3, 2019

Lecturer:
Dharini Balasubramaniam

Submitted By:
140011146

Contents

1 Introduction **1**
1.1 Analogy to architecture of buildings 1
1.2 Definitions 2
1.3 Fundamental concepts 2

2 Software architecture styles **4**
2.1 Style selection 6

3 Results **7**

4 Conclusion and evaluation **7**

1 Introduction

1.1 Analogy to architecture of buildings

There is a strong analogy between the architecture of software systems and the architecture of buildings. Requirements are gathered, a design is created to satisfy the requirements, the design is refined to create blueprints and the building is constructed following the blueprints, to be used and occupied. In software systems, requirements are specified, a high-level design is created, code is written to implement the design and the system is deployed and used. This analogy offers several insights.

1. The concept of a building having an architecture that is a separate form which is liked to the physical structure itself. The architecture during the design can be compared to the architecture of the physical structure after construction. In a software system, that architecture should also exist in an independent form, linked to the code that implements the architecture.
2. The properties of structures are induced by the design of their architectures, for example a castle with thick walls is designed to have defensive properties against attacks. In a software system, properties such as resilience against security attacks are determined by the design of the architecture.
3. There is a distinctive role and character of the architect, separate from that of the construction workers. In the analogy, this means that skill in programming is not sufficient for the creation and conception of complex software systems.
4. The process is not as important as the architecture. Simply following a standard process will not guarantee that a successful building will be created which meets the original requirements. Architects and engineers must keep its design and qualities in mind and a process is only present to serve those ends, not be an end in itself.
5. Architecture develops from existing knowledge and previous work. New buildings are not designed from first principles and re-discovery of materials.

We also see the goals of building architecture and mostly similar to those of software architecture.

- Good quality
- On time
- On budget
- Satisfies customer requirements

1.2 Definitions

There are many different definitions of software architecture:
Definition 1. ***Software architecture*** *is the fundamental organisation of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution. [IEEE 1471 - 2000]*
Definition 2. ***Software architecture*** *is the structure or structures of the system, which comprise software elements, the externally visibly properties of these elements, and the relationships among them. [Bass, Clements and Kazman]*
Definition 3. ***Software architecture*** *comprises of descriptions of elements from which systems are built, interactions among these elements, pattern that guide their composition, and constraints on these patterns. [Shaw and Garlan]*

The main motivation for software architecture is to have a high-level specification of the system which captures important properties and can be used as a basis for development and management activities. A software architecture also allows the design to be verified or analysed, either manually or automatically. Software architecture can serve as a basis for a variety of areas:

- Recognition of common problem classes
- Separation of concerns and complexity management
- Satisfying requirements (functional and non-functional)
- Development focus
- Stakeholder communication
- System design and implementation
- Maintenance and evolution
- Reuse
- Cost estimation
- Project and process management

1.3 Fundamental concepts

There are two aspects to software architecture that key concepts fall under: the system and the stakeholders.

1. System
    - Components
    - Connectors (interactions)
    - Configurations
    - Properties
    - Styles/patterns
    - Rationale
    - Interfaces
2. Stakeholders
    - Views and viewpoints
    - Requirements (functional and non-functional)
    - Rationale (again)

- Priorities
- Risks
- Tradeoffs

1.3.1 Component

Definition 4. *A software component is an architectural entity that (1) encapsulates a subset of the system's functionality and/or data, (2) restricts access to that subset via an explicity defined interface, and (3) has explicitly defined dependencies on its required execution context.*

Components are the building block of architectures. These are typically units of computation or a data store (processing and data elements). They encapsulate the processing and data in a system's architecture. Examples are things like a server, database, client, etc. The extent of the context captured by a component can include:

- The components required **interface** to services provided by other components in a system that this components depends on to operate.
- The availability of specific resources, such as data files that the components relies on
- The required software environment, such as programming language, middleware and operating system
- The hardware configurations needed to execute the component

An important aspect to software components is making sure they are reusable, as a software architecture comprises of many components with possible overlap. This has implications for how the problem solve, how they are distributed and how they are allocated. A suitable decomposition strategy is critical to components of a software architecture.

1.3.2 Connector

Definition 5. *A software connector is an architectural elements tasked with effecting and regulating interactions among components.*

Connectors are the elements that deal with the interaction among a system's components. They model the rules that govern the interactions, with functionality include

- Communication
- Coordination
- Conversion
- Facilitation

Examples of connectors include procedure calls, shared memory, pipes, synchronous message passing, etc. They can be either simple or semantically very rich. Due to a shift towards microservice architectures, connectors are increasingly important as the facilitate the interaction between the microservices.

Note that while components provide application-specific services, connectors are typically application-independent. The characteristics of a procedure call, distributor, adaptor, etc. are independent of the components they service.

1.3.3 Configuration

Definition 6. *An architectural configuration is a set of specific associations between the components and connectors of a software architecture.*

Configurations are used to define the topology of the system. A software system can be viewed as a connected graph of components and edges and the configuration defines this graph. The configuration can capture aspects such as concurrency and distribution. Some composite elements may already have a set configuration. The aim of having configurations is to minimise dependencies.

1.3.4 Architectural styles and patterns

Definition 7. *An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.*

Architectural styles are reusable architectural templates. The idea is that many existing and different systems may all have similar design choices. These solutions have been evolved over time and carried on to be more elegant, efficient, scalable, etc. compared to creating a new architecture from scratch. Multiple of these patterns can be combined in a single system. Each pattern implies the existence of some components and interactions, for example a client-server implies a single server and multiple client components.

1.3.5 Rationale

The rationale is an important aspect to the architecture, which is about why the system should be designed. This is linked heavily to customer requirements for an architecture, or the various intent and assumptions during the design process.

1.3.6 Properties

Properties of a software architecture are the non-functional requirements and quality attributes that define the performance or other requirements of the system. Ideally, these properties are quantified and verifiable, for example throughput in messages per second.

2 Software architecture styles

There are a number of different domains and problems which have solutions with similar structure and properties. Architecture styles capture the experience and learning from previous systems into a reusable design for new systems. There are a few advantages to using architecture styles

- Reduced development time and cost - Less time is spent designing and analysing the system because there are inherent properties to some architectural styles
- Improved system quality - Again, the properties of the chosen architectural style will guarantee some system properties, which helps evaluate system quality on a high-level

The notion of architectural style is useful from both descriptive and prescriptive points of view. **Descriptively**, architectural style defines a particular codification of design elements and formal arrangements. Prescriptively, style limits the kinds of design elements and their formal arrangements. So the style can constrain both the design elements and the relationships between the design elements.

There is also a differentiation that should be made between **design** patterns and **architectural patterns**. Design patterns provide a template for subsystems or components of a system, but not a template for the entire system itself. This is a difference in the scale of each type of pattern. Design patterns do not influence the fundamental structure of a software system and only affect a single subsystem. In other words, it can be said that design patterns help implement architectural patterns.

Example styles:

- **Layered**
  - Virtual machines
  - Client-server
- **Dataflow styles**
  - Batch-sequential
  - Pipe-and-filter
- **Shared memory**
  - Blackboard
  - Rule-based
- **Implicit invocation**
  - Publish-subscribe
  - Event-based

Note that a style or pattern does not entirely define an architecture, but rather a family of architectures which obey a common set of constraints. Most architectures are heterogeneous, which uses a combination of patterns, models and technologies.

2.1 Style selection

Style name	*Summary*	*Use it when...*	*Avoid it when...*
Language-influenced styles			
Main program and subroutines	Main program controls program execution, calling multiple subroutines	...application is small and simple	...complex data structures are needed ...future modifications are likely
Object-oriented	Objects encapsulate state and accessing functions	...close mapping between external entities and internal objects is sensible ...many complex and interrelated data structures	...strong independence between components necessary, very high performance required
Layered			
Virtual machines	Virtual machine offers services to the layers above it	...many applications can be based upon a single, common layer of services ...interface service specification resilient when implementation of a layer must change	...many levels are required (causes inefficiency) ... data structures must be accessed from multiple layers
Client-server	Clients request service from a server	...centralised computation and data at a single location (the server) promotes manageability and scalability ...end-user processing limited to data entry and presentation	...centrality presents a single point of failure risk ...network bandwidth limited ...client machine capabilities rival or exceed the server's
Dataflow styles			
Batch-sequential	Separate programs executed sequentially with batched input	...problem easily formulated as a set of sequential, severable steps	...interactivity or concurrency between components necessary or desirable ...random access to data required
Pipe and filter	Separate programs (filters) executed, potentially concurrently. Pipes route data streams between filters	...filters are useful in more than one application ...data structures easily serialisable	...interaction between components required

Style name	*Summary*	*Use it when...*	*Avoid it when...*
Shared memory			
Blackboard	Independent programs access and communicate exclusively through a global repository (the blackboard)	...all calculations center on a common, changing data structure ...order of processing dynamically determined and data-driven	...programs deal with independent parts of the common data ...interface to common data susceptible to change ... interactions between the independent programs require complex regulation
Rule-based	Use rules in a knowledge base to resolve queries	...problem data and queries expressible as simple rules over which inference may be performed.	...number of rules is large ...interaction between rules is present ...high-performance required
Implicit Invocation			
Publish-subscribe	Publishers broadcast messages to subscribers	...components are very loosely coupled ...subscription data is small and efficiently transported	...middleware to support high-volume data is unavailable
Event-based	Independent components asynchronously emit and receive events communicated over event buses	...components are concurrent and independent ...components are heterogeneous and network-distributed	...guarantees on real-time processing of events is required

The selection of which style/pattern do use depends on a number of factors:

- Types of components/connects used in the pattern
- Mechanisms by which control is shared, allocated and transferred among the components
- Ways of communicating data
- Interaction of data and control
- Invariants defined

3 Results

4 Conclusion and evaluation