Review

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Architecture

Software Architecture

innovate achieve lead

- A software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software design activity.

componen

- A Component is an encapsulated part of a software system. A component has an interface. Components serve as the building blocks for the structure of a system. At a programming-language level, components may be represented as modules, classes, objects or a set of related functions.

Categorisation of Components

innovate achieve lead

D.E. Perry, A.L. Wolf

- Processing elements
- Data elements
- Connecting elements

Object Oriented Paradigm

- Controller components
- Coordinator components
- Interface components
- Service provider components
- Information holder components
- Structuring components

Relationship

- A relationship denotes a connection between components.
- A relationship may be static or dynamic.
- Static relationships show directly in source code.
- They deal with the placement of components within an architecture.
- Dynamic relationships deal with temporal connections and dynamic Interaction between components.
- They may not be easily visible from the static structure of source code.

View

- A view represents a partial aspect of a software architecture that shows specific properties of a software system

Views:

D. Sonl. R. Nord, C. Hofmeister:

- Conceptual architecture: components, connectors.. ..
- Module architecture: subsystems, modules, exports, imports..,.
- Code architecture: files, directories. libraries, includes ...
- Execution architecture: tasks, threads, processes ...

P.B. Kruchten: The 4 + 1 View Model of Architecture, IEEE Software. November 1995

- Logical view: the design's object model, or a corresponding model such as an entity relationship diagram.
- Process view: concurrency and synchronization aspects.
- Physical view: the mapping of the software onto the hardware and its distributed aspects.
- Development view: the software's static organization in its development environment.
- Scenarios(+1): We use a small subset of important scenarios - instances of use cases - to show that the elements of the four views work together seamlessly.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Patterns

Pattern System

- A pattern system for software architecture is a collection of patterns for software architecture, together with guidelines for their implemen- tation. combination and practical use in software development.

Pattern System Requirements

- It should comprise a sufficient base of patterns.
- It should describe all its patterns uniformly.
- It should expose the various relationships between patterns.
- It should organize its constituent patterns.
- It should support the construction of software systems.
- It should support its own evolution

Pattern Classification

- It should be simple and easy to learn, rather than complex, hard to understand, and use.
- It should consist of only a few classification criteria, rather than of a multi-dimensional pattern space that organizes patterns according to every theoretically-possible pattern property.
- Each classification criterion should reflect natural properties of patterns, for example the kinds of problems the patterns address, rather than artificial criteria such as whether patterns belong to a pattern language or not.
- It should provide a 'roadmap' that leads users to a set of potentially-applicable patterns, rather than a rigid 'drawer-like' schema that tries to support finding the one 'correct' pattern.
- The schema should be open to the integration of new patterns without the need for refactoring the existing classification.

Pattern Categories

- Architectural patterns can be used at the beginning of coarse-grained design, when specifying the fundamental structure of an application.
- Design patterns are applicable towards the end of coarse-grained design, when refining and extending the fundamental architecture of a software system, for example deciding on the basic communication mechanisms between subsystems. Design patterns are also applicable in the detailed design stage for specifying local design aspects, such as the required support for multiple implementations of a component.
- Idioms are used in the implementation phase to transform a software architecture into a program written in a specific language.

Problem Categories

- From Mud to Structure includes patterns that support a suitable decomposition of an overall system task into cooperating subtasks.
- Distributed Systems includes patterns that provide infrastructures for systems that have components located in different processes or in several subsystems and components.
- Interactive Systems includes patterns that help to structure systems with human-computer interaction.
- Adaptable Systems includes patterns that provide infrastructures for the extension and adaptation of applications in response to evolving and changing functional requirements.

Pattern Selection

1. Specify the problem.
2. Select the pattern category that corresponds to the design activity you are performing.
3. Select the problem category that corresponds to the general nature of the design problem.
4. Compare the problem descriptions.
5. Compare benefits and liabilities.
6. Select the variant that best implements the solution to your design problem
7. Select an alternative problem category.

Properties of Software Systems

- A functional property deals with a particular aspect of a system's functionality, and is usually related to a specified functional requirement. A functional property may either be made directly visible to users of an application by means of a particular function, or it may represent aspects of its implementation, such as the algorithm used to compute the function.
- A non-functional property denotes a feature of a system that is not covered, by its functional description. A non-functional property typically addresses aspects related to the reliability, compatibility, cost, ease of use, maintenance or development of a software system.

Software Design

- Software design is the activity performed by a software developer that results in the software architecture of a system. It is concerned with specifying the components of a software system and the relationships between them within given functional and non-functional properties.
- Conventional wisdom has been to use terms like 'software architecture', 'software architectural design', or 'coarse-grained design' for the high-level structural subdivision of the system, and 'design' or 'detailed design' for more detailed planning.
- We denote the whole activity of constructing a software system as 'software design' and the resulting artefacts as 'software architecture'.

Patterns

- Our patterns build on the immense practical experience in software development gathered by designers and programmers over the last three to four decades.
- None of the patterns we describe is artificially constructed, neither by us nor by anyone else-they evolved over time.
- Software developers recognized that particular solutions solved a problem better than others, and so they reused these solutions again and again.
- Some of the patterns we describe have existed for a long time. For example, the Pipes and Filters pattern has been known since the 1960s, and the Model-View-Controller pattern since the late 1970's. Without this practical experience, no patterns would exist.
- Patterns also build explicitly on the many principles that have been developed for structured programming-patterns are not dedicated solely to object technology

Architectural Style

- An architectural style defines a family of software systems in terms of their structural organization. An architectural style expresses com- ponents and the relationships between them, with the constraints of their application, and the associated composition and design rules for their construction.
  - An architectural style expresses a particular kind of fundamental structure for a software system together with an associated method that specifies how to construct it.
  - An architectural style also comprises information about when to use the architecture it describes, its invariants and specializations, as well as the consequences of its application.

Framework

- A framework is a partially complete software (sub-) system that is intended to be instantiated. It defines the architecture for a family of (sub-) systems and provides the basic building blocks to create them. It also defines the places where adaptations for specific functionality should be made. In an object-oriented environment a framework consists of abstract and concrete classes.
- A framework for applications in a specific domain is called an application framework.
  - Frozen spots define the overall architecture of a software system-its basic components and the relationships These remain unchanged in any instantiation of the application framework.
  - Hot spots represent those parts of the application framework that are specific to individual software systems. Hot spots are designed to be generic- they can be adapted to the needs of the application under development.

Enabling Techniques for Software Architecture:

innovate achieve lead

- Abstraction
- Encapsulation
- Information Hiding
- Modularization
- Separation of Concerns
- Coupling and Cohesion
- Sufficiency, Completeness and Primitiveness
- Separation of Policy and implementation
- Separation of Interface and Implementation
- Single Point of Reference
- Divide-and-Conquer

Patterns

- They explicitly build on enabling techniques for constructing well-defined software systems, such as
  - information hiding and
  - the separation of interface and implementation.
- They stress the importance of non-functional properties, such as
  - Usability, Modifiability, Performance, Testability, Security, Interoperability and Reliability.
- They complement existing problem-independent software development processes and methods with guidelines for solving specific recurring design and implementation problems.

Benefits of Patterns

- They help with the recognition of common paradigms, so that high-level relationships between software systems can be understood and new applications built as variations on old systems.
- They provide support for finding an appropriate architecture for the software system under development.
- They provide support for making principled choices among design alternatives.
- They help with the analysis and description of high-level properties of complex software systems. They provide support for change and evolution of software systems.