BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Architectures

SECLZG651/SSCLZG653

Parthasarathy

innovate achieve lead

BITS Pilani

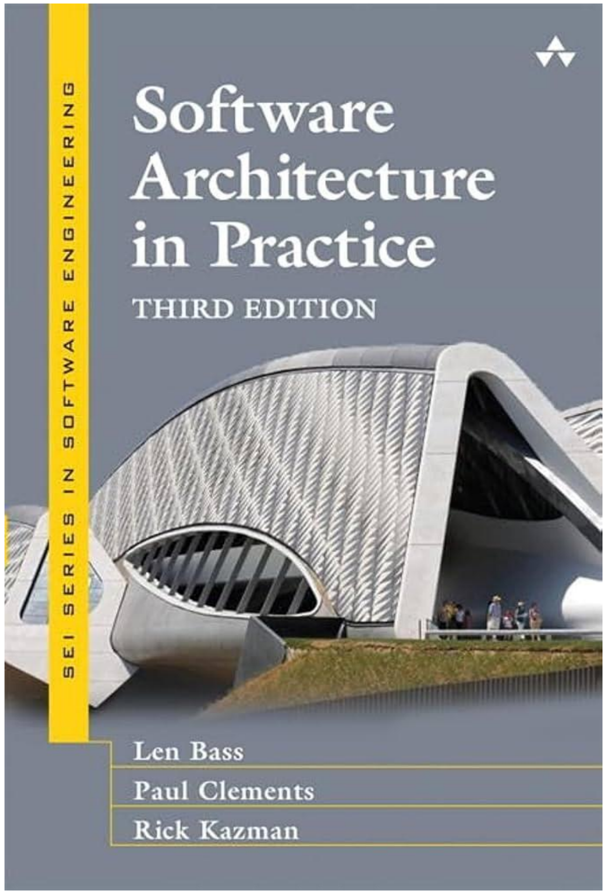
Pilani|Dubai|Goa|Hyderabad

SECLZG651/SSCLZG653 – CS#1 Introduction to Software Architectures

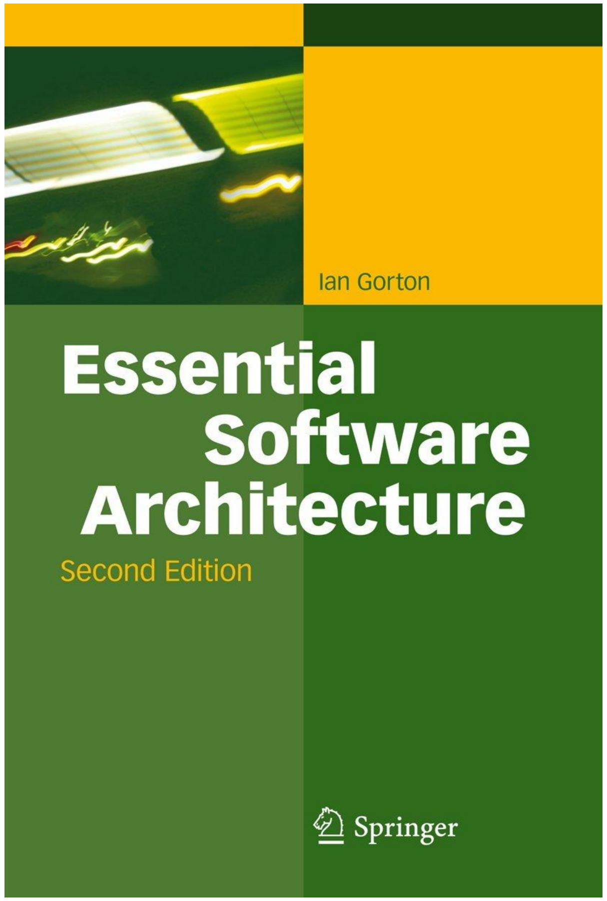
Agenda for CS #1

1) Introduction to ZG651/ZG653
   - Course handout
   - Books & Evaluation components
   - How to make the most out of this course ?
2) Motivation & Synergies between Software Engineering/Systems & SA
3) Introduction to Software Architecture
   - What is Software Architecture?
   - Definitions of Software architecture
   - Architecture structure and patterns
   - Good architecture
4) Quick check, Q&A
5) Importance of software architecture
6) Contexts of Software architecture
7) Q&A!

Text Book

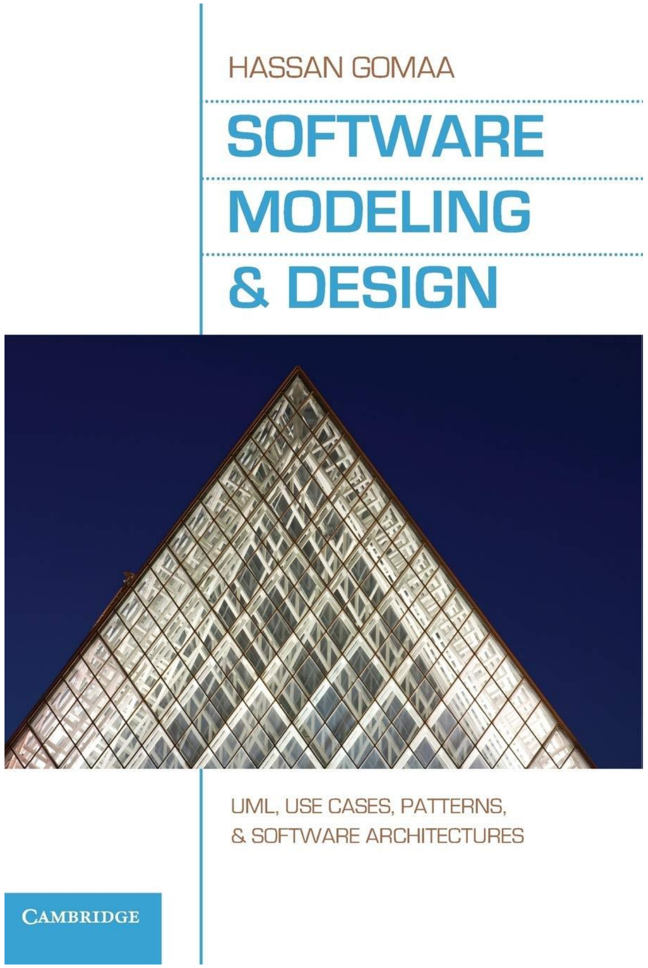

SEI SERIES IN SOFTWARE ENGINEERING
Software Architecture in Practice
THIRD EDITION
Len Bass
Paul Clements
Rick Kazman

Software Architecture in Practice, 3rd Edition or above, Len Bass et al.



Essential Software Architecture, 2nd Edition, Ian Gorton.

Reference Books

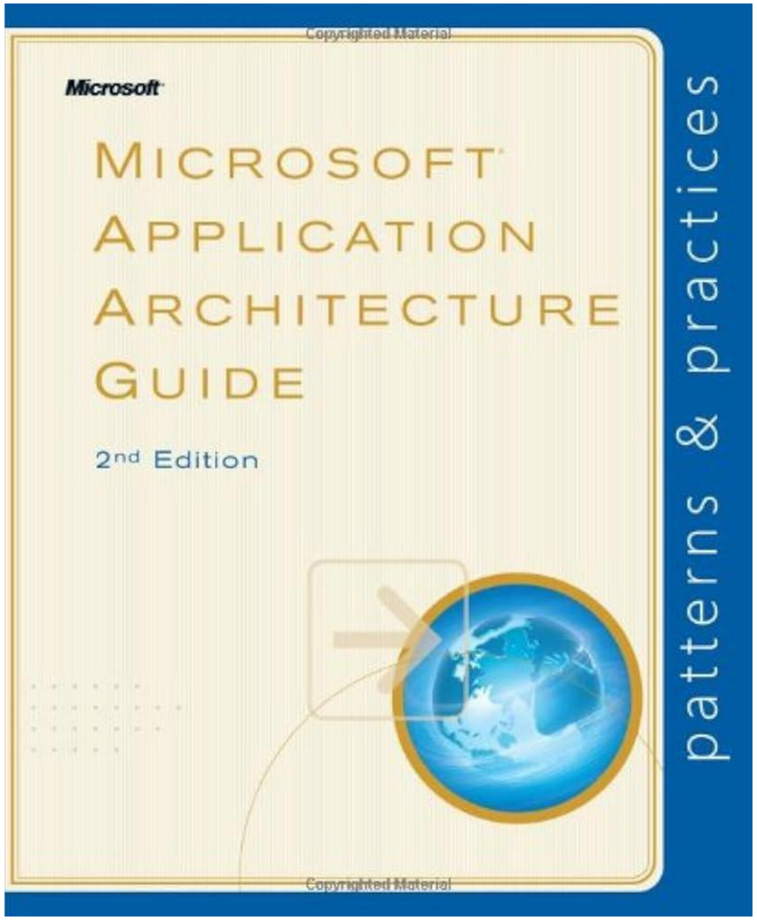


HASSAN GOMAA

SOFTWARE MODELING & DESIGN

UML, USE CASES, PATTERNS, & SOFTWARE ARCHITECTURES

CAMBRIDGE



Copyrighted Material

Microsoft

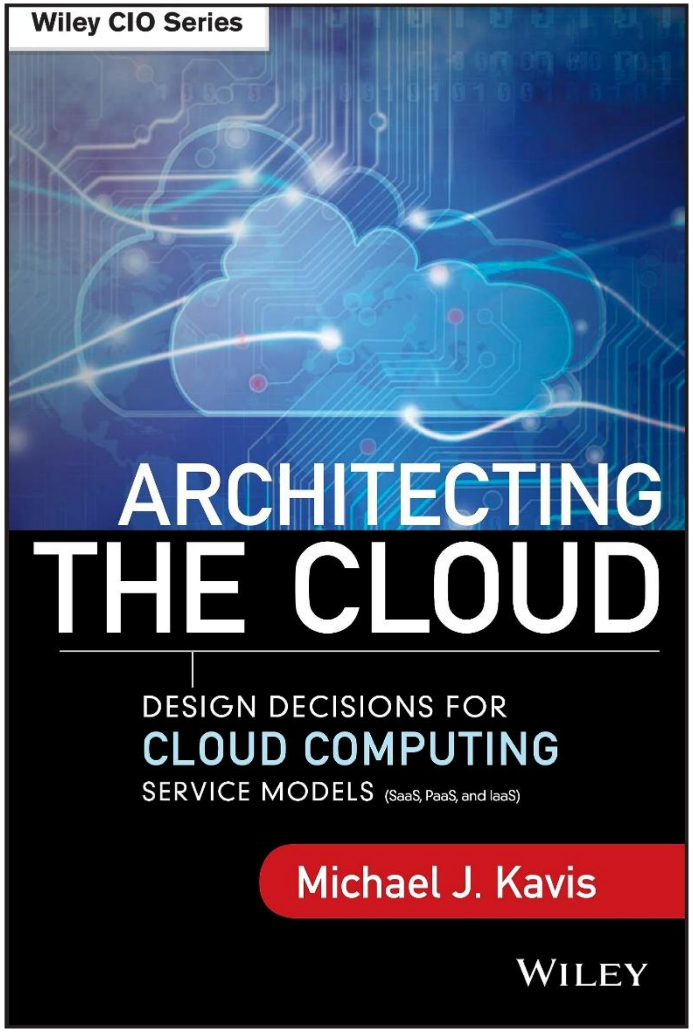
MICROSOFT APPLICATION ARCHITECTURE GUIDE

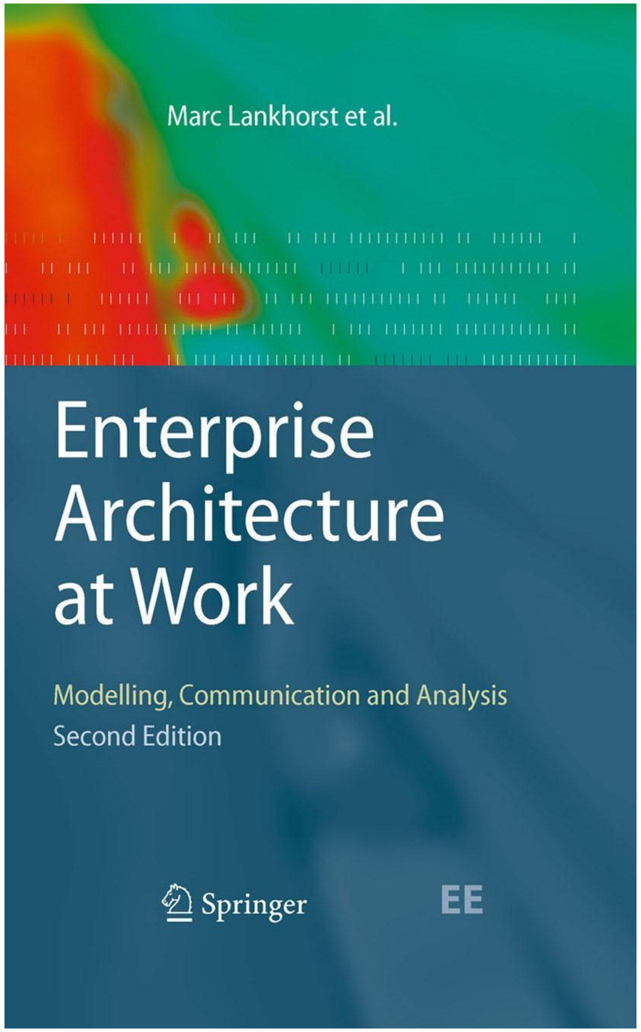
2nd Edition

patterns & practices

Copyrighted Material

Reference Books


Wiley CIO Series
ARCHITECTING THE CLOUD
DESIGN DECISIONS FOR CLOUD COMPUTING SERVICE MODELS (SaaS, PaaS, and IaaS)
Michael J. Kavis
WILEY


Marc Lankhorst et al.
Enterprise Architecture at Work
Modelling, Communication and Analysis
Second Edition
Springer
EE

And Many more …
Refer Course handout!!

Ground Rules!

- Be Regular 😅
- Mentally present – Observe!! Listen!! 😉
- Keep your questions for the Q&A section …
- Use the Discussion Forum in Canvas effectively
- Solve the exercises regularly!
- Go an extra mile ☺

$$\boxed{1^{365} = 1}$$

$$\boxed{1.01^{365} = 37.8}$$

Motivation for Software Architecture?

Aspiring Software Architect and allied areas ?

- Awesome! This role requires you to:
  - Build Scalable Systems
  - Make Design Decisions with Confidence
  - Bridge the Gap Between Business and Engineering
  - Handle Complexity with Elegance
- For this, you need a strong foundation of software architectures! Hence, this course is an essential part of your SS/SE journey.

Code is the bricks, but architecture is the blueprint of the house. Without understanding software architecture, you may build something that works, but not something that lasts

Informally what is meant by (Software) Architecture ?

- Essentially a blueprint of a software system that helps ***stakeholders*** to understand how the system would be once it is implemented
- What's should be there in this blueprint?
  - A description at a higher level of abstraction than objects and lines of codes

    So that
  - Stakeholders understand and reason about without getting lost into a sea of details!

Who are Stakeholders?

A complex software has multiple stakeholders who expect certain features of the software

Stakeholder	Area of Concern
Chief Technologist	• Does it adhere to organization standards ?
Database Designer	• What information to be stored, where, how, access mechanism??? • Information security issues?
Application Development team	• How do I implement a complex scenario? • How should I organize my code? • How do I plan for division of work?
Users/Customers	• Does it perform as per my requirement? • What about the cost/budget? • Scalability, performance and reliability of the system? • How easy it is to use? • Is it always available?
Infrastructure Manager	• Performance and scalability • Idea of system & network usage • Indication of hardware and software cost, scalability, deployment location • Safety and security consideration • Is it fault tolerant-crash recovery & backup
Release & Configuration Manager	• Build strategy • Code management, version control, code organization
System Maintainer	• How do I replace of a subsystem with minimal impact ? • How fast can I diagnosis of faults and failures and how quickly I can recover?

Why Architecture needs to be described?

- Each stakeholder has his own interpretation of the systems
  - Sometimes no understanding at all
  - Architect is the middleman who co-ordinates with these stakeholders
- How will everyone be convinced that his expectations from the system will be satisfied?
- Even when the architect has created the solution blueprint, how does she handover the solution to the developers?
- How do the developers build and ensure critical aspects of the system?
- Misunderstanding leads to incorrect implementation
  - Leads to 10 times more effort to fix at a later stage

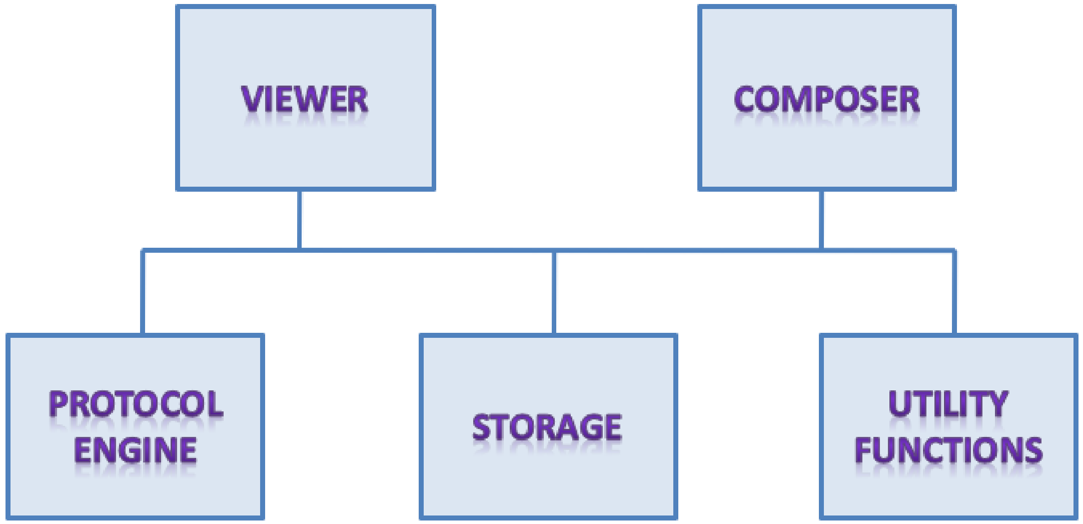
Any Large Software Corporation

- Hundreds of concurrent projects being executed
  - 10-100 team size
- Projects capture requirements, there are architects, and large Development teams
- Architect start with requirements team & handover to Development teams

Software Architecture Definition

- No unique definition though similar…(look at SEI CMU)
- .. "**structure** or structures of the system, which comprise **software elements**, the **externally visible properties** of those elements, and the **relationships** among them"(Bass, Clements and Kazman, Software Architecture in Practice, 2nd Ed)
- "description of elements from which systems are built, **interactions** among those elements, **patterns** that guide their **composition**, and **constraints** on these patterns. In general, a particular system is defined in terms of a collection of **components** and interactions among these components"(Shaw and Garlan "Software Architecture: Perspectives on an Emerging Disciplines")
- "description of the **subsystems** and **components** of a software system and the **relationship** between them. Subsystems and components are typically specified in different **views** to show the relevant **functional** and **nonfunctional** properties of a software system" (F. Buschmann et al)

Is this Architecture ?



What we understand

- The system has 5 elements
- They are interconnected
- One is on the top of another

Typically, we describe architecture as a collection of diagrams like this

What’s Ambiguous?

- Visible responsibilities
  - What do they do?
  - How does their function relate to the system
  - How have these elements been derived, is there any overlap?
- Are these processes, or programs
  - How do they interact when the software executes
  - Are they distributed?
- How are they deployed on a hardware
- What information does the system process?

What’s Ambiguous?

- Significance of connections
  - Signify control or data, invoke each other, synchronization
  - Mechanism of communications
- Significance of layout
  - Does level shown signify anything
  - Was the type of drawing due to space constraint

What should Architecture description have?

- A structure describing
  - Modules
    - Services offered by each module
    - and their interactions- to achieve the functionality
  - Information/data modeling
  - Achieving quality attributes
  - Processes and tasks that execute the software
  - Deployment onto hardware
  - Development plan

What should Architecture description have?......

- A behavioral description
  - describing how the structural elements execute “important” and “critical” scenarios
    - E.g. how does the system authenticates a mobile user
    - How does the system processes 1 TB of data in a day
    - How does it stream video uninterruptedly during peak load
  - These scenarios are mainly to implement various quality attributes

Architecture Is a Set of Software Structures

- A structure is a set of elements held together by a relation.
- Software systems are composed of many structures, and no single structure holds claim to being the architecture.
- There are three important categories of architectural structures.
  1. Module
  2. Component and Connector
  3. Allocation

Module Structures

- Some structures partition systems into implementation units, which we call modules.
- Modules are assigned specific computational responsibilities, and are the basis of work assignments for programming teams.
- In large projects, these elements (modules) are subdivided for assignment to sub-teams.

Component-and-connector Structures

- Other structures focus on the way the elements interact with each other at runtime to carry out the system's functions.
- We call runtime structures *component-and-connector (C&C) structures*.
- In our use, a component is always a runtime entity.
  - Suppose the system is to be built as a set of services.
  - The services, the infrastructure they interact with, and the synchronization and interaction relations among them form another kind of structure often used to describe a system.
  - These services are made up of (compiled from) the programs in the various implementation units – modules.

Allocation Structures

- Allocation structures describe the mapping from software structures to the system's environments
  - organizational
  - developmental
  - installation
  - Execution
- For example
  - Modules are assigned to teams to develop, and assigned to places in a file structure for implementation, integration, and testing.
  - Components are deployed onto hardware in order to execute.

Which Structures are Architectural?

- A structure is architectural if it supports
- reasoning about the system and the system's properties.
- The reasoning should be about an attribute of the system that is important to some stakeholder.
- These include
  - functionality achieved by the system
  - the availability of the system in the face of faults
  - the difficulty of making specific changes to the system
  - the responsiveness of the system to user requests,
  - many others.

Architecture is an Abstraction

- An architecture comprises software elements and how the
- elements relate to each other.
  - An architecture specifically omits certain information about elements that is not useful for reasoning about the system.
  - It omits information that has no ramifications outside of a single element.
  - An architecture selects certain details and suppresses others.
  - Private details of elements—details having to do solely with internal implementation—are not architectural.
- The architectural abstraction lets us look at the system in terms of its elements, how they are arranged, how they interact, how they are composed, what their properties are that support our system reasoning, and so forth.
- This abstraction is essential to taming the complexity of an architecture.
- We simply cannot, and do not want to, deal with all of the complexity all of the time.

Every System has a Software Architecture

- Every system comprises elements and relations among them to support some type of reasoning.
- But the architecture may not be known to anyone.
  - Perhaps all of the people who designed the system are long gone
  - Perhaps the documentation has vanished (or was never produced)
  - Perhaps the source code has been lost (or was never delivered)
- An architecture can exist independently of its description or specification.
- Documentation is critical.

Structures and Views

- A *view* is a representation of a coherent set of architectural elements, as written by and read by system stakeholders.
  - A view consists of a representation of a set of elements and the relations among them.
- A *structure* is the set of elements itself, as they exist in software or hardware.
- In short, a view is a representation of a structure.
  - For example, a module *structure* is the set of the system's modules and their organization.
  - A module *view* is the representation of that structure, documented according to a template in a chosen notation, and used by some system stakeholders.
- Architects design structures. They document views of those structures.

Structures Provide Insight

- Structures play such an important role in our perspective on software architecture because of the analytical and engineering power they hold.
- Each structure provides a perspective for reasoning about some of the relevant quality attributes.
- For example:
  - The module structure, which embodies what modules use what other modules, is strongly tied to the ease with which a system can be extended or contracted.
  - The concurrency structure, which embodies parallelism within the system, is strongly tied to the ease with which a system can be made free of deadlock and performance bottlenecks.
  - The deployment structure is strongly tied to the achievement of performance, availability, and security goals.
  - And so forth.

Architectural Patterns

- Architectural elements can be composed in ways that solve particular problems.
  - The compositions have been found useful over time, and over many different domains
  - They have been documented and disseminated.
  - These compositions of architectural elements, called architectural patterns.
  - Patterns provide packaged strategies for solving some of the problems facing a system.
- An architectural pattern delineates the element types and their forms of interaction used in solving the problem.
- A common module type pattern is the Layered pattern.
  - When the uses relation among software elements is strictly unidirectional, a system of layers emerges.
  - A layer is a coherent set of related functionality.
  - Many variations of this pattern, lessening the structural restriction, occur in practice.

Architectural Patterns

- Common component-and-connector type patterns:
- Shared-data (or repository) pattern.
  - This pattern comprises components and connectors that create, store, and access persistent data.
  - The repository usually takes the form of a (commercial)
  - database.
  - The connectors are protocols for managing the data, such as SQL.
- Client-server pattern.
  - The components are the clients and the servers.
  - The connectors are protocols and messages they share among each other to carry out the system's work.

Architectural Patterns

- Common allocation patterns:
- Multi-tier pattern
  - Describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium.
  - This pattern specializes the generic deployment (software-to-hardware allocation) structure.
- Competence center pattern and platform pattern
  - These patterns specialize a software system’s work assignment structure.
  - In competence center, work is allocated to sites depending on the technical or domain expertise located at a site.
  - In platform, one site is tasked with developing reusable core assets of a software product line, and other sites develop applications that use the core assets.

What Makes a “Good” Architecture?

- There is no such thing as an inherently good or bad architecture.
- Architectures are either more or less fit for some purpose
- Architectures can be evaluated but only in the context of specific stated goals.
- There are, however, good rules of thumb.

Process “Rules of Thumb”

- The architecture should be the product of a single architect or a small
- group of architects with an identified technical leader.
  - This approach gives the architecture its conceptual integrity and technical consistency.
  - This recommendation holds for Agile and open source projects as well as “traditional” ones.
  - There should be a strong connection between the architect(s) and the development team.
- The architect (or architecture team) should base the architecture on a prioritized list of well-specified quality attribute requirements.
- The architecture should be documented using views. The views should address the concerns of the most important stakeholders in support of the project timeline.
- The architecture should be evaluated for its ability to deliver the system’s important quality attributes.
  - This should occur early in the life cycle and repeated as appropriate.
- The architecture should lend itself to incremental implementation,
  - Create a “skeletal” system in which the communication paths are exercised but which at first has minimal functionality.

Structural "Rules of Thumb"

- The architecture should feature well-defined modules whose functional responsibilities are assigned on the principles of information hiding and separation of concerns.
  - The information-hiding modules should encapsulate things likely to change
  - Each module should have a well-defined interface that encapsulates or "hides" the changeable aspects from other software
- Unless your requirements are unprecedented your quality attributes should be achieved using well-known architectural patterns and tactics specific to each attribute.
- The architecture should never depend on a particular version of a commercial product or tool. If it must, it should be structured so that changing to a different version is straightforward and inexpensive.
- Modules that produce data should be separate from modules that consume data.
  - This tends to increase modifiability
  - Changes are frequently confined to either the production or the consumption side of data.

Why is Software Architecture Important?

1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.
8. An architecture can provide the basis for evolutionary prototyping.
9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that form the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.
13. An architecture can be the foundation for training a new team member.

The Many Contexts of Software Architecture

- Architecture in a Technical Context
- Architecture in a Project Life-Cycle Context
- Architecture in a Business Context
- Architecture in a Professional Context
- Stakeholders
- How Is Architecture Influenced?
- What Do Architectures Influence?

Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in