Software System/Design & Architecture

Eng. Muhammad Fahad Khan
Assistant Professor
Department of Software Engineering
Lecture Objectives

- Scope of Software Architectures
- Fundamental principles and guidelines for software architecture design
Even in the field of software engineering, we often come across different forms of architecture. For example, in addition to the concept of software architecture, we may encounter concepts such as enterprise architecture, system architecture, organizational architecture, information architecture, hardware architecture, application architecture, infrastructure architecture, and so on. You will also hear other terms, each of which defines a specific scope of the architecting activities.

However, the scope of some of these terms can be inferred from Figure . As you consider this figure and the discussion that follows, there are almost certainly elements of it that you disagree with or that you use differently within your organization. But that is exactly the point -- to show that these terms do exist in the industry, but that there is no consensus on their meaning.
Architectural Scope

Enterprise

System

- Software
- Hardware
- Organization
- Information
The Scope of Different Terms

The elements shown in previous slide are:

- **Software architecture**, which is the main focus of this article as defined earlier.
- A **hardware architecture**, which considers elements such as CPUs, memory, hard disks, peripheral devices such as printers, and the elements used to connect these elements.
- An **organizational architecture**, which considers elements that are concerned with business processes, organizational structures, roles and responsibilities, and core competencies of the organization.
- An **information architecture**, which considers the structure by which information is organized.
- Software architecture, hardware architecture, organizational architecture, and information architecture, which are all subsets of the overall system architecture.
- An **enterprise architecture**, which is similar to a system architecture in that it, too, considers elements such as hardware, software, and people. However, an enterprise architecture has a stronger link to the business in that it focuses on the attainment of the business objectives and is concerned with items such as business agility and organizational efficiency. An enterprise architecture may cross company boundaries.
Scope of Software Architecture

- Enterprise Architecture
- The Business Perspective
- The Application Perspective
- The Information Perspective
- The technology perspective
An enterprise architecture (EA) is a conceptual tool that assists organizations with the understanding of their own structure and the way they work. It provides a map of the enterprise and is a route planner for business and technology change.

Normally an enterprise architecture takes the form of a comprehensive set of cohesive models that describe the structure and the functions of an enterprise. Important uses of it are in systematic IT planning and architecting, and in enhanced decision making.
Enterprise Architecture

The individual models in an EA are arranged in a logical manner, and this provides an ever-increasing level of detail about the enterprise, including:

- Its objectives and goals.
- Its processes and organization.
- Its systems and data.
- The technology used.
Enterprise Architecture

Software Infrastructure Services

Data Storage Security Development Tools

Software Infrastructure Services Workflow Integration Data Access Authorization User Interface

Core Business Services

Core Mission Services

Copyright 2012 @ M. Fahad Khan
The Business Perspective

The business perspective describes how a business works. It includes broad business strategies along with plans for moving the organization from its current state to an visualized future state. It will typically include the following:

• The enterprise's high-level objectives and goals.
• The business processes carried out by the entire enterprise, or a significant portion of the enterprise.
• The business functions performed.
• Major organizational structures.
• The relationships between these elements.
Collaborative Business Perspective (Level 1)

Financial Performance
- Liquidity Ratios
- Leverage Ratios
- Profitability
- ROI

Market Analysis
- Market Competition
- Market Share

Product Development
- Basic Information
- Design & Development
- Quality & Tracking

Copyright 2012 @ M. Fahad Khan
The Application Perspective

The application perspective defines the enterprise's application portfolio and is application-centered. This view will typically include:

- Descriptions of automated services that support the business processes.
- Descriptions of the interaction and interdependencies (interfaces) of the organization's application systems.
- Plans for developing new applications and revising old applications based on the enterprises objectives, goals, and evolving technology platforms.

- The application perspective may represent cross-organization services, information, and functionality, linking users of different skills and job functions in order to achieve common business objectives.
Building the business case for workforce mobile apps

1. Is the organisation ready?
   - Current state
     - How are remote or mobile workers currently supported?
     - What strategic role does IT play in the organisation?
     - How are competitors using mobile computing internally?
     - Is the post-personal computer a challenge or opportunity?

2. Why create an app?
   - App business case
     - Will the app improve basic connectivity?
     - Will the app improve or create new ways to collaborate?
     - Will the app benefit a specific role or activity?
     - How does the app address the consumerisation of IT?

3. What will it achieve?
   - Link purpose to measures
     - Better personal productivity
     - Enhanced or innovative service
     - Specific business process
     - Information management
     - Collaboration
     - Knowledge management & learning
     - Employee engagement
     - Other emergent purpose

4. What are the use cases?
   - Bring-your-own-device policy
   - Mobile IT architecture
   - Risks

Copyright 2012@ M.Fahad Khan
The Information Perspective

The information perspective describes what the organization needs to know to run its business processes and operations. It includes:

- Standard data models.
- Data management policies.
- Descriptions of the patterns of information production and consumption in the organization.

- The information perspective also describes how data is bound into the work flow, including structured data stores such as databases, and unstructured data stores such as documents, spreadsheets, and presentations that exist throughout the organization.
Data from the connected device and the legacy data is processed in two different flows. Based on the insights in the analytic process, it is possible to introduce advanced real-time analytics and actions.
The Technology Perspective

The technology perspective lays out the hardware and software supporting the organization. It includes, but is not limited to:

- Desktop and server hardware.
- Operating systems.
- Network connectivity components.
- Printers.
- Modems.

The technology perspective provides a logical, vendor-independent description of infrastructure and system components that are necessary to support the application and information perspectives. It defines the set of technology standards and services needed to execute the business mission.
Fundamental Principles & Guidelines for Software Architecture Design
**Key Design Principles**

**Separation of concerns**

A key principle of software development and architecture is the notion of *separation of concerns*. At a low level, this principle is closely related to the Single Responsibility Principle of object oriented programming. The general idea is that one should avoid co-locating different concerns within the design or code. For instance, if your application includes business logic for identifying certain noteworthy items to display to the user, and your application formats such items in a certain way to make them more noticeable, it would violate separation of concerns if both the logic for determining which items were noteworthy and the formatting of these items were in the same place. The design would be more maintainable, less tightly coupled, and less likely to violate the Don’t Repeat Yourself principle if the logic for determining which items needed formatted were located in a single location (with other business logic), and were exposed to the user interface code responsible for formatting simply as a property.
Key Design Principles

**Single Responsibility principle** Each component or module should be responsible for only a specific feature or functionality.

**Principle of Least Knowledge** (also known as the Law of Demeter or LoD). A component or object should not know about internal details of other components or objects.
Key Design Principles

Minimize upfront design Only design what is necessary. In some cases, you may require upfront comprehensive design and testing if the cost of development or a failure in the design is very high. In other cases, especially for agile development, you can avoid big design upfront (BDUF). If your application requirements are unclear, or if there is a possibility of the design evolving over time, avoid making a large design effort prematurely. This principle is sometimes known as YAGNI ("You ain’t gonna need it").
Key Design Principles

Don’t repeat yourself (DRY) You should only need to specify intent in one place. For example, in terms of application design, specific functionality should be implemented in only one component; the functionality should not be duplicated in any other component.
Design Practices
Design Practices

Keep design patterns consistent within each layer Within a logical layer, where possible, the design of components should be consistent for a particular operation. For example, if you choose to use the Table Data Gateway pattern to create an object that acts as a gateway to tables or views in a database, you should not include another pattern such as Repository, which uses a different paradigm for accessing data and initializing business entities. However, you may need to use different patterns for tasks in a layer that have a large variation in requirements, such as an application that contains business transaction and reporting functionality.
Design Practices

**Do not duplicate functionality within an application** There should be only one component providing a specific functionality—this functionality should not be duplicated in any other component. This makes your components cohesive and makes it easier to optimize the components if a specific feature or functionality changes. Duplication of functionality within an application can make it difficult to implement changes, decrease clarity, and introduce potential inconsistencies.
Design Practices

**Prefer composition to inheritance** Wherever possible, use composition over inheritance when reusing functionality because inheritance increases the dependency between parent and child classes, thereby limiting the reuse of child classes. This also reduces the inheritance hierarchies, which can become very difficult to deal with.

```
Class A
  \_____ Class B
  \_____ Class C
  \_____ Class D
```
Design Practices

**Establish a coding style and naming convention for development** Check to see if the organization has established *coding style and naming standards*. If not, you should establish common standards. This provides a consistent model that makes it easier for team members to *review code they did not write*, which leads to better maintainability.
Design Practices

Maintain system quality using automated QA techniques during development. Use unit testing and other automated Quality Analysis techniques, such as dependency analysis and static code analysis, during development. Define clear behavioral and performance metrics for components and sub-systems, and use automated QA tools during the build process to ensure that local design or implementation decisions do not adversely affect the overall system quality.
TestComplete™ 7
See what’s new »

QA Analyst / Performance Testing

Automate Your Tests with TestComplete and Visual Studio Team System.

Copyright 2012 @ M. Fahad Khan
Design Practices

Consider the operation of your application. Determine what metrics and operational data are required by the IT infrastructure to ensure the efficient deployment and operation of your application. Designing your application’s components and sub-systems with a clear understanding of their individual operational requirements will significantly ease overall deployment and operation. Use automated QA tools during development to ensure that the correct operational data is provided by your application’s components and sub-systems.
Application Layers

**Separate the areas of concern** Break your application into distinct features that overlap in functionality as little as possible. The main benefit of this approach is that a feature or functionality can be optimized independently of other features or functionality. In addition, if one feature fails, it will not cause other features to fail as well, and they can run independently of one another. This approach also helps to make the application easier to understand and design, and facilitates management of complex inter dependent systems.
Application Layers

Be explicit about how layers communicate with each other. Allowing every layer in an application to communicate with or have dependencies upon all of the other layers will result in a solution that is more challenging to understand and manage. Make explicit decisions about the dependencies between layers and the data flow between them.
Application Layers

Do not mix different types of components in the same logical layer. Start by identifying different areas of concern, and then group components associated with each area of concern into logical layers. For example, the UI layer should not contain business processing components, but instead should contain components used to handle user input and process user requests.
Keep the data format consistent within a layer or component. Mixing data formats will make the application more difficult to implement, extend, and maintain. Every time you need to convert data from one format to another, you are required to implement translation code to perform the operation and incur a processing overhead.
Key Design Considerations

This guide describes the major decisions that you must make, and which help to ensure that you consider all of the important factors as you begin and then iteratively develop your architecture design. The major decisions, briefly described in the following sections, are:

- Determine the Application Type
- Determine the Deployment Strategy
- Determine the Appropriate Technologies
- Determine the Quality Attributes
- Determine the Crosscutting Concerns
Determine the Application Type

• Choosing the appropriate application type is the key part of the process of designing an application. Your choice is governed by your specific requirements and infrastructure limitations. Many applications must support multiple types of client, and may make use of more than one of the basic archetypes. This guide covers the following basic application types:
  • Applications designed for mobile devices.
  • Rich client applications designed to run primarily on a client PC.
  • Rich Internet applications designed to be deployed from the Internet, which support rich UI and media scenarios.
  • Service applications designed to support communication between loosely coupled components.
  • Web applications designed to run primarily on the server in fully connected scenarios.
WEB APPLICATIONS

Access and unlock the potential of your business anywhere. Web applications are internet-based solutions that allow you to connect with your business’s operations wherever there is internet access. IRDS is dedicated to offering convenient applications that will open the capabilities of your productivity.

Read More...
Determine the Deployment Strategy

Your application may be deployed in a variety of environments, each with its own specific set of constraints such as physical separation of components across different servers, a limitation on networking protocols, firewall and router configurations, and more. Several common deployment patterns exist, which describe the benefits and considerations for a range of distributed and non-distributed scenarios. You must balance the requirements of the application with the appropriate patterns that the hardware can support, and the constraints that the environment exerts on your deployment options. These factors will influence your architecture design.
Determine the Appropriate Technologies

When choosing technologies for your application, the key factors to consider are the type of application you are developing and your preferred options for application deployment topology and architectural styles. Your choice of technologies will also be governed by organization policies, infrastructure limitations, resource skills, and so on. You must compare the capabilities of the technologies you choose against your application requirements, taking into account all of these factors before making decisions.
Determine the Quality Attributes

Quality attributes—such as security, performance, and usability—can be used to focus your thinking on the critical problems that your design should solve. Depending on your requirements, you might need to consider every quality attribute covered in this guide, or you might only need to consider a subset. For example, every application design must consider security and performance, but not every design needs to consider interoperability or scalability. Understand your requirements and deployment scenarios first so that you know which quality attributes are important for your design. Keep in mind that quality attributes may conflict; for example, security often requires a tradeoff against performance or usability.
website usability

Are people getting lost on your website? A Website Usability Review could be the answer.
Determine the Crosscutting Concerns

• Crosscutting concerns represent key areas of your design that are not related to a specific layer in your application. For example, you should consider implementing centralized or common solutions for the following:
  • A logging mechanism that allows each layer to log to a common store, or log to separate stores in such a way that the results can be correlated afterwards.
  • A mechanism for authentication and authorization that passes identities across multiple layers to permit granting access to resources.
  • An exception management framework that will work within each layer, and across the layers as exceptions are propagated to the system boundaries.
  • A communication approach that you can use to communicate between the layers.
  • A common caching infrastructure that allows you to cache data in the presentation layer, the business layer, and the data access layer.
COMING UP NEXT

A Technique for Architecture & Design
Assignment 02

Give one real life example of software/Application that is developed using software architecture design guidelines.

Due Date: 1 October 2012
Question in my mind is?

Should I ask this?

hmmmmmmmmm?

Sorry I was sleeping sir!

Copyright 2012@ M.Fahad Khan
If you have any query please feel free to ask

**Phone:** +92-51-9047-746

**Fax:** +92-51-9047-420

**Email:** fahad.khan@uettaxila.edu.pk

University Of Engineering & Technology Taxila Pakistan