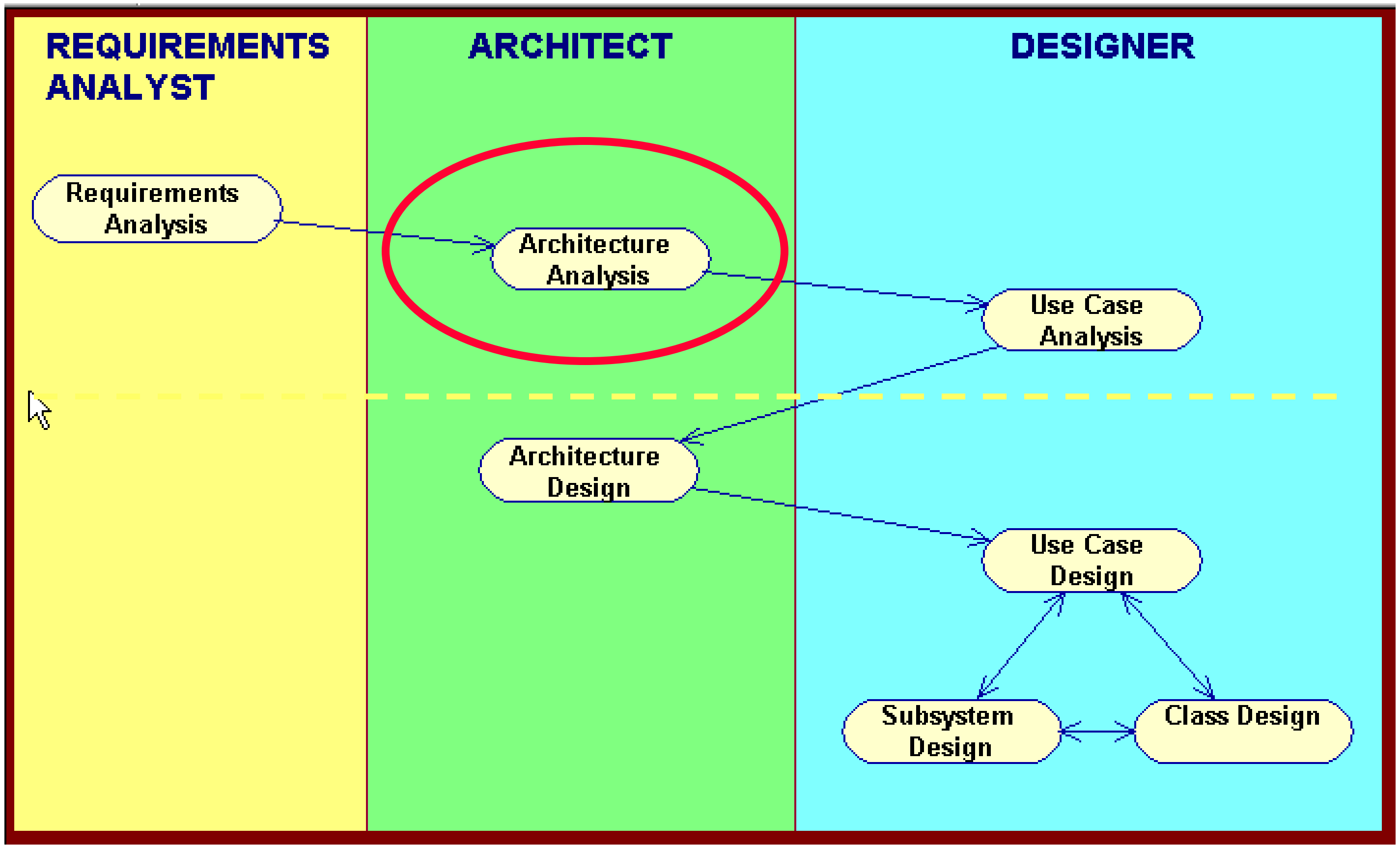
Architectural Analysis

The whole is contained in every part.

Basic RUP OOAD Activities



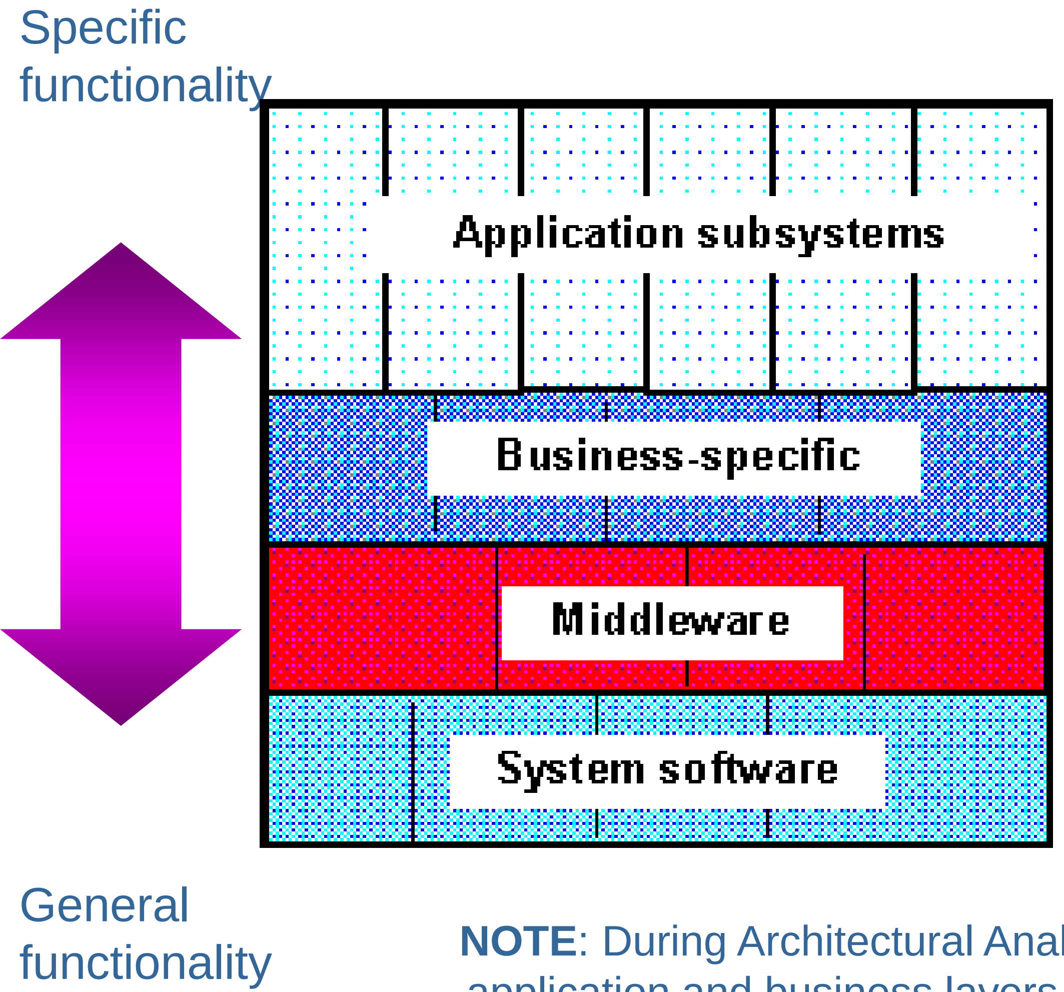
Architectural Analysis Topics

- Hi-Level System Architecture
  1. Layers
  2. Analysis Mechanisms
  3. Key Abstractions

Design Patterns

- A design pattern is a customizable solution to a common design problem
  - Describes a common design problem
  - Describes the solution to the problem
  - Discusses the rationale, results and trade-offs of applying the pattern
- Design patterns provide the capability to
  - reuse successful designs
  - Capture master-class design expertise
- Architectural patterns are design patterns applied to software architecture. Example: the Layering Pattern

Typical Layered Architecture



Distinct application subsystem that make up an application - contains the value adding software developed by the organization.

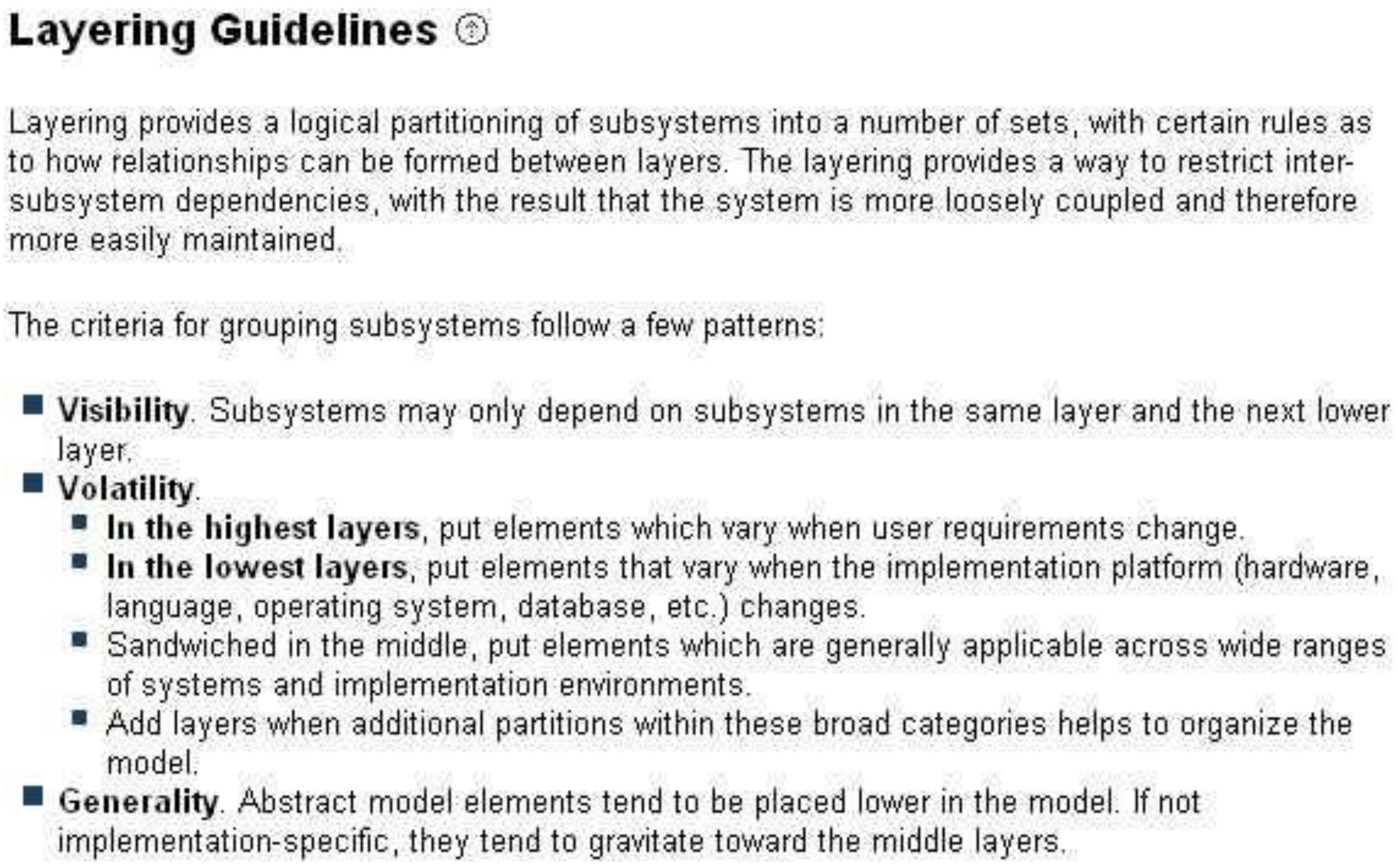
Business specific - contains a number of reusable sybsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.

NOTE: During Architectural Analysis, the focus is on the upper layers – application and business layers. The middleware and system software layers will be detailed in Architectural Design.

Part 1 - The Notion of Architectural Layers



Layering Guidelines

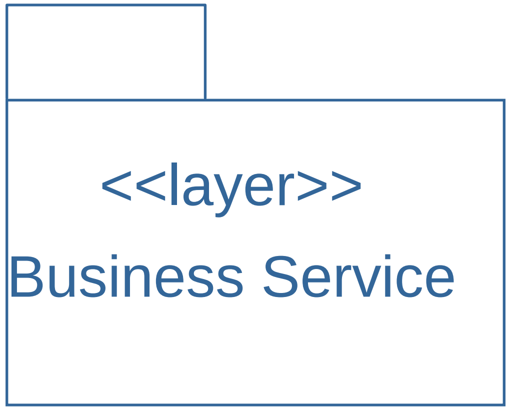
Layering provides a logical partitioning of subsystems into a number of sets, with certain rules as to how relationships can be formed between layers. The layering provides a way to restrict inter-subsystem dependencies, with the result that the system is more loosely coupled and therefore more easily maintained.

The criteria for grouping subsystems follow a few patterns:

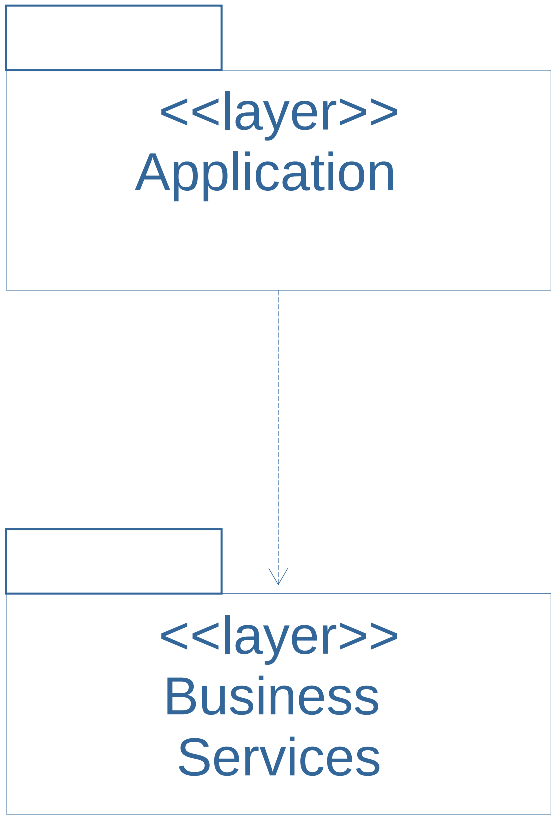
- **Visibility**. Subsystems may only depend on subsystems in the same layer and the next lower layer.
- **Volatility**.
  - **In the highest layers**, put elements which vary when user requirements change.
  - **In the lowest layers**, put elements that vary when the implementation platform (hardware, language, operating system, database, etc.) changes.
  - Sandwiched in the middle, put elements which are generally applicable across wide ranges of systems and implementation environments.
  - Add layers when additional partitions within these broad categories helps to organize the model.
- **Generality**. Abstract model elements tend to be placed lower in the model. If not implementation-specific, they tend to gravitate toward the middle layers.

UML for Modeling Architectural Layers

- Use stereotyped packages to show Architectural layers
- Use the “standard” <<layer>> stereotype



Model Organization : High-Level Example



MUMScrum Layers

- What are the simple layers for or our MUMScrum project?
- At the analysis stage the layers should be the same for all implementations:
  - .NET
  - Java EE
  - Spring Hibernate
  - Android
  - Etc.

MUMScrum -- Layers or MVC or Both?

- MVC is a design pattern for separating the model (data), view (user interface) and controller (business logic) concerns and creating associations between them.
- Often MVC is considered to be strictly for user interface: e.g., Spring MVC, or .NET MVC.
- Developing a layers architecture can be used with MVC. In the layers we may make decisions on things like Operating Systems, Databases, Networks, etc.
- We call these ***architectural mechanisms*** at the analysis stage.

Part 2 - The Notion of Architectural Mechanisms

- A project's architectural mechanisms are standards for how common design problems are to be solved on this project
  - Promotes uniformity of solutions and increases possibility of reuse, ease of maintenance and minimizes entropy of multiple solution strategies
  - An architectural mechanism is really an architectural pattern for the project.
- Whenever possible, represent in diagrams as a placeholder
  - Short-hand representation for complex behavior

- Example: Plan to use a legacy database (since it is a known constraint) and represent the database as a placeholder in diagrams. Don't need to design the database or the infrastructure for connecting since company already knows how this should be done – mechanism is known.

Identifying Analysis Mechanisms

Two approaches:

• Top Down -- use well-known approaches in the context of your new application (e.g. persistence)

• Bottom Up -- identify some pattern in emerging application that is relevant for the current application and possibly others -- e.g., a rules engine or other useful framework

Sample Analysis Mechanisms

- Distribution of components on multiple tiers or via web services – e.g. SOAP or RESTful
- Transaction management
- Concurrency
- Persistence -- e.g. Object Relational Mapping - ORM
- Security -- e.g. Java Authentication and Authorization
- Error and Exception detection / handling / reporting
- Rules engine
- Wrapping of legacy systems
- Web Application Framework
- User Interface Framework

MUMScrum Analysis Mechanisms

- What are some analysis mechanisms for our MUMScrum project?

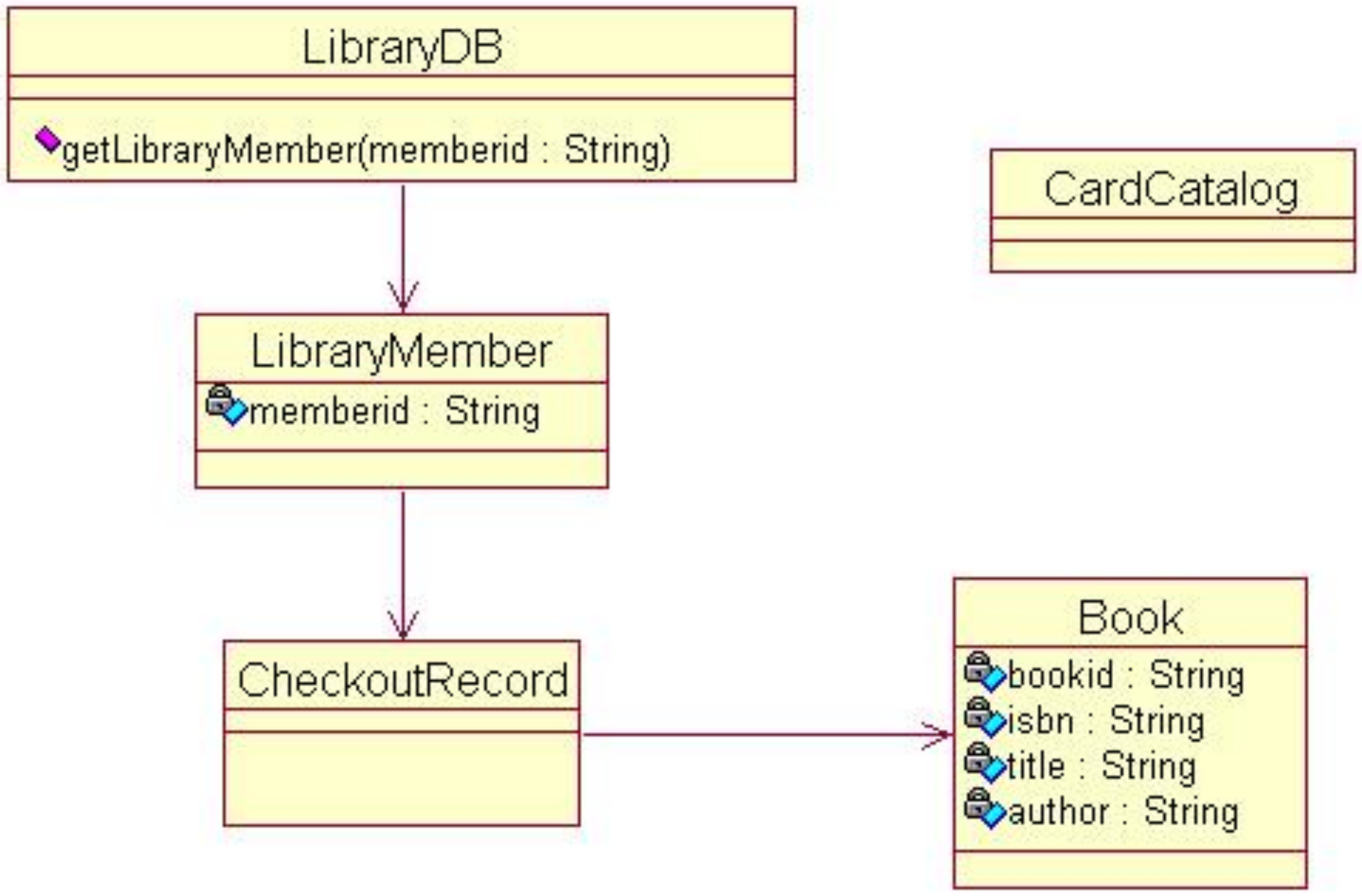
Part 3 -- Key Abstractions Guidelines

- Key abstractions are concepts identified during requirements analysis. Examples: Customer, Product, LegacyDB, Course, Section, Faculty, etc.
- Goal is "analysis classes" common across the use cases; these will be refined and fleshed out as the project evolves
- Architect will come up with these based on
  - Statement of the Problem
  - List of Requirements
  - Past experience with similar systems
  - Understanding of business domain

Key Abstractions Guidelines (cont)

- Include obvious entities acted upon by the use cases, e.g.,
  - Actors for which system has internal data (e.g., customer, student, etc; not librarian, not product manager, etc)
  - External systems which require system interaction (e.g. databases, messaging service) and a GUI (if there is one)
  - Entities that are important parts of the business operation
    - Example: CheckoutRecord in the library example
- Include any obvious attributes and operations. Key abstraction diagrams are class diagrams.

Example: Key Abstractions



Compro Admissions Forms Key Abstractions

- Each group list some key abstractions for our MUMScrum project.

Architecture Analysis Summary

1. Activity performed by system architect in preparation for use case analysis.
2. Identify an initial system architecture--preferably one that separates the project into nonchanging and changing subsets, and the changing elements should be grouped such that those grouped together tend to change together.
3. Architectural mechanisms are identified which are known solution patterns to complex problems. We might specify ORM and Web Application but not the specific frameworks.
4. Key abstractions are identified which will provide a common set of key domain elements across use case realizations.

Review of Architectural Analysis

- Describe the purpose of Architectural Analysis.
- Describe the rationale for defining analysis mechanisms.
- How are key abstractions identified during architectural analysis? Why are they identified?
- Describe the advantages of a layered architecture with an example of typical layers.