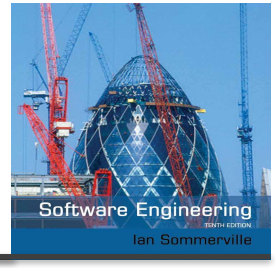
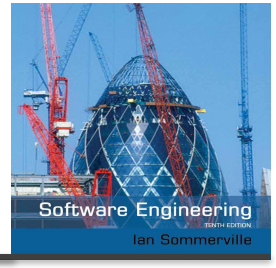


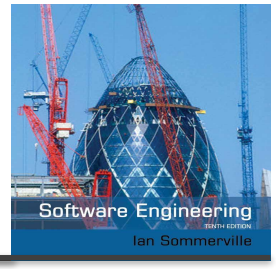
Architectural Design

Architectural design



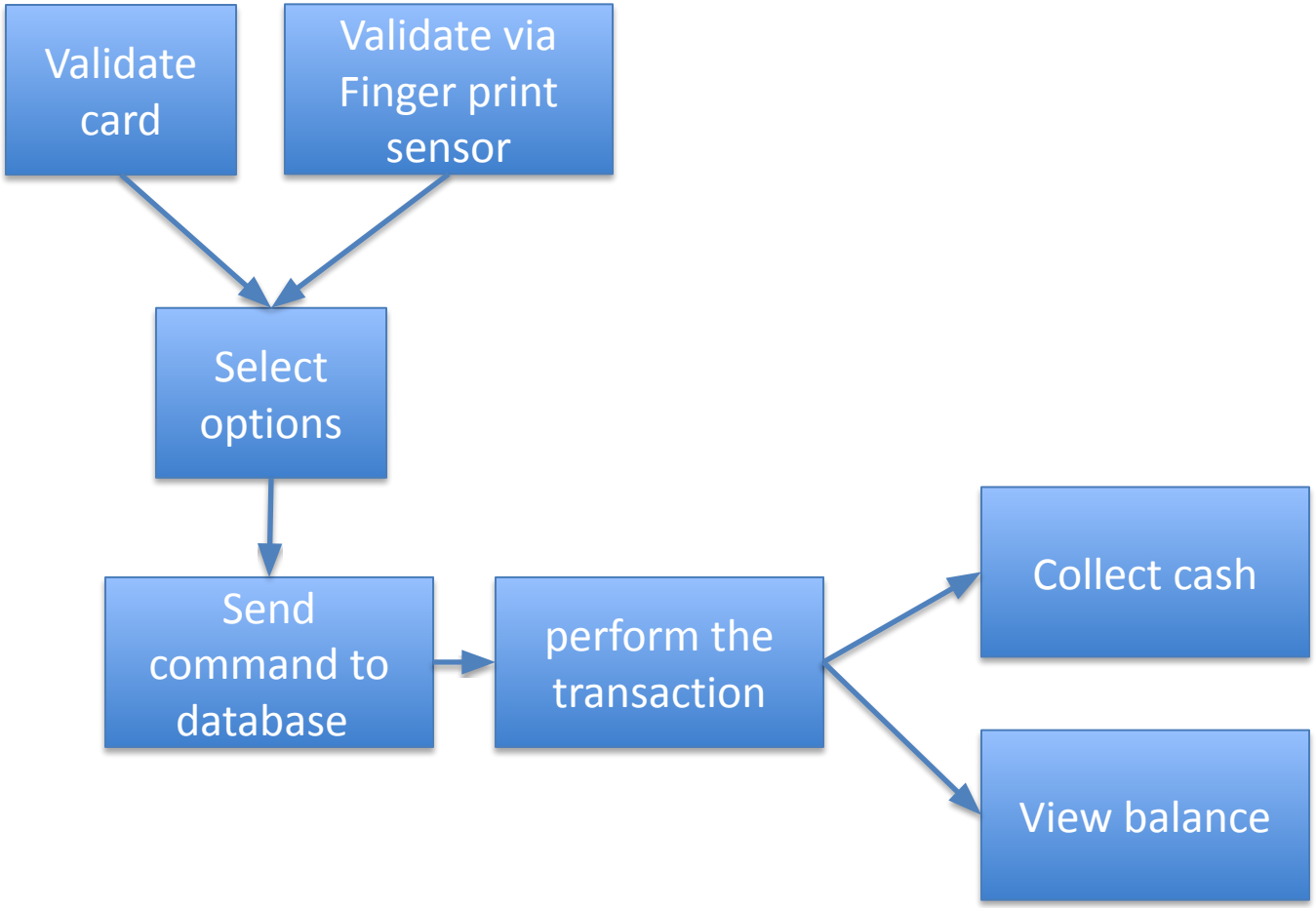
- 1st stage in software design process
- concerned with:
  - understanding how a software system should be organized
  - designing the overall structure of that system.
- It is the critical link between design and requirements engineering
  - as it identifies the main structural components in a system and the relationships between them.
- **output of this process is an architectural model** that describes how the system is organized as a set of communicating components.

Agility and architecture

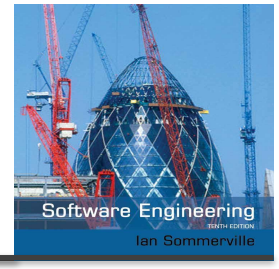


- It is generally accepted that an **early stage of agile processes is to design an overall systems architecture.**
- Refactoring is easy BUT **refactoring the system architecture is usually expensive** because a lot of components have to be changed to adapt to the architectural changes.

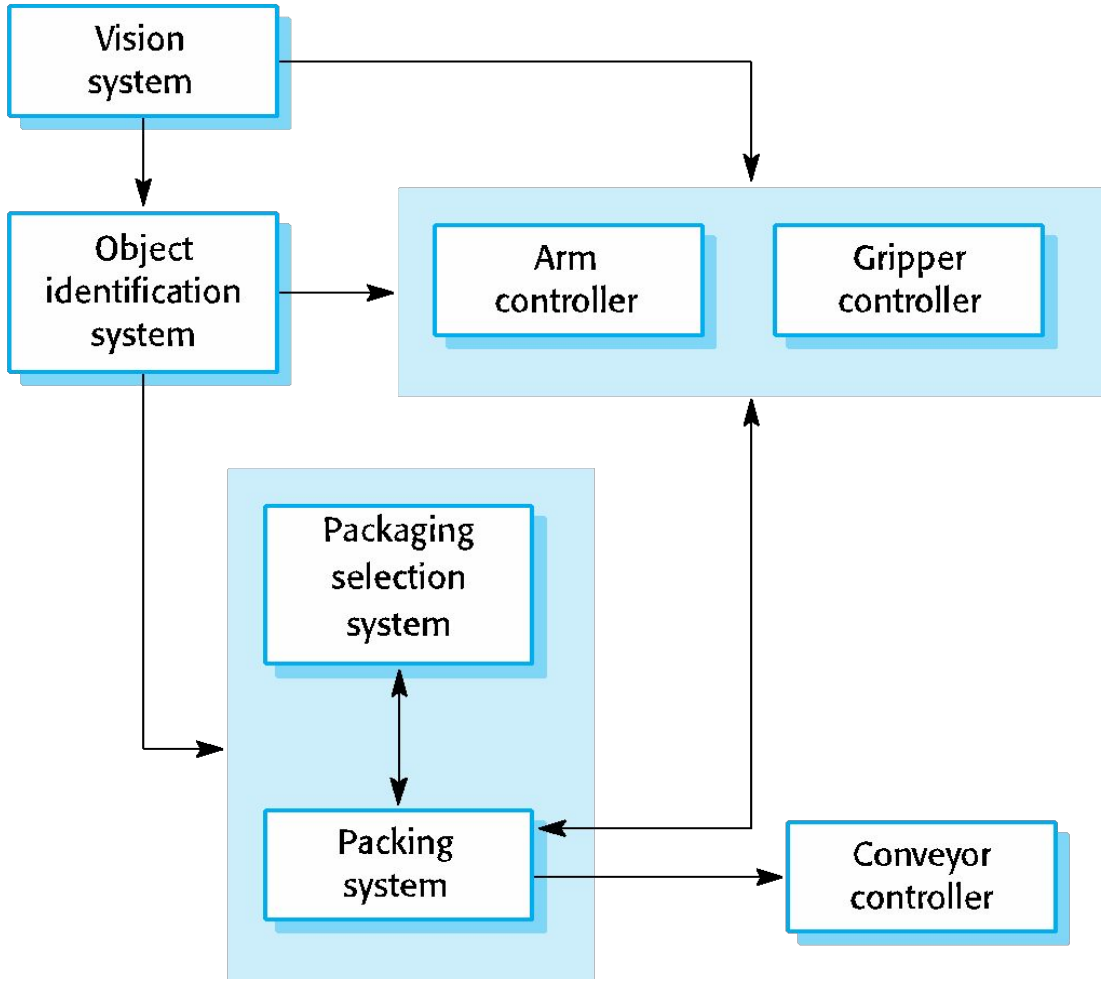
The architecture of an ATM Machine



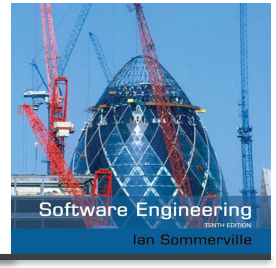
The architecture of a packing robot control system



- Pack different kind of objects.
- Classify it to a type
- Package accordingly
- And put on to another conveyor for shipment

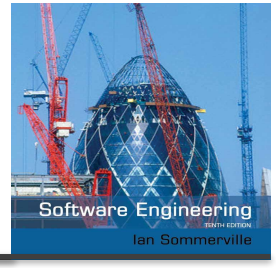


RE & Design process overlapping



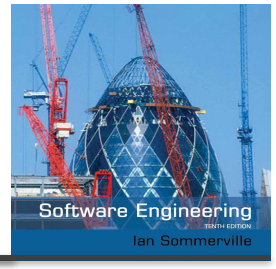
- Practically the **requirement specification process is overlapped with architectural design procedure**
- It is **impossible that the specification does not includes design information**
- You need to **identify the main architectural components** as these reflects the **high level features of the system.**
- so the **RE process must show a sort of abstract system arch showing where you associate subsystems .**
- You may **use this decomposition to discuss & finetune requirements with stakeholders**

Architectural abstraction



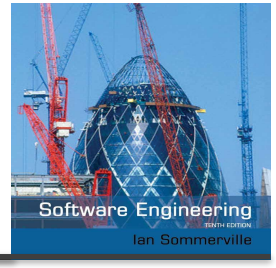
- You can design software architecture at two levels of abstraction:
- **Architecture in the small:**
  - concerned with the **architecture of individual programs**.
  - Depicts **how an individual program is decomposed into components.**
- **Architecture in the large:**
  - concerned with the **architecture of complex enterprise systems**
  - that **include other systems, programs, and program components**.
  - These **enterprise systems are distributed over different computers, which may be owned and managed by different companies.**

Why Software Architecture is important?



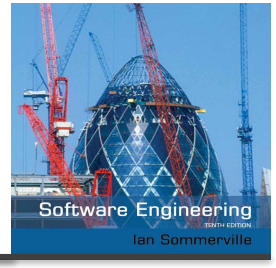
- Non – functional requirements are applied to the system as a whole.
- System architecture affects the non-functional requirements of the system.
- Like, robustness, performance, maintainability etc.

Advantages of explicit architecture



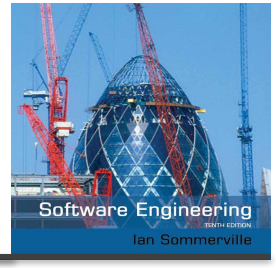
- **Stakeholder communication**
  - High level presentation of a system.
  - Architecture may be used as a focus of discussion by system stakeholders.
- **System analysis**
  - Means that analysis of whether the system can meet its non-functional requirements is possible.
- **Large-scale reuse**
  - As architecture is the compact manageable details pf system organization & its interoperations. So,
  - Same system architecture is often used for similar applications.
  - The architecture may be reusable across a range of systems.
  - Product-line architectures are the approach to reuse same architecture.

Architectural representations



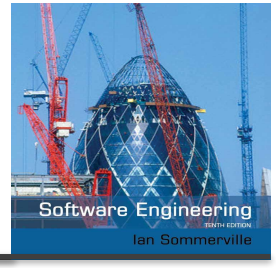
- Often modeled as **informal block diagrams showing entities and relationships**.
- Components are represented as **Box**
- Sub components are represented as **nested boxes**.
- **Arrows** depicts the data and control signals flowing from components.

Box and line diagrams



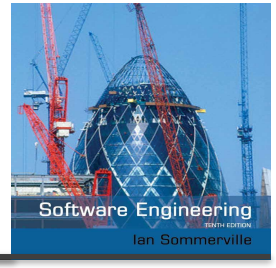
- Very abstract
- They do not show:
  - the nature of component relationships
  - the externally visible properties of the sub-systems (fault tolerance, shared resource usage etc.) .
- However, useful for communication with stakeholders and for project planning.

Ways to use the architectural models



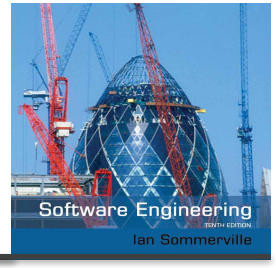
- As a **way of facilitating discussion about the system design**
  - A high-level architectural view of a system
  - **Useful for project planning with system stakeholders**
  - **Not cluttered with detail.**
  - **Easily relatable**
  - After that, the system as a whole can be discussed in detail without understanding implementation details.
  - Since it identifies the key components, so **tasks planning & assignment will become easy**.
- As a **way of documenting an architecture that has been designed**
  - The aim here is to **produce a complete system model** that shows the different components in a system, their interfaces and their connections.
  - **System evolution & understanding becomes easy**

architectural models' representation



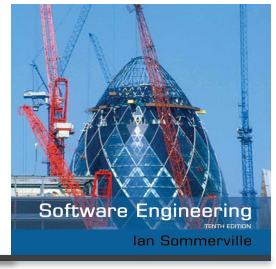
- For documenting architectural details, better to use an ADL(Architecture Description Language).
- As they are recognized globally
- Some ADLs are : UML, ArchiMate

- Drawbacks;
  - Time consuming
  - Expensive solution



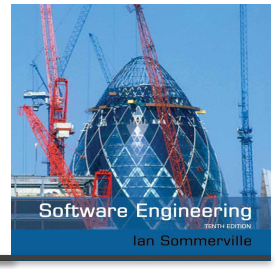
Architectural design decisions

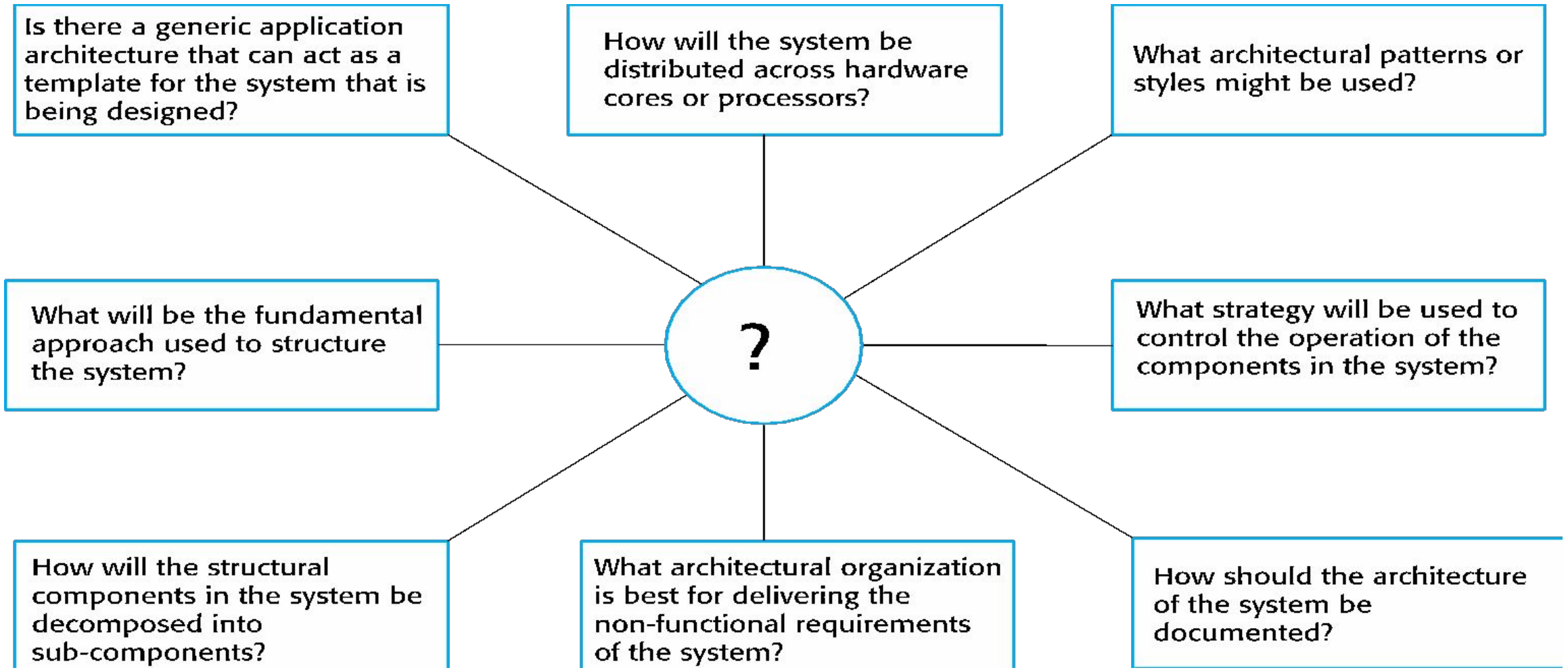
Architectural design decisions



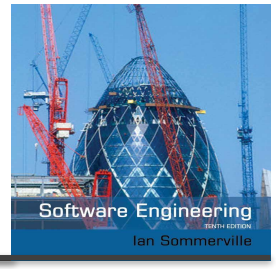
- Architectural design is a **creative process** to develop a system via fulfilling FRs & NFRs.
- **No formulaic design process**
- Depends upon:
  - System type
  - Background
  - Experience of system architect
  - Requirements
- So it's a **series of decisions to made**, **not a sequence of activities to be performed.**
- However, **a number of common decisions span all design processes** and these decisions affect the non-functional characteristics of the system.

Architectural design decisions by system architect



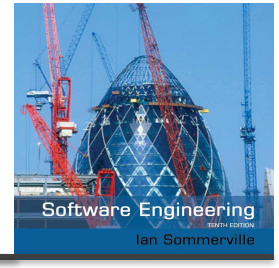


Architecture reuse

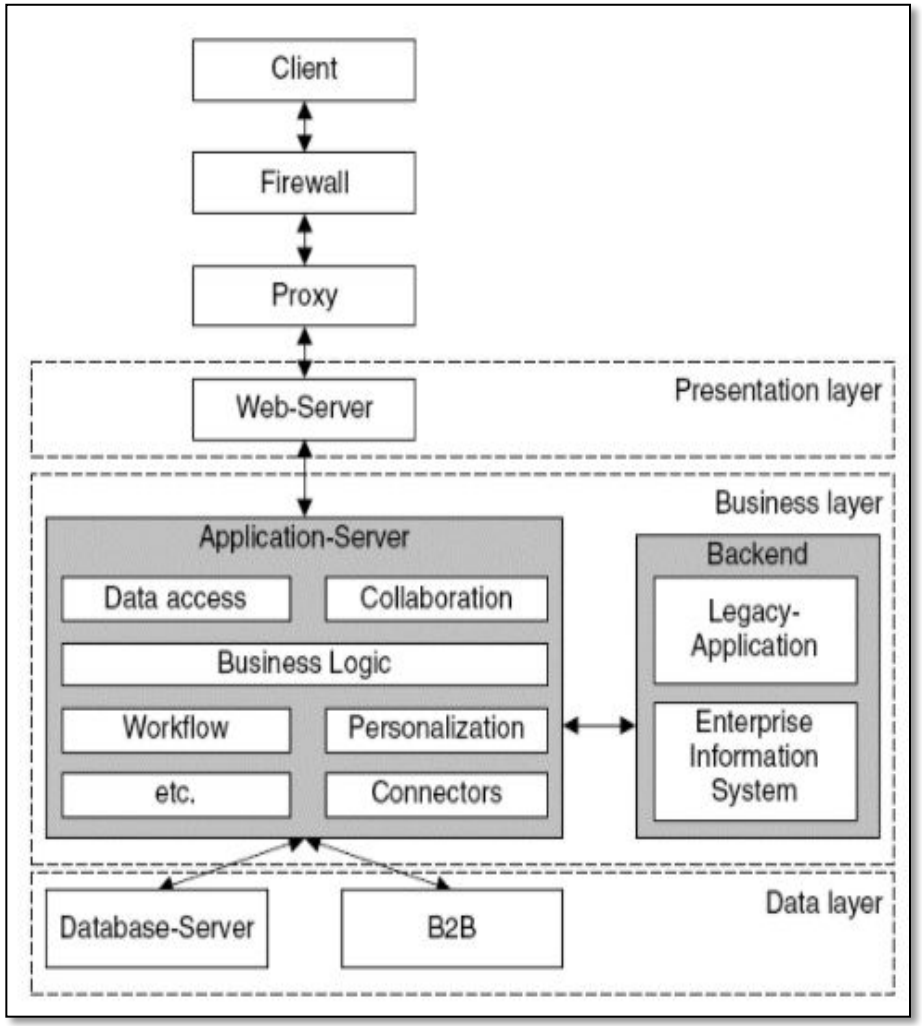


- **Systems in the same domain often have similar architectures that reflect domain concepts**.
- **Application are built around a core architecture with variants that satisfy particular customer requirements.**
  - An IMS must have basic CRUD features. Extra features could be added as per customer requirements.
- So, **decide carefully to which extent architecture can be reused from a similar system**.
- **The architecture of a system may be designed around one of more architectural patterns or 'styles'.**
  - An architectural pattern -> description of system organization.(client-server org.)
- **Selecting a pattern depends upon its weaknesses & strengths**. We will discuss it later in detail.
  - **Pattern selection depends on NFRs as it has a direct impact of software architecture**.

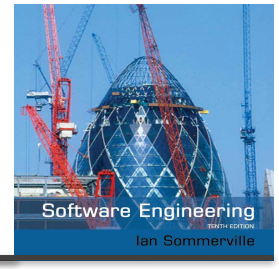
Architecture and system characteristics what to do if following NFRs are critical?



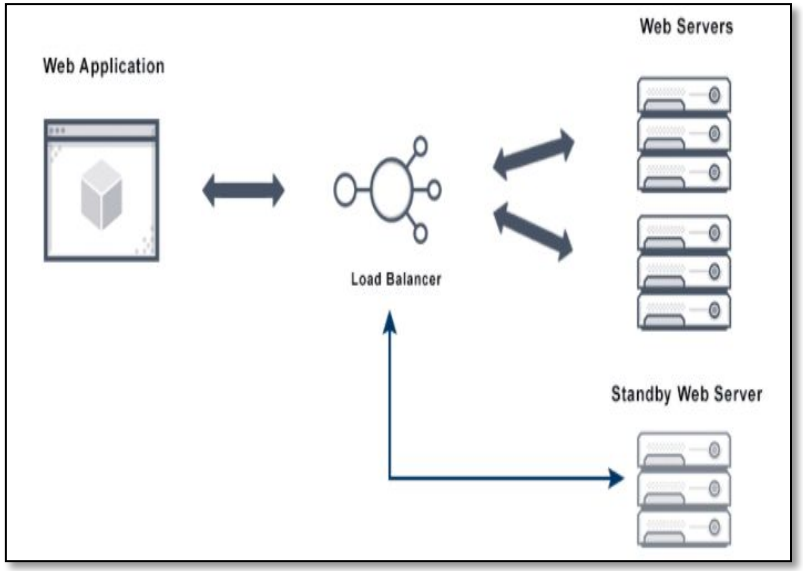
- Performance
  - **Localize critical operations** within a small no. of components **deployed on same system**
  - **Don't use distributed systems** (creates delay in Message passing)
  - **Use large** rather than fine-grain **components to minimize communications.**
  - Effectively **use multiple processors** in your system **for task scheduling.**
- Security
  - **Use a layered architecture with critical assets in the inner layers.**
  - E.g., multi tier architectures



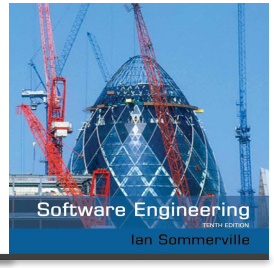
Architecture and system characteristics what to do if following NFRs are critical?



- Safety
  - **Localize safety-critical features in a small number of sub-systems.**
  - Reduces the cost of safety validations
  - and it **has a separate shutdown system that safely shuts down in case of any failure**.
- Availability
  - Architecture should be **designed to Include redundant components**
  - So its **possible to replace/update component in case of fault occurrence.**
  - E.ge., fault tolerant system architectures

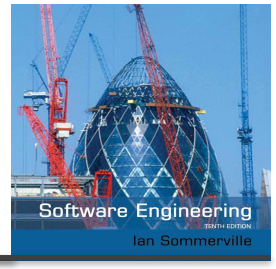


Architecture and system characteristics what to do if following NFRs are critical?



- Maintainability
  - **Design using self grained components that can be changed**.
  - **Shared data structures is avoided** (e.g., static variables are shared objects between multiple instances)

Conflicting critical requirements ?



- Performance improve with lesser no. of components and maintainability improves with more subcomponents.
- Security is the critical req. for almost all applications these days.
- Solution is to either compromise over less critical requirement or use multiple architectural styles for separate parts of systems.