**Design Phase**

**Introduction**

The software project design phase is composed of designing the solution if the current system in terms of logically and physically.

The logical design is composed of designing:

* Form and report
* Interfaces and dialog(splash screen, log on screens, Main menu screens, Input screen, Out put screen)
* Normalization (ERD)

The physical design is composed of:

* Creating database tables and fields
* Hierarchical or system structure design
* Developing the logic of code using the algorithm.

Recalling our discussion of software construction process, once the requirements of a software system have been established, we proceed to design that system. During the design phase, the focus shifts from what to how. That is, at this stage we try to answer the question of how to build the system. The objective of the design process is to analyze and understand the system in detail so that features and constituent components of at least one feasible solution are identified and documented. The design activity provides a roadmap to progressively transform the requirements through a number on stages into the final product by describing the structure of the system to be implemented. It includes modeling of the data structures and entities, the physical and logical partitioning of the system into components, and the interfaces between different components of the system as well as interfaces to the outside world. Sometimes design of algorithms is also included in this activity.

**6.2 Managing Complexity of a Software System**

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems

that have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. To manage the complexity of the system we need to apply the principles of separation of concern, modularity, and abstraction. This leads to designs that are easy to understand and hence easy to maintain. Separation of concern, modularity, and abstraction are different but related principles. Separation of concern allows us to deal with different individual aspects of a problem by considering these aspects in isolation and independent of each other. A complex system may be divided into smaller pieces of lesser complexity called modules. This is the classic divide-and-conquer philosophy – if you cannot solve a complex problem, try to break it into smaller problems that you can solve separately and then integrate them together in a systematic fashion to solve the original problem. One major advantage of modularity is that it allows the designer to apply the principle of separation of concern on individual modules.

**Software Design Process**

Software design is not a sequential process. Design of a software system evolves through a number of iterations. The design process usually involves developing a number of different models, looking at the system from different angles and describing the system at various levels of abstraction. Like the various different models used during requirement engineering domain models, these models complement each other. As stated earlier, software design provides a road map for implementation by clearly describing how the software system is to be realized. A activities performed at this stage include design of the software architecture by showing the division of system into sub-systems or modules, the specification of the services provided by these sub-systems and their interfaces with each other, division of each sub-system into smaller components and services and interfaces provided by each one of these components. Data modeling is also an essential activity performed during the design phase. This includes the identification of data entities and their attributes, relationships among these entities, and the appropriate data structures for managing this data.

**Software Design Strategies**

Software design process revolves around decomposing of the system into smaller and simpler units and then systematically integrates these units to achieve the desired results. Two fundamental strategies have been used to that end. These are functional or structured design and object oriented design. In the functional design, the structure of the system revolves around functions. The entire system is abstracted as a function that provides the desired functionality (for example, the main function of a C program). This main function is decomposed into smaller functions and it delegates its responsibilities to these smaller functions and makes calls to these functions to attain the desired goal. Each of these smaller functions is decomposed into even smaller functions if needed. The process continues till the functions are defined at a level of granularity where these functions can be implemented easily. In this design approach, the system state, that is the data maintained by the system, is centralized and is shared by these functions. The object-oriented design takes a different approach. In this case the system is decomposed into a set of objects that cooperate and coordinate with each other to implement the desired functionality. In this case the system state is decentralized and each object is held responsible for maintaining its own state. That is, the responsibility of marinating the system state is distributed and this responsibility is delegated to individual objects. The communication and coordination among objects is achieved through message passing where one object requests the other object if it needs any services from that object. The object-oriented approach has gained popularity over the structured design approach during the last decade or so because, in general, it yields a design that is more maintainable than the design produced by the functional approach.

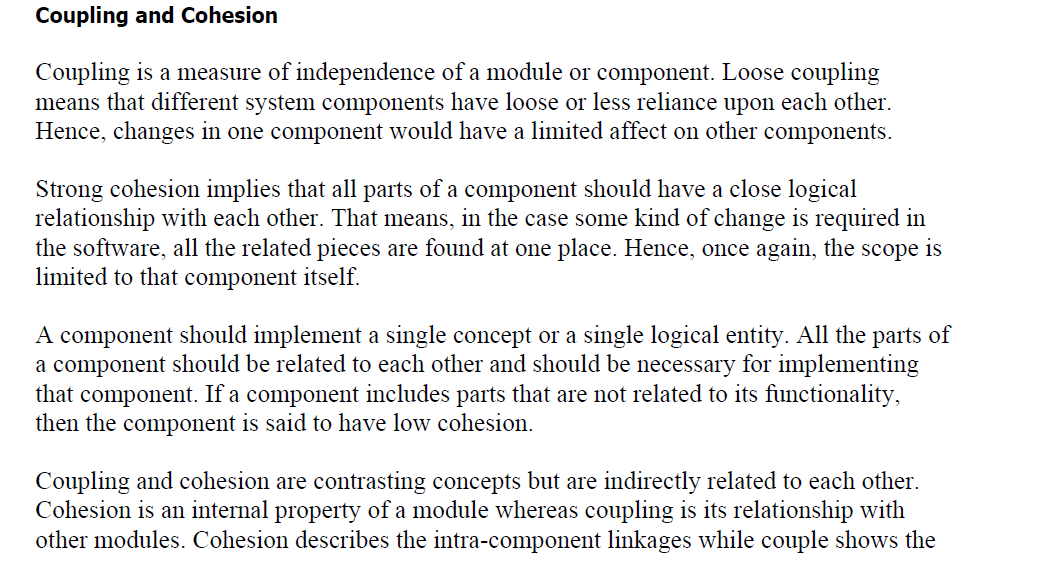
**Software Design Qualities**

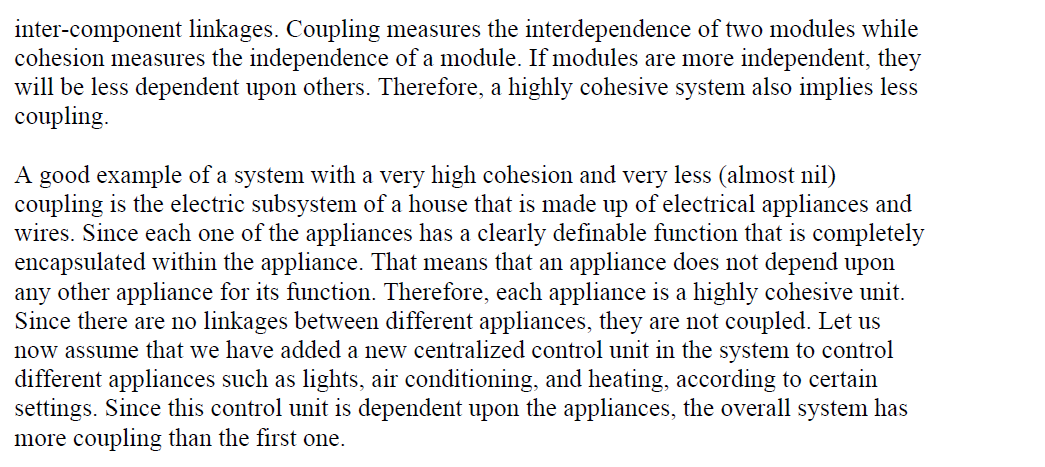
A software design can be looked at from different angles and different parameters can be used to measure and analyze its quality. These parameters include efficiency, compactness, reusability, and maintainability. A good design from one angle may not seem to be suitable when looked from a different perspective. For example, a design that yields efficient and compact code may not be very easy to maintain. In order to establish whether a particular design is good or not, we therefore have to look at the project and application requirements. For example, if we need to design an embedded system for the control of a nuclear reactor or a cruise missile, we would probably require a system that is very efficient and maintainability would be of secondary concern. On the other hand, in the case of an ordinary business system, we would have a reversal in priorities.

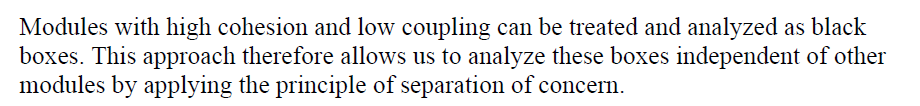
**Maintainable Design**

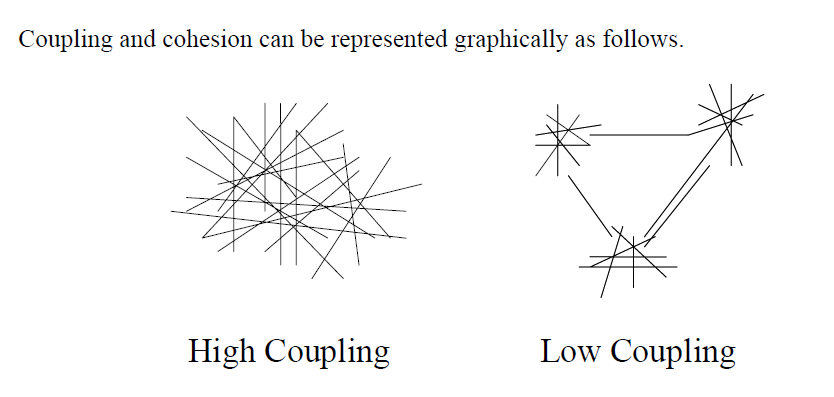
Since, in general, maintenance contributes towards a major share of the overall software cost, the objective of the design activity, in most cases, is to produce a system that is easy to maintain. A maintainable design is the one in which cost of system change is minimal and is flexible enough so that it can be easily adapted to modify existing functionality and add new functionality. In order to make a de

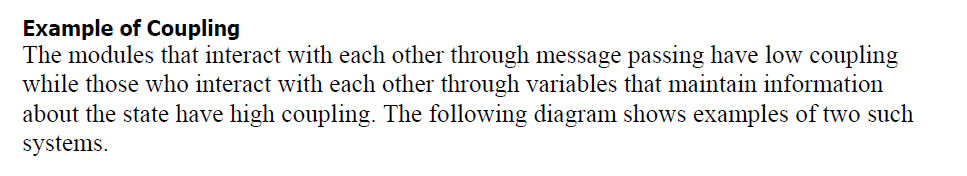
sign that is maintainable, it should be understandable and the changes should be local in effect. That is, it should be such that a change in some part of the system should not affect other parts of the system. This is achieved by applying the principles of modularity, abstraction, and separation of concern. If applied properly, these principles yield a design that is said to be more cohesive and loosely coupled and thus is easy to maintain.

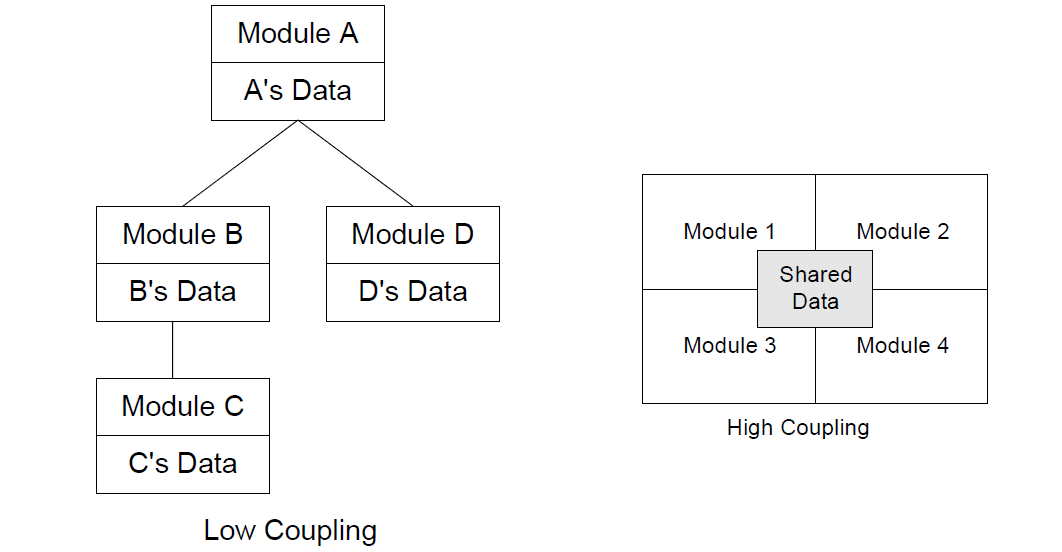


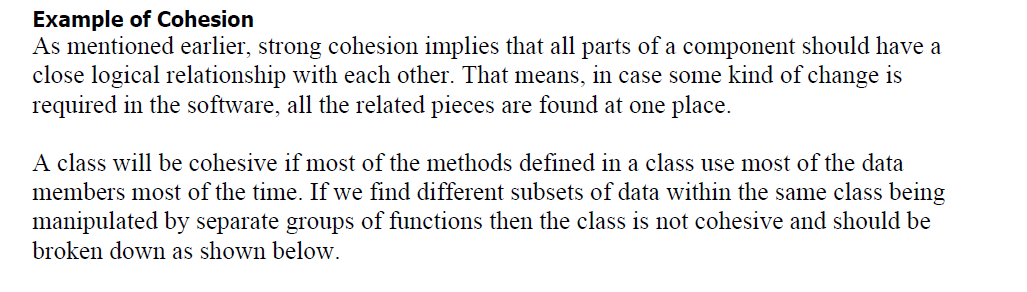


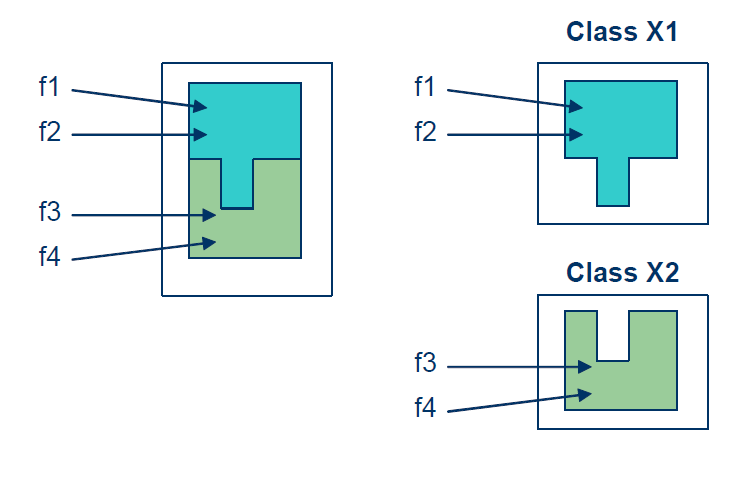


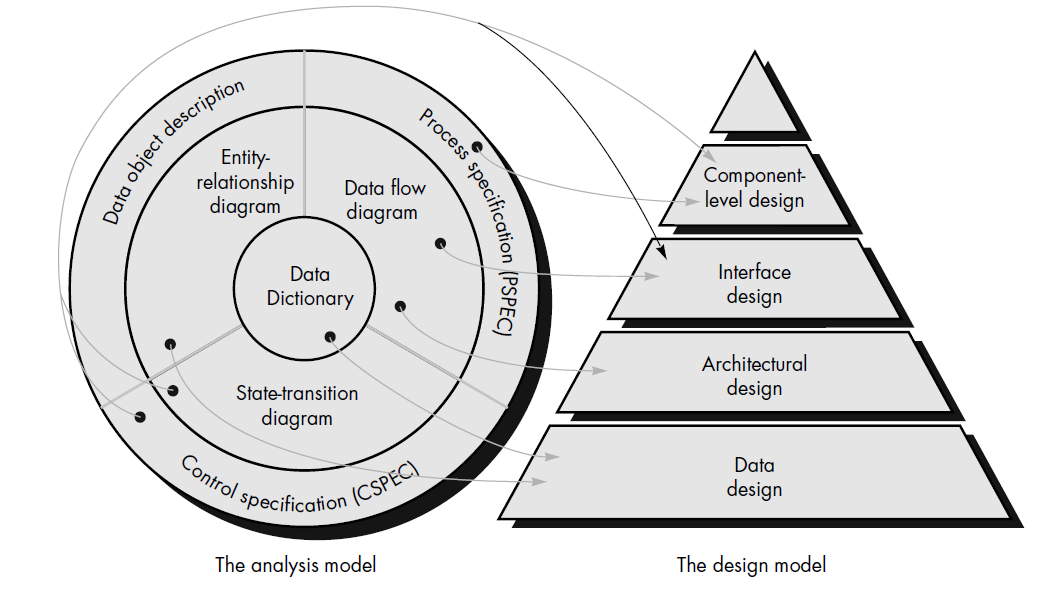




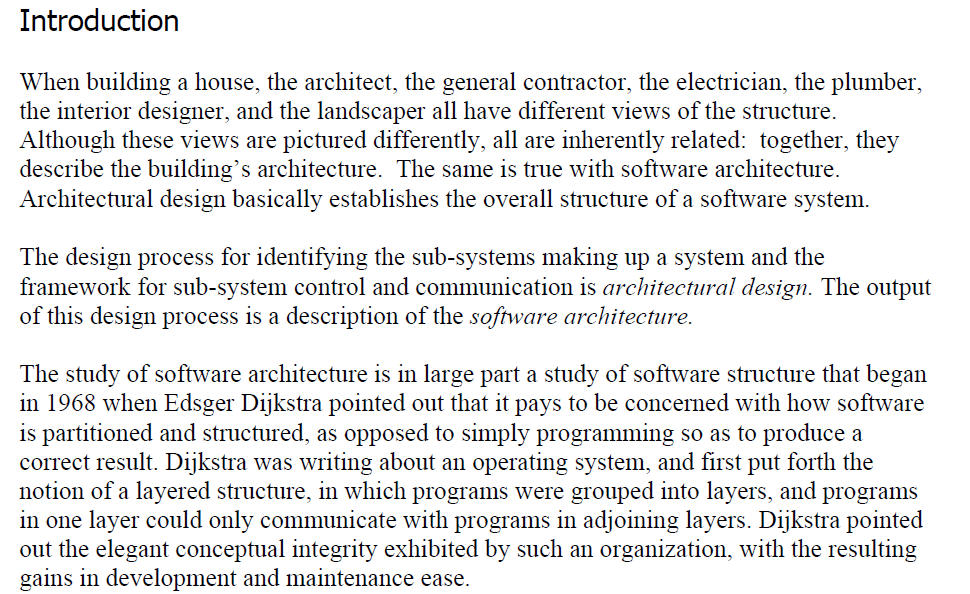






****

**Software system and Architecture**



Object Identification system

Conveyor Controller

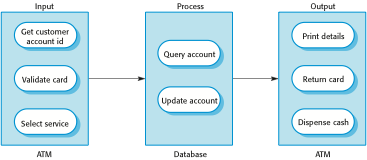
Vision system

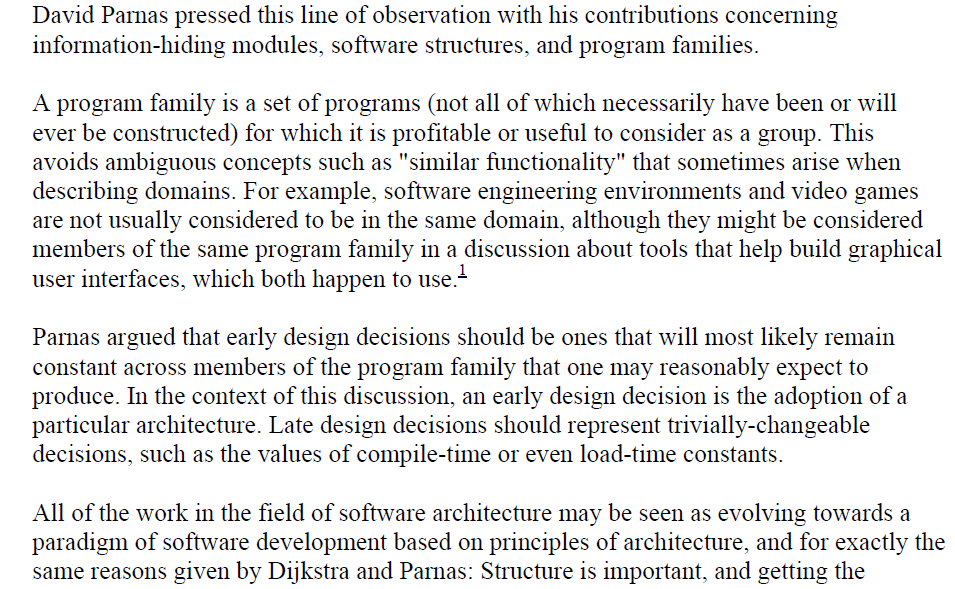
Packing system

Packaging selecting system

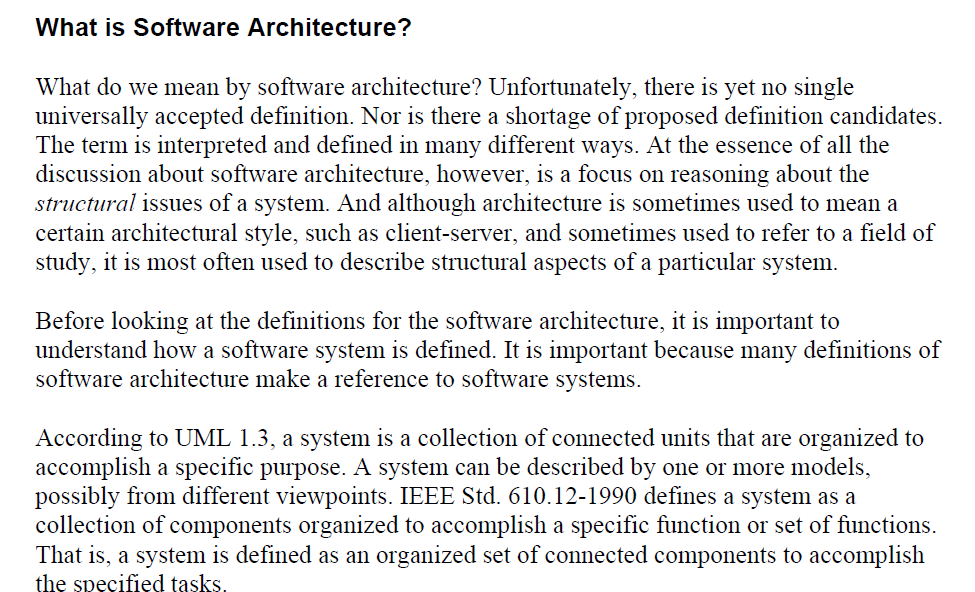
Gripper Controller

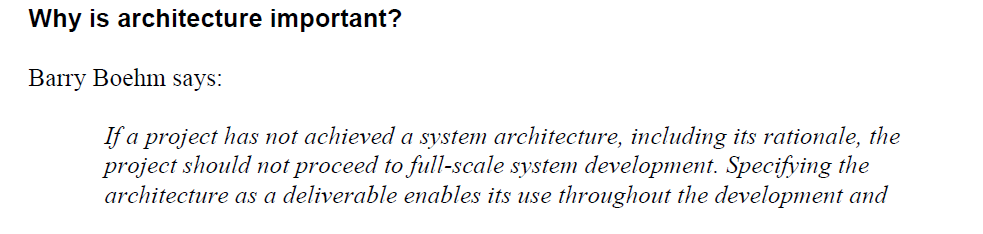
Arms Controller

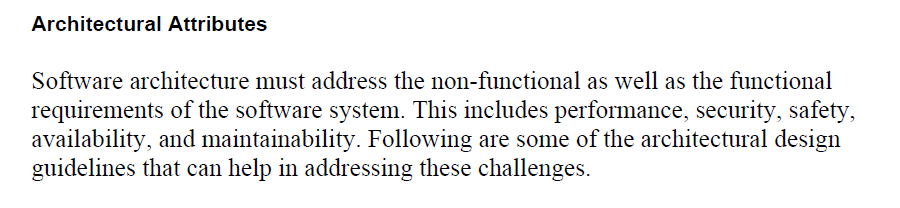


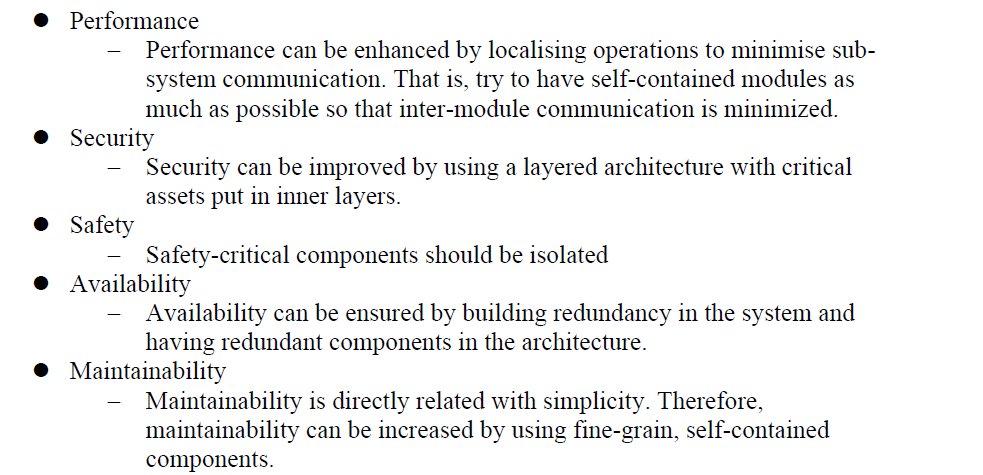


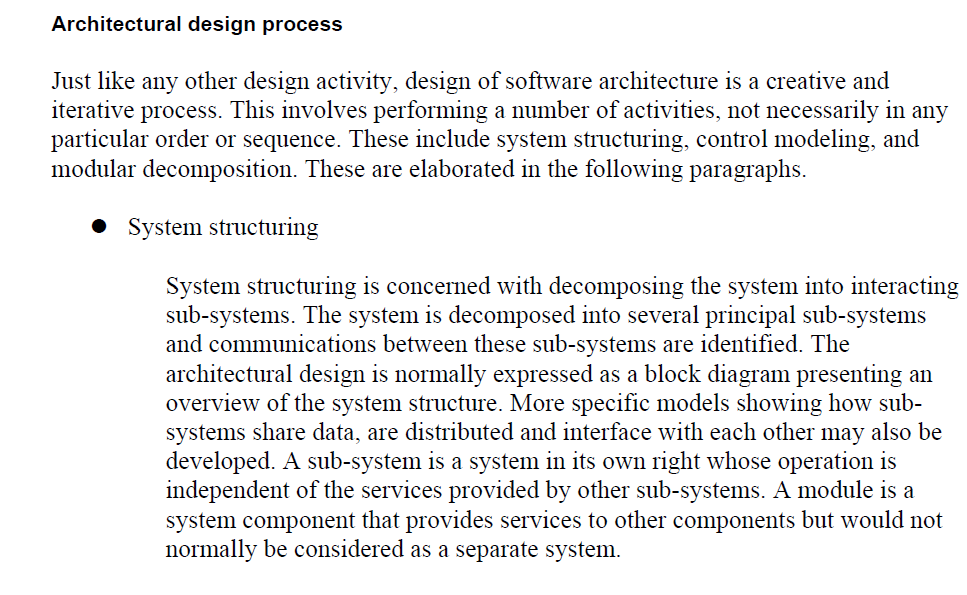


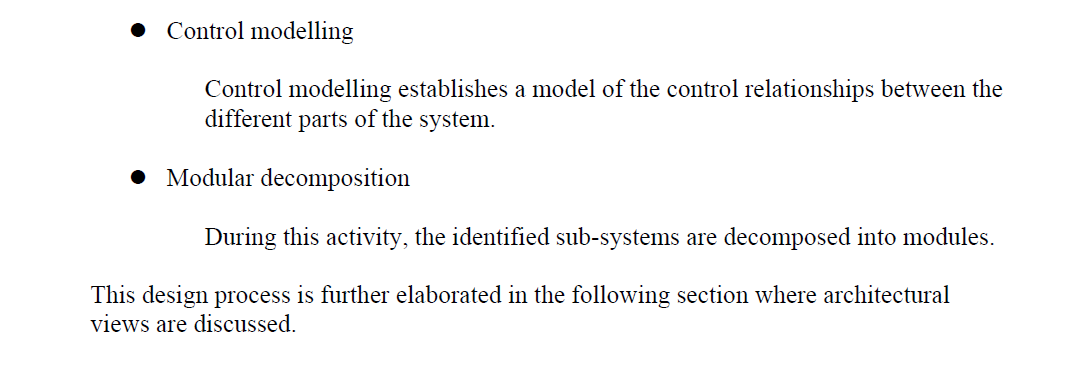


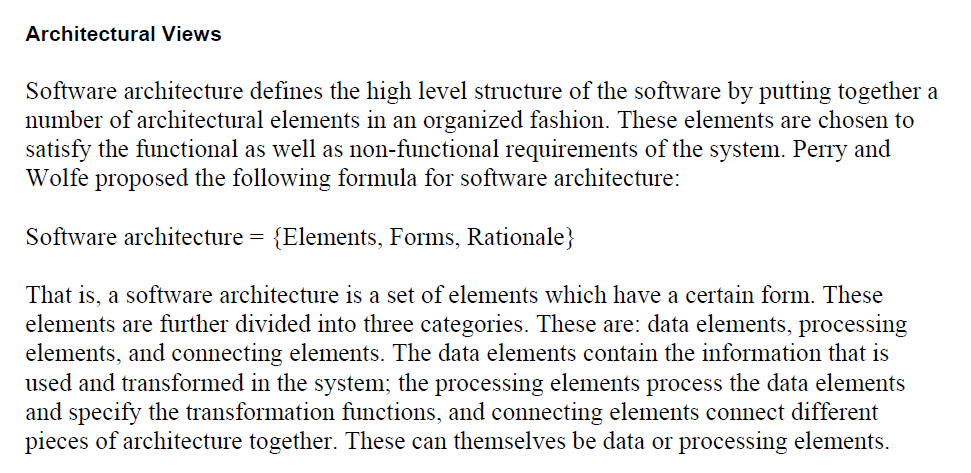


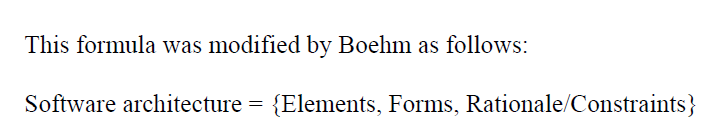


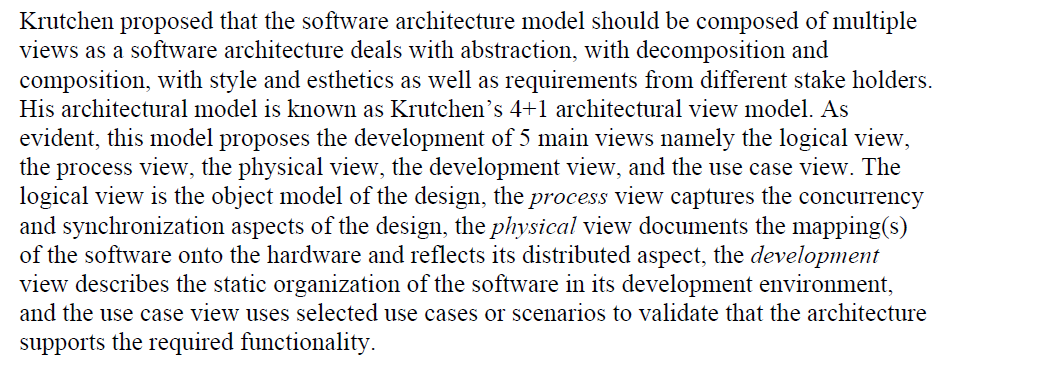


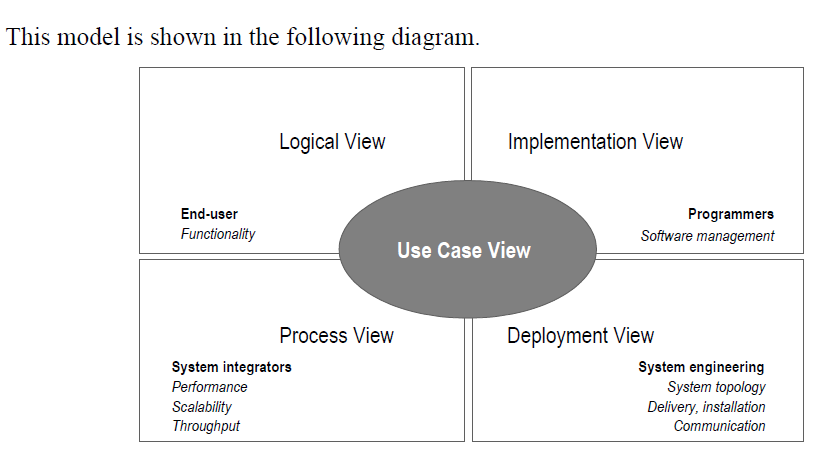




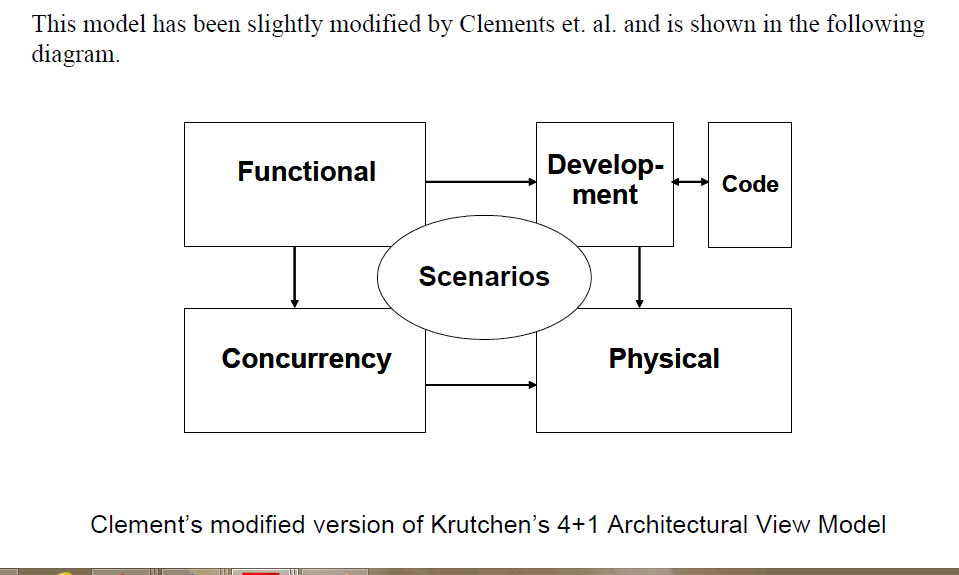


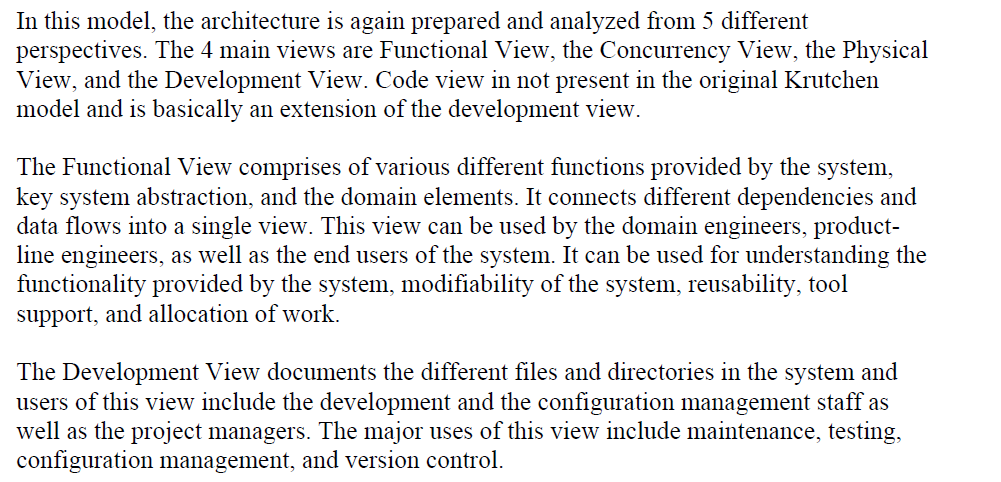


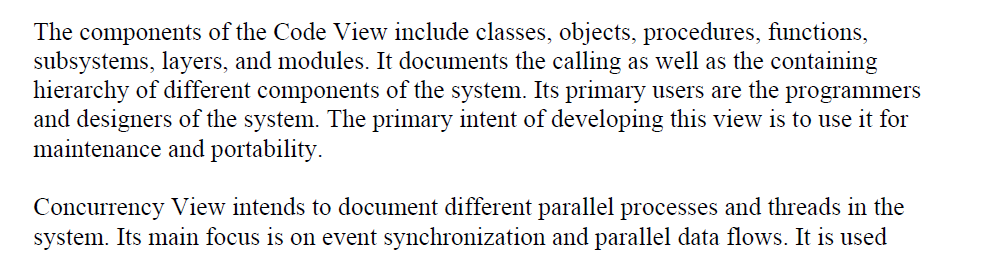
****

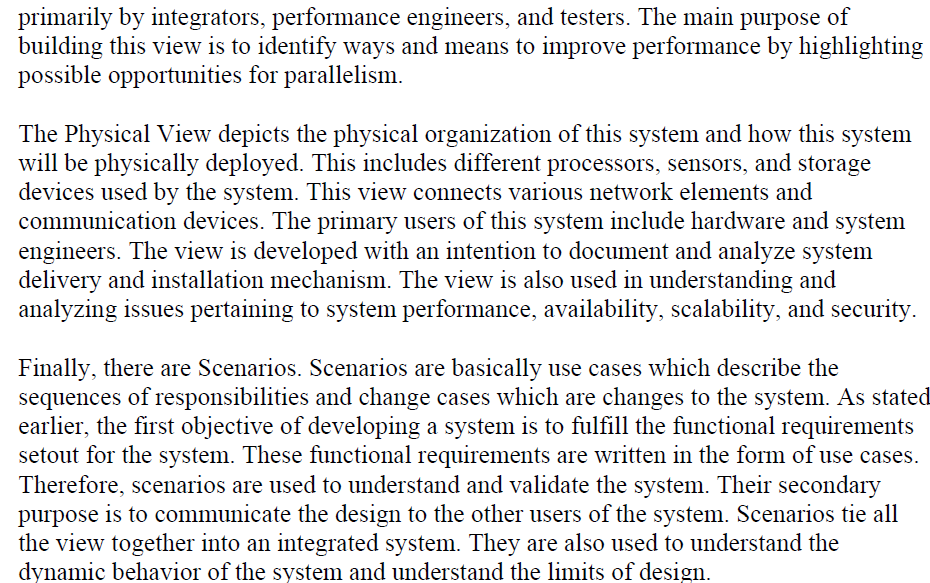


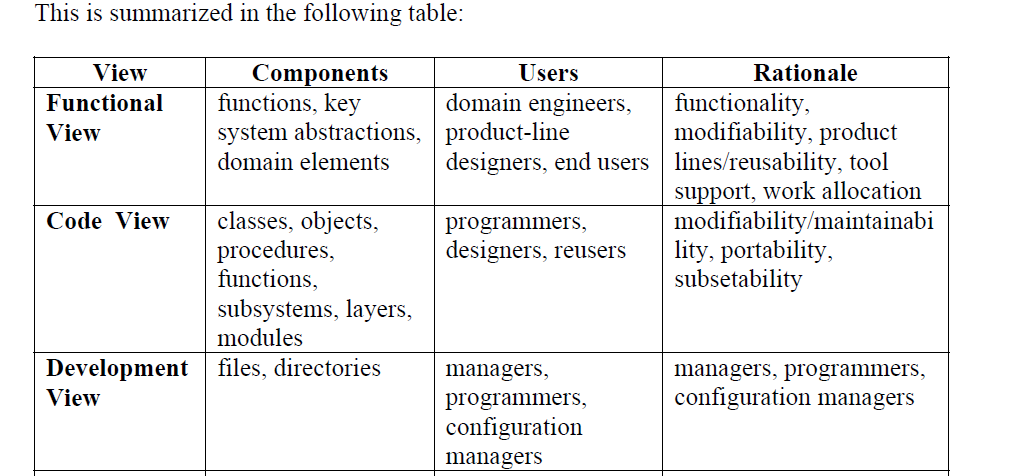
Conceptual Physical

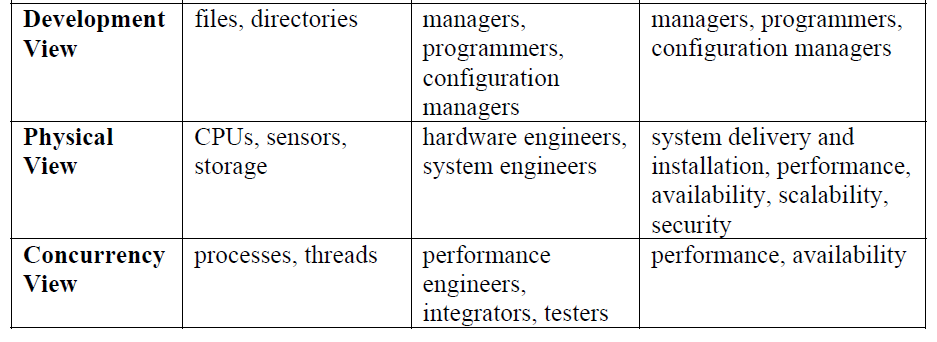


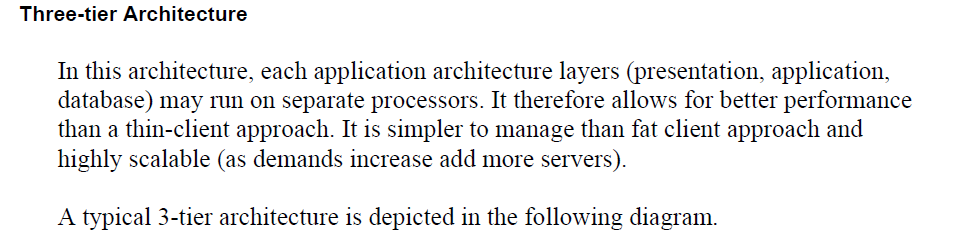


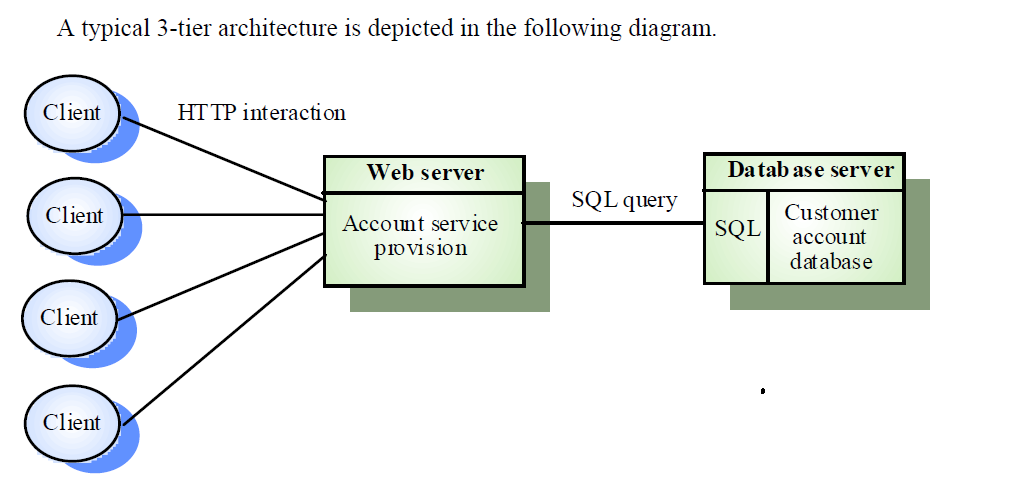


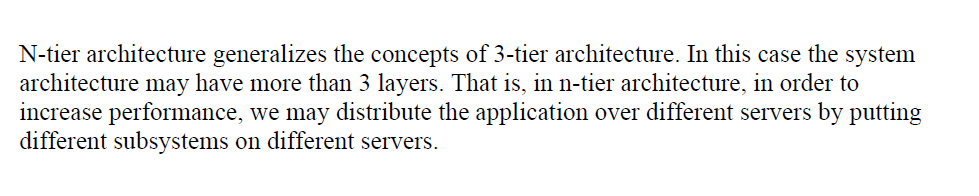


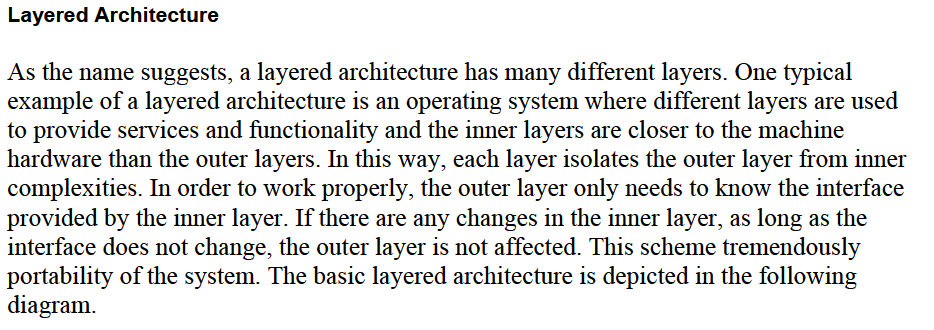


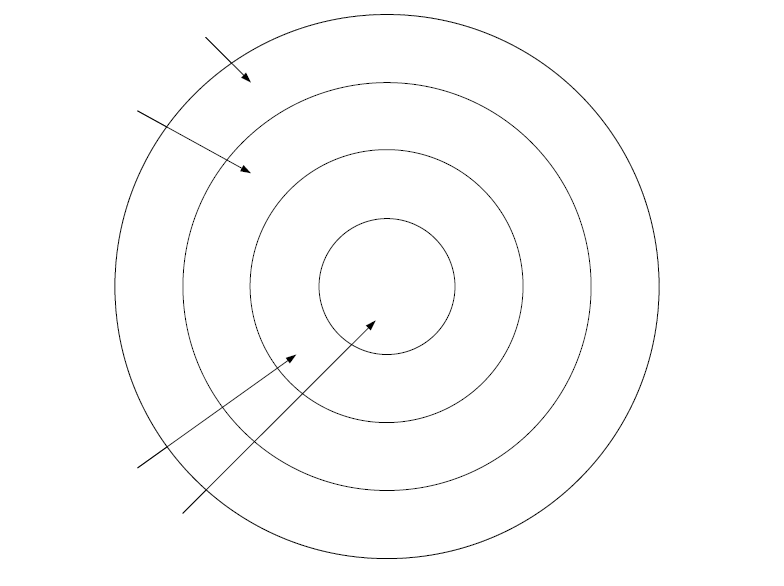


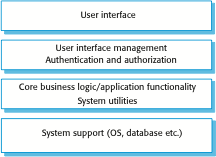










Example 1

Example2

