Chapter 17: System Design Process

Upon acceptance of the system requirement specification (SysRS), the customers tentatively freeze the requirements. By this time the system developers are expected to have understood these requirements, and design of the system should begin. The IT project manager establishes the system design process. The system designer develops the system architecture to understand the requirements and partitions the system requirements between software and hardware requirements.

System Design

System design begins from the baseline of system requirements. The building of a system requires identification of hardware for which the system software will be developed. The requirements for the system design will be derived from the SysRS (Figure 17−1). Identification of hardware depends on the customer, who provides the necessary tools and instruments that will be used by the system developers. This process assists the system developers so that the system is tested in accordance with the requirements. Further discussion of hardware is beyond the scope of this book.

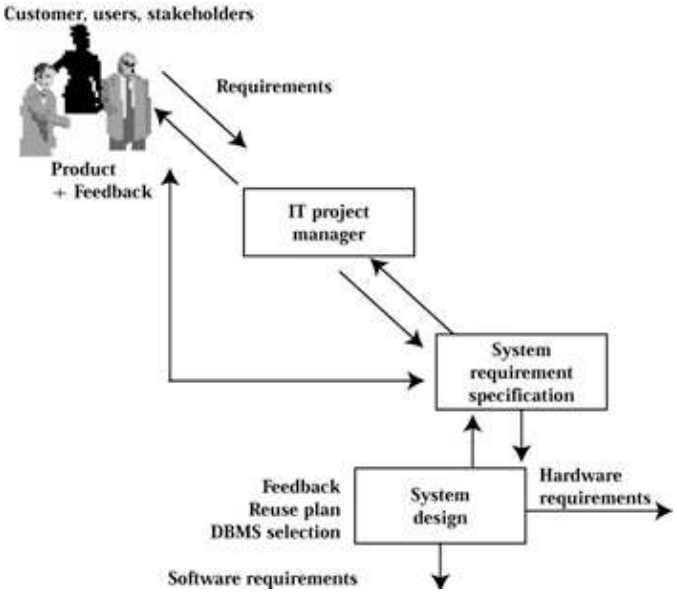

Figure 17−1: IT system design process

Identification of software and hardware depend on whether it is a new development, a reuse of the available existing system, a reengineering of the system, or a reverse engineering process. Commercial off−the−shelf (COTS) software and hardware are available that can suit the requirements. The knowledge and experience of the system or domain expertise play a major role in this selection decision. The domain expert can explore on the Internet for reuse assets. The selection must be the right assets for the right requirements.

- What the system must be capable of accomplishing
- How well the system product must perform in quantitative terms
- The environment in which system products must operate
- Human and system interface requirements
- Constraints that affect design solutions

System Design Characteristics

System design consists of requirements for hardware and software. The requirements have the following characteristics:

- Are unique and complete
- Do not overlap and are not duplicated
- Are necessary in the development of system engineering
- Are feasible and implementable
- Must not be outside the capability of current technology
- Must be consistent and stated unambiguously
- Must be evaluated
- Must have a feasible way to determine that the requirements have been satisfied and are traceable forward and backward
- Must state what will be implemented, and system design must state how to implement them

Three types of requirements are generally identified from the SysRS, which is the systems functional baseline:

- Explicit requirements are part of the functional baseline.
- Implicit requirements are implied by nature or by special knowledge of the system although not stated or expressed in the functional baseline. Implicit requirements may include requirements that are derived from a specified technical understanding of the system.
- Derived requirements are also unstated in the functional baseline but may be deduced from the structure, relationship, or nature of the system.

The system designer correctly and seamlessly maps these requirements in the system design.

Configuration Item Selection Criteria

The system designer identifies computer software configuration items (CSCIs) and hardware configuration items (HWCIs). Selection of a CSCI and HWCI is difficult. The software developer must conduct a proper analysis and select one or more CSCIs. The selection criteria depend on the following factors:

- System requirements
- Hardware
- Requirements allocation
- User acceptance
- Schedule
- Cost

The system requirements are partitioned and allocated to the lesser number of CSCIs for effective cost and time. The greater the number of CSCIs, the more work is required in the creation of the volume of documents, which further leads to more manpower needed in document checking. These schemes need more time and cost. The ideal number of selections is from one to three CSCIs.

Allocation of Requirements to CSCI

The system designer determines the best allocation of software requirements to a CSCI. The system designer

defines a set of software requirements for each CSCI by showing the system architecture and its interfaces. The designer explores the operational concepts of the system and discusses its functional and operational environments.

Figure 17−2 illustrates a sample system architecture, representing the allocation of system hardware and software resources. Graphically, it represents the picture of the system when built. It is the top−level model that shows all of the interfaces with external entities. The context of the system is presented graphically. If more than one CSCI has been selected, the system architecture should be shown for each CSCI for clarification. The designer should give meaningful names to the CSCI and interfaces, carry out the names throughout the system software development, and document these names in the system data dictionary. This top−level graphic becomes the foundation of the system software development.

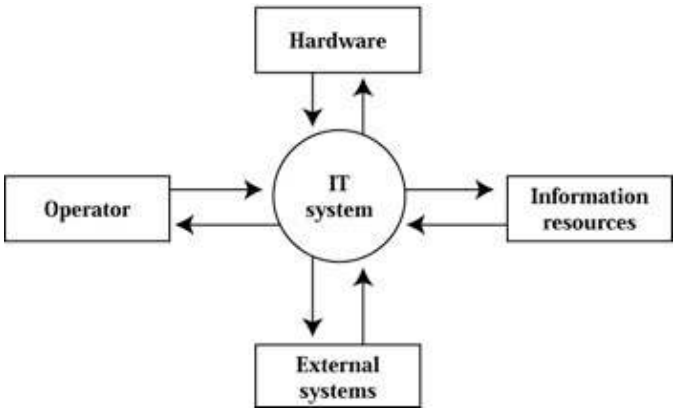

Figure 17−2: A system architecture

The designer must segregate the requirements between automated and manual. Only the automated requirements will be allocated to the CSCI for software development. The automated requirements also consist of environmental and operational requirements. All of these requirements are derived from SysRS.

IT System Architectural Model

The IT system architectural model is a generic picture of related systems in the domain. The architectural model is a plug and play, an open architecture designed around a client/server model. Figure 17−3 shows a sample of a generic domain architectural model. The model consists of a standard reusable environment, a set of standard off−the−shelf reusable components, and a set of reusable programming standards that describe how to add new functionality to the environment. The model is designed to run on hardware platforms such as personal computers (PCs) and workstations. It extends the Windows paradigm, which allows applications to coexist by providing the capability of applications to share data and services and functions at the server level.

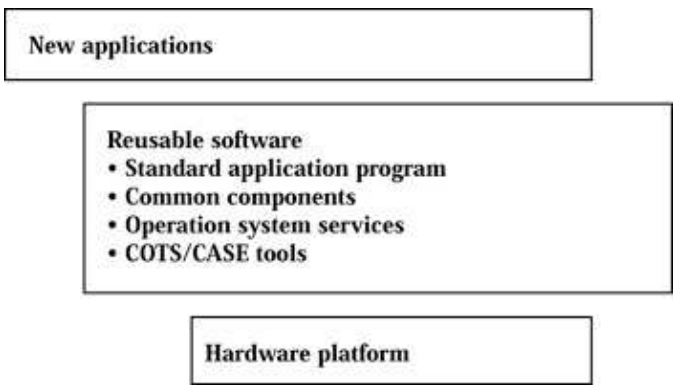

Figure 17−3: A sample of a generic system architecture model

Conventionally, software developers build the common function from scratch for every system. Each developer implements the functions differently, which leads to incompatibility between the related systems. Thus most of the production systems have similar functions, but they cannot easily interoperate. Surprisingly an organization is investing a great deal of time and money in rebuilding the same functions.

The concept is to construct an architecture model for building interoperable systems, an infrastructure for supporting related applications. The model facilitates an automated process for software integration and tools, provides a collection of reusable software components, and provides an approach and methodology for software reuse.

Exploration of Reuse Assets

Exploration of reuse assets for the system via the Internet is a good management technique. Reuse of well−tested components saves cost and time in the systems development. A good manager identifies the domain at early stages of the project for maximum use of reuse assets. Conducting a thorough analysis of existing systems in the organization can identify the domain or domains in the organization. Some systems may be unique, and others may be interrelated. A domain may be made up of subsets, referred to as *subdomains*. The designer should scope the domain and identify a whole domain and its subdomains.

A domain is a group or family of related systems. All systems in that domain share a set of common capabilities and/or data. The domains identification for a system is an important issue; the domain should be within the systems environment and boundary. The system developer selects a domain expert, an individual who has good knowledge of the domain and its subdomains. The domain expert explores candidates for system reuse components (i.e., subdomains) as he or she progresses in all phases of the systems development. The selection of a domain and its subdomains successfully leads to identification of system reuse components during the domain analysis.

Establishment of a System Reuse Plan

Establishment of a system reuse plan (SRP) is important to identify various phases, activities, roles, schedules, and project organizations. The purpose of an SRP is to provide guidelines and a road map to conduct day−to−day business for the system reuse center in a business organization, which consists of necessary events, actions, and processes to operate a reuse center in an organization. This includes responsibilities of various sections, practitioners, and management. The designer should only consider the plan as a guideline to follow during the development life cycle. He or she should not consider it to be untouchable. To quote a famous saying, Plans are prepared to be changed. The SRP shall contain the following:

- SRP purpose
- Various responsibilities and duties
- Operation and maintenance of the reuse center
- Customer and user participation
- Configuration management (CM)
- Quality assurance (QA) and control
- Training
- Repository system establishment and maintenance
- Reuse life−cycle phases

SRP Purpose

The purpose of the SRP is to define clear guidelines for practitioners to practice reuse in the systems development. An experienced manager establishes a reuse center and forms a team of domain experts who define the purpose, various responsibilities and duties, customer and user participation, CM of reusable assets, QA and quality control, training, repository system establishment and maintenance, and reuse life−cycle phases.

Various Responsibilities and Duties

Various responsibilities and duties relate to domain architecture, domain analysis and design, establishment and maintenance of a repository, networking, training, and management of a data review board. These include domain asset engineers, domain repository administrators, domain system integrators, domain assets quality administrators, domain assets testers, domain assets CM, and support groups.

The domain manager is an expert in the domain and management. He or she is responsible for management of the team. The domain manager ensures that the repository is functional and assets are tested in accordance with the software reuse plan. He or she is an expert in designing and maintaining the domain infrastructure architecture. He or she is responsible for reporting the progress of the reuse project to higher management. The managements commitment is important for the success of the reuse center.

The *domain engineer*, or the domain expert, conducts domain analysis and identifies and builds assets for reuse. He or she uses customer requirements to analyze and find out desirable reusable assets. The domain engineer explores other repositories on the Internet and selects from an availability of assets for the domain. His or her role is to support and satisfy the clients needs. The domain engineer also maintains domain infrastructure architecture.

The *domain analyst* performs domain analysis and identifies, collects, organizes, and represents all relevant information in a domain. This information is based on the existing systems in the domain; knowledge collected from domain experts; underlying policies, procedures, and standards; and state−of−the−art technologies.

The *domain designer* identifies a reusable asset, which is suitable to its domain. He or she has access to other repositories via the Internet and Intranet. He or she explores and researches for the right asset at the right time. The domain designer maintains a description of the available assets in the repository to be accessed by developers. Each asset has a unique identifier, and each interface is documented in the repository with proper description, protocol, constraints, and names of systems that are reusing it.

The description covers hardware, design, and programming. It also includes the processing resources and characteristics of its use. These characteristics include the following parameters:

- Memory size (amount of internal memory [absolute, spare, or both] of the computer)
- Word size (number of bits in each computer word)
- Processing speed (computer processor capacity [absolute, spare, or both])
- Character set standard
- Instruction set architecture
- Interrupt capabilities of the hardware
- Direct memory access
- Channel requirements
- Auxiliary storage

- Growth capabilities of any part of the processing resource
- Diagnostic capabilities
- Allocation of pertinent processing resources to each CSCI

Operation and Maintenance of the Reuse Center

This is the responsibility of the domain manager and his or her staff. The manager delegates duties and responsibilities to each member of the staff to perform certain tasks. The manager supervises the performance of his or her staff.

Customer and User Participation

This is an important factor in the success of the reuse effort since practitioners generally like to be involved in all reuse activities. Their input is important because the customer and users are the ones who use the reuse assets. They should be informed on a regular basis about the progress of the reuse center. They have proper access and services that are provided by the reuse center. There may be a fee involved for certain services.

Configuration Management

CM is the process of managing the configuration assets with a proper unique number scheme. These assets in the CM are accessible by practitioners for reuse. The CM is responsible for processing and validating submitted CM information regarding the potential, modified, or reengineered assets. All reusable assets are configuration managed. The CM personnel are the only ones with write privileges to the CM database.

Once a candidate asset is accepted and adapted for reuse, it is placed into CM control. The CM−controlled assets are changeable only if an anomaly has been discovered or the asset has been modified in use. Any asset anomalies discovered are identified with a fault report and submitted for proper analysis and disposition. The data review board for asset reuse consists of representatives from various diversified fields and systems. This board decides the course of action for the identified anomaly. It certifies and submits the corrected asset to the CM for update in the CM database. The process of building systems identifies reusable assets and retrieves them from the CM database. The CM tracks the assets that are modified and incorporated into systems via the software change order.

Quality Assurance and Control

This ensures that the reuse processes and the policy, procedures, and standards are implemented and followed. The quality section monitors, audits, and checks the correctness of various reuse documents, models, functions, and activities. They verify and validate the reuse assets.

Training

The reuse center may provide training as a service to customers and users. The training may cover topics such as repository usage, reuse assets management, reuse basics, software development with reuse, domain analysis, and design processes. The training also includes the operation and reuse of the repository. The reuse center develops and maintains a repository users manual.

Repository System Establishment and Maintenance

This is the responsibility of the software reuse center team. One of their major tasks is to select the best repository system suitable to the domains needs. The team maintains the repository to ensure that it is functional and operational at all times. The repository is easily accessible by the customers and users.

Reuse Life−Cycle Phases

These involve selection and certification of reusable assets. The phases populate the repository with the certified reusable asset with proper description and identification. The customers and users should know about the new asset. All reuse assets accepted follow the ritual for the adaptation and certification process before submitting to the CM. The reuse life−cycle phases are as follows:

- Delineation indicates how well the reusable asset is described in documentation, comments, and conformance to the policy, procedure, and standard.
- Relevance indicates how well the reusable asset matches the requirements of the functional and application domain.
- Coherence indicates the overall design quality.
- Connectivity indicates how well the reusable asset connects to other reusable assets comprising the functional and application domain.
- Inerrancy indicates how error−free the asset is and how well it has been tested.
- Mobility indicates how easily the reusable asset can be ported to new platforms and environments.

Case Study 17−1

This case study involves identification of the domain and subdomains associated with maintaining a personal checkbook. The domain is Financial Accounting (Figure 17−4). Existing subdomains must now be identified that are relevant to the task of maintaining a checkbook. Applicable subdomains include functions such as bookkeeping, accounts payable, accounts receivable, and taxable items (Figure 17−4 depicts this domain and subdomains relationship). Each subcomponent may be reused, possibly resulting in savings of money and time. In terms of software code, interfaces may have to be written if they are not yet available to glue those reusable assets.

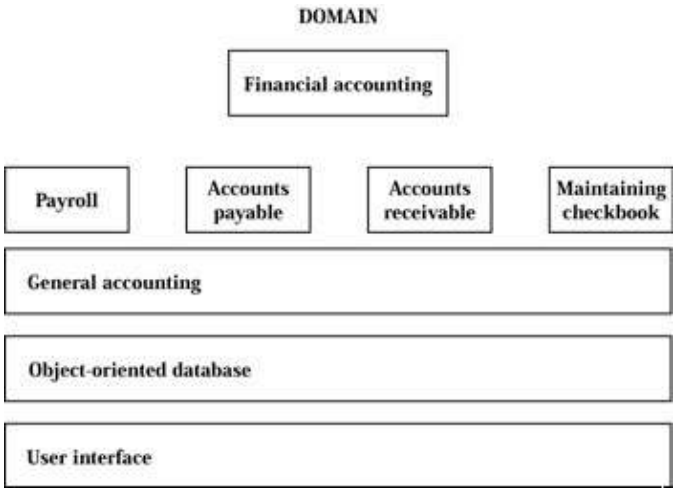

Figure 17−4: Financial accounting domain and subdomains

Domain models represent a graphic presentation for software development. The reuse process is to understand the customers requirements and then graphically show them by logical models. Domain models determine objects and establish a relationship between other objects within the domain infrastructure.

Selection Criteria of a Good Database Management System

The selection criteria of a good database management system (DBMS) depend on application requirements, budget, and vendor support. The following factors enhance the selection criteria:

- Performance
- Ease of use
- Migration
- Data modeling capabilities

Performance

Performance of a DBMS depends on the way in which it handles the manipulation of complex data models. Manipulation of stored data should be as fast as possible and within the allocated budget and time. The DBMS operation requires a minimum overhead cost per operation. Overhead is related to pointer dereferencing (object traversal), locking, network access, and disk access.

High performance data manipulation requires that if a pointer or reference to an object is given, then the operation of obtaining data from that object must not incur undue overhead. This operation is called *dereferencing* a pointer.

Objects can be locked on a segment of the database (i.e., a page) or in a single configuration that the user defines. Only a single lock action is needed for an entire page, segment, or configuration of objects, which provides significant performance gains over other locking mechanisms.

Data caching occurs when a sequence of transactions accesses the same objects. A high probability exists that the data, which are accessed in the next transaction, will already be cached in workstation memory. Since network accesses are expensive operations, the reduction of the number of data transfers is critical to the success of a distributed DBMS. Batching objects that will be transmitted between the client and server are typically only one network request and are used when a group of referenced objects are sent into the client cache. This strategy minimizes network traffic and allows the client to proceed through a transaction without contacting the server until a transaction is committed.

Often an application uses only a portion of a database, and that portion can be stored contiguously in a small section of the database. Related objects can be clustered together in a database. This clustering increases performance because disk access time for contiguous data is faster than random access. The disks can typically read or write many sequential blocks in the same time required for disk head to move to a random location.

Ease of Use

Ease of use is in the integration between the database system and the host programming language. No inheritance exists from a special persistent object base class. Different objects of the same type may be persistent or transient within the same program. This concept dramatically improves developer productivity and code reuse. Ease of use further leads to ease of learning. In object−oriented languages, the types,

variables, and classes should be declared in the same way. Similarly, objects manipulate, access variables, set variables, and call functions, all of which should all be managed in the same manner.

No translating code is recommended in the ease of use. The programmer should not write code that translates between the disk resident representation of data and the in−memory representation that is used during execution. This is referenced to as single level storage. In contrast, developers who use the file system storage model must create a linear on−disk representation for their objects and write code that will map its in−memory representation.

Migration

Migration is a convenient way for the application and customer data to be converted into existing code, data−stored files, and libraries. Many programmers who are interested in the use of a DBMS should add persistence to existing applications that deal with transient objects and new applications that can be built from scratch. This is possible because basic data operations, such as dereferencing pointers and getting and setting data members, are semantically the same for persistent and transient objects. Variables do not need their type declarations changed when persistent objects are used.

Data Modeling Capabilities

Data modeling capabilities mean that the DBMS data model is designed so that developers can use the full power of the language for application development. With C++ and Ada, this means full support for the complete language and includes virtual functions, inheritance, polymorphism, encapsulation, and parameterized types. Additional data modeling features include the following:

1. Collections
2. Queries on collections
3. Relationships between objects
4. Versioning of objects

Collections

Collections are objects that group together other objects. Collections are abstract structures that resemble arrays in programming language or tables in a relational database. Like C++ arrays, collections store many instances of the same type or a derived type. Users may optionally describe intended usage by estimating frequencies of various operations (e.g., iteration, insertion, and removal), and the collection library will automatically select an appropriate representation. Furthermore, a policy can be associated with the collection and dictate how the representation should change in response to changes in the collections cardinality. These performance−tuning facilities simplify the developers interaction from the coding of data structures to the description of access patterns.

The collection classes for insertion, removal, and retrieval of elements of a given collection also provide methods that perform set theoretic operations, such as union and intersection, and set theoretic comparisons, such as subset. Often objects have embedded collections (e.g., a person object may contain a set of children). Collections also store all objects or a subset of all objects of the same type (e.g., all employees or managers). Such collections can be arbitrarily large. Access patterns differ greatly among applications and even over time within a single application. A single representation type will probably not provide adequate performance for all such access patterns of support for multiple representations of collections.

Queries on Collections

Queries on collections provide functionality similar to querying a relational database in that the programmer may not necessarily know how a particular object is found by following pointers. Instead, the programmer provides a query expression that selects the target object or objects that are based on the values that are contained in it or on the relationships between it and other objects. Many query languages lack the support of multiattribute, multiple condition, class extents, and distributed query functionality.

The best approach treats queries as ordinary expressions in C++ or C. A good DBMS should provide indexes that permit more efficient implementations. A query optimizer examines a variety of strategies and chooses the least expensive way for the execution of a query. The DBMS can index paths through objects and collections and not just fields directly contained in objects. In complex applications in which speed is crucial, pointers and embedded collections obviate the need for joins. Therefore a typical query will likely be over a small number of top−level collections. Selection predicates involve paths through objects and embedded collections. These paths express the same sorts of connections as those that join terms that are expressed in relational queries. Join optimization is of less concern since the path has materialized in the database with interobject references and embedded collections.

Relationships Between Objects

Relationships between objects are useful when complex objects are modeled, such as designs, parts hierarchies, documents, and multimedia information. Each relationship is composed of two or more objects that are

constrained and consistent with one another in a particular fashion. The user declares the constraints on the data members who comprise the relationship.

Versioning of Objects

Versioning of DBMS objects stores and manipulates multiple versions of an object simultaneously. This is useful for modeling many activities, such as writing a document or developing a design, in which iterative processes require experimentation with various modifications. A user can check the version of an object or group of objects, make changes, and then check the changes back into the main branch of project development. In the interim, other users continually use previous versions and therefore are not impeded by concurrency conflicts on their shared data regardless of the duration of the editing sessions that are involved.

Types of Data Modeling Capabilities

- Configurations provide a means of grouping related objects that should evolve together. An object that is composed of several objects, such as a document, may constitute a unit for versioning purposes. Configuration provides the user with specified granularity of both evolution and concurrency control. Since versioning is logical multiple representations of the same thing, a mechanism must allow a process to look at a distinct version.
- Workspaces provide a private work area so that one or more processes can view a configuration. For each application area, the system designer must determine what types of workspaces must be created. Users can then employ workspaces that selectively share their work in progress. Workspaces can inherit from other workspaces. A designer can specify that a workspace should, by default, inherit whatever is in the teams shared workspace. Individual new versions can be added as overriding of this default makes changes.
- Versioning history graphs provide a model that allows the designer and developers to visualize the evolution of versions. These graphs keep track of the different versions as the entity or entities evolve

through time. The versioning and persistence of an object are independent of type. This means that instances of any type may be versioned and that versioned and nonversioned instances can be operated on by the same user code. This can easily take an existing piece of code that has no notion of versioning and use it on versioned data.

Data Dictionary

The object, attributes, and their relationships are documented in a data dictionary. This document consists of a written description of objects, formalizes the identification of objects, forms part of the formal system specification, and separates descriptions that are written for each object. This is a live document and should start at the beginning of the system engineering and enhance throughout the systems development and maintenance phases.

System Design Document

The system design document (SysDD) contains the highest level of design information for the IT system. The SysDD describes the allocation of system requirements to CSCIs. The major outlines for the SysDD are as follows:

- Operational concepts
- System architecture
- System design features
- Processing resources
- Requirements traceability

Operational Concepts

The system developer summarizes the customers needs that must be met by the IT system. The primary mission of the system will be described. The operational environment describes the environment in which the system will be employed. The support environment describes the operational system during the production and deployment phase of the life cycle. It covers the use of multipurpose or automated test equipment and maintenance criteria.

System Architecture

System architecture describes the internal structure of the IT system. The CSCIs and HWCIs will be identified and their purpose summarized. The purpose of each external interface will be explained. Graphic system architecture will be included for clarification.

System Design Features

System design consists of the identification and description of the relationships of each HWCI, CSCI, and manual operation of the IT system. Each CSCI will be named uniquely. Each requirement from SysRS to the CSCI will be identified. Each external interface to the system that is addressed by the CSCI will be recorded as follows:

- Bits per second
- Word length

- Message format
- Frequency of messages
- Priority rules
- Protocol

Any design constraints on the CSCI will be described. The internal interfaces to the system will be described, and the systems state and mode will be discussed.

Processing Resources

Processing resources cover hardware, programming, design, coding, and use characteristics of the processing resource. These characteristics include the following parameters:

- Memory size (amount of internal memory [absolute, spare, or both] of the computer)
- Word size (number of bits in each computer word)
- Processing speed (computer processor capacity [absolute, spare, or both])
- Characteristic set standard
- Instruction set architecture
- Interrupt capabilities of the hardware
- Direct memory access
- Channel requirements
- Auxiliary storage
- Growth capabilities of any part of the processing resource
- Diagnostic capabilities
- Allocation of pertinent processing resources to each CSCI
- Networking requirements
- E−mail capabilities

Requirements Traceability

Requirements traceability provides traceability of the requirements that were allocated to the HWCIs, CSCIs, and manual operations back to the requirements of the SysRS as follows:

SysRS < = > **CSCI requirements** + **HWCIs requirements** +
Manual operation requirements

The traceability should be tabulated in a requirements traceability matrix. This traceability will carry out forward and backward throughout the software development documents.

System Design Checklist

The IT manager should check the following parameters in the system design phase:

- System engineering
- Operations
- Maintenance
- Testing
- Training
- Reuse
- Software

- Facilities
- Personnel
- Logistic support
- Security
- Safety
- Risk
- Optimization