B.E.

Sixth Semester Examination, December-2008

Principles of Software Engineering (CSE-302-E)

Note : Attempt any five questions.

Q. 1. (a) What is meant by the term 'Software Process' ? Explain.

Ans. Software Process : A software process is the set of activities and associated results that produce a software product. There are four fundamental process activities that are common to all software processes. These are :

(i) Software specification where customers and engineers define the software to be produced and the constraints on its operation.

(ii) Software development where the software is designed and programmed.

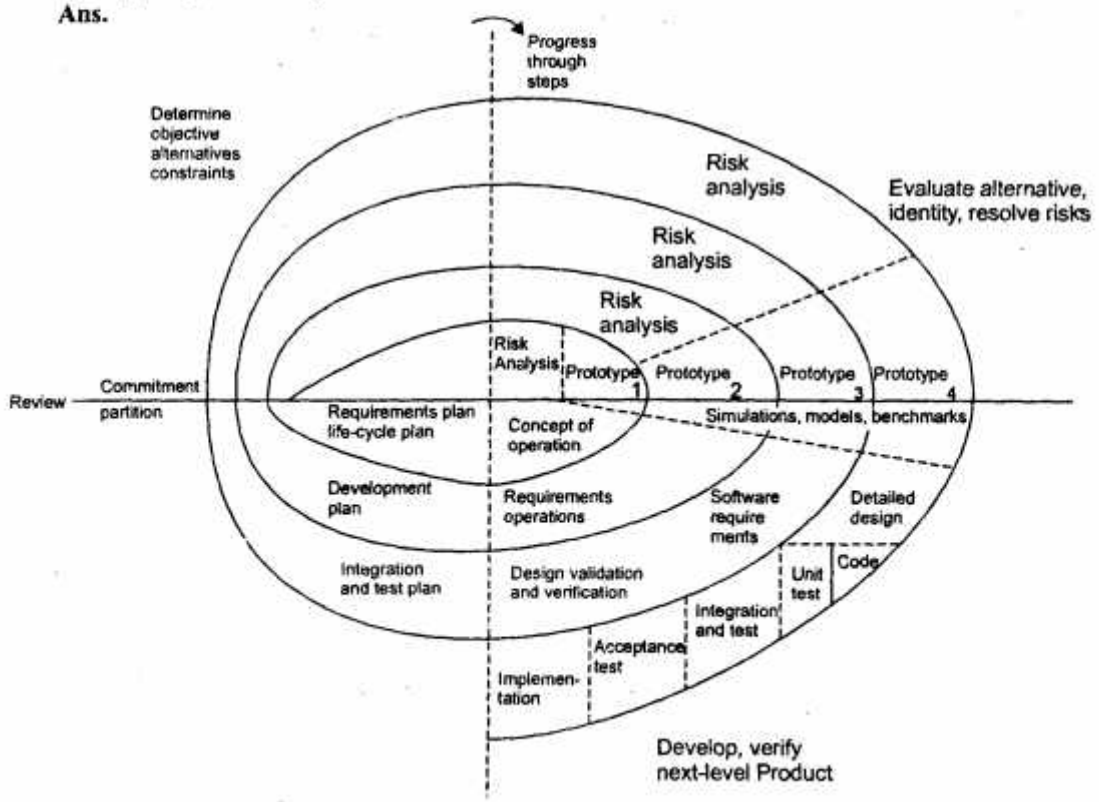
(iii) Software validation where the software is checked to ensure that it is what the customer requires.

(iv) Software evolution where the software is modified to adapt it to changing customer and market requirements.

A software process model is a simplified description of a software process that presents one view of that process. Process models may include activities that are part of the software process, software products and the roles of people involved in software engineering.

Q. 1. (b) Explain the Spiral model in detail.

Ans.



Spiral Model : The spiral model of the software process was originally proposed by Boehm. Rather than represent the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral. Each loop in the spiral represents a phase of the software process.

Each loop in the spiral is split into four sectors :

(i) Objective Setting : Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified.

(ii) Risk Assessment and Reduction : For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce risk.

(iii) Development and Validation : After risk evaluation, a development, model for system is chosen. If safety risks are the main consideration, development based on formal transformations may be the most appropriate and so on.

(iv) Planning : The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

The main difference between the spiral model and other software process models is the explicit recognition of risk in the spiral model. Informally, risk simply means something that can go wrong.

Q. 2. (a) What are project metrics? Differentiate between size-oriented metrics and function-oriented metrics.

Ans. Software project measure tactical that is, project metrics and indicators derived from them are used by the project manager and a software team to adapt project works flow and technical activities. The first application of the project metrics on the most software project occurs during estimation. Metrics collected from the past projects are used as a basis from which effort and time duration estimates are made for current software works. Project metrics are used to minimize the development schedule by guiding the adjustments necessary to avoid delays and mitigate potential problem and risks.

Project metrics are also used to access product quality on an ongoing basis and when necessary. Modify the technical approach to improve quality.

Size Oriented Metrics : Size oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of software that has been produced. If a software organization maintains simple records a table of size oriented measure. Size oriented metrics are not universally accepted as the best way to measure the process of software developed.

Function Oriented Metrics : Function oriented software metrics is basically used to measures functionality delivered by the application as normalization value. Since functionality cannot be measured directly, it must be delivered indirectly using other direct measure. Function oriented metrics were first proposed by Albrecht, who suggested a measure called the function point. Function point is derived using an empirical relationship based on countable measures of software information domain and assessments of software complexity.

Q. 2. (b) What are software risks? What is meant by risk identification and risk projection?

Ans. Software risk is a problem that could cause some loss or threaten the success of a software project, but which hasn't happened yet. These potential problems might have an adverse impact on the cost, schedule, or technical success of the software project, the quality of software products or project team morale.

Major software risks are :

(i) Personnel shortfalls.

(ii) Unrealistic schedules and budgets.

(iii) Developing the wrong functions and properties.

(iv) Developing the wrong user interface.

(v) Gold-plating.

(vi) Continuing stream of requirements changes.

(vii) Shortfalls in externally furnished components.

(viii) Shortfalls in externally performed tasks.

(ix) Real-time performance shortfalls.

(x) Straining computer science capabilities.

Risk identification is a systematic attempt to specify threats to the project plan. The purpose of risk identification is to develop a list of risk items called risk statement. Risk identification can be facilitated with the help of a checklist of common risk areas for software projects or by examining the contents of an organizational database of previously identified risks and mitigation strategies.

Risk projection also called risk estimation, attempts to rate each risk in two ways the likelihood or probability that the risk is real.

There are four risk projection steps :

(i) Establish a scale that reflects the perceived likelihood of a risk.

(ii) Delineate the consequences of the risk.

(iii) Estimate the impact of the risk on the project and the product.

(iv) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Q. 3. (a) What is an ER diagram? Where can it be used? What symbols are used in drawing an ER diagram?

Ans. The data flow diagram depicts the flow of events and data within a system but omit consideration of the other very important part of the system. The structure of the data. Data cannot magically move from process to process or from process to data store without computer programs understanding the structure of the data. As data is stored in a database in records that are defined by a collection of fields. This data structure is called the database schema or data model. No system design is complete without a data model.

The Entity-Relationship (E-R) diagram can be used to represent the information contained within the data model. Each file in the database (each entity) is shown as a rectangle with the name of the entity inside the rectangle. Since all files in a database are related the diagram shows lines that depict the relationship between pairs of entities.

Analysts either draw ERD's by hand or use software like Microsoft Access to draw them.

The cardinality of the relationship between entities is shown at the ends of each line. ER diagrams are used to show the data model because people tend to understand pictures when they are not familiar with a topic and few people are familiar with data modelling.

E-R Diagram consists of the following major symbol :

(i) Rectangles, which represents entity sets

(ii) Ellipses, which represents attributes

(iii) Diamonds, which represents relationship sets.

(iv) Lines, which link attributes to enti / sets and entity sets to relationship sets

(v) Double ellipses, which represents multivalued attributes

(vi) Dashed ellipses, which denote derived attributes.

(vii) Double lines, which indicates total participation of an entity in a relationship set.

Q. 3. (b) Write a detailed note on SRS.

Ans. "Software Requirement Specification is a document, that completely describe what the proposed software should do, without describing how the software will do it."

Software Requirement Specification state the goal and objective of the software, describe in the context of the computer based system.

A Software Requirement Specification (SRS) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe all of the the interactions that the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation.

Characteristics of SRS : IEEE standards of SRS, state that SRS should be :

(i) Correct : This is like motherhood and apple pic. Of course we want specification to be correct. No one writes a specification that they know is incorrect.

(ii) Unambiguous : An SRS is unambiguous if and only if every requirement stated therein has only one interpretation.

(iii) Complete : A simple judge of this its that it should be all that is needed by the software designers to create the software.

(iv) Consistent : The SRS should be consistent within itself and consistent to its reference documents.

(v) Ranked for Importance : Very often a new system has requirements that are really marketing wish lists. Some may not be achievable.

(vi) Verifiable : SRS should provide a quantitative requirement like "Every key stroke should provide a user response within 100 milliseconds."

(vii) Modifiable : Having the same requirement in more than one place may not be wrong but tends to make the document not maintainable.

Q. 4. (a) Explain the complete architectural design process.

Ans. Hofmeister discuss how the architectural design stage forces software designers to consider key design aspects early in the process. They suggest that the software architecture can serve as a design plan that is used to negotiate system requirements and as a means of structuring discussions with clients, developers and managers. They also suggest that it is an essential tool for complexity management. It hides details and allows the designers to focus on the key system abstractions. The system architecture affects the performance, robustness, distributability and maintainability of a system. The particular style and structure chosen for an application may therefore depend on the non-functional system requirements :

(i) Performance : If performance is a critical requirement, the architecture should be designed to localise critical operations within a small number of sub-systems with as little communication as possible between these sub-systems. This may mean using relatively large-grain rather than fine-grain components to reduce component communications.

(ii) Security : If security is a critical requirement, a layered structure for the architecture should be used, with the most critical assets protected in the innermost layers and with a high level of security validation applied to these layers.

(iii) Safety : If safety is a critical requirement, the architecture should be designed so that safety-related operations are all located in either a single sub-system or in a small number of sub-systems. This reduces the costs and problems of safety validation and makes it possible to provide related protection systems.

(iv) Availability : If availability is a critical requirement, the architecture should be designed to include redundant components and so that it is possible to replace and update components without stopping the system. Fault-tolerant system architectures for high-availability systems.

(v) Maintainability : If maintainability is a critical requirement, the system architecture should be designed using fine-grain, self-contained components that may readily be changed. Producers of data should be separated from consumers and shared data structures should be avoided.

Q. 4. (b) Define the following terms :

abstraction, refinement, functional independence, cohesion, coupling.

Ans. Abstraction : Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time.

There are two types of abstraction :

(i) Data abstraction

(ii) Control abstraction

Data abstraction is the enforcement of a clear separation between the abstract properties of a data type and the concrete details of its implementation.

Control abstraction is one of the main purposes of using programming languages. Without control abstraction, a programmer would need to specify all the register/binary-level steps each time if simply wanted to add or multiply a couple of numbers and assign the result to a variable.

Refinement : Refinement is actually a process of elaboration. The process should proceed from a highly conceptual model (abstractions) to lower level details. The refinement of each module is done until we reach the statement level of our programming language.

Functional Independence : Functional independence refers to the phenomenon in which functions are not interrelated i.e., if we have two functions each will work independently/separately without affecting the other function. The output and input of one function is not dependent on the other.

Cohesion : Cohesion is a measure of the functional strength of a module. Following are the types of cohesion :

(i) Coincidental Cohesion : A module is said to have coincidental cohesion if it performs a set of tasks that are related to each other very loosely.

(ii) Logical Cohesion : A module is said to be logically cohesive, if all the elements of the module perform similar operations.

(iii) Temporal Cohesion : When a module contains tasks that are related by the fact that all the tasks must be executed in the same time-span, module is said to exhibit temporal cohesion.

Some othe cohesions are—Procedural cohesion, Communicational cohesion, Sequential cohesion and Functional cohesion.

Coupling : Coupling indicates how closely two modules interact or how interdependent they are.

The degree of coupling between two modules depends on their interface complexity. Following are types of couplings :

(i) Data Coupling : Two modules are data coupled, if they communicate via an elementary data item which is passed as a parameter between the two.

(ii) Stamp Coupling : Two modules are stamp coupled, if they communicate via a composite data item.

(iii) Control Coupling : If data from one module is used to direct the order of execution of instructions in another.

(iv) Common Coupling : Two modules are common coupled, if they share some global data areas.

(v) Content Coupling : If their code is shared.

Q. 5. (a) Differentiate between :

(i) Unit testing and integration testing

(ii) White box testing and black box testing

Ans. (i) Unit Testing : In computer programming, unit testing is a test that validates that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/ child class.

It is also known as component testing i.e., mean testing individual component of program.

A systematic way of testing small portions of code. Unit test are written with full knowledge of implementation details.

It is also known as program testing. It is carried out on the module/program, while make up system. In it, step can be conducted in parallel for multiple module.

Testing of individual hardware and software unit or group of related units.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

Integration Testing : Integration testing is testing, in which software and hardware components or both are combined together and tested to evaluate the interaction between them. Using White Box and Black Box Techniques, tester verifies that units work together, when they are integrated into larger code base.

Primary objective of integration testing is to test the module interface.

To plan there, integration test case, tester look at high and low-level design document. Integration testing is a systematic technique for constructing the software architecture i.e., focus on testing multiple module working together. The entire system is viewed as a collection of subsystems determined during the system and object design.

The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items.

(ii) White Box Testing : White box testing is also known as structural testing or glass box testing. It deals with the initial logic and structure of the code. Therefore, tester has knowledge of coding and knowledge i.e., internal working of the code. Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases. It is one of the very important technique that avoid most of error i.e., not detected during or completion of coding or project, because program is translated into

programming language, it likely that some typing errors are detected. It is very important technique to develop sophisticated project, whenever chances of occurring errors has less or minor, during testing of entire project and during implementation time.

Black Box Testing : Black box testing is also known as behavioural or functional or opaque testing, black box testing are designed to validate the functional requirements of the software, without regard to the internal working of the program. It is not alternative of white box testing. Here test cases are decided, on the basis of requirement specification of that module, rather than logic of program. It is complementary approach that is likely to uncover a different class of errors that white box testing.

While black box testing applied to software engineering, the tester would only know "legal" inputs and what expected output should be, but he or she did not have knowledge about, how the program actually arrives at these outputs.

White box testing is opposite to black box testing, where test data are derived from direct examination of the code to be tested. In it, test cases cannot be determined, until the code has actually written. Unlike white box testing, which is performed early in testing process. It tends to apply during later stage of testing. In it, test cases tell us something about presence or absence of class errors.

Q. 5. (b) Write short notes on :

(i) Boundary value analysis

(ii) Software re-engineering

Ans. (i) Boundary Value Analysis : Boundary value analysis, is a test case design technique. Boundary value analysis is complementary to equivalence partitioning. It leads to selection of test cases, that exercise boundary values. Rather than selecting any elements of equivalence classes. Boundary value analysis leads to selection of test cases at the edges of classes. Rather than focusing on input condition, boundary value analysis derive test cases from output domain as well.

Boundary value analysis is similar in many respect to those provided by equivalence partitioning.

(i) If input condition specifies a range bound by a value A and B, test cases should be designed with the value A and B, just above and just below A and B.

(ii) If input condition specifies a number of values, test cases should be developed that exercise the maximum and minimum number of values.

(iii) Just above and just below are tested.

(iv) Guideline 1 and 2 applied to output condition.

(ii) Software Re-engineering : Software re-engineering is concerned with re-implementing legacy systems to make them more maintainable. Re-engineering may involve re-documenting the system, organising and restructuring the system, translating the system to a more modern programming language and modifying and updating the structure and values of the system's data.

Re-engineering a software system has two key advantages over more radical approaches to system evolution.

(i) Reduced Risk : There is a high risk in re-developing business-critical software. Errors may be made in the system specification or there may be development problems.

(ii) Reduced Cost : The cost of re-engineering is significantly less than the cost of developing new software.

Q. 6. (a) What is Software Configuration Management ? Dicusss.

Ans. Software Configuration Management : Software Configuration Management {SCM} is a roadmap for a software project development. SCM is an umbrella activity i.e., applied throughput the software process, because it can occur at any time.

It is essential part of good project management and soled engineering practice, because change can occur at any time.

It is critical to producing and ensuring the integrity of the software products that meet customer requirement, satisfy user needs.

While software project build, it is obviously projects changes time to time, or depending upon customer requirement, SCM control the changes by identifying the work products that are likely to change, establishing the relationship among them, by defining mechanism for managing different versions of these work products, controlling changes that are imposed and auditing and reporting on the changes that are made.

"SCM is a set of activities that have been developed to manage changes throughout life cycle of the computer software."

"SCM ability to control and manage changes in software project."

Goals of software configuration management :

(i) SCM activity are personal.

ii) Affected groups and individual are informed of the status and content of software baseline.

(iii) **Configuration Control :** Controlling the release of a product and its changes.

(iv) **Configurations Identification :** What code are we working with?

(v) **Status Accounting :** Recording and reporting the status of components.

(vi) **Review :** Ensuring completeness and consistency among components.

(vii) **Build Management :** Managing the process and tools used for builds.

(viii) **Process Management :** Ensuring adherence to the organisation's development process.

(ix) **Teamwork :** Facilitates team interactions related to the process.

(x) **Defect Tracking :** Making sure every defect has traceability back to the source.

Q. 6. (b) Explain SQA activities in detail.

Ans. SQA stands for Software Quality Assurance. The major software quality assurance activities are following :

(i) Configuration Management Monitoring : SQA assures that software Configuration Management [CM] activities are performed in accordance with the CM plans, standards and procedures. SQA reviews the CM plans for compliance with software CM policies and requirements and provides follow-up for non-conformance's. SQA audits the CM functions for adherence to standards and procedures and prepares reports of its findings.

(ii) Verification and Validation Monitoring : SQA assures verification and validation (V and V) activities by monitoring technical reviews, inspection and walkthroughs.

Inspection : Inspection is a formal evaluation technique, in which software requirements, design or code are examined in detail by person or group of person, other than author to detect fault, violation of development standard and other problems. Software inspection, improve the software quality and increase software productivity. Inspection also helps improve software quality by checking properties, like standard compliance, modularity clarity and simplicity.

Review : Formal software reviews should be conducted at the end of each phase of the life cycle to identify problems and determine whether the interim product meets all applicable requirements. In formal reviews, actual work done is compared with established standards. SQA's main objective in reviews is to assure that the Management and Development Plans have been followed, and that the product is ready to proceed with the next phase of development. Although the decision to proceed is a management decision. SQA is reponsible for advising management and participating in the decision.

Q. 7. Describe CASE (Computer Aided Software Engineering) tools in detail.

Ans. Computer Aided Software Engineering [CASE] tools help the project manager, the software developer and other key personnel to improve their productivity in the development team.

Following are the CASE tools :

(i) Software Rquirements Tools : A number of tools are proposed for modelling, tracing and analyzing requirements. CASE diagramming or upper CASE tools represent the system requirements visually using structured or object oriented methodologies.

(ii) Software Design Tools : These tools are used to support the system design stage of SDLC and in most of the cases are extensions of CASE tools used for requirements analysis stage. They can be used to support design, verification and optimization.

(iii) Software Construction Tools : Software consturction tools are the tools which are used to code and implement the software and hence transform the software requirements into working product. These tools are required even after the product is installed so as to remove defects and maintain it. Broadly they can be classified as program editors, compilers, interpreters and debuggers.

(iv) Software Testing Tools : These are automated tools that support various activities of software stage of SDLC.

They can be classified as :

Test Generators : These are the tools which assists the developers in test case design and their documentation e.g., MS Access, Quick List.

Test Execution and Evaluation Tools : These tools support the process of test case execution and also evaluate the results of execution. The various tools under this category are :

(i) Capture/Playback tools to automate execution of test cases.

(ii) Coverage analysis tools to ensure all parts of code i.e., statement, decision, conditions, paths, loops etc. are executed.

(iii) Memory testing tools of test memory related problems.

(iv) Simulators replace the actual hardware/software which interact with the software to be tested and also evaluate system.

Test Management Tools : These tools support management of different test artifacts. There are automated tools to support reviews and inspections also.

(v) Software Maintenance Tools : Tools under this category are :

Comprehension tools to assist in human comprehension and visualization of programs.

Reengineering tools which allow the change of existing format of a program to a new format i.e., a new language or new database or new technology.

(vi) Software Quality Tools : These tools are used to automate techniques like static analysis, reviews, inspections etc. to ensure software quality.

(vii) Configuration Management Tools : These tools support version control and other activities related to configuration management and control of changes made to the documents.

(viii) Project Management Tools : Tools under this category automate size estimations, cost estimation, schedule estimation, risk management activities etc. Some of the tools are MS-Project, Excel, COCOMO, FPA, etc.....

Q. 8. Write short notes on :

(i) ISO 9001 standard

(ii) DFD

Ans. (i) ISO 9000 Model : ISO 9000 is a family of standards for quality management systems. ISO 9000 is maintained by ISO, the International Organization for Standardization and is administered by accreditation and certification bodies. Some of the requirements in ISO 9001 (which is one of the standards in the ISO 9000 family) include

(i) a set of procedures that cover all key processes in the business;

(ii) monitoring processes to ensure they are effective;

(iii) keeping adequate records;

(iv) checking output for defects, with appropriate and corrective action where necessary;

(v) regularly reviewing individual processes and the quality system itself for effectiveness; and

(vi) facilitating continual improvement.

A company or organization that has been independently audited and certified to be in conformance with ISO 9001 may publicly state that it is "ISO 9001 certified" or "ISO 9001 registered." Certification to an ISO 9000 standard does not guarantee the compliance (and therefore the quality) of end products and services; rather, it certifies that consistent business processes are being applied. Indeed, some companies enter the ISO 9001 certification as a marketing tool.

ISO 9000 includes stadards :

ISO 9000-2000, Quality Management Systems : Fundamentals and Vocabulary : Covers the basics of what quality management systems are and also contains the core language of the ISO 9000 series of standards. A guidance document, not used for certification purposes. but important reference document to understand terms and vocabulary related to quality management systems. In the year 2005, revised ISO 9000:2005 standard has been published, so it is now advised to refer to ISO 9000 : 2005.

ISO 9001:2000 Quality Management Systems : Requirements is intended for use in any organization which designs. develops, manufactures, installs and/or services any product or provides any form of service. It provides a number of requirements which an organization needs to fulfill if it is to achieve customer satisfaction through consistent products and services which meet customer expectations. It includes a requirement for the continual (i.e., planned) improvement of the Quality Management Systems, for which ISO 9004:2000 provides many hints.

This is the only implementation for which third-party auditors may grant certification. It should be noted that certification is not described as any of the 'needs' of an organization as a driver for using ISO 9001 but does recognize that it may be used for such a purpose.

ISO 9004 : 2000 Quality Management Systems : Guidelines for Performance Improvements covers continual improvement. This gives you advice on what you could do to enhance a mature system. This standard very specifically states that it is not intended as a guide to implementation.

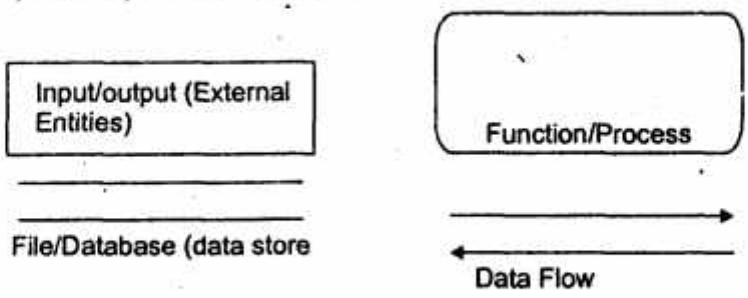
There are many more standards in the ISO 9001 family many of them not even carrying "ISO 900x" numbers. For example, some standards in the 10,000 range are considered part of the 9000 family: **ISO 10007 : 1995** discusses Configuration management, which for most organizations is just one element of a complete management system. ISO notes : "The emphasis on certification tends to overshadow the fact that there is an entire family of ISO 9000 standards... Organizations stand to obtain the greatest value when the standards in the new core series are used in an integrated, both with each other and with the other standards making up the ISO 9000 family as a whole".

Note that the previous members of the ISO 9000 family, 9001 , 9002 and 9003, have all been integrated into 9001. In most cases, an organization claiming to be "ISO 9000 registered " is referring to ISO 9001.

(ii) Data Flow Diagram : A Data Flow Diagram (DFD) is a graphical representation of the "flow" of date through an information system. A data flow diagram can also be used for the visualization of data processing (structured design). It is common practice for a designer to draw a context-level DFD first which shows the interaction between the system and outside entities. This context-level DFD is then "exploded" to show more detail of the system being modeled.

Developing a DFD helps in identifying the transaction data in the data model.

There are different notations to draw data flow diagrams, defining different visual representations for processes, data stores, dataflow, and external entities.



Components of DFD :

(i) External Entities define the sources and destinations of information entering and leaving the system. An external entity can be a person, system, or organization that has pre-defined behaviour. External entities are mandatory on context diagrams but optional on data flow diagrams.

Examples of external entities include :

(i) End user,

(ii) Purchasing Department,

(iii) Inventory System.

External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

(ii) Data Flow : Flows define the interfaces between the components within the system, and the system and its external components. Data flows are the pipelines through which data are transmitted between any two components on a DFD. The composition of data is known and defined in a data dictionary.

Examples of data flows are :

(i) purchase order,

(ii) customer profile,

(iii) account number,

(iv) product.

(iii) Store : Stores represents information (i.e., data or control) at rest. Stores are used when different processes need to share information but are active at different times. Information can be written to a store and read from a store.

(iv) Process : Processes are also known as data transforms. Processes transform input flows into output flows in a defined manner.

A process is a distinct activity (or set of activities) described by its inputs and outputs. A process describes a unique behaviour that has a beginning and an end. A process is performed repeatedly.

DFD Levels :

Context Data Flow Diagram : This level shows the overall context of the system and it's operating environment and shows the whole system as just one process. It does not usually show data stores, unless they are ''owned'' by external systems, e.g., are accessed by but not maintained by this system, however, these are often shown as external entities.

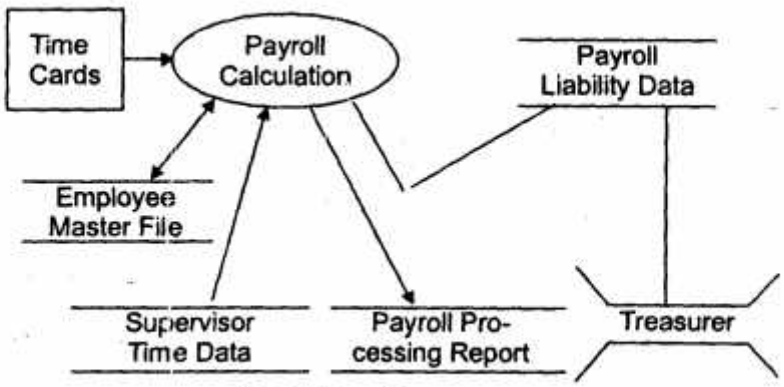
Level 0

This level shows all processes at the first level of numbering, data stores, external entities and the data flows between them. The purpose of this level is to show the major high level processes of the system and their interrelation. A process model will have one. and only one, level 0 diagram. A level 0 diagram must be balanced with it's parent context level diagram, i.e., there must be the same external entities and the same data flows, these can be broken down to more detail in the level 0, e.g , the ''enquiry'' data flow could be split into ''enquiry request'' and ''enquiry results'' and still be valid.

Level 1

This level is a decomposition of a process shown in a level 0 diagram, as such there should be a level 1 diagram for each and every process shown in a level 0 diagram. In this example processes 1.1, 1.2 & 1.3 are all children of process 1, together they wholly and completely describe process 1, and combined must perform the full capacity of this parent process. As before, a level 1 diagram must be balanced with it's parent level 0 diagram.

Data Flow Diagram of Payroll Processing



Data Flow Diagram