

B.Tech 6th Semester Examination, 2014

Model Answer

Subject:- *Software Engg.*

Paper Code:- *057614*

Sets (I) / (II)

Ans1- i) ^ae ii) ^ba iii) ^cd iv) ^de v) ^ed vi) ^fe vii) ^gc viii) ^hd ix) ⁱb x) ^ja

Ans2- a)

| Criteria | Black Box Testing | White Box Testing |
|--------------------------|--|---|
| Definition | Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester | White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. |
| Levels Applicable To | Mainly applicable to higher levels of testing: Acceptance Testing System Testing | Mainly applicable to lower levels of testing: Unit Testing Integration Testing |
| Responsibility | Generally, independent Software Testers | Generally, Software Developers |
| Programming Knowledge | Not Required | Required |
| Implementation Knowledge | Not Required | Required |
| Basis for Test Cases | Requirement Specifications | Detail Design |
| Techniques Used | Boundary-value analysis, guessing, Race conditions, effect graphing, Syntax testing, State transition testing, Graph matrix, Equivalence partitioning | Error Code Coverage, Segment coverage, Compound condition coverage and loop coverage, Data Flow testing, Path Testing |

Name of Setter: - *J. P. Singh*

Designation:- *Asst. Prof.*

Address:- *Dept of CSE,
NIT Patna.*

Signature of Setter *[Signature]*

1
SUMIT KUMAR
HOD, C&E, VVIT PURNEA
9884249879

B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

Ans2- b) **Integration Testing** is a level of the software testing process where individual units are combined and tested as a group.

- i. **Big Bang** is an approach to Integration Testing where all or most of the units are combined together and tested at one go. This approach is taken when the testing team receives the entire software in a bundle. So what is the difference between Big Bang Integration Testing and System Testing? Well, the former tests only the interactions between the units while the latter tests the entire system.
- ii. **Top Down** is an approach to Integration Testing where top level units are tested first and lower level units are tested step by step after that. This approach is taken when top down development approach is followed. Test Stubs are needed to simulate lower level units which may not be available during the initial phases.
- iii. **Bottom Up** is an approach to Integration Testing where bottom level units are tested first and upper level units step by step after that. This approach is taken when bottom up development approach is followed. Test Drivers are needed to simulate higher level units which may not be available during the initial phases.
- iv. **Sandwich/Hybrid** is an approach to Integration Testing which is a combination of Top Down and Bottom Up approaches.

Ans3- a) SRS characteristics are-

- i) **Correct** – The SRS should be made up to date when appropriate requirements are identified.
- ii) **Unambiguous** – When the requirements are correctly understood then only it is possible to write an unambiguous software.
- iii) **Complete** – To make SRS complete, it should be specified what a software designer wants to create software.
- iv) **Consistent** – It should be consistent with reference to the functionalities identified.
- v) **Specific** – The requirements should be mentioned specifically.
- vi) **Traceable** – What is the need for mentioned requirement? This should be correctly identified.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

Ans3-b) **CAPABILITY MATURITY MODEL (CMM)**: CMM is a strategy for improving the software process, irrespective of the actual life cycle model used. Software Engineering Institute (SEI) of Carnegie-Mellon University developed CMM in 1986. CMM is used to judge the maturity of the software processes of an organization and to identify the key practices that are required to increase the maturity of these processes.

Different levels of CMM:

- 1) **Initial (maturity level 1)** -At this, the lowest level, there are essentially no sound software engineering management practices in place in the organization. Instead everything is done on adhoc basis. In maturity level 1 organization, the software process is unpredictable.
- 2) **Repeatable (maturity level 2)** -At this level, policies for managing a software project and procedures to implement those policies are established. Planning and managing new projects is based on experience with similar projects. An objective in achieving level 2 is to institutionalize effective management processes for software projects, which allow organizations to repeat successful practices, developed on other projects. The software process capability at level 4 organizations can be summarized as disciplined.
- 3) **Defined (maturity level 3)** - At this level, the standard process for developing and maintaining software across the organization is documented, including both software engineering and management processes. Processes established at level 3 are used to help the software managers and technical staff to perform more effectively. The software process capability at level 4 organizations can be summarized as standard and constituent.
- 4) **Managed (maturity level 4)** -At this level, the organization sets quantitative quality goals for both software products and processes. Productivity and quality are measured for important software process activities across all projects as part of an organizational measurement program. The software process capability at level 4 organizations can be summarized as "predictable".
- 5) **Optimizing (maturity level 5)** -At this level, the entire organizations focused on continuous process improvement. The organizations have the means to identify weaknesses and strengthen the process proactively, with the goal of preventing the occurrence of defects. . The software process capability at level 4 organizations can be summarized as continuously improving.

Ans4- a) **Cohesion**: Cohesion refers to the strength of relationship among elements within a module. Cohesion represents how strongly the internal elements of a module are bound to each other.

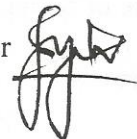
- i. **Coincidentally cohesion** – The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

- ii. **Logically cohesion** – A module that performs the tasks that are logically related with each other is called logically cohesive.
- iii. **Temporal cohesion** – The module in which the tasks need to be executed in some specific time span is called temporal cohesive.
- iv. **Procedural cohesion** – When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
- v. **Communicational cohesion** – When the processing elements of a module share the data then such module is called communicational cohesive.

- Ans4- b) The cyclomatic complexity can be computed by any one of the following ways.
1. The numbers of regions of the flow graph correspond to the cyclomatic complexity.
 2. Cyclomatic complexity, $V(G)$, for the flow graph, G , is defined as: $V(G) = E - N + 2$,
E -- number of flow graph edges, N -- number of flow graph nodes
 3. $V(G) = P + 1$
where P is the number of predicate nodes contained in the flow graph.

For the given code- Cyclomatic complexity $V(G) = E - N + 2$
 $E = 13, V = 11$
 $V(G) = 13 - 11 + 2$
 $= 2 + 2$
 $= 4$

Cyclomatic Complexity = 4

Name of Setter: -

Designation:-

Address:-

Signature of Setter

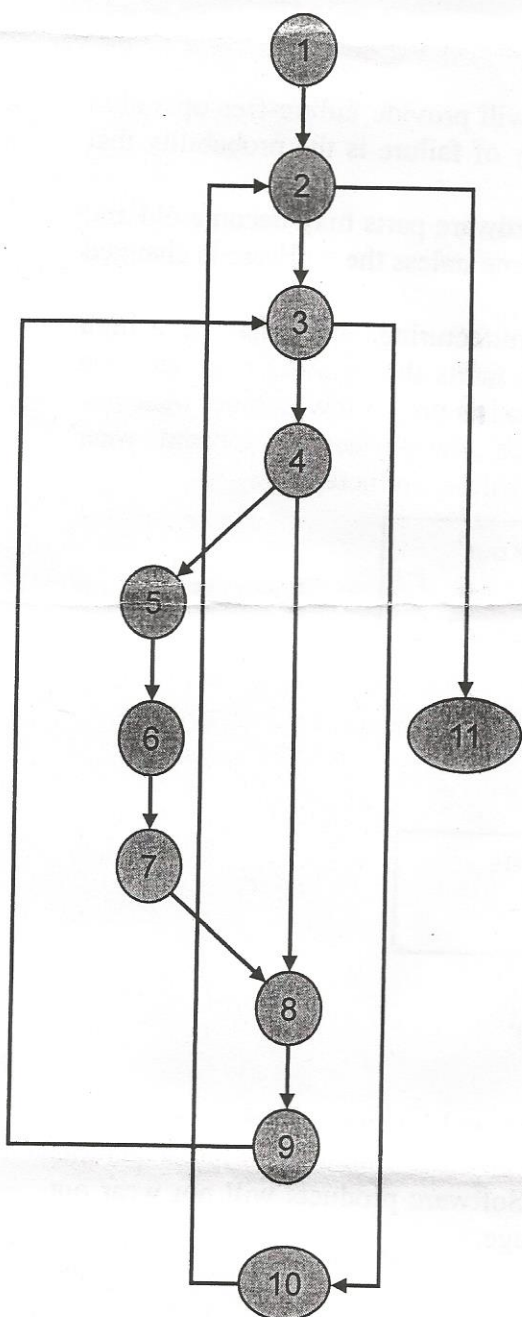


B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)



Name of Setter: -

Designation:-

Address:-

Signature of Setter

B.Tech 6th Semester Examination, 2014
Model Answer

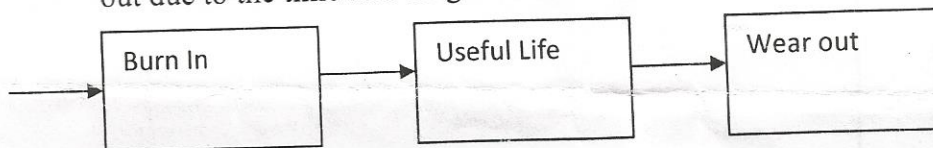
Subject:-

Paper Code:-

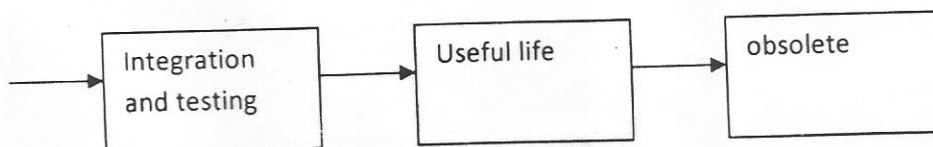
Sets (I) / (II)

Ans5- a) **Software reliability** is the probability that software will provide failure-free operation in a fixed environment for a fixed interval of time. Probability of failure is the probability that the software will fail on the next input selected.

- 1) Software reliability is not a direct function of time. Hardware parts may become old and wear out with time, but software will not change over time unless the software is changed or modified intentionally.
- 2) In Hardware reliability, in the first phase of the manufacturing, there may be a high number of faults. But after discovering and removing faults this number may decrease and gradually in the second phase (Useful life), there exists only a few number of faults. After this phase, there will be wear out phase in which, the physical component wear out due to the time and usage and the number of faults will again increase.



Phases of hardware when considering reliability



Phases of software when considering reliability

But in software reliability, at the first phase, i.e while testing and integration there will be high number of faults, but after removing the faults, there exists only a few number of faults and this process of removing the faults continues at a slower rate. Software products will not wear out with time and usage, but may become outmoded at a later stage.

Name of Setter: -

Designation:-

Address:-

Signature of Setter

B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

Ans5- b) The COCOMO (Constructive Cost Estimation Model) is proposed by **DR. Berry Boehm in 1981** and that's why it is also known as COCOMO'81. It is a method for evaluating the cost of a software package. Software cost estimation should be done through three stages:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Complete/Detailed COCOMO Model

BASIC COCOMO MODEL: Basic COCOMO Model is good for quick, early, rough order of magnitude estimate of software cost. It does not account for differences in hardware constraints, personal Quality and experience, use of modern tools and techniques, and other project attribute known to have a significant influence on software cost, which limits its accuracy. It gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2} \text{ PM}$$

$$\text{Tdev} = b_1 \times (\text{Effort})^{b_2} \text{ Months}$$

Where, KLOC is the estimated size of the software product expressed in Kilo Lines of Code, a_1 , a_2 , b_1 , b_2 are constants for each category of software products,

Tdev is the estimated time to develop the software, expressed in months,

Effort is the total effort required to develop the software product, expressed in person months (PMs).

Estimation of development effort:

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: $\text{Effort} = 2.4(\text{KLOC})^{1.05} \text{ PM}$

Semi-Detached: $\text{Effort} = 3.0(\text{KLOC})^{1.12} \text{ PM}$

Embedded: $\text{Effort} = 3.6(\text{KLOC})^{1.20} \text{ PM}$

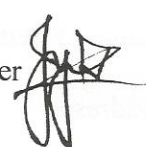
PM: Person Months

Name of Setter: -

Designation:-

Address:-

Signature of Setter



Subject:-

Paper Code:-

Sets (I) / (II)

Estimation of development time:

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: $T_{dev} = 2.5(\text{Effort})^{0.38}$ Months

Semi-detached: $T_{dev} = 2.5(\text{Effort})^{0.35}$ Months

Embedded: $T_{dev} = 2.5(\text{Effort})^{0.32}$ Months

DETAILED/ADVANCED COCOMO MODEL: A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics.

The Detailed COCOMO Model differs from the Intermediate COCOMO model in that it uses effort multipliers for each phase of the project. These phase dependent effort multipliers yield better estimates because the cost driver ratings may be different during each phase. In Advanced COCOMO Model the cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

Example: A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

Database part

Graphical User Interface (GUI) part

Communication part

Of these, the communication part can be considered as **Embedded software**. The database part could be **Semi-detached software**, and the GUI part **Organic software**. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

Ans6- The Halstead Metrics routine computes Maurice Halstead's software science metrics for each procedure or module contained in the parameter prc. Altogether, there are six metrics, all of which are functions of the following four variables.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

n1 = the number of unique operators
n2 = the number of unique operands
N1 = the total number of operators
N2 = the total number of operands

The six software science metrics include:

Vocabulary: $n = n1 + n2$ (the total number of unique operators and operands)

Length: $N = N1 + N2$ (the sum of all occurrences of operators and operands)

Volume: $V = N \log_2(n)$ (quantifies the total size of a given procedure)

Program Difficulty: $D = \frac{1}{2} (n1 * N2 / n2)$ (reflects the effort required to understand, code, and maintain a given procedure)

Language Level: $L = V / D^2$ (indicates how well a programmer uses features of the language)

Effort: $E = D * V$ (indicates a level of program complexity in units of time that it takes to write, modify, or maintain a piece of code).

Given $n1 = 8, n2 = 8, N1 = 13, N2 = 16$.

Vocabulary $n = n1 + n2; n = 8 + 8; n = 16$

Program Length $N = N1 + N2; N = 13 + 16; N = 29$

Estimated Program Length $N^* = n1 \log_2 n1 + n2 \log_2 n2; N^* = 8 \log_2 (8) + 8 \log_2 (8); N^* = 48$

Volume $V = N \log_2 n; V = 29 \log_2 16; V = 29 * 4; V = 116$

Potential Volume $V^* = (2+n) \log_2 (2+n); V^* = 18 \log_2 18; V^* = 22.59490$

Program Level $L = V / D^2; L = 116 / 8^2; L = 116 / 64; L = 1.8125$

Estimated Program Level $L^* = 1 / D; L^* = 1 / 8; L^* = 0.125$

Difficulty $D = \frac{1}{2} (n1 * N2 / n2); D = \frac{1}{2} (8 * 16 / 8); D = 8$

Effort $E = D * V; E = 8 * 116; E = 928$

Ans 7- a)

Criteria **Verification**

Validation

The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.

Definition

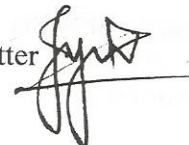
The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

| | | |
|--|---|---|
| <p><i>Objective</i></p> <p><i>Question</i></p> <p><i>Evaluation Items</i></p> <p><i>Activities</i></p> | <p>To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.</p> <p>Are we building the product <i>right</i>?</p> <p>Plans, Requirement Specs, Design Specs, Code, Test Cases</p> <ul style="list-style-type: none">▪ Reviews▪ Walkthroughs▪ Inspections | <p>To ensure that the product actually meets the user's needs, and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.</p> <p>Are we building the <i>right</i> product?</p> <p>The actual product/software.</p> <ul style="list-style-type: none">▪ Testing |
|--|---|---|

Ans7- b) i) **Configuration Management:** Due to several reasons, software changes during its life cycle. As a result of the changes made, multiple versions of the software exist at one time. These changes must be managed, controlled and documented properly in order to have reliable systems. Configuration management helps the developers to manage these changes systematically by applying procedures and standards and by using automated tools. Though multiple versions of the software exist in the tool repository, only one official version of set of project components exist called baseline. The different components of the baseline are called configuration items. Some examples of configuration items are project plan, SRS, Design document, test plans, user manuals etc. A group of people constitute Configuration Control Board (CCB) which controls the changes to be made to the software. Whenever changes are to be made the following steps are followed:

- (i) Submit the change request along with details to CCB
- (ii) CCB accesses the change request after proper evaluation.
- (iii) Depending upon the results, the request is either accepted or rejected or can be deferred for the future assessment.
- (iv) If accepted, proper plan is prepared to implement the change.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



Subject:-

Paper Code:-

Sets (I) / (II)

(v) Once changes are made, after validating by the quality personnel, all configuration items are updated.

Some popular configuration management tools are Clear CASE, Visual Source Safe etc.

Ans7- b) ii) **SRS**: Software Requirement Specification Document is the output of requirement analysis stage of the software development life cycle. It documents all types of requirements and constraints imposed on the end product. This document is important because it is used in all the successive stages of SDLC. Any error introduced here will result in to incomplete and bad quality product.

The characteristics of a good quality SRS are:

- (i) Correctness (ii) Completeness (iii) Consistency (iv) Unambiguousness
- (v) Ranking for importance and/ or stability (vi) Modifiability (vii) Verifiability
- (viii) Traceability (ix) Design Independent (x) Understandable by customer.

Ans8- a) Reverse Engineering is a process of analyzing software with a view to understanding its design and specification.

- In reverse engineering, source code and executable code are the input.
- It may be part of a re-engineering process but may also be used to re-specify a system for re-implementation.
- Reverse engineering often precedes re-engineering but is sometimes worth wise in its own right.
- Builds a program database and generates information from this.
- Program understanding tools (browsers, cross reference etc.) may also be used in this process.
- Design and specification may be reverse engineer to:
 - Serves as input to SRS for program replacement.
 - Be available to help program maintenance

The main objectives of the Reverse Engineering process are:

- (i) It helps the companies to understand the complexities of the system
- (ii) Helps the analyst to generate useful lost information about legacy systems
- (iii) Can be used to identify reusable components for analysis and future use
- (iv) Helps in generating graphical representation of the system from different perspectives e.g. ER diagram, DFD, class diagram etc.
- (v) Can be used as a part of Reengineering process.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



B.Tech 6th Semester Examination, 2014
Model Answer

Subject:-

Paper Code:-

Sets (I) / (II)

(vi) Over a period of time modifications made to the software also result into unexpected problems. The anomalies can be detected using reverse engineering techniques.

Ans8- b) **Bottom up design:** This approach leads to a style of design where we decide how to combine these modules to provide larger ones; to combine those to provide even larger ones, and so on, till we arrive at one big module which is the whole of the desired program. Since the design progressed from bottom layer upwards, the method is called bottom up design. The main argument for this design is that if we start coding a module soon after its design, the chances of recoding is high; but the coded module can be tested and design can be validated sooner than a module whose sub modules have not yet been designed.

Top down design: A top down design approach starts by identifying the major modules of the system, decomposing them into their lower level modules and iterating until the desired level of details is achieved. This is stepwise refinement; starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

Hybrid design: Pure top-down or pure bottom up approaches are often not practical therefore hybrid approach which combines the above two approaches is often used.

Ans9- a) i) **Lines of code (LOC)** is a software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. LOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced.

Advantages:-

1. Scope for Automation of Counting: Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the LOC in a program. However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.

2. An Intuitive Metric: Line of Code serves as an intuitive metric for measuring the size of software due to the fact that it can be seen and the effect of it can be visualized. Function Point is more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience.

Name of Setter: -

Designation:-

Address:-

Signature of Setter



Disadvantages

1. Lack of Accountability: Lines of code measure suffers from some fundamental problems. Some think it isn't useful to measure the productivity of a project using only results from the coding phase, which usually accounts for only 30% to 35% of the overall effort.
2. Lack of Cohesion with Functionality: Though experiments have repeatedly confirmed that effort is highly correlated with LOC, functionality is less well correlated with LOC. That is, skilled developers may be able to develop the same functionality with far less code, so one program with less LOC may exhibit more functionality than another similar program. In particular, LOC is a poor productivity measure of individuals, because a developer who develops only a few lines may still be more productive than a developer creating more lines of code.
3. Adverse Impact on Estimation: Because of the fact presented under point (a), estimates based on lines of code can adversely go wrong, in all possibility.
4. Developer's Experience: Implementation of a specific logic differs based on the level of experience of the developer. Hence, number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does, though they use the same language.

Ans9- a) ii) **Function Point**: Function point measures the functionality from the user point of view, that is, on the basis of what the user request and receives in return. Therefore, it deals with the functionality being delivered, and not with the lines of code, source modules, files etc. Measuring size in this way has the advantage that size measure is independent of the technology used to deliver the function.

Importance of function point:

- This is independent of the languages tools, or methodology used for implementation.
- They can be estimated from requirement specification or design specification.
- They are directly linked to the statement of request.

Ans9- b) **Structured Analysis**- Its purpose is to analyze users' requirement, carry out functional decomposition of a system and represent the same through some standard graphical tools. The analysis consists of interpreting the system concept (or real world) into data and control terminology, that is into data flow diagrams. The flow of data and control from bubble to data store to bubble can be very hard to track and the number of bubbles can get to be extremely large. One approach is to first define events from the outside world that require the system to react, then assign a bubble to that event, bubbles that need to interact are then connected until the system is defined. This can be rather overwhelming and so the bubbles are usually grouped into

higher level bubbles. Data Dictionaries are needed to describe the data and command flows and a process specification is needed to capture the transaction/transformation in format.

Structured Design- Structured Design is a systematic methodology to determine design specification of software. The basic principles, tools and techniques of structured methodology are discussed in this chapter. It covers the four components of software design, namely, architectural design, detail design, data design and interface design. The following are concepts, tools and techniques of structured design:

- Coupling and cohesion
- Structure chart
- Transaction analysis and transform analysis
- Program flowchart
- Structured flowchart
- HIPO documentation

Marks distribution :-

Q ① _____ $7 \times 2 = 14$

Q ② (a) + (b) _____ $7 + 7 = 14$

Q ③ (a) + (b) _____ $7 + 7 = 14$

Q ④ (a) + (b) _____ $7 + 7 = 14$

Q ⑤ (a) + (b) _____ $7 + 7 = 14$

Q ⑥ _____ 14

Q ⑦ (a) + [(b) - (i) + (ii)] _____ $[7 + (0.5 \times 2)] = 14$

Q ⑧ (a) + (b) _____ $7 + 7 = 14$

Q ⑨ [(a) + (i) + (ii)] + (b) _____ $[(0.5 \times 2) + 7] = 14$

Sum 7
23/8/14

SUMIT KUMAR
MOD, CSE DEPT.
VVIT PURNIA
9334249879