**Software Engineering**

**Ques1**: What is software engineering? What are the key areas of software engineering?

(Câu 1 Công nghệ phần mềm là gì? Liệt kê các lĩnh vực trong công nghệ phần mềm.)

**Ans**: Software engineering is a broad field that touches upon all aspects of developing and supporting a software system.

The key areas of software engineering are:

- Technical and business processes

- Specific methodologies and techniques

- Product characterization and metrics for measurements

- People skills and team work

- Project coordination and management

**Ques2**: What are program requirements?

**Ans**: Program requireents are statements that define and qualify what the program needs to do.

- Defined by client, with help from engineer

- Functional – define what must be done

- Non-functional – qualify the functional ones

**Ques3**: Types of non-functional requirements

**Ans**: 3 types:

- Performance requirements

- Real-time requirements

- Modifiability requirements

**Ques4**: Issues involved in the complexity of software?

**Ans**: Breadth issue

- Major functions

- Features within each functional area

- Interfaces to other external systems

- Simultaneous users

- Types of data and data structures

Deep issue (linkages)

- Amount of nesting

- Number of “chained” relationships

**Ques5**: Non-technical and technical issues in system development.

**Ans**: Technical issues:

- Problem and solution simplifcation

+ Decomposition

+ Modularization

+ Separation

+ Incremental iterations

- Technology and tool choices

+ Development platform

+ Development language

+ Database

+ Network

+ Configuration management

- Process and Methodology

+ Choice of process

+ Choice of methodologies

+ Choice tools to support the process

Non-technical issue

- Project effort estimation and scheduling

+ Needs to consider and estimate more items

+ Needs to coordinate more items in terms of pre-requisites and co-requisites

+ Needs to consider more potentials of risks and variations

- Assignments and Communications

+ More people with an increased variety of skills

+ More communications among the people

+ More errors and modifications

**Ques6**: What factors are considered to estimate effort for supporting system?

**Ans**:

**Ques7**: 8 principles for ethics and professional practices in software engineering by IEEE-CS/ACM.

**Ans**: 8 principles for ethics and professional practices in software engineering

- Software engineers shall act consistently with the public interest

- Software engineers shall act in a manner that is in the best interest of their client and employer, consistent with the public interest

- Software engineers shall ensure that their products and related modifications meet the highest professional standards possible

- Software engineers shall maintain integrity and independence in their professional judgment

- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance

- Software engineers shall advance the integrity and reputation of the profession consistent with the public interest

- Software engineers shall be fair to and supportive of their colleagues

- Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

**Ques8**: Davis’s Early 15 principles in system development.

**Ans**: Davis’s Early 15 principles

- Make quality number 1

- High quality software is possible

- Give products to customers early

- Determine the problem before writing the requirements

- Evaluate design alternatives

- Use an appropriate process model

- Use different languages for different phases

- Minimize intellectual distances

- Put techniques before tools

- Get it right before you make it faster

- Inspect code

- Good management is more important than good technology

- People are the key to success

- Follow with care

- Take responsibility

**Ques9**: Waterfall model Chap 4 - 74

**Ans**:

1. Requirements must be specified in the first step.

2. Four main tasks must be completed in sequence: requirements, design, code, and test, followed by packaging.

3. Output of one stage feeds into the next stage in sequence, and thus easily tracked by management.

4. The software project may be tracked as it moves sequentially through specific and identifiable stages.

**Requirements**

**Design**

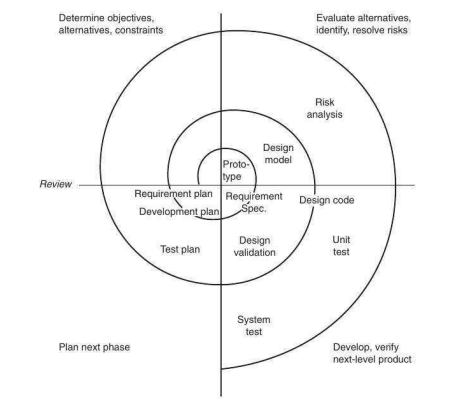
**Code**

**Test**

**Integrate and Package**

**Ques10**: Spiral model Chap4 63

**Ans**:



- Software development activities are cycled through 4 phases

- A Risk averse process

1. Identify the objectives, alternatives, or constraints for each cycle of the spiral.

2. Evaluate the alternatives relative to the objectives and constraints. In performing this step, many of the risks are identified and evaluated.

3. Depending on the amount of and type of identidied risks, develop a prototype, more detailed evaluation, an evolutionary development, or some other step to further reduce risk of achieving the identified objective. On the other hand, if the risk is substantially reduced, the next step may just be a task such as requirements, design, or code.

4. Validate the achievement of the objective and plan for the next cycle.

**Ques11**: What are the problems with traditional processes in system development.

**Ans**: Problems with traditional processes in system development

- Lengthy development times

- Inability to cope with changing requirements

- Assumption that requirements are completely understood before the project begins

- Too much reliance on heroic developer effort

- Complex methodology

- Waste/duplication of effort

**Ques12**: What are Agile (linh hoạt/nhanh) processes? **Chap5** 98

**Ans**: Agile processes are a family of software development methodologies (phương pháp luận) that produce software in short iterations (lặp đi lặp lại) and allow for greater changes in design.

(Agile software development is an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).[1] It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change)

The following is a list of characteristics and methods that depict (miêu tả) Agile methods: (6)

- Short releases and iterations: Divide the work into small pieces. Release a software to the customer as often as possible.

- Incremental design: Don’t try to complete the design up front because not enough is known early about the system anyway. Delay design decisions as much as possible, and improve the existing design when more knowledge is acquired.

- User involvement: Rather than trying to produce formal, complete, immutable standards at the beginning, ask the users involved with the project to provide constant feedback. This usually leads to a better-suited system.

- Minimal documentation: Do only the neccessary amount of documentation, which is just a means to an end. The source code is a big part of the actual documentation.

- Informal communication: Maintain constant communication, but not neccessarily through formal documents. People communicate better informally. This approach works as long as understanding is achieved.

(Không cần dùng các văn bản theo chuẩn)

- Change: Assume that the requirements and environment will change, and try to find good ways to deal with this fact.

(Đặt ra các tình huống và giải quyết các vấn đề thay đổi yêu cầu, môi trường)

**Ques13**: Practices in Extreme Programming.

**Ans**: 12 pratices in XP

1. Planning

- Determine features to be included in next release

- Business priorities and technical estimates

2. Short release

- Get a working system quickly

- Release new versions in short cycles (2 to 4 weeks)

- Base new detail plans on customer feedback

3. Metaphor

- Use a metaphor instead of an architecture

4. Simple design

- Eliminate unnecessary complexity as soon as possible

- Design can be changed in later versions

5. Test-driven development

- Continuous and automated

- Write tests before writing code

6. Design improvement (Refactoring)

- Remove duplication, improve communication, and simplify or add needed flexibility

7. Pair programming

- Two programmers working at same machine

8. Collective ownership

- Anyone can change any code at any time

9. Continuous integration

- Integrate and test every time a task is completed (many times a day)

10. Sustainable pace

- 40 hours a week reasonable

- Never overtime two consecutive weeks

11. On-site customer

- Include real customer on the team

12. Coding standards

- Everyone needs to use the same rules

**Ques14**: What are major requirements engineering activities?

**Ans**: Requirements engineering - A set of activies related to the development and agreement of the final set of requirement specifications.

The following requirements engineering activities are involved in a software project:

- Elicitation

- Documentation and definition

- Specification

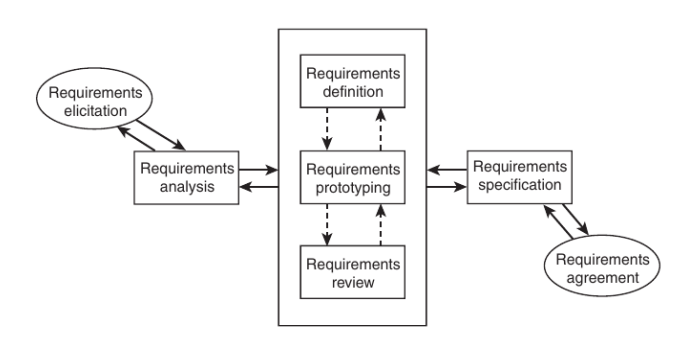
- Prototyping

- Analysis

- Review and validation

- Agreement and acceptance

**Ques15**: Requirements engineering preparation process.

**Ans**: 

**Ques16**: Criteria for requirement prioritization

**Ans**: Establishing the priority of the requirements may be based on many criteria, including the following:

- Current customer demands

- Competition and current market condition

- Future customer needs

- Immediate sales advantage

- Critical problems in the existing product

**Ques17**: Pipes-and-filters architecture.

**Ans**:

Software architecture is the structure or structures which comprise elements, their externally visible properties, and relationships among them.

Pipes-and-filters architecture: A style widely used for Unix scripts, and for signal processing applications. It consists of a series of processes connected by “pipes”. The output of a process serves as the input of the next one; processes no not need to wait until the previous process finishes but can start processing input as soon as some of the input is available. Most of the time the topology is linear, but occasionally there could be forks. Although the most popular application of this style is in combining Unix commands, it is also the conceptual model for many audio and video processing applications.

**Ques18**: Client-Server architecture.

**Ans**:

Client-server architecture: A style showing a clear demarcation between clients and servers, which reside on different nodes in a network. Components interact through basic networking protocols or through remote procedure calls (RPC). Usually there will be many clients accessing the same server. The client-server architecture was heavily influenced by various hardware changes and hardware cost. First, with less powerful terminals or client boxes, many of the processing resided on the server boxes. As the client machines improved in power and dropped in price, more functions were placed in the clients. An interesting side note on this development of placing more functions on the clients or personal desktops is that it created the need to support these clients. A whole profession called IT desktop support became a neccessity as a result of the powerful client desktops.

**Ques19**: Shared-data architecture.

**Ans**:

Shared-data architecture

- Two component types – data repository and data accessor

- Data repository – provides reliable permanent storage

- Data accessors – access data in repositories, perform computations, and may put the results back

- Communication between data accessors is only through the repository

- Two variations possible

+ Black board style: if data is posted in a repository, all accessors are informed; i.e. shared data source is an active agent

+ Repository style: passive repository

- Eg. database oriented systems; web systems; programming environments,..

Example: - A student registration system of a university

- Repository contains all the data about students, courses, schedules,…

- Accessors like administration, approvals, registration, reports which perform operations on the data

**Ques20**: Relational database design

**Ans**:

- Most databases use relational technology

- Relations (tables)

+ Two-dimensional sets

+ Rows (tuples), Columns (attributes)

+ Row may be entity, relationship or attribute

+ Primary key

+ Foreign keys

**Ques21**: UML diagrams in object-oriented development.

**Ans**:

**Ques22**: Volume metric by Halstead to measure the lexical complexity of the source program.

**Ans**:

n1 = number of distinct operators

n2 = number of distinct operands

N1 = sum of all occurrences of n1

N2 = sum of all occurrences of n2

Program vocabulary or unique tokens, n = n1 + n2

Program length, N = N1 + N2

Volume, V = N\*()

Potential volume, V@ = (2 + n2@) log2 (2+n2@)

- Volume of a function should be >20 &<1000

- Volume of a parameterless one-line function that is not empty is about 20; a volume >1000 tells that the function probably does too many things

- The volume of a file should >100 & < 8000

**Ques23**: Levels of coupling between modules.

**Ans**: The amount or degree of coupling is generally divided into five distinct levels, listed from the worst to the best:

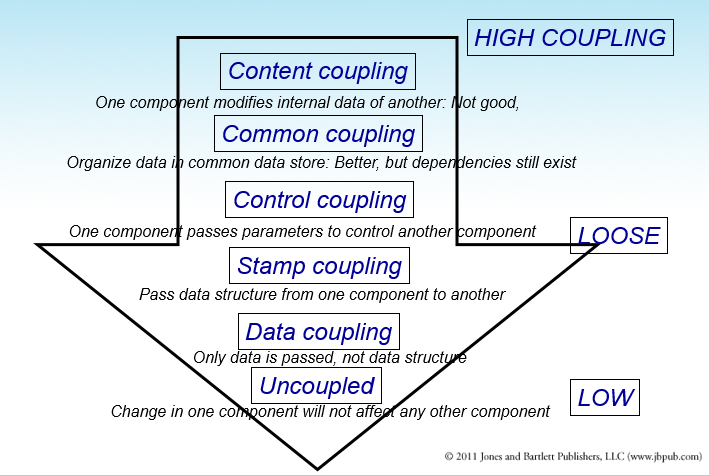
- Content coupling

- Common coupling

- Control coupling

- Stamp coupling

- Data coupling



**Ques24**: Cohesion levels of a module

**Ans**:

- There are many levels of cohesion

+ Coincidental

+ Logical

+ Temporal

+ Communicational

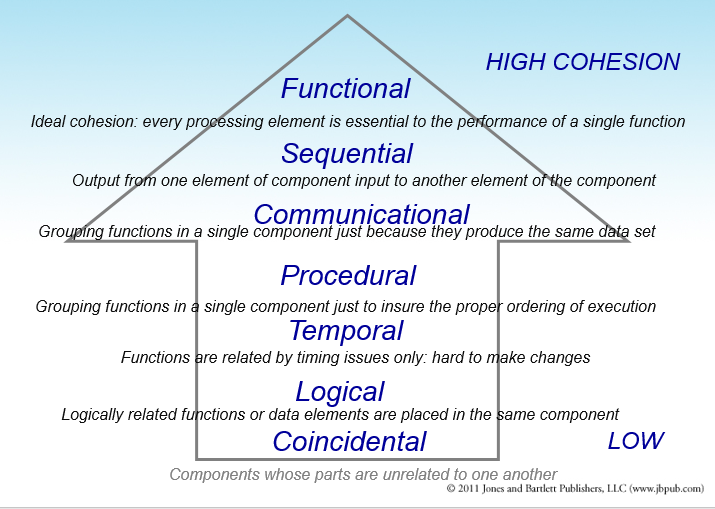
+ Sequential

+ Functional

- Coincidental is lowest, functional is highest

- Scale is not linear

- Functional is considered very strong



**Ques25**: 8 rules for user interface design by Shneiderman and Plaisant.

**Ans**: Shneiderman and Plaisant (2005) have identified the following 8 rules of interface design:

1. Strive for consistency

2. Enable frequent users to use short cuts

3. Offer informative feedback

4. Design dialogues to yield closure

5. Offer error prevention and simple error handling

6. Permit easy reversal of actions

7. Support internal locus of control

8. Reduce short-term memory

**Ques26**: Common coding errors.

**Ans**:

- Memory Leaks

+ Memory that is not freed when no longer needed

- Freeing an already freed resource

- NULL dereferencing

+ Trying to access the contents of a location that points to NULL

- Lack of unique addresses

+ Caused by aliasing

- Array index out of bound

- Arithmetic exceptions

- Off by one

+ Starting at 1 when we should start at 0

- Enumerated data types

+ Cause overflow and underflow

- Illegal use of & instead of &&

- String handling errors

- Buffer overflow

**Ques27**: Programming practices.

**Ans**: Some programming pratices

- Control constructs: Use only a few structured constructs (rather than using a large number of constructs)

- Goto: Use them sparingly, and only when the alternatives are worse

- Information hiding: Use information hiding

- Use-defined types: use these to make the programs easier to read

- Nesting: Avoid heavy nesting of if-then-else

+ Easier to do if conditions are disjoint

- Module size: Should not be too large – generally means low cohesion

- Module interface: make it simple

- Robustness: Handle exceptional situations

- Side effects: Avoid them, document when exist

+ Use of global variables, etc.

- Switch case with default

- Empty catch block: always have some default action rather than empty

- Empty if, while: bad practice

- Read return: should be checked for robustness

- Return from finally: should not return from finally

- Correlated parameters: Should check for compatibility

+ Parameter needs to match the operation

**Ques28**: Java coding standards.

**Ans**:

- Naming conventions

+ Package name should be in lower case

\* *mypackage, edu.iitk.maths*

+ Type names should be nouns and start with uppercase

*\* Day, DateOfBirth, EventHandler*

+ Variable names should be nouns in lowercase

\* *name, amount*

+ Variables with large scope should have long names; variables with small scope short names

+ Loop iterators should be i, j, k…

+ Constant names should be all uppercase

\* *PI, MAX\_INTERACTIONS*

+ Method names should be verbs starting with lower case

\* *getValue()*

+ Private class variables should have the \_ suffix

\* *private int value\_*

+ Prefix is should be used for Boolean methods

\* *isStatus*

+ Avoid negative Boolean variable names

\* *isNotCorrect*

+ Use term compute for methods that computes something

\* *computeMean()*

+ Use term find for methods that looks up something

\* *findMin()*

+ Exception classes should be suffixed with Exception

\* *OutOfBoundException*

- Files

+ Source files should have .java extension

+ Each file should contain one outer class and the name should be same as file

+ Line length should be less than 80

\* If longer continue on another line

+ Special characters should be avoided

- Statements

+ Variables should be initialized where declared in the smallest possible scope

+ Declare related variables together; unrelated variables should be declared separately

+ Class variables should never be declared public

+ Loop variables should be initialized just before the loop

+ Avoid using break and continue in loops

+ Avoid executable statements in conditionals

+ Avoid using the do… while construct

+ Avoid complex conditional expressions

\* Introduce temporary Boolean variables instead

+ Avoid executable statements in conditionals

- Commenting and layout

+ Single line comments for a block should be aligned with the code block

+ There should be comments for all major variables explaining what they represent

+ A comment block should start with a line with just /\* and end with a line with \*/

+ Trailing comments after statements should be short and on the same line and shifted far enough to separate them from statements

**Ques29**: Test driven development method.

**Ans**:

- This coding process changes the order of activities in coding

- In TDD, programmer first writes the test scripts and then writes the code to pass the test cases in the script

- This is done incrementally

- Is a relatively new approach, and is a part of the extreme programming (XP) approach

- In TDD, you write just enough code to pass the test

- I.e. code is always in sync with the tests and gets tested by the test cases

+ Not true in code first approach, as test cases may only test part of functionality

- Responsibility to ensure that all functionality is there is on test case design, not coding

- Help ensure that all code is testable

- Focus shifts to how code will be used as test cases are written first

+ Helps validate user interfaces specified in the design

+ Focuses on usage of code

- Functionality prioritization happens naturally

- Has possibility that special cases for which test cases are not possible get left out

- Code improvement through refactoring will be needed to avoid getting a messy code

**Ques30**: Checklist in code inspections.

**Ans**: Checklists are generally used to focus the attention on defects

- Some items in a checklist

+ Do data definitions exploit the typing capabilities of the language

+ Do all pointers point to something

\* Any dangling pointers

\* Checked for NULL when being used

+ Are all vars and pointers initialized

+ Are all array indexes within bounds

+ Will all loops always terminate

\* Are loop termination conditions correct

\* Is number of loop iterations off by one

+ Are divisors checked for zero

+ Do actual and formal parameters match

+ Are all variables used

\* Are all output variables assigned

+ Can statements placed in loop be placed outside loop

+ Are the labels unreferenced

+ Any security flaws

+ Is input data being checked

+ Are the local coding standards met

**Ques31**: Categorization of software testing techniques.

**Ans**:

Who does the testing? Basically, we have three options here:

- Programmers

- Testers

- Users

What is tested? There are three main levels:

- Unit testing

- Functional testing

- Integration and system testing

Why are we testing? Which specific kinds of defects are we trying to detect or which risks are we trying to mitigate. There are diffirent types of testing conducted for different purposes:

- Acceptance testing

- Conformance testing

- Configuration testing

- Performace testing

- Stress testing

- User-interface testing

How do we generate and choose test cases? We can choose test cases based on the following:

- Intuition

- Specification

- Code

- Existing test case

- Faults

**Ques32**: Software testing levels.

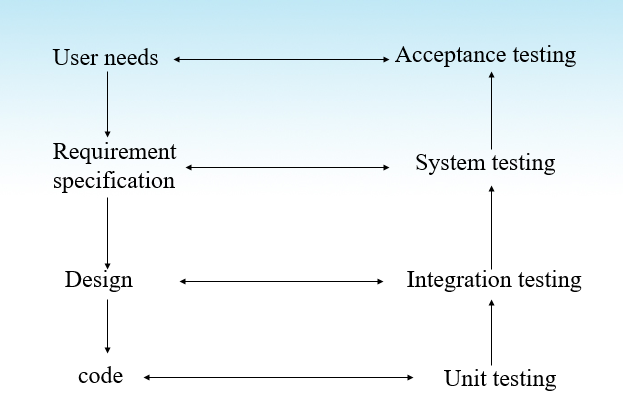
**Ans**:

- The code contains requirement defects, design defects, and coding defects

- Nature of defects is different for different injection stages

- One type of testing will be unable to detect the different types of defects

- Different levels of testing are used to uncover these defects



**Ques33**: Software testing technique: equivalence class partitioning

**Ans**: Equivalence class partitioning

- Black-box technique

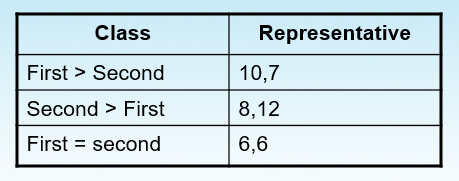
- Divide the input into several classes, deemed equivalent for purposes of finding errors.

+ If fails for one equivalent class, assume it fails for all equivalent classes; if passes for one, passes for all

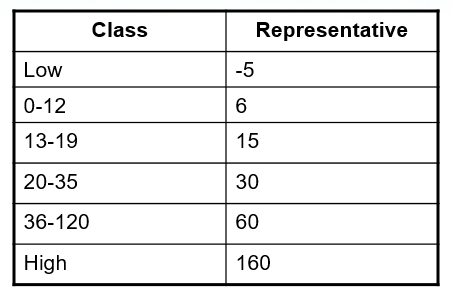
- Representative for each class used for testing.

- Equivalence classes determined by intuition and specs

Ex: largest



Ex: age groups



**Ques34**: Software testing technique: boundary value analysis.

**Ans**:

- Boundaries are error-prone

- Do equivalence-class partitioning, add test cases for boundaries (boundary, +1, -1)

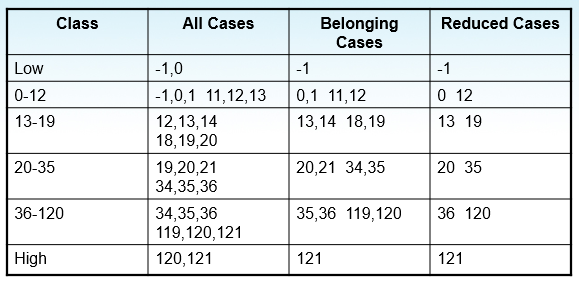
- Reduced cases: consider boundary as falling between numbers

+ boundary is 12, normal: 11,12,13; reduced: 12,13 (boundary between 12 and 13)

- Large number of cases (~3 per class)

- Only for ordinal values (or no boundaries)

Ex: age groups



**Ques35**: Software testing technique: path analysis.

**Ans**:

- White-Box technique

- Two tasks

+ Analyze number of paths in program

+ Decide which ones to test

- Decreasing coverage

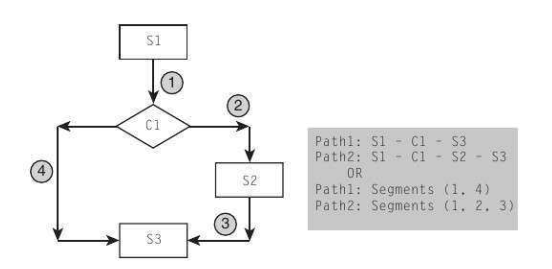
+ Logical paths

+ Independent paths

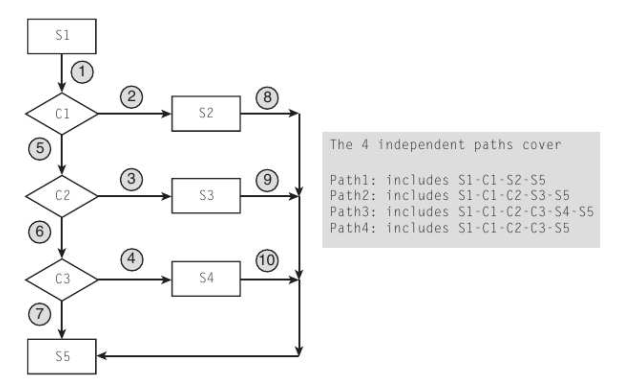
+ Branch coverage

+ Statement coverage

Ex: A simple logical structure



Ex: A CASE structure



**Ques36**: Items that are managed in software configuration management.

**Ans**:

- Software artifacts

- Change control

- System builds

**Ques37**: Why is software configuration management needed? What is functionality needed for configuration management?

**Ans**: Because:

- To satisfy the basic project objective of delivering the product to the client

+ What files should comprise the product

+ What versions of these files

+ How to combine these to make the product

- Just for this, versioning is needed, and state of different items has to be tracked

- There are other functions of CM

Functionality is needed for CM:

- Give states of programs

- Give latest version of a program

- Undo a change and revert back to a specified version

- Prevent unauthorized changes

- Gather all sources, documents, and other information for the current system

**Ques38**: Parts storage and access model for software configuration management

**Ans**:

- Basic functions:

+ Create a part

+ Delete a part

- Access functions

+ View a part

+ Modify a part

+ Return a part

- Control and service functions

+ Import part(s)

+ Export part(s)

+ List parts

+ Set release or version numbers

+ Increment release or version numbers

+ Change part name, version, release, artifact type, etc

+ Gather parts

+ Merge into a part

+ Promote parts

+ compare parts

+ Lock / unlock pars

+ Where used and cross reference the parts

**Ques39**: What activities are needed for defect support? What is the trend of the number of defects dicovered over time?

**Ans**: Activities are needed for defect support:

- Project the # of problems and problem arrival rate

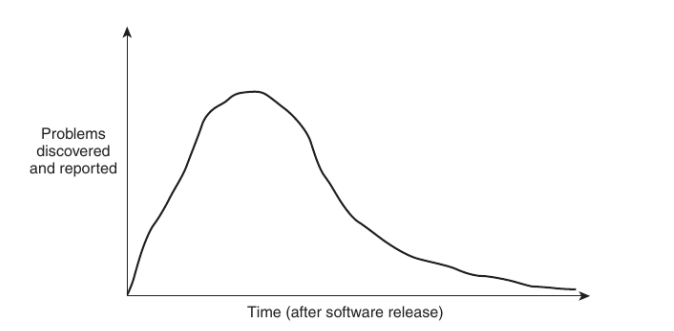
- Estimate and plan the needed support resources

- Educate and build the defect support team

- Defect reporting and tracking

- Defect identification, fix, and release

Trend of the number of defects discovered over time



- During the period right after release, many problems are discovered and reported.

- The amount of problems discovered eventually decreases; at the same time the nature of the problem discovered becomes more difficult to diagnose.

**Ques40**: What activities are needed in product “sun-setting” period.

**Ans**:

- Stop any product’s additional feature and enhancement

- Fix only the high severity problems

- Announce new replacement product

- Encourage new and existing customers to move to new product

- Notify all old users on the old product of the planned termination date

- Provide names of other vendors who are willing to support the old product to the customers who chooses to stay

- Terminate the customer product and withdraw the product from market

**Ques41**: What items are needed in a maintenance change request form?

**Ans**:

- Change Request #

- Request Date

- Requestor Name

- Request Status

- Brief Change Request Description

- Areas Impacted by the Change Request

- Estimated Effort

- Inclusion in Maintenance Rel.#

**Ques42**: Phases and goals of software project management.

**Ans**:

Phases of software project management:

- Planning

- Organizing

- Status monitoring

- Adjustment

Goals of software project management

- End results of the project satisfy the customer’s needs

- All the desired and the needed product/project attributes (quality, security, productivity, cost, etc.) are met

- Target milestones and the over-all schedule are met.

- Team members are operating effectively and at a high level of morale

- Required tools and other resources are made available and are effectively utilized

**Ques43**: Planning phase of software project management.

**Ans**:

- The 1st step of project planning is to understand the requirements of the project.

+ This step itself may be a mini-project

- Then the following 4 steps are included in the rest of project planning:

+ Estimate

\* the work effort,

\* the schedule, and

\* the needed resources

+ Clearly define and establish measurable goals for the project

+ Determine the project resource allocations of

\* people,

\* process,

\* tools, and

\* facilities

+ Identify and analyze the project risks

**Ques44**: Organizing phase of software project management.

**Ans**:

- Once a project plan is formulated or partially formulated, organizing may start

+ Organization structure needs to be designed

+ Human resource hiring needs to start and be completed along with acquisition of other resources

+ Any required education and training have to be completed

+ Mechanisms for tracking must be established

\* Risk tracking and mitigation

\* Project goal monitoring

**Ques45**: Work Breakdown Structure for estimating complete project work effort.

Ans:

- Examine and determine the external deliverables of the project.

- Identify the steps and tasks required to produce each of the deliverables, including the tasks that are required to produce any intermediate internal deliverables

- Sequence the tasks, showing any potential for parallelism

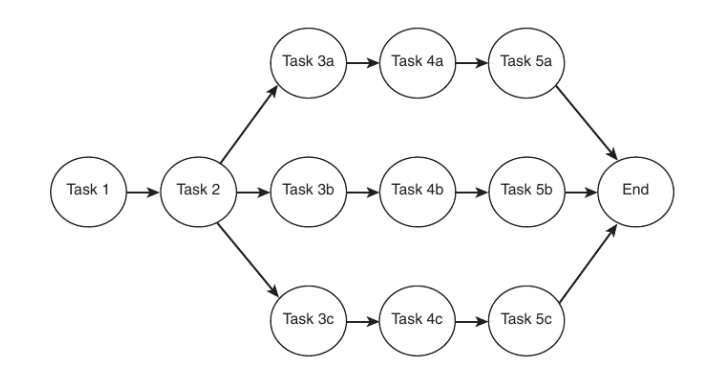
- Provide an estimate to complete each of the tasks

- Provide an estimate of the productivity of the personnel that is most likely to be assigned to each of the tasks

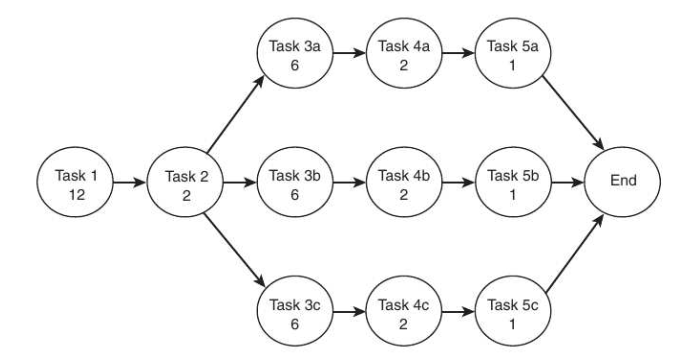
- Calculate the time required to accomplish each task

- For each of the external deliverable, lay out the timeline of all the tasks needed to produce that deliverable and label the resources that will be assigned to the tasks.

Ex: A WBS network of tasks



Task network with estimated time units



Initial schedule estimate

