CSE6214 Tutorial 7 Answers

By Dr. Mohana

Q1: Why is it important that we understand the skills of the users when creating the user interface of a software?

In order to produce user-friendly software, the features available should suit the different skill levels of the users. Novices would require more help features, and experts would require less help features but more controls and short cuts. So, it is important that the software provide for the corresponding requirements otherwise the software could be considered not user-friendly.

Q2: What is task analysis and how will it improve the user interface design of a software?

Task analysis is the process to understand the tasks and subtasks involved in order to design the interfaces to suit the tasks. The following techniques are involved:

- Examination of use cases
- Task Elaboration
- Object Elaboration
- Workflow Analysis
- Hierarchical Representation

Q3: Discuss Component-level Design.

- It occurs after the first iteration of architectural design has been completed.
- At this stage, the overall data and program structure of the software has been established.
- The intent is to translate the design model into operational software.
- But the level of abstraction of the existing design model is relatively high, and the abstraction level of operational program is low.

Q3: Discuss Component-level Design. (con’t)

- This Component-level Design defines the data structures, algorithms, interface characteristics and communication mechanisms allocated to each software component.

Q4: Why is Component-level Design important?

- We have to be able to determine whether the software will work before we build it.
- The Component-level Design represents the software in a way that allows you to review the details of the design representations (such as the data, architectural and interface designs).
- It also provides a means for assessing whether data structures, interfaces and algorithms will work.

Q5: What are the steps in Component-level Design?

- Design representations of data, architecture and interfaces form the foundation for component-level design.
- The class definition or processing narrative for each component is translated into a detailed design that makes use of diagrammatic or text-based forms that specify internal data structures, local interface detail, and processing logic.

Q5: What are the steps in Component-level Design? (con’t)

- Design notation encompasses UML diagrams and supplementary forms.
- Procedural design is specified using a set of structured programming constructs.
- It is often possible to acquire existing reusable software components rather than building new ones.

Q6: What are the advantages of Component-Based Development?

- Reduce the required time to develop the functionalities that could be implemented by existing components
- Reuse of components that have been developed, refined, and well-tested.
- The framework for component-based systems provide flexibility and ease of modification to include new components that could be required in future development and maintenance work.

Q7: Discuss the options for reusable components to be identified and selected for integration with a software that is being developed.

- The reusable components could be obtained from the following sources:
  - – Commercial off-the-shelf (COTS) components
  - – Internally-developed reusable components
- The interfaces for these components must be compatible with the architecture of the system that is to be built.

CSE6214 Tutorial 8 Answers

By Dr. Mohana

Q1:Discuss the need to design Web applications. When is it necessary to design them?

- when content and function are complex
- when the size of the WebApp encompasses hundreds of content objects, functions, and analysis classes
- when the success of the WebApp will have a direct impact on the success of the business

Q2:For the end-users, what are the important quality dimensions of Web applications?

- Time – changes to the Web site at different times
- Structural – organization and connections of the parts in the Web site
- Content – quality of the contents
- Accuracy and Consistency – accuracy and consistency of data/contents
- Response Time and Latency – timely responses to requests
- Performance – Web site connection and performance variations at different times of the day

Q3:Describe each level of the WebE Design Pyramid.

- Interface design – design of the Web pages
- Aesthetic design – aesthetic considerations for the Web site
- Content design – design of content objects
- Navigation design – design of the navigation throughout the Web site
- Architecture design – structural representation of the Web application
- Component design – processing requirements of the Web application

Q4:What are the important considerations in the development of mobile applications?

Refer to lecture slides:

- Multiple hardware and software platforms
- Many development frameworks and programming languages.
- Many app stores with differing acceptance rules and tool requirements
- Short development cycles
- User interface limitations
- Complex camera/sensor interaction
- Effective use of context
- Power management
- Security and privacy models/policies
- Device limitations (computation and storage)
- Integration of external services
- Testing complexities

Q5: Discuss the typical mistakes in mobile application development and suggest some approaches to address them.

Some of the typical mistakes:

- Kitchen sink – trying to do too much; determine the important and relevant functions first
- Inconsistency – interfaces and items designed inconsistently; use standard interfaces and maintain consistency in the application flow
- Overdesigning – complexity in the design; simplify the design, reduce complexity

Q5: Discuss the typical mistakes in mobile application development and suggest some approaches to address them. (con't)

Some of the typical mistakes:

- Lack of speed – too much processing by the application; assess the processing requirements and improve/optimize the efficiency of algorithms
- Verbiage – too much textual contents; simplify the descriptions, display only the necessary info
- Non-standard interaction – interaction items that are not conventional; use common interaction items

Q5: Discuss the typical mistakes in mobile application development and suggest some approaches to address them.

Some of the typical mistakes:

- Help-and –FAQ-itis – lack of Help and FAQ; put some Help or FAQ to assist the users on major functions

Tutorial – Lecture 8 – Part B: Activity 1 – Cohesion

Describe the following types of cohesion:

- Functional
- Layer
- Communicational

Tutorial – Lecture 8 – Part B: Activity 1 – Cohesion

Functional – Exhibited primarily by operations, this level of cohesion occurs when a component performs a targeted computation and then returns a result.

Layer – Exhibited by packages, components and classes, this type of cohesion occurs when a higher layer accesses the services of a lower layer, but lower layers do not access higher layers.

Communicational – All operations that access the same data are defined within one class. In general, such classes focus solely on the data in question, accessing and storing it.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

Match the following coupling categories with its descriptions. You should select the best answer from Table 1.

Table 1 – List of Coupling Categories

Common coupling	**Data coupling**	**Control coupling**
Routine call coupling	Type use coupling	Couple coupling
Inclusion/Import coupling	Content coupling	External coupling
Stamp coupling	Unusual coupling	Good coupling

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2a) Content coupling

- Occurs when one component “surreptitiously” modifies data that is internal to another component
- This violates information hiding – a basic design concept.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2b) Common coupling

- Occurs when a number of components all make use of a global variable.
- Although this is sometimes necessary (e.g. for establishing default values that are applicable throughout an application), common coupling can lead to uncontrolled error propagation and unforeseen side effects when changes are made.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2c) _Control coupling_

- Occurs when operation A () invokes operation B() and passes a control flag to B.
- The control flag then "directs" logical flow within B.
- The problem with this form of coupling is that an unrelated change in B can result in the necessity to change the meaning of the control flag that A passes.
- If this overlooked, an error will result.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2d) Stamp coupling

- Occurs when ClassB is declared as a type of an argument of an operation of ClassA.
- Because ClassB is now a part of the definition of ClassA, modifying the system becomes more complex.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2e) Data coupling

- Occurs when operations pass long strings of data arguments.
- The “bandwidth” of communication within classes and components grows and the complexity of the interfaces increases.
- Testing and maintenance are more difficult.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2f) _Routine call coupling_

- Occurs when one operation invokes another.
- This level of coupling is common and is often necessary.
- However, it does increase the connectedness of a system.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2g) Type use coupling

- Occurs when component A uses a data type defined in component B.
- For example: this occurs whenever "a class declares an instance variable or a local variable as having another class for its type"
- If this type definition changes, every component that uses the definition must also change.

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2h) Inclusion/Import coupling

- Occurs when component A imports or includes a package or the content of component B

Tutorial – Lecture 8 – Part B: Activity 2 – Coupling

2i) External coupling

- Occurs when a component communicates or collaborates with infrastructure components (e.g.: operating systems functions, database capability, and telecommunications functions).
- Although this type of coupling is necessary, it should limit to a small number of components or classes within a system.

CSE6214 Tutorial 9 Answers

By Dr. Mohana

Tutorial – Lecture 9: Software Quality Assurance– Part A

1. Search the following URL: http://en.wikipedia.org/wiki/ISO_9000 . Discuss the significance of ISO 9000 certification to an organization that develops software products.
2. What are the variations of the ISO standards that are more specific to software industry?

Q1: Discuss the significance of ISO 9000 certification to an organization that develops software products.

- An organization that is certified would have software process in place to ensure quality products.
- ISO 9000 standards specify that quality management system is created

Q2: What are the variations of the ISO standards that are more specific to software industry?

- The TickIT guidelines are an interpretation of ISO 9000 produced by the UK Board of Trade to suit the processes of the information technology industry, especially software development.
- ISO/IEC/IEEE 90003:2018 provides guidelines for the application of ISO 9001:2015 to computer software.

Tutorial – Lecture 9: Software Quality Assurance– Part A

3. Search the following URL: http://en.wikipedia.org/wiki/Capability_Maturity__Model_Integration. Discuss the significance of CMMI certification to an organization that develops software products.

4. If your organization is certified as CMMI Level 3, what does it indicate? Can your organization be ISO certified also?

Q3: Discuss the significance of CMMI certification to an organization that develops software products.

- An organization is certified at Level 1 – 5, where certification at each level specifies that the organization have the software processes defined for the particular level

Q4: If your organization is certified as CMMI Level 3, what does it indicate? Can your organization be ISO certified also?

The organization has Defined processes, specifically the following processes:

Maturity Level 2 - Managed	• CM - Configuration Management • MA - Measurement and Analysis • PMC - Project Monitoring and Control • PP - Project Planning • PPQA - Process and Product Quality Assurance • REQM - Requirements Management • SAM - Supplier Agreement Management

Q4: If your organization is certified as CMMI Level 3, what does it indicate? Can your organization be ISO certified also?

The organization has Defined processes, specifically the following processes:

Maturity Level 3 - Defined	• DAR - Decision Analysis and Resolution • IPM - Integrated Project Management • OPD - Organizational Process Definition • OPF - Organizational Process Focus • OT - Organizational Training • PI - Product Integration • RD - Requirements Development • RSKM - Risk Management • TS - Technical Solution • VAL - Validation • VER - Verification

Tutorial – Lecture 9: Software Quality Assurance– Part B

1. Discuss how the quality of a software product can be determined.
2. Which is more important, to have a quality product or quality process? Discuss the benefits of each and give a comparison.
3. Explain the “Software Quality Dilemma” and what can be done to manage it.
4. What are Software Quality Assurance Activities?
5. Explain whether having ISO certification would ensure that Software Quality Assurance activities would be performed by an organization.

Q1:Discuss how the quality of a software product can be determined.

The quality of a software product is determined by evaluating how well it satisfies both functional and non-functional requirements.

- Conformance to requirements: The software must correctly implement all specified functional requirements.
- Quality attributes (factors): These include correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, and interoperability.
- User satisfaction: A quality product meets user expectations and performs reliably in real-world usage.

Q1:Discuss how the quality of a software product can be determined.

The quality of a software product is determined by evaluating how well it satisfies both functional and non-functional requirements.

- Measurement and metrics: Defect density, failure rates, performance benchmarks, and usability metrics help quantify quality.
- Verification and validation: Reviews, inspections, and testing ensure the product is built correctly and is the right product for users.

Q2: Which is more important, to have a quality product or quality process? Discuss the benefits of each and give a comparison.

Both are important, but a quality process is generally more critical in the long term.

Quality Product – Benefits

- Meets customer needs and expectations.
- Reduces immediate rework and maintenance costs.
- Improves user satisfaction and organizational reputation.

Quality Process – Benefits

- Produces consistent and predictable results.
- Reduces defects early in development.
- Improves productivity and project control.
- Enables continuous improvement across projects.

Q2: Which is more important, to have a quality product or quality process? Discuss the benefits of each and give a comparison.

Aspect	Quality Product	Quality Process
Focus	End result	How the result is achieved
Timeframe	Short-term success	Long-term sustainability
Repeatability	Not guaranteed	High
Risk Reduction	Limited	Strong

Q3: Explain the "Software Quality Dilemma" and what can be done to manage it.

Software Quality Dilemma

The software quality dilemma arises because:

- High quality requires time, cost, and effort, and
- Projects are often under pressure to deliver faster and cheaper.
- As a result, quality-related activities (reviews, testing, documentation) are often reduced or skipped, increasing defects and long-term costs.

Q3: Explain the "Software Quality Dilemma" and what can be done to manage it.

Managing the Dilemma

- Invest in early defect prevention (reviews, inspections).
- Adopt incremental and iterative development.
- Use risk-based testing to focus on critical components.
- Educate stakeholders that quality saves cost over time.
- Integrate quality activities into every development phase, not just testing.

Q4: What are Software Quality Assurance Activities?

Software Quality Assurance activities are planned and systematic actions that ensure software processes and products conform to standards and requirements. SQA focuses on preventing defects, not just detecting them.

Common SQA activities include:

- Process definition and standard enforcement
- Formal technical reviews and inspections
- Auditing software work products
- Ensuring compliance with standards and procedures
- Test planning and execution oversight
- Defect tracking and reporting
- Metrics collection and analysis
- Configuration management support

Q5: Explain whether having ISO certification would ensure that Software Quality Assurance activities would be performed by an organization.

No, ISO certification alone does not guarantee effective SQA activities.

- ISO standards (e.g., ISO 9001) ensure that documented processes exist and are followed.
- Certification confirms process compliance, not product quality.
- An organization may follow documented procedures but still perform SQA activities superficially.
- ISO certification supports SQA by providing a structured framework, but:
  - The commitment of management,
  - The competence of staff, and
  - The proper execution of SQA practices are what truly ensure software quality.

CSE6214 Tutorial 10 Answers

By Dr. Mohana

Tutorial – Lecture 10: Software Testing Techniques– Part A

1. Two main approaches in testing are white box testing and black box testing. Explain white box and black box testing approaches and contrast these two testing approaches.

Q1: Explain white box and black box testing approaches and contrast these two testing approaches.

- White box testing is an approach to test where the test cases are derived from the source code structure. It is also known as structural testing. Its relatively applies to small program units. Tester analyzes the code and uses the knowledge about the structure component to derive test data. The code will be analyzed to determine how many test cases needed to guarantee all of the statements in the program are executed at least once during the testing process.

Q1: Explain white box and black box testing approaches and contrast these two testing approaches.

- Black box testing is an approach where test cases are derived from component or program specification. The system is a black box whose behavior can only be determined by studying the inputs and the related outputs. The test is concern with the system functionality. The tester presents inputs to the component and examines the corresponding outputs. If outputs are not predicted then the test successfully detected a problem with the software. Black box testing complements the white box testing.

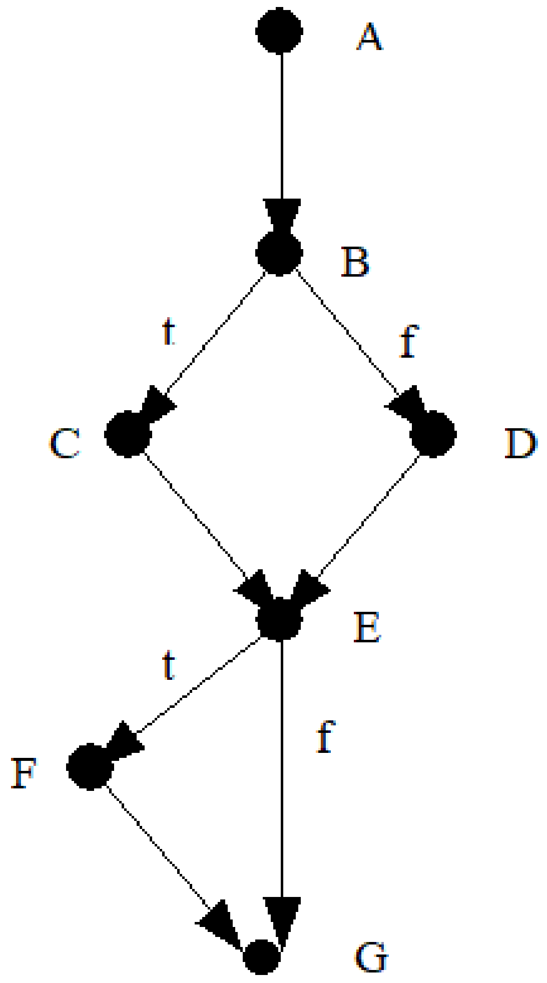
Tutorial – Lecture 10: Software Testing Techniques– Part A

2. Given the following program:

```
A     input (mark)
B     if mark < 45
C           then print "Fail"
D           else print "Pass"
E     if mark > 80
F           then print "With Distinction"
G     end
```

(a) Draw the flowgraph for this program.

Q2 (a) Draw the flowgraph for this program.



Tutorial – Lecture 10: Software Testing Techniques– Part A

2. Given the following program:

(b) Based on the flowgraph, determine the cyclomatic complexity of this program.

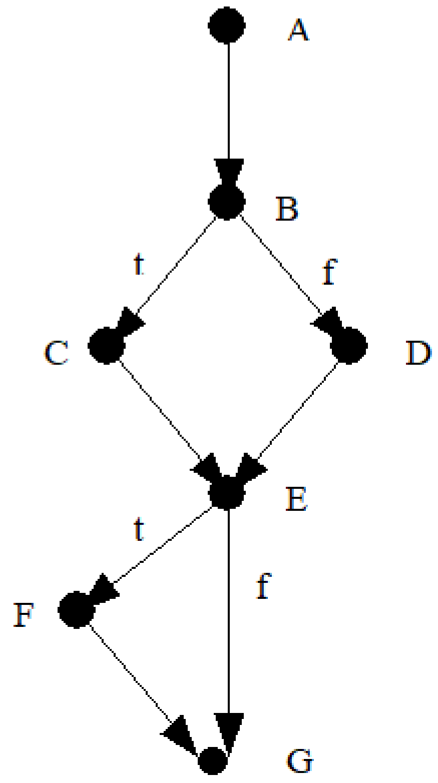
(c) How many test cases is required to completely test all the paths of this program?

Q2 (b) Based on the flowgraph, determine the cyclomatic complexity of this program.

- $M = E - N + 2 * P$

- M= Cyclomatic complexity
- E = number of edges
- N = number of nodes
- P = number of end points

Q2 (b) Based on the flowgraph, determine the cyclomatic complexity of this program.



- E = 8
- N = 7
- P = 1

- M = E – N + 2 * P
- M = 8 – 7 + 2(1)
- M = 3

Q2 (c) How many test cases is required to completely test all the paths of this program?

- Test cases = 3
- There are two decision points:

  B: mark < 45

  E: mark > 80
- To cover all paths, we need to consider combinations of these decisions.

Q2 (c) How many test cases is required to completely test all the paths of this program?

Paths:

- B True (mark < 45) → C → E False (mark ≤ 80) → G
  - Output: Fail
- B False (mark ≥ 45) → D → E False (mark ≤ 80) → G
  - Output: Pass
- B False (mark ≥ 45) → D → E True (mark > 80) → F → G
  - Output: Pass + With Distinction

Q2 (c) How many test cases is required to completely test all the paths of this program?

Test Cases Needed

- **Case 1**: mark = 30 → Path 1 (*Fail*)
- **Case 2**: mark = 60 → Path 2 (*Pass*)
- **Case 3**: mark = 90 → Path 3 (*Pass + With Distinction*)
- Thus, **3 test cases** are required to completely test all possible execution paths.

Tutorial – Lecture 10: Software Testing Techniques– Part A

3. How does unit testing differ for object-oriented testing as compared to conventional software unit testing?
4. What is scenario-based testing?
5. Describe the how test cases are derived from behavior models to facilitate interclass testing?

Q3: How does unit testing differ for object-oriented testing as compared to conventional software unit testing?

- You can no longer test a single operation in isolation (the conventional view of unit testing) but rather, as part of a class.
- Rather than testing an individual module, the smallest testable unit is the encapsulated class.
- Because a class can contain a number of different operations and a particular operation may exist as part of a number of different classes, the meaning of unit testing changes dramatically.

Q4: What is scenario-based testing?

- The user tasks described in the use-cases are used to construct the test cases.
- It is used to uncover errors that occur when actors interact with the software (focus is on user behavior, not product behavior).
- Generally, it is better to spend more time reviewing the use cases as they are created than spending more time on testing.

Q5: Describe the how test cases are derived from behavior models to facilitate interclass testing?

- Test cases must cover all states in the state transition diagram
- Breadth first traversal of the state model can be used (test one transition at a time and only make use of previously tested transitions when testing a new transition)
- Test cases can also be derived to ensure that all behaviors for the class have been adequately exercised

CSE6214 Tutorial 11 Answers

By Dr. Mohana

Tutorial – Lecture 11: Software Testing Strategies– Part A

1. Describe the difference between verification and validation.
2. Why is a highly coupled module difficult to unit test?
3. Why is regression testing an important part of any integration testing procedure?

Q1: Describe the difference between verification and validation.

- Verification focuses on the correctness of a program by attempting to find errors in function or performance.
- Validation focuses on conformance to requirements—a fundamental characteristic of quality.

Q2: Why is a highly coupled module difficult to unit test?

- A highly coupled module interacts with other modules, data and other system elements.
- Therefore, its function is often dependent of the operation of those coupled elements.
- In order to thoroughly unit test such a module, the function of the coupled elements must be simulated in some manner.
- This can be difficult and time consuming.

Q3: Why is regression testing an important part of any integration testing procedure?

- The goal of integration testing is to make sure that independent modules that work correctly on their own do not interfere with one another when added to the same program (unforeseen side effects are always possible).
- Regression testing checks for defects propagated to other modules by changes made to an existing program.

Tutorial – Lecture 11: Software Testing Strategies– Part B

Match the following classes and approaches of testing with its descriptions. You should select the best answer from Table 1.

Table 1 – List of Tests Classes and Approaches.

Deployment testing	Stress testing	Black-box testing
Performance testing	Beta test	Recovery Testing
White-box testing	Unit testing	Bottom-up Integration Testing
Regression Testing	Alpha testing	Top-down Integration Testing
Security testing	Smoke testing	Component testing

Tests Classes and Approaches

1. Unit testing focuses verification effort on the smallest unit of software design, the software component or module
2. Bottom-up Integration Testing begins construction and testing with atomic modules which refers to the components at the lowest levels in the program
3. Top-down Integration Testing is an incremental approach to construction of the software architecture.

Tests Classes and Approaches

4. Security testing attempts to verity that protection mechanisms built into a system will protect it from improper penetration

5. Regression Testing is the re execution of some subset of tests that have been conducted to ensure that changes have not been propagated unintended side effects

6. Component testing provides a number of benefits when it is applied on complex, time-critical software projects

Tests Classes and Approaches

7. Alpha testing is conducted in a controlled environment. It is conducted at the developer's site by a representative group of end user.

8. Recovery Testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed

9. Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency or volume

Tests Classes and Approaches

10. Deployment testing is also called configuration testing. It examines all installation procedures and specialized installation software that will be used customers, and all the documentation that will be used to introduce the software to end-users.

11. Performance testing is designed to test the run-time performance of software within the context of an integrated system

12. Smoke testing is an integrating testing approach that is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project in a frequent basis.

Tests Classes and Approaches

13. Beta test is a testing that is conducted at one or more end-user sites.

14. White-box testing or also called as glass-box testing is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases.

15. Black-box testing examines some fundamental aspects of a system with little regard for the internal logical structure of the software.

CSE6214 Tutorial 12 Answers

By Dr. Mohana

Tutorial – Lecture 12: Software Project Management– Part A

1. Briefly describe the four P's in software project management.
2. List a few examples of project metrics.
3. Explain the use of size-oriented and function-oriented metrics.
4. What is required to produce a good estimation for a software project?
5. What is software scope?

Q1: Briefly describe the four P's in software project management.

- People — the most important element of a successful project
- Product — the software to be built
- Process — the set of framework activities and software engineering tasks to get the job done
- Project — all work required to make the product a reality

Q2: List a few examples of project metrics.

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Q3: Explain the use of size-oriented and function-oriented metrics.

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they go "critical,"
- adjust work flow or tasks,
  - evaluate the project team's ability to control quality of software work products

Q4: What is required to produce a good estimation for a software project?

- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used

Q5: What is software scope?

Software scope describes

- the functions and features that are to be delivered to end-users
- the data that are input and output
- the "content" that is presented to users because of using the software
- the performance, constraints, interfaces, and reliability that bound the system.

Tutorial – Lecture 12: Software Project Management– Part B

a) What is the importance of drawing an (Activity on Arrow) AOA diagram in finding a critical path?

Part B (a)

- By drawing a network diagram, you can figure out the critical path of your project.
- In project management, a critical path is the sequence of project network activities which add up to the longest overall duration.
- This determines the shortest time possible to complete the project.
- Any delay of an activity on the critical path directly impacts the planned project completion date (i.e., there is no float on the critical path).

Part B (a)

It means:

- Path - is a connected sequence of activities leading from the starting event to the ending event
- Critical path - is the longest path (time) which determines the project duration. These are the activities that must be completed in as little time as possible, in order that the duration of the project can be minimized. The critical path can be found by identifying the activities that have no float time. These activities must be closely supervised, since any delay will delay the completion of the project.

Part B (a)

It means:

- Critical activities - refer to all of the activities that make up the critical path. This is an activity which is on the critical path. If this activity is delayed, then the project will not be able to be completed on time.
- Critical path analysis - is the way of showing how a lengthy and complex project (e.g., a building project) can be completed in the shortest possible time. The project is broken down into a number of separate activities, and each activity is then placed in the correct sequence, so to minimize the duration of the project

Part B (b) Search the Internet on the characteristics of an AOA Diagram.

- Construct from left to right
- Any activity can have only one start node
- No two activities can have the same start and end node
- Dummy activities are used in AOA
- Start the network by finding those activities that have no predecessors
- Calculate activity times using probabilistic or deterministic means

Part B (c) Find out the meaning and the symbols that represent Activity, Event and Dummy Activities in AOA diagram.

i. Activity - A task or job which takes time & use up resources

Represented by labeled arrow

ii. Event - An instantaneous point representing the start or finish of an activity

Represented by node

Part B (c) Find out the meaning and the symbols that represent Activity, Event and Dummy Activities in AOA diagram.

iii. Dummy activity - Represented by dotted arrow

- Used when relationships between activities require no work
- Applied when:
  - Two activities start and end similarly
  - Succeeding activities have partial dependencies on predecessor activities

Tutorial – Lecture 12: Software Project Management– Part B

Activity 2 – Draw an AOA diagram based on Table 1

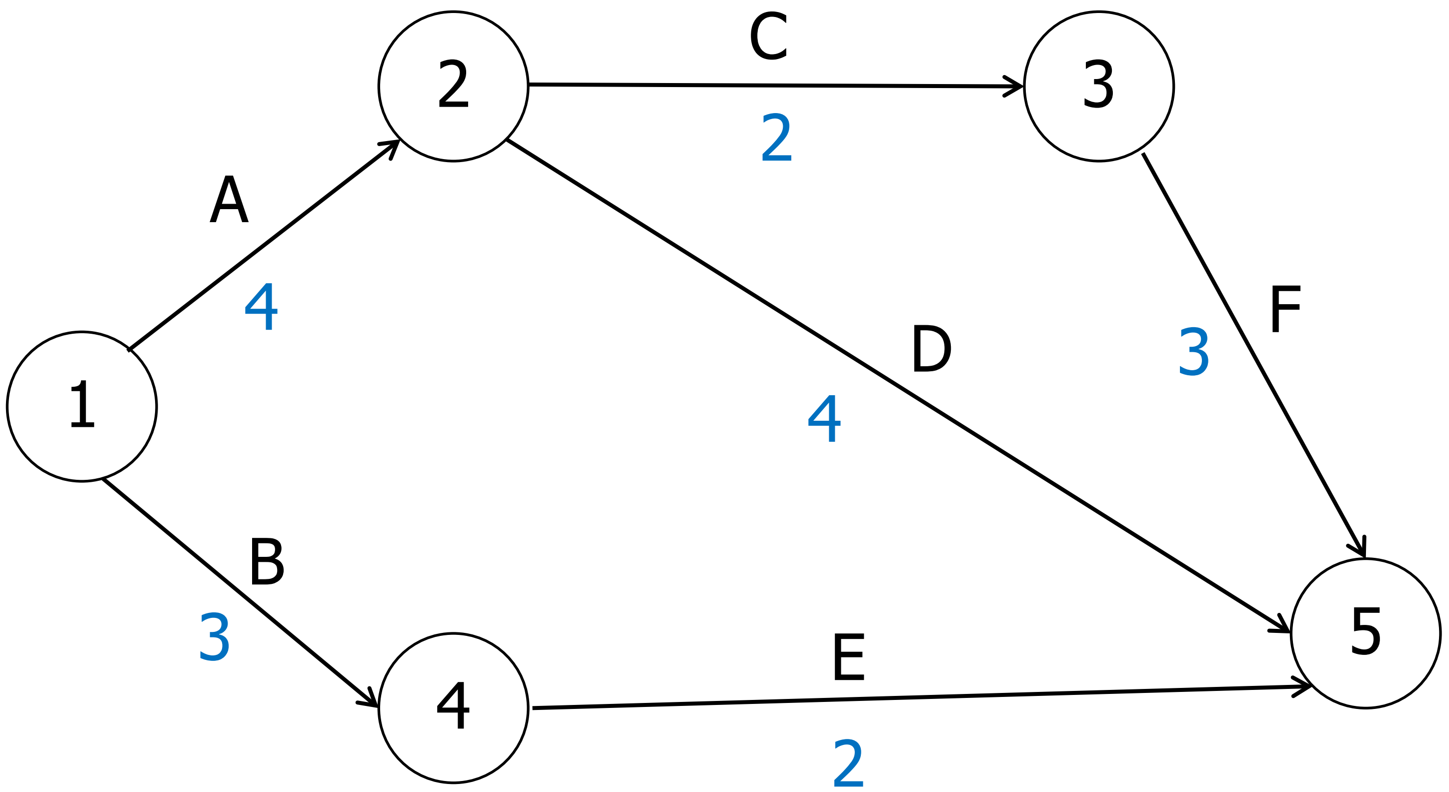
Table 1

Activity	Immediate Predecessor	Duration (month)
A	-	4
B	-	3
C	A	2
D	A	4
E	B	2
F	C	3

Activity 2

Start by drawing the initial node, and the arrows for A and B (no predecessors). Then draw C and D from the end of A, followed by E from the end of B. Next draw F from the end of C. Merge all the end nodes into a single final node.

Activity 2



There are 3 paths: A-C-F, A-D, B-E. A-C-F is the longest path (4+2+3=9) so it is the critical path.

Tutorial – Lecture 12: Software Project Management– Part B

Activity 3 – Draw an AOA Diagram and calculate the longest path using Table 2

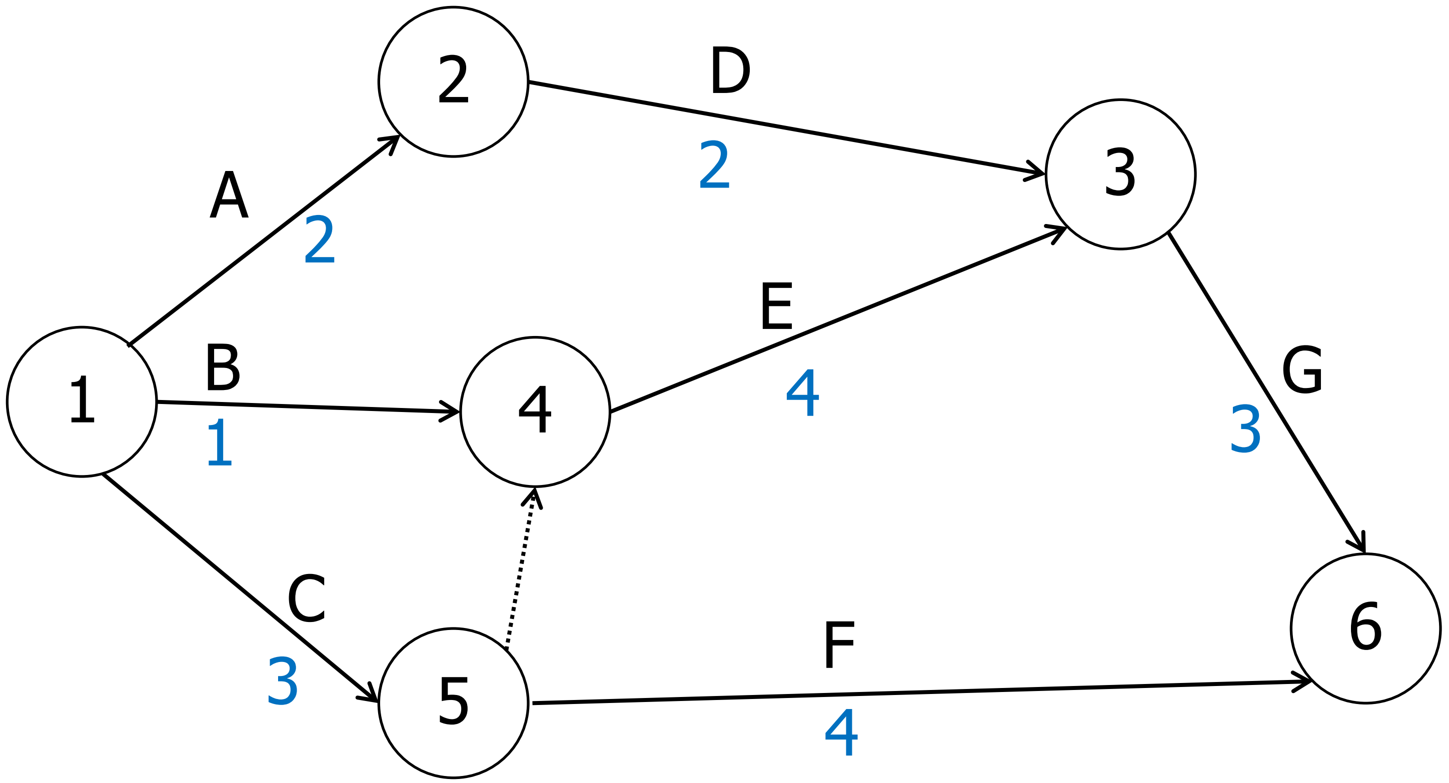
Table 2

Activity	Immediate Predecessor	Duration (week)
A	-	2
B	-	1
C	-	3
D	A	2
E	B, C	4
F	C	4
G	D, E	3

Activity 3

Start by drawing the initial node, and the arrows for A, B and C (no predecessors). Then draw D from the end of A, followed by F from the end of C. For E, because the predecessors are B + C, meaning E has to start from the end of both, a dummy arrow (dotted line) is added from C to B and then draw E from the end of B. Merge the ends of D and E, and draw G from the merged node. Merge all the end nodes into a single final node. See the following diagram.

Activity 3



There are 4 paths: A-D-G, B-E-G. C-(dummy)-E-G, C-F. The longest path is C-(dummy) –E-G (3+4+3=10). This is the critical path since these activities cannot be delayed without impacting the project.

CSE6214 Tutorial 13 Answers

By Dr. Mohana

Tutorial – Lecture 13: Software Maintenance and Control– Part A

1. Describe the main elements of a project that must be monitored and controlled in order for the project to be progressing well.

Solution:

- Performance
- Quality
- Cost
- Time

Q1: Main elements of a project that must be monitored and controlled

Performance

- Refers to how well the project is meeting its defined objectives, scope, and deliverables.
- Monitoring performance ensures that the project outputs align with stakeholder requirements and specifications.
- Example: Tracking whether a software module implements all required features and performs as expected during testing.

Q1: Main elements of a project that must be monitored and controlled

Quality

- Ensures that the project deliverables meet the required standards and are free from defects.
- Quality control includes inspections, reviews, and testing activities to verify conformance to requirements.
- Example: Conducting code reviews and usability testing to maintain software reliability and user satisfaction.

Q1: Main elements of a project that must be monitored and controlled

Cost

- Refers to keeping the project within the approved budget.
- Continuous cost monitoring prevents overspending and helps identify areas where savings can be made.
- Example: Comparing actual expenses with the planned budget in each phase, and adjusting resource allocation to avoid overruns.

Q1: Main elements of a project that must be monitored and controlled

Time

- Ensures that the project activities are completed within the planned schedule.
- Monitoring time involves tracking milestones, deadlines, and dependencies between tasks.
- Example: Using tools such as Gantt charts or project management software to ensure that critical tasks are completed on time and that the project remains on track.

Tutorial – Lecture 13: Software Maintenance and Control– Part A

2. Describe the types of changes that could happen in a software engineering project.

Solution:

- Changes in business requirements
- Changes in technical requirements
- Changes in user requirements

Q2: Types of changes that could happen in a software engineering project

Changes in Business Requirements

- Business goals and priorities may evolve due to market competition, regulatory policies, or organizational strategy.
- Such changes can affect the overall project scope, expected outcomes, or even the target users.
- Example: A company may initially request a desktop-based system but later shift focus to a cloud-based solution to remain competitive.

Q2: Types of changes that could happen in a software engineering project

Changes in Technical Requirements

- Advancements in technology or limitations in existing tools can lead to technical modifications.
- This includes adopting new platforms, frameworks, or tools to improve system performance, scalability, or maintainability.
- Example: A project planned for on-premises deployment may change to adopt a microservices architecture using Docker and Kubernetes.

Q2: Types of changes that could happen in a software engineering project

Changes in User Requirements

- Users often refine their needs after interacting with prototypes, demos, or early versions of the system.
- These changes typically focus on usability, features, or workflow improvements to make the system more effective for end users.
- Example: Users may request additional reporting features or a more intuitive interface after testing the beta version of the software.

Tutorial – Lecture 13: Software Maintenance and Control– Part A

3. List the 3 broad categories of information that make up the software configuration.

Solution:

- Programs
- Documents
- Data

Tutorial – Lecture 13: Software Maintenance and Control– Part A

4. What is a Software Configuration Management (SCM)?

Solution:

SCM is the discipline for systematically controlling the changes that take place during development. Consists of the following activities:

- Identification - What are the configuration items?
- Control - How changes should be controlled?
- Status Accounting - What changes have been made?
- Auditing - Is there conformance to the system?

Tutorial – Lecture 13: Software Maintenance and Control– Part A

5. Explain the importance of version control and change control in ensuring quality of the software product.

Solution:

Version control and change control ensures that there is systematic documentation of the changes to the software. Any quality issues can be referred to the specific version of the software and how changes have been applied to it.

Tutorial – Lecture 13: Software Maintenance and Control– Part A

6. Describe the different types of maintenance activities after a software application has been deployed.

Solution:

- *Corrective Maintenance* - diagnosis and correction of reported errors
- *Adaptive Maintenance* - modifications to properly interface with changing environments $\rightarrow$ new hardware, OS, devices
- *Perfective Maintenance* - implementing new system requirements after system is successful

Tutorial – Lecture 13: Software Maintenance and Control– Part A

7. State a few reasons to perform software reverse engineering and software re-engineering.

Solution:

Software reverse engineering

- There is no documentation or the documentation is incomplete
- The software was updated without updating the documentation
- There are errors in existing documentation

Tutorial – Lecture 13: Software Maintenance and Control– Part A

7. State a few reasons to perform software reverse engineering and software re-engineering.

Solution:

Software re-engineering

- To improve the functionality of existing modules
- To improve the structure of existing architecture
- To add new features and functions to the software
- Existing software could no longer be supported or updated for new requirements