TESTING

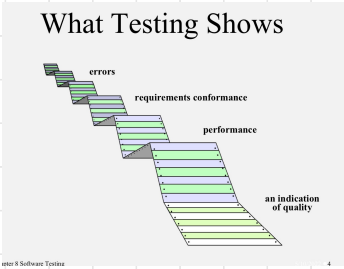
CHP 8

Software Testing

↳ is a process of analyzing software for the purpose of finding bugs

WHY

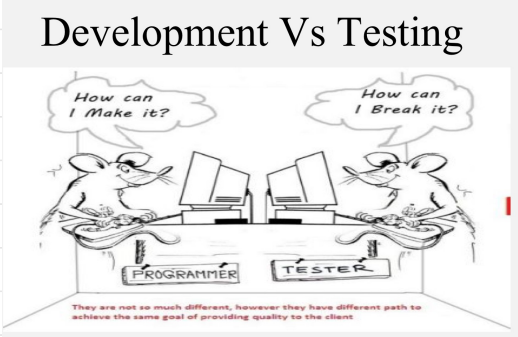
1. To provide stakeholders with info about quality of software under test
2. To detect failures so they can be corrected → Defect detection
3. Help establish that it does not function properly under specific conditions → Reliability estimation



Testing Criteria

↳ execute a program under +ve, -ve conditions by manual and automated testing
↳ should help locate error
↳ should be repeatable
↳ It checks for
  ↳ Specification
  ↳ Performance
  ↳ Functionality

SPF



Types of Testing

1. Manual Testing
2. Automatic Testing
3. Program Testing
   - ↳ Validation Testing
   - ↳ Defect Testing
4. Development Testing
   - ↳ unit testing
   - ↳ component testing
   - ↳ system testing
5. Release Testing
6. User Testing → acceptance testing

Testing Stages

6. Unit Testing
7. Object class Testing
8. Automated Testing
9. Partition Testing
10. Guideline based Testing

Testing Strategies

10. Component Testing
11. Interface Testing
12. System Testing
13. Usecase Testing
14. Sequence Diagram Testing
15. Testing Policies
16. Regression Testing
17. Path Testing
18. Release Testing
19. Requirement Testing
20. Feature Testing
21. Performance Testing
    - ↳ Load Testing
    - ↳ Stress Testing
22. User Testing
    - ↳ Alpha Testing
    - ↳ Beta Testing
    - ↳ Acceptance Testing

When to START Testing

- an early start to testing reduces
  - cost
    - time to rework and produce error free software
- In SDLC, testing can started from Requirement Gathering phase and continued till Deployment of the software
- also depends on development mode being used

When to STOP Testing

- a software can never be 100% tested (as testing is a never ending process)
  - Testing Deadlines
  - Completion of test case execution
  - Completion of function to a certain Point
  - bug rate falls below a certain level and no high priority bugs are identified
  - Management decision

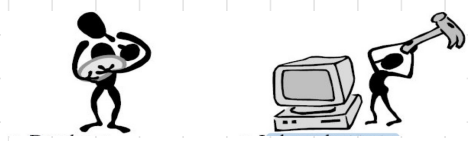
Who should Test

DEVELOPER

- understands the system
- will test gently
- is driven by deadlines } CON

INDEPENDENT TESTER

- must learn system } CON
- will attempt to break system
- is driven by quality



The role of the independent tester is to remove the conflict of interest inherent when the builder is testing his or her own product.

Chapter 8 Software Testing

1. Manual Testing

- testing a software manually
- w/o any automated tool / script
- Tester uses test cases to test a software

2. Automation Testing

- automated manual testing
- using scripts
- used to re-run test senarios that were performed manually

If a manual test costs $X to run the first time, it will cost $X to run every time thereafter.
An automated test can cost 3 to 30 times $X the first time, but will cost about $0 after that.

Chapter 8 Software Testing

Program Testing

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Program Testing Goals

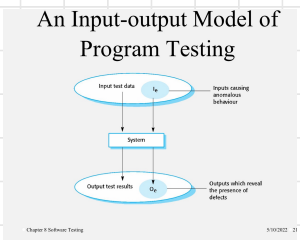
- To demonstrate to the developer and the customer that the software meets its requirements.
  - For custom software, this means that there should be at least one test for every requirement in the requirements document. For generic software products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.
- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - Defect testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

3. Validation Testing → Program Testing Goal 1

↳ expect system to perform correctly
↳ by using a set of test cases
that reflect systems expected use
↳ sucessful test shows developer and
system customer that system operates as intended

4. Defect Testing → Program Testing Goal 2

↳ test cases are designed to expose defects
↳ a sucessful test makes the system
perform incorrectly, and expose defect



V & V Confidence

↳ aims to establish confidence that the system is fit for purpose
↳ depends on
↳ software purpose
↳ how critical software is to an organization
↳ user expectations
↳ users may have low expectations of certain kinds of software
↳ marketing enviorment
↳ getting a product early to the market may be more imp than finding defects

Verification VS Validation

- Verification:
  "Are we building the product right".
- The software should conform to its specification.
- Validation:
  "Are we building the right product".
- The software should do what the user really requires.

Software Inspections → static verification

↳ concerned with analysis of the static system representation to discover problems

- May be supplement by tool-based document and code analysis.

↳ effective technique for discovering program errors

↳ people examine source representation with aim of finding anomalies and defects

↳ doesn't require execution of system so may be used before implementation

PROS

↳ static process

↳ no concern for interaction b/w errors ← as errors can hide other errors

↳ incomplete system can be inspected w/o additional costs

CONS

↳ can't check conformance ← hard with customers real requirements

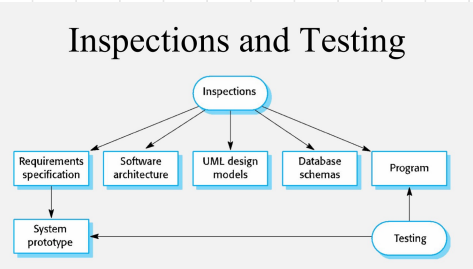
↳ can't check non functional characteristics

e.g. performance, usability

Software Testing → dynamic verification

↳ concerned with exercising and observing product behavior

- The system is executed with test data and its operational behaviour is observed.

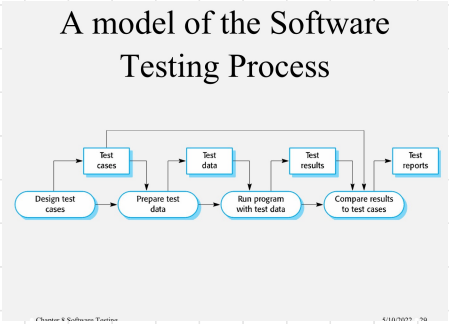


↳ both are complementary and not opposing verification techniques

↳ both should be used during V & V process

Testing Stages

1. Development Testing
   - ↳ system is used during development to discover bugs and defects
2. Release Testing
   - ↳ where a sperate testing team tests a complete version of the system before its released to users
3. User Testing
   - ↳ where users of a system test the system in their own enviorment



4. Development Testing

- ↳ tests all activities that are carried out by the team developing the system
  - ↳ Unit testing
  - ↳ Component Testing
  - ↳ System testing

carried out by the team developing the system.

- Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
- Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
- System testing, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

5. Unit Testing

→ should be automated

- ↳ tests individual components in isolation
- ↳ it is a defect testing process
- ↳ Units may be
  - ↳ individual functions
  - ↳ object classes
  - ↳ composite components

Choosing unit test cases

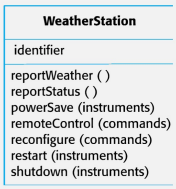
- ↳ unit test cases should
  1. reflect normal operation of a program and show component works as expected
  2. based on testing experiences where common problems arise
     - ↳ use abnormal inputs to check
       - ↳ these are properly processed
       - ↳ does not crash the component

6. Object class Testing

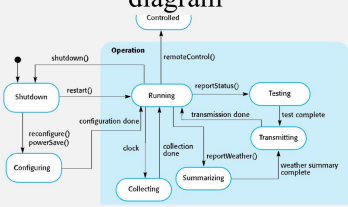
- ↳ tests all operations associated with an object
- ↳ setting and interrogating all obj attributes
- ↳ exercising the obj in all possible states

- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

The Weather Station Object Interface



Weather station state diagram



Weather Station Testing

- Need to define test cases for reportWeather, calibrate, test, startup and shutdown.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions
- For example:
  - Shutdown -> Running-> Shutdown
  - Configuring-> Running-> Testing -> Transmitting -> Running
  - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

7. Automated Testing

↳ use test automation framework (JUnit) to write and run your program tests

components

↳ setup part
  ↳ initialize the system with the test case
↳ call part
  ↳ you call the obj/method to be tested
↳ Assertion part
  ↳ you compare the result of the call with the expected results
  ↳ if true → test successful
  ↳ if false → test failed

- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.

TESTING STRATEGIES

1. Partition Testing

↳ where you identify group of inputs that have common characteristics and should be processed in the same way

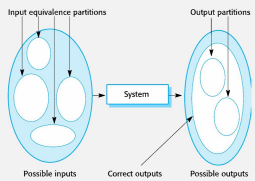
↳ test cases should be chosen from each partition

2. Guideline based Testing

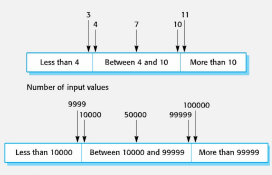
↳ where you use testing guidelines to choose test cases

- These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Equivalence Partitioning



Equivalence Partitions



Testing Guidelines (sequences)

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Test with sequences of zero length.

General Guidelines Based Testing

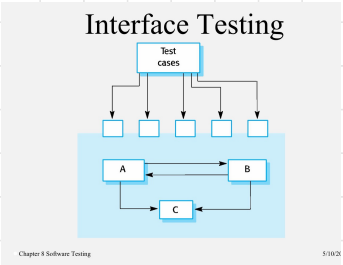
- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

11. Interface Testing

↳ detect faults due to interface errors

↳ INTERFACE TYPE

- Parameter interfaces Data passed from one method or procedure to another.
- Shared memory interfaces Block of memory is shared between procedures or functions.
- Procedural interfaces Sub-system encapsulates a set of procedures to be called by other sub-systems. Objects and reusable components have this form of interface.
- Message passing interfaces Sub-systems request services from other sub-systems. A return message includes the results of executing the service. Some object-oriented systems have this form of interface, as do client–server systems.



↳ INTERFACE ERRORS

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect. For example, a binary search method may be called with a parameter that is an unordered array.
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed.

↳ INTERFACE TESTING GUIDELINES

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

10. Component Testing

- Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

12. System Testing

- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behaviour of a system.

- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

System & Component Testing

EMERGENT BEHAVIOR → often all the system is compiled and tested then unwanted behaviors are discovered

↳ performance

↳ synchronization

↳ security

↳ scalability

PSSS

13. Use Case Testing

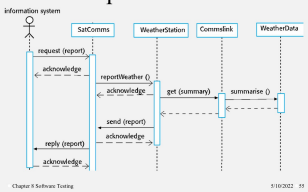
↳ can be used as basis for system testing

↳ tests each use case

↳ as use case involves several system components

↳ so testing them forces these interactions

Collect Weather Data Sequence Chart



Test cases Derived from Sequence Diagram

- An input of a request for a report should have an associated acknowledgement. A report should ultimately be returned from the request.
  - You should create summarized data that can be used to check that the report is correctly organized.
- An input request for a report to WeatherStation results in a summarized report being generated.
  - Can be tested by creating raw data corresponding to the summary that you have prepared for the test of SatComms and checking that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object.

Testing Policies

- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

↳ Kis cheez ki testing

↳ There are some limitations

↳ evaluation

Test Driven Development (TDD)

↳ testing and code development is interleaved

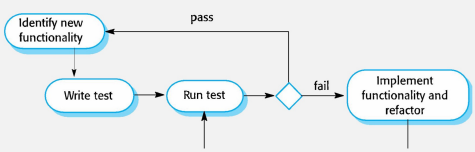
↳ tests are written before code

↳ Passing the tests is the critical driver of development

↳ code and tests are developed incrementally

↳ don't move on until code passes the test

Test-driven Development



Pros

↳ Code Coverage

↳ all written code has atleast 1 test

↳ Regression Testing

↳ a regression test suite is developed incrementally

↳ Simplified Debugging

↳ when test fails, it obvious where problem lies

↳ System documentation

↳ test themselves are a form of documentation that describes what the code should be doing

TDD Process Activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.

Test Scenario ID	Login-1		Test Case ID	Login-1B	
Test Case Description	Login – Negative test case		Test Priority	High	
Pre-Requisite	NA		Post-Requisite	NA	
Test Execution Steps:					

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	Launch successful
2	Enter invalid Email & any Password and hit login button	Email id : invalid@xyz.com Password: ******	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	IE-11	Pass	Invalid login attempt stopped
3	Enter valid Email & incorrect Password and hit login button	Email id : valid@xyz.com Password: ******	The password that you've entered is incorrect. Forgotten password?	The password that you've entered is incorrect. Forgotten password ?	IE-11	Pass	Invalid login attempt stopped

Project Name	Bank Webiste Functionality							
Module Name	Login Functionality							
Created By								
Created Date								
Executed By								
Executed Date								

Test Case ID	Test Case Description	Pre Steps	Test Step	Preconditions	Test Data	Expected Result	Actual Result	Status	Comments
Test the Login Functionality in Banking	Verify login functionality with valid username & password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Valid password	password: XXXXXXX@1	Credential can be entered	As expected	Pass	
			Click on login button			User logged	User logged successfully	Pass	
Test the Login Functionality in Banking	Verify login functionality with valid username & invalid password		Navigate to login page			Able to see the login page	As expected	Pass	
			Enter valid username	Valid Username	username: choudaryac97@gmail.com	Credential can be entered	As expected	Pass	
			Enter valid password	Invalid password	password: XXXXXXX@2	Credential can be entered	As expected	Pass	
			Click on login button			User logged	Unsuccessful login	Fail	

TC1.0 - Create Contact			
Project Name	Project ABC	Date	3/3/2004
Test Case Description	The purpose of this Test Case is to test the creation of a new contact. Login --> Contacts --> New Contact --> Logout	Requirements	3, 54, 123, 45
Function / Module Under Test	Contact	Test Type	Manual
Written by			
Goals	To create a contact in CMS as a basis for building relationship, opportunity, and account information in the system		
Test Setup	N/A		
Dependencies	Appropriate username, password and role		

ID	Process	Detailed Step	Expected Results	User	Pass/Fail (Criteria)	Data Input	Siebel Reference
1	Login	Type User ID into the User Name field	Field is populated			User Name	Login
2		Type Password into the Password field	Field is populated			Password	
3		Click the Login Button	Siebel Launches				
4	Navigate_Contact	Click on Contacts tab	Contacts - Rolodex View opens	What type of user or username goes here	View opens without error.		Contacts
5	Search_Contact_By_Name	Click on Search view tab	Contacts-Search view opens				
6		Select "Name" search method in Search By field	Field is populated, appropriate form controls display on applet				
7		Type [First Name] in First field	Field is populated			First Name	
8		Type [Last Name] in Last field	Field is populated			Last Name	
9	Create_Contact	Click the New button	New record displays in Contacts list applet				

REQUIREMENTS TRACEABILITY MATRIX

	Test Case ID	TC001	TC002	TC003	TC004	TC005	Total Test Case ID
Requirement ID							
REQ 1		X			X		2
REQ 2		X		X			2
REQ 3		X	X		X	X	4
REQ 4					X		1
REQ 5			X	X	X		3
REQ 6			X			X	2

14. Regression Testing

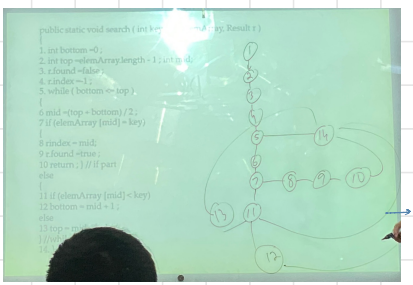
- ↳ checks that changes have not broken previously working code
- ↳ in manual testing, regression testing is expensive
- ↳ all tests are rerun everytime a change is made
- ↳ tests must run sucesshully before the change is commited

15. Path Testing

↳ ensure test cases are such that each path through the program is executed atleast once

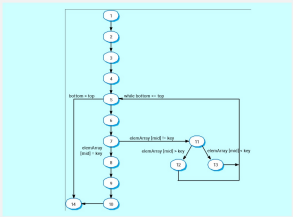
steps

1. Draw a control Graph
2. Find a basis set of paths
3. Generate test cases to exercise each path



→ this one wrong

Binary Search Flow Graph



Independent Paths

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 5, 6, 7, 11, 13, 5, …
- Test cases should be derived so that all of these paths are executed
- A dynamic program analyser may be used to check that paths have been executed

16. Release Testing

↳ tests a particular release of a system that is intended for use outside of development team

↳ a black box testing process where tests are derived from system specification

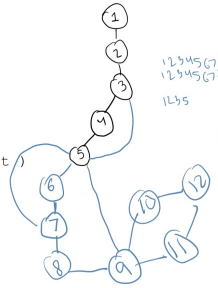
↳ It's a form of system testing

- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.

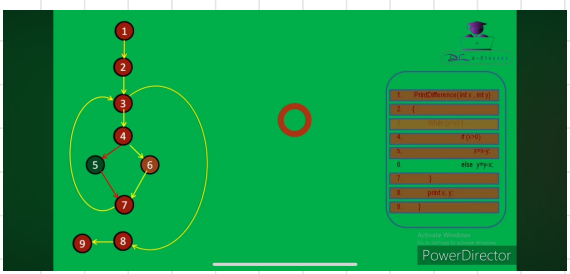
d) Perform path testing on following code snippet. [5 points]

```
1 PROGRAM maxsum ( maxint, value : INT )
2 INT result := 0 ; i := 0 ;
3 IF value < 0
4 THEN value := - value ;
5 WHILE ( i < value ) AND ( result <= maxint )
6 DO i := i + 1 ;
7 result := result + i ;
8 OD;
9 IF result <= maxint
10 THEN OUTPUT ( result )
11 ELSE OUTPUT ( "too large" )
12 END.
```



Cyclomatic complexiM: E - N + 2xP

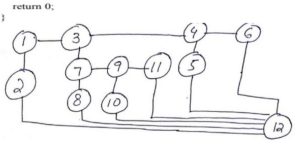
E: edges, N: nodes, P: no. of end points



Q1. a. For the given piece of code do the following tasks:

- Draw a control flow graph after labeling the code.
- Calculate Cyclomatic complexity
- Determine all the independent paths
- Generate test cases for any two independent paths (Assume inputs for testcases)

```
#include <iostream>
using namespace std;

int main() {
    int num_wheels, max_speed;
    string engine_type;

    // Get input for num_wheels, max_speed, and engine_type

    if (num_wheels == 2 && engine_type == "gasoline") {
        cout << "It's a motorcycle" << endl;
    } else if (num_wheels == 4 && engine_type == "gasoline") {
        if (max_speed > 200) {
            cout << "It's a sports car" << endl;
        } else {
            cout << "It's a sedan" << endl;
        }
    } else if (num_wheels == 4 && engine_type == "electric") {
        cout << "It's an electric car" << endl;
    } else if (num_wheels > 4 && engine_type == "gasoline") {
        cout << "It's a truck" << endl;
    } else {
        cout << "Not a recognized vehicle" << endl;
    }

    cout << "End of program" << endl;

    return 0;
}
```



(b) CC = E - N + 2
= 16 - 12 + 2
= 6

(c) 3 independent paths.

1. 1 - 2 - 12
2. 1 - 3 - 7 - 8 - 12
3. 1 - 3 - 7 - 9 - 10 - 12
4. 1 - 3 - 7 - 9 - 11 - 12
5. 1 - 3 - 4 - 5 - 12
6. 1 - 3 - 4 - 6 - 12

Inputs			Expected output
Num_wheels	Engine_type	Max_speed	
2	Gasoline	100	motorcycle
4	Gasoline	250	Sportscar
4	electric	250	It's an electric car
6	electric	250	Not recognized vehicle

Q1. a. For the given piece of code shown in figure 1., do the following tasks:

i. Draw a control flow graph after labeling the code.

ii. Calculate Cyclomatic complexity

iii. Determine all the independent paths

iv. Generate test cases for any two independent paths (Assume inputs for testcases)

```
import java.util.Arrays;
class SelectionSort {
    public static void main(String args[]) {
        int[] array = {20, 12, 10, 15, 2};
        for (int step = 0; step < array.length - 1; step++) {
            int min_idx = step;
            for (int i = step + 1; i < array.length; i++) {
                // Select the minimum element in each loop.
                if (array[i] < array[min_idx]) {
                    min_idx = i;
                }
            }
            // put min at the correct position
            int temp = array[step];
            array[step] = array[min_idx];
            array[min_idx] = temp;
        }
        System.out.println("Sorted Array in Ascending Order: ");
        System.out.println(Arrays.toString(array));
    }
}
```

17. Requirements Based Testing

↳ examines each requirement and and tests for it

- Mentcare system requirements:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Requirements Tests

- Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
- Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
- Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

A Usage Scenario for the Mentcare System

George is a nurse who specializes in mental healthcare. One of his responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, George logs into the Mentcare system and uses it to print his schedule of home visits for that day, along with summary information about the patients to be visited. He requests that the records for these patients be downloaded to his laptop. He is prompted for his key phrase to encrypt the records on the laptop.

One of the patients that he visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. George looks up Jim's record and is prompted for his key phrase to decrypt the record. He checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so he notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. Jim agrees so George enters a prompt to call him when he gets back to the clinic to make an appointment with a physician. George ends the consultation and the system re-encrypts Jim's record.

After, finishing his consultations, George returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for George of those patients who He has to contact for follow-up information and make clinic appointments.

Features Tested by Scenario

- Authentication by logging on to the system.
- Downloading and uploading of specified patient records to a laptop.
- Home visit scheduling.
- Encryption and decryption of patient records on a mobile device.
- Record retrieval and modification.
- Links with the drugs database that maintains side-effect information.
- The system for call prompting.

18. Performance Testing

↳ tests should reflect the profile of use of the system

load Testing

↳ plan a series of tests where load is steadily increased until system performance becomes unacceptable

↳ identifies
- ↳ max capacity of software
- ↳ software behavior at peak time

stress Testing

↳ system is deliberately overloaded to test its failure behavior

↳ identifies breaking point

- This testing can be performed by testing different scenarios such as:
  - Shutdown or restart of network ports randomly
  - Turning the database on or off
  - Running different processes that consume resources such as CPU, memory, server, etc.

User Testing

↳ users provide input on system testing

- The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

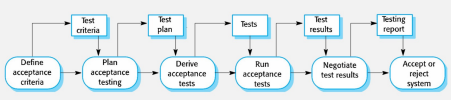
TYPES OF USER TESTING

↳ Alpha Testing
  ↳ users work with development team to test software at developers site

↳ Beta Testing
  ↳ release of software is made aul to users to experiment and tell problems to developers

↳ Acceptance testing
  ↳ customers test a system to decide wether or not its ready to be accepted from the system developers and deployed in customers enviroment

The Acceptance Testing Process



Stages in the Acceptance Testing Process

- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

Agile methods and acceptance testing

- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is 'typical' and can represent the interests of all system stakeholders.

Quality Management

CHP 24

Software Quality Management

↳ ensures required level of quality is achieved

↳ there are 3 principal concerns

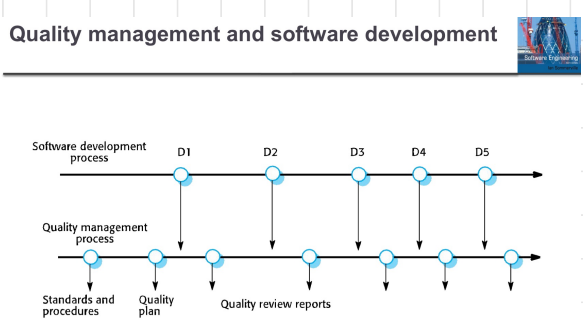
1. establishing a framework of organizational processes and standards that lead to high quality software
   ↳ a concern at the organizational level
2. application of specific quality processes and checking if these planned processes have been followed
   ↳ a concern at the project level
3. establishing a quality plan for a project
   The quality plan should
   ↳ set out quality goals
   ↳ define what processes and standards to be used
   ↳ a concern at the project level

Quality Management Activities

↳ provides an independent check on the software development process

↳ It checks the project deliverables to ensure they are consistent with organizational standards and goals

↳ quality team should be independent from the development team so they can report quality issues w/o biasnes



Quality Planning

- It sets out desired product qualities
- how these are assessed
- defines most significant quality attributes

Quality Plans

- It should define quality assessment process
  - which organisational standards should be applied
  - where necessary
  - define new standards to be used
- They should be short, succint docs → if too long no one will read them

Outline for quality Plans

- Product Introduction
  - description of the product
  - its intended market
  - quality expectations for the product
- Product Plans
  - the critical release dates and responsibilities
  - Plans for distribution and Product servicing
- Process descriptions
  - development and service processes to be used
  - Standards to be used for Product development and management
- Quality goals
  - quality goals and Plans for the produt
  - including identification and justification of critical product quality attributes
- Risks and risk management
  - key risks that might affect Product quality
  - actions to be taken to address these risks

scope of quality management

↳ It is imp for large complex systems

↳ for smaller systems

↳ quality management needs less documentation

↳ should focus on establishing a quality culture

quality documentation

↳ is a record of progress

↳ It supports continuity of development as the development team changes

Software Quality

Non functional characteristics

↳ defines subjective quality of a software system

↳ It reflects practical user experience

- This reflects practical user experience – if the software's functionality is not what is expected, then users will often just work around this and find other ways to do what they want to do.

↳ If the software is unreliable or too slow then it is impossible to achive their goals

Software fitness for purpose

- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?
- Have programming and documentation standards been followed in the development process?

Quality conflicts

↳ Its impossible for a system to be optimized for all these attributes

e.g increase robustness ⟶ loss of performance

↳ Hence quality plan should define

↳ most imp quality attributes for the software

↳ include definition of quality assessment process

↳ an agreed way of assessing some quality

e.g maintainibility/ robustness is present in product

Software quality attributes

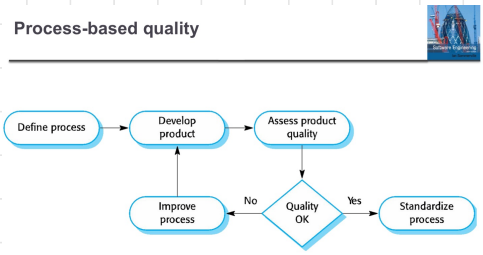
Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Process and Product Quality

↳ Product quality is influenced by process quality

↳ It is imp as product quality attributes are hard to assess

- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.



SOFTWARE STANDARDS

Software Standards

↳ defines required attributes of a product/process

↳ standards may be
  ↳ international
  ↳ national
  ↳ organizational/project standards

Importance of Standards

↳ Encapsulation of best practices
  ↳ avoids repetion of past mistakes

↳ Provide continuity
  ↳ new staff can understand the organisation

↳ Provides a framework defining what quality means
  ↳ organizations view of quality

Product Standards

- applied to software product being developed
- They include
  - document standards — e.g. structure of req doc
  - documentation standards — e.g. standard comment header
  - coding standards — defines how a programming lang should be used

Process Standards

- defines process that should be followed — during software development
- They include
  - definitions of specifications
  - design and validation processes
  - process support tools
  - description of documents — that should be written during these processes

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Standards Cons

- aren't seen as up-to-date by Software Engineers
- often involve too much bureaucratic (government) form filling
- if they are supported by software tools
  - tedious form filling work is involved

Standards Development

- involves practitioners in development
  - engineers should understand the rationale underlying standard
- review standards and their usage regularly
  - as standards can quickly become outdated
- detailed standards should have specialized tool support
  - excessive clerical work is the most significant complaint against standards
  - web based form are not good enough

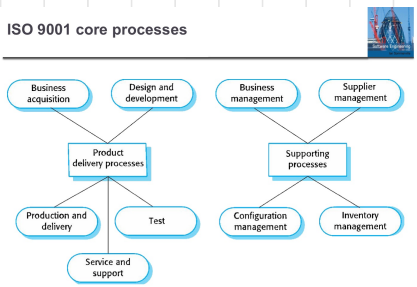
ISO 9001 standards framework

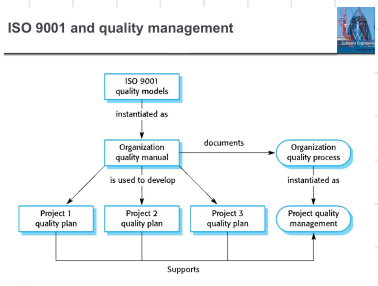
↳ an international set of standards

↳ a framework for developing software standards

↳ applies to organizations that design, develop, maintain products

- It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.





ISO 9100 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognised.

ISO 9100 software quality

↳ inadequate certification as it defines quality to be conformance to standards

↳ it takes no account of quality

the software. For example, a company could define test coverage standards specifying that all methods in objects must be called at least once.

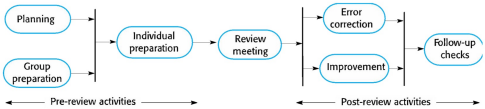
↳ unfortunately, this standard can be met by incomplete software testing that doesn't include tests with diff method parameters

method parameters. So long as the defined testing procedures are followed and test records maintained, the company could be ISO 9001 certified.

Phases in the review process

- Pre-review activities
  - review planning and preparation
- The review meeting
  - the author of the program/doc being reviewed 'walks through' the doc with the review team
- Post review activities
  - addresses the problems and issues raised during review meeting

The software review process



Quality Management in Agile Development

- It is informal → not doc based
- establishes a quality culture
  - where all team members feel responsible for software quality and take actions to ensure quality is maintained

✧ The agile community is fundamentally opposed to what it sees as the bureaucratic overheads of standards-based approaches and quality processes as embodied in ISO 9001.

Issues with Agile QM and large systems

✧ When a large system is being developed for an external customer, agile approaches to quality management with minimal documentation may be impractical.

- If the customer is a large company, it may have its own quality management processes and may expect the software development company to report on progress in a way that is compatible with them.
- Where there are several geographically distributed teams involved in development, perhaps from different companies, then informal communications may be impractical.
- For long-lifetime systems, the team involved in development will change. Without documentation, new team members may find it impossible to understand development.

Shared good Practice

↳ check before check-in

- Programmers are responsible for organizing their own code reviews with other team members before the code is checked in to the build system.

↳ Never break the build

- Team members should not check in code that causes the system to fail. Developers have to test their code changes against the whole system and be confident that these work as expected.

↳ Fix Problems when you see them

- If a programmer discovers problems or obscurities in code developed by someone else, they can fix these directly rather than referring them back to the original developer.

Reviews in agile methods

↳ It is informal → not doc based

↳ In Scrum

↳ there is a review meeting after each iteration → sprint review

where quality issues and problems are discussed

↳ In Extreme Programming

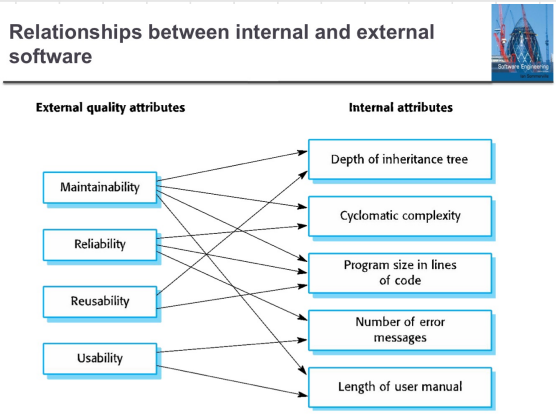
↳ pair programming ensures that code is

constantly being examined and reviewed by another team member

SOFTWARE MEASUREMENT

Types of process metric

- Process completion time
  - total time devoted to process
- Resources required for a process
  - travel costs
  - computer resources
  - total effort in person-days
- NO. of occurences of an event
  - no. of defects discovered
  - no. of req changes requested
  - no. of bugs reports
  - avg no. of lines of code modified



Problems with measurement

- impossible to quantify ROI of introducing an organizational metrics program
- there are no standards for measurement and analysis
- software processes are not standardized in many companies
  - are poorly defined and controlled
- less work on software measurement focused on configure ERP systems or COTS →
- adds additional overhead to processes

✧ Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.

Emperical software engineering

- a research area with experiments on
  - software systems
  - collection of data about real projects

to form and validate hypotheses about software Engineering methods and techniques

✧ Research on empirical software engineering, this has not had a significant impact on software engineering practice.

✧ It is difficult to relate generic research to a project that is different from the research study.

Measurement Surprises

Reducing the no of faults in a program leads to an increased no of help desk calls

- The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
- A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Key points

- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability etc. Software standards are important for quality assurance as they represent an identification of 'best practice'. When developing software, standards provide a solid foundation for building good quality software.
- Reviews of the software process deliverables involve a team of people who check that quality standards are being followed. Reviews are the most widely used technique for assessing quality.

Key points

- In a program inspection or peer review, a small team systematically checks the code. They read the code in detail and look for possible errors and omissions. The problems detected are discussed at a code review meeting.
- Agile quality management relies on establishing a quality culture where the development team works together to improve software quality.
- Software measurement can be used to gather quantitative data about software and the software process.

LINES OF CODE (LOC) AND FUNCTIONAL POINTS (FP)

LOC and FP are used in 2 ways

1. as estimation variables to size each element of the software → S
   ↳ during software project estimation
2. as baseline metrics collected from past projects
   ↳ used in conjunction with estimation variables
   to develop cost and effort projections

expected value for estimation variable

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

estimated cost: total estimated LOC × cost per line of code → ?

estimated effort: $\frac{\text{estimated cost}}{\text{labour rate}}$ → ?

effort: $\frac{\text{total LOC}}{\text{LOC/pm}}$

cost = effort × labour rate

Example of LOC

- The mechanical CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database.
- Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

9)

- A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic, 4600 LOC; most likely, 6900 LOC; and pessimistic, 8600 LOC. Applying Equation 26.1, the expected value for the 3D geometric analysis function is 6800 LOC.

S_{opt} = 4600 LOC
S_m = 6900 LOC
S_{pt} = 8600 LOC

$$S = \frac{4600 + 4(6900) + 8600}{6}$$

S = 6800 LOC

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm. Based on a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,600 (or 432,000 by rounding up) and the estimated effort is 54 person-months

Labour rate = $8000 per month
cost per line of code = $ 13

avg LOC = 620 LOC / per month ?

estimated cost = 33,200 × 13 = 431600
estimated effort = $\frac{431600}{8000}$ = 54 person per month

$$E = \frac{33200}{600} = 53.54$$

$$E = \frac{cost}{LR} \rightarrow cost = E \times LR$$

Q) estimated LOC
A = 5000 C = 10000
B = 3000 D = 2000

velocity = 500 LOC/pm
Labour rate = 10000

Date: / /2024 SE Quiz#4 Student Id/ Section:

Instructions:
- Duration: 6 mins
- Questions should be answered in order. Out of order answers will not be graded.

1. Write and explain two (out of the five) attributes that should be encompassed by effective software metrics.
2. You are about to design a new software for your company. Assume that an estimate of 5000 LOC is established for component A, 3000 for component B, 10000 for component C, and 2000 for component D of the software. Further assume that the historical velocity for previous software of this type is 500 LOC/pm in your organization. The burdened labour rate is Rs. 1000. What would be the estimated effort and cost of the software?

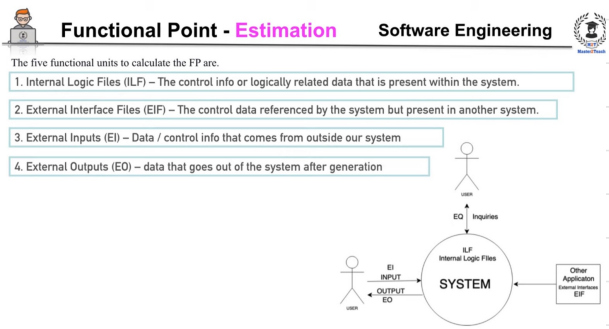
Write answers below and overleaf:

Total LOC = 5000 + 3000 + 10 000 + 2000 = 20 000
Effort = $\frac{20000}{500}$ = 40

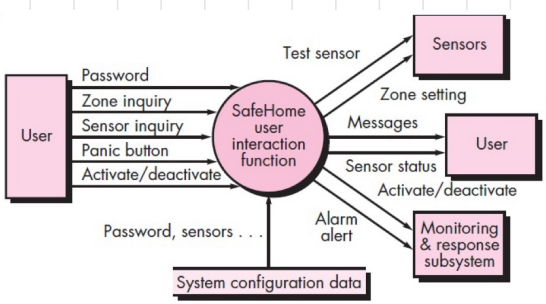
Cost = 40 × 10,000 = 400,000

Functional Point Metric (FP) → effective Software metrics

1. No. of external inputs (EIs)
   - ↳ originates from user
   - e.g user input, password, update ILF (write, delete, update)
2. NO. of external outputs (EOs)
   - ↳ derived data within the application
   - e.g reports, error msses, anything on screens
3. No. of external inquiries (EIs)
   - ↳ an online input that results in immediate software response
   - e.g prompt interrupt
4. No. of internal logical files (ILFs)
   - ↳ files
   - e.g database, dictionary
5. NO. of external interface files (EIFs)
   - e.g shared database



A data flow model for *SafeHome* software



The data flow diagram is evaluated to determine a set of key information domain measures required for computation of the function point metric.

Three external inputs—password, panic button, and activate/deactivate.
Two external inquiries—zone inquiry and sensor inquiry.
One ILF (system configuration file)
Two external outputs (messages and sensor status)
Four EIFs (test sensor, zone setting, activate/deactivate, and alarm alert)

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		count × simple
External Inputs (EIs)	3	×	3	4	6	=	9
External Outputs (EOs)	2	×	4	5	7	=	8
External Inquiries (EQs)	2	×	3	4	6	=	6
Internal Logical Files (ILFs)	1	×	7	10	15	=	7
External Interface Files (EIFs)	4	×	5	7	10	=	20
Count total							50

The count total shown in Figure 23.3 must be adjusted using Equation (23.1). For the purposes of this example, we assume that $\Sigma(F_i)$ is 46 (a moderately complex product). Therefore,

$$FP = 50 \times [0.65 + (0.01 \times 46)] = 56$$

Based on the projected FP value derived from the requirements model, the project team can estimate the overall implemented size of the *SafeHome* user interaction

Value adjustment factors are used to provide an indication of problem complexity.

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

0 (not important or applicable) to 5 (absolutely essential)

value based on 14 responses

$$FP = (\text{count total}) \left(0.65 + (0.01 \times \Sigma F_i)\right)$$

given in question

$$FP = 50 \left(0.65 + (0.01 \times 46)\right)$$
$$= 56$$

- Assume that past data indicates that one FP translates into 60 lines of code (an object-oriented language is to be used) and that 12 FPs are produced for each person-month of effort. These historical data provide the project manager with important planning information that is based on the requirements model rather than preliminary estimates. Assume further that past projects have found an average of three errors per function point during requirements and design reviews and four errors per function point during unit and integration testing. These data can ultimately help you assess the completeness of your review and testing activities.

$\frac{20 + 4(24) + 30}{6}$

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320 → UFP

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5

52 → ΣF_i

FP = 375

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8000 per month, the cost per FP is approximately $1230.
Based on the FP estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

1 FP = 60 LOC
12 FP produced for each person pm

complexity adjustment factor

depend on 14 question

$$CAF = 0.65 + (0.01 \times \Sigma f_i)$$

Functional Points

unadjusted FP

$$FP = UFP \times CAF$$

$$FP = 320 \left(0.65 + (0.01 \times 52)\right)$$
$$= 375$$

6.5 FP/pm, LR = 8000, cost per FP = 1230

estimated cost = 375 × 1230 = 461000

estimated effort = $\frac{461000}{8000} = 57.6 \approx 58$ person months

ishma hafeez
notes

LOC BASED ESTIMATION EXAMPLE

- As per historic data, it is deduced that:
  - Organization can write lines of code = LOC = 620 LOC/person
  - Payment to a personnel = Cost= $ 8000/month
  - Cost/LOC = $\frac{8000}{620}$ = 12.9 ≈ $ 13
  - Effort required for the project= $\frac{\sum \text{Estimated LOC for each function}}{LOC}$
    $= \frac{5300+6800+2300+3350+8400+4950+2100}{620}$
    $= \frac{33200}{620} \approx 53$ persons
  - Project cost to be paid = $\sum_{\text{Estimated LOC for each function}}$ * Cost/LOC
    = (5300+6800+2300+3350+8400+4950+2100) * 13
    = $ 4,31,600

A "person month" is the metric for expressing effort (amount of time) personnel devote to a spec project. Approximately equals to 140 hours

FP = 50 * 0.93 = 46.5

FP BASED ESTIMATION

- If lets say team productivity is 60FP/person-week, then

  Effort = FP/productivity
  = FP/60
- If team size =5, then

  Project duration = Effort/ team size
  = Effort/5 weeks

c) Suppose the requirement specification for the E-commerce Website Development has been carefully analyzed and the following estimates have been obtained. There is a need for 11 inputs, 11 outputs, 7 inquiries, 22 files, and 6 external interfaces. Also, assume outputs, queries, files function point attributes are of low complexity and all other function points attributes are of medium complexity.

Calculate Unadjusted Function Point by using the predefined weights for each function point in each category. (Redraw the whole table on your answer sheet). [10points]

Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	11	*	3	4	6	=
2. Number of external Output (EO)	11	*	4	5	7	=
3. Number of external Inquiries (EQ)	7	*	3	4	6	=
4. Number of internal Files (ILF)	22	*	7	10	15	=
5. Number of external interfaces(EIF)	6	*	5	7	10	=
Count-total →						

→ UFP

Question 1: [pt-50,CLO-4]

a. Given the following values:

External inputs (EI):
- 3 simple
- 4 average
- 1 Complex

External Outputs (EO):
- 6 Average
- 2 Complex

External Enquiries (EE):
- 8 Average

Internal Logical Files (ILF):
- 5 Complex

External Interface Files (EIF):
- 3 Average
- 4 Complex

	Simple	Average	Complex	UFP
External Inputs	2	4	6	28
External Outputs	3	5	7	44
External Enquiries	2	4	6	32
Internal Logical Files	5	10	15	80
External Interface Files	4	7	10	61
				245

Table 1

Questions	Scale
1. Are the inputs, outputs, files or inquiries are complex?	Average (3)
2. Is the code design to be reused?	Moderate (2)
3. Is performance critical?	Significant (4)
4. Is the system designed for multiple installations in different organizations?	Average (3)
5. Are there distributed processing functions?	Essential (5)

Table 2

Now do the following tasks:
- Refer to table 1, calculate Unadjusted Functional Point (UFP). [pt-5]
- Refer to table2, Calculate Complexity Adjustment Factor (CAF) for the questions based on the given scale for each question. [pt-5]
- Calculate functional points for the given values. [pt-2]
- What is the impact of the value of functional points on software size & its associated cost? Give a brief reason for your answer. [pt-3]

b. As the project manager of Softneck software house, you are tasked with estimating the total effort and cost for the upcoming Alpha Project. The Alpha Project team consists of six members, each charging $800 per person-month (pm). Based on previous similar projects, the average productivity is determined to be 620 lines of code per person-month (LOC/pm). For the Alpha Project, the estimated line of code is 33,200 LOC. With this information, calculate the total effort and cost required for the Alpha Project. [pt-10]

c. The estimates provided by the five estimators in the second round of the wide-band Delphi technique are referred in table 3:

Estimators	Pessimistic Effort (P)	Optimistic Effort (O)	Expected Effort (E)
Kainat Memon	99	80	92
Faizan Sheikh	80	50	70
Farah Shah	91	72	82
Rafy Meon	85	60	72
Ghulam Mujtaba	87	62	72

Table 3

Based on the estimation given in the above table, answer the following questions:

i. What is the weighted estimate for each estimator? [pt-2]
ii. What is the average joint estimate? [pt-2]
iii. What is the 95% confidence interval for the estimate? [pt-1]

EI: $3\times2 + 4\times4 + 6\times1 = 28$

EO: $6\times5 + 2\times7 = 44$

EE: $8\times4 = 32$

ILF: $5\times5 = 80$

EIF: $3\times7 + 4\times10 = 61$

$245 \rightarrow$ UFP

i) UFP = 245

ii) V = 3+2+4+3+5 = 17

CAF = 0.65 + (0.01 × 17) = 0.82

iii) FP = UFP × CAF

= 245 × 0.82

= 200.9

b) Labour = $800 pm

avg productivity = 620 LOC/pm

LOC = 33200 LOC

$$\text{effort} = \frac{\text{Total LOC}}{\text{Prev}} = \frac{33,200}{620\ \text{LOC}} = 53.5$$

$$\text{cost} = 53.5 \times 800 = 42800$$

c): expected value for estimation variable

$$S = \frac{S_{Opt} + 4S_m + S_{Pess}}{6}$$

$$= \frac{80 + 4(92) + 99}{6}$$

= 91.6 for Kainat Memon

⋮

Repeat for all

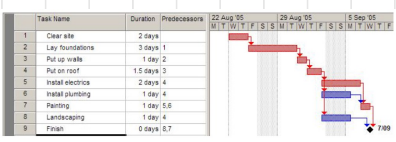
GANT & PERT CHARTS

↳ They are born CPM (critical Path method) tools to
  ↳ manage tasks in big and complex projects
  ↳ let PM (Project managers) organise
    ↳ time
    ↳ People
    ↳ equipment
    ↳ money
  ↳ allow PM to monitor progress of a project

Gaant Chart (name)

↳ a timeline with tasks
that can be connected to eachother

NOTE SPELLINGS → not all caps!

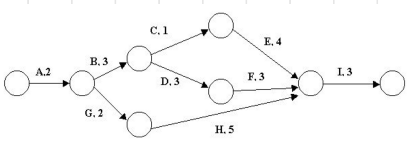


	Task Name	Duration	Predecessors
1	Clear site	2 days	
2	Lay foundations	3 days	1
3	Put up walls	1 day	2
4	Put on roof	1.5 days	3
5	Install electrics	2 days	4
6	Install plumbing	1 day	4
7	Painting	1 day	5,6
8	Landscaping	1 day	4
9	Finish	0 days	8,7

22 Aug '05 | 29 Aug '05 | 5 Sep '05

PERT Chart (acronym in all CAPS)

↳ In Gaant Chart, the length of a tasks bar
is proportional to the length of the task
This RARELY applies to PERT



Early Start	Duration	Early Finish
Task Name		
Late Start	Slack	Late Finish

Critical Path:

↳ sequence of tasks that take
longest time to complete
↳ It is shortest possible time Project can be finished in
↳ any task on it is critical task

No critical task can have its duration changed without affecting the end date of the project.

Slack (aka Float)

↳ amount of times a task can be extended
before it affects the other tasks

CRITICAL TASKS have NO SLACK

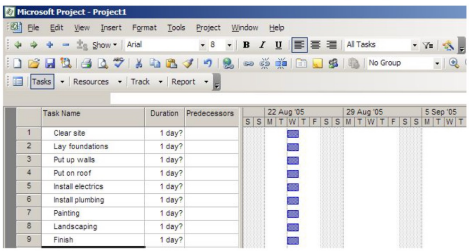
Float (or Slack) = late start − early start

Slack = late finish − early finish

GANTT CHART

Making a Gantt chart

- Step 1 – list the tasks in the project



	Task Name	Duration	Predecessors
1	Clear site	1 day?	
2	Lay foundations	1 day?	
3	Put up walls	1 day?	
4	Put on roof	1 day?	
5	Install electrics	1 day?	
6	Install plumbing	1 day?	
7	Painting	1 day?	
8	Landscaping	1 day?	
9	Finish	1 day?	

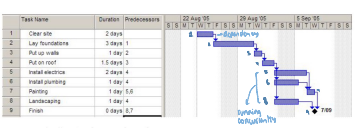
- Step 2 – add task durations

	Task Name	Duration	Predecessors
1	Clear site	2 days	
2	Lay foundations	3 days	
3	Put up walls	1 day	
4	Put on roof	1.5 days	
5	Install electrics	2 days	
6	Install plumbing	1 day	
7	Painting	1 day	
8	Landscaping	1 day	
9	Finish	0 days	

- Step 3 – add *dependencies* (which tasks cannot start before another task finishes)

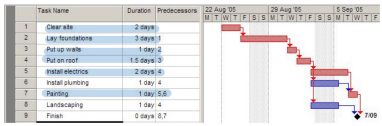
	Task Name	Duration	Predecessors
1	Clear site	2 days	
2	Lay foundations	3 days	1
3	Put up walls	1 day	2
4	Put on roof	1.5 days	3
5	Install electrics	2 days	4
6	Install plumbing	1 day	4
7	Painting	1 day	5,6
8	Landscaping	1 day	4
9	Finish	0 days	8,7

Notes



- The arrows indicate **dependencies**.
- Task 1 is a **predecessor** of task 2 – i.e. task 2 cannot start before task 1 ends.
- Task 3 is **dependent** on task 2. Task 7 is dependent on two other tasks
- Electrics, plumbing and landscaping are **concurrent** tasks and can happen at the same time, so they overlap on the chart. All 3 can start after task 4 ends.
- Painting must wait for both electrics and plumbing to be finished.
- Task 9 has zero duration, and is a **milestone**

- Step 4 – find the **critical path**



	Task Name	Duration	Predecessors
1	Clear site	2 days	
2	Lay foundations	3 days	1
3	Put up walls	1 day	2
4	Put on roof	1.5 days	3
5	Install electrics	2 days	4
6	Install plumbing	1 day	4
7	Painting	1 day	5,6
8	Landscaping	1 day	4
9	Finish	0 days	8,7

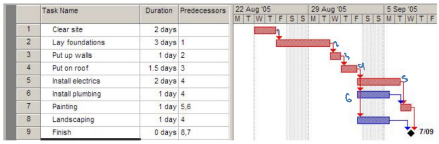
The critical path is the sequence of tasks from beginning to end that takes the **longest time** to complete.

It is also the **shortest possible time** that the project can be finished in.

Any task on the critical path is called a **critical task**.

No critical task can have its duration changed without affecting the end date of the project.

- MS Project can work out the critical path for you!
- The length of the critical path is the sum of the lengths of all critical tasks (the red tasks 1,2,3,4,5,7) which is 2+3+1+1.5+2+1 = 10.5 days.
- In other words, the minimum amount of time required to get all tasks completed is 10.5 days
- The other tasks (6,8) can each run over-time before affecting the end date of the project



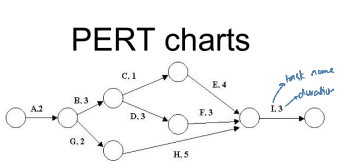
	Task Name	Duration	Predecessors
1	Clear site	2 days	
2	Lay foundations	3 days	1
3	Put up walls	1 day	2
4	Put on roof	1.5 days	3
5	Install electrics	2 days	4
6	Install plumbing	1 day	4
7	Painting	1 day	5,6
8	Landscaping	1 day	4
9	Finish	0 days	8,7

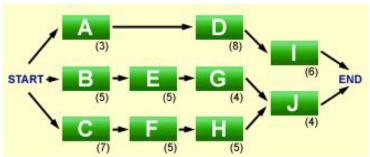
- The amount of time a task can be extended before it affects other tasks is called *slack* (or *float*).
- *Float = Late Start – Early Start OR Late Finish – Early Finish*
- Task 6 can take an extra day and a half before it affects the project's end date, so each has **1.5 day's slack**.

Critical tasks, by definition, can have NO slack.

Thus if you are ever asked, "*Can the duration of a critical task be changed without affecting the end date of the project?*", the answer is always **NO**!

PERT CHART





Activity on Arrow

Tasks →

Task name letters

Start/end of tasks ○ nodes

Task duration numbers

Activity on Node

tasks ○ nodes

Connectors →

SIDE BAR

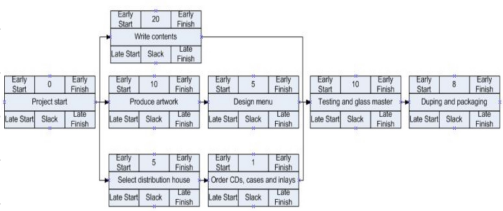
Activity	Predecessor	Time estimates			Expected time
		Opt. (*O*)	Normal (*M*)	Pess. (*P*)	
A	—	2	4	6	4.00
B	—	3	5	9	5.33
C	*A*	4	5	7	5.17
D	*A*	4	6	10	6.33
E	*B, C*	4	5	7	5.17
F	*D*	3	4	8	4.50
G	*E*	3	5	8	5.17

$$T_E = \frac{(O + 4M + P)}{6}$$

T_E ↙ expected time

being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

- *Early Start – The earliest time that an activity can start*
- *Early Finish – The earliest time that an activity can finish*
- *Late Start – The latest time that an activity can start*
- *Late Finish – The latest time that an activity can finish*



18

Total Float

$$TF = LS - ES \quad | \quad TF = LF - EF$$

$$FF = \min ES_{Sucessor} - ES_{activity} - Duration_{activity}$$

↙ Free Float

$$FF \leq TF$$

↙ how long an activity can be delayed

MEMBOOB

Technology

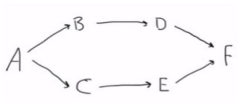
NAV Tech

Reg Soum Karachi

A

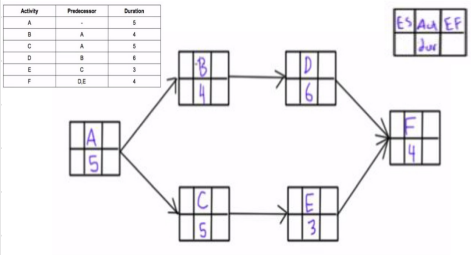
Example: Activity Network

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D,E	4



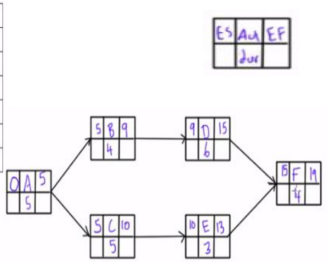
Example: Activity Network

Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D,E	4



Example: Activity Network- Early Start and Early finish

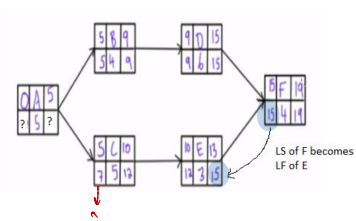
Activity	Predecessor	Duration
A	-	5
B	A	4
C	A	5
D	B	6
E	C	3
F	D,E	4



A,B,D,F = 5+4+6+4=19
A,C,E,F= 5+5+3+4= 17

Therefore, ABDF is critical path

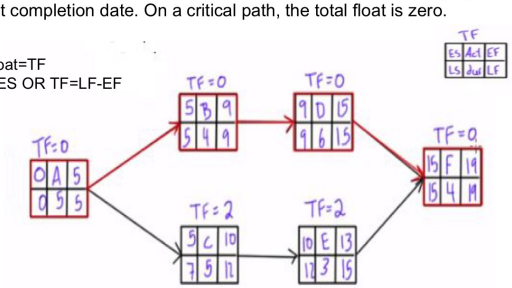
Example: Activity Network- Late Start and Late finish



Example: Activity Network- Total Float

Total float is how long an activity can be delayed, without delaying the project completion date. On a critical path, the total float is zero.

Total Float=TF
TF=LS-ES OR TF=LF-EF



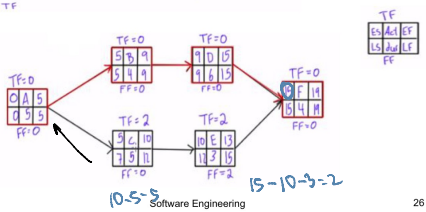
Example: Activity Network- Free Float

Free float is how long an activity can be delayed, without delaying the Early Start of its successor activity.

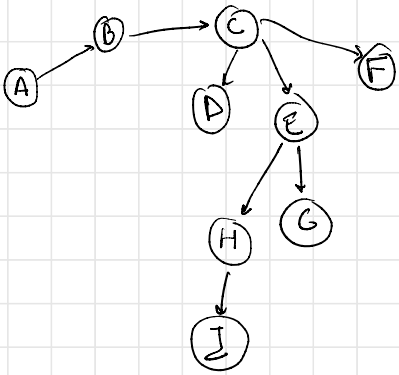
Free Float = FF = Minimum $ES_{successors}$ − $ES_{activity}$ − $Duration_{activity}$

FF ≤ TF

Or FF= ES(successor) – EF (activi



Activities	Completion Time	Immediate Predecessor
A. Excavate	5 days	
B. Foundation	2 days	A
C. Frame	12 days	B
D. Electrical	9 days	C
E. Roof	5 days	C.
F. Masonry	8 days	C.
G. Interior	10 days	E.
H. Exterior	7 days	E.
I. Landscape	5 days	H.

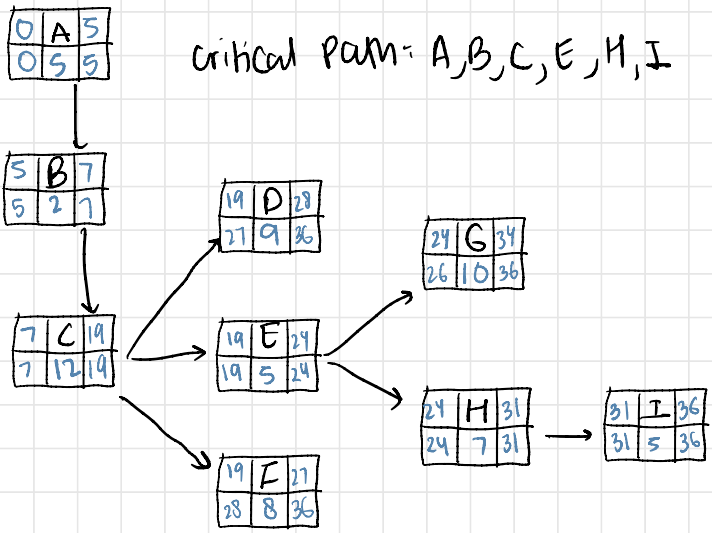


i. Draw the activity network diagram. How many paths are in the network, and what are they along with their duration. [4 points]

ii. Calculate the early start, early finish, late start and late finish for all activities. [4 points]

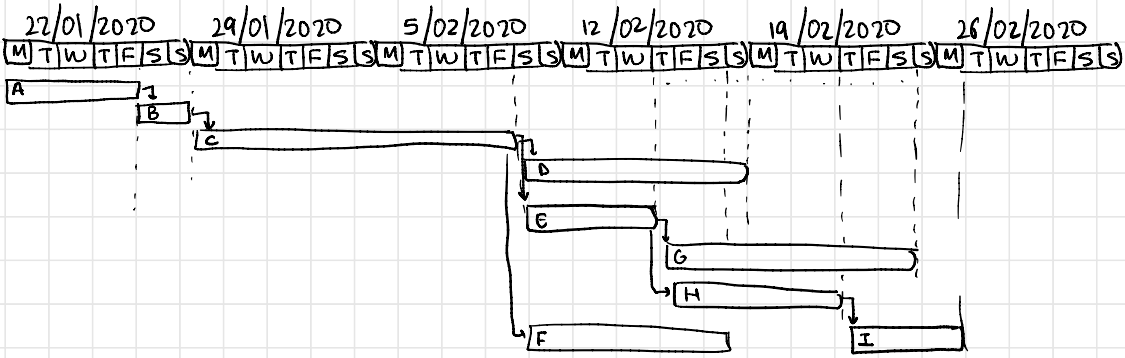
iii. Determine the critical path. [2 points]

iv. Draw the Gantt chart showing the project schedule with starting date 22/01/2020. [2 points]



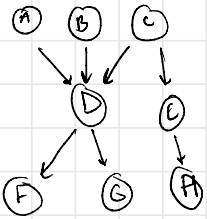
Critical Path: A, B, C, E, H, I

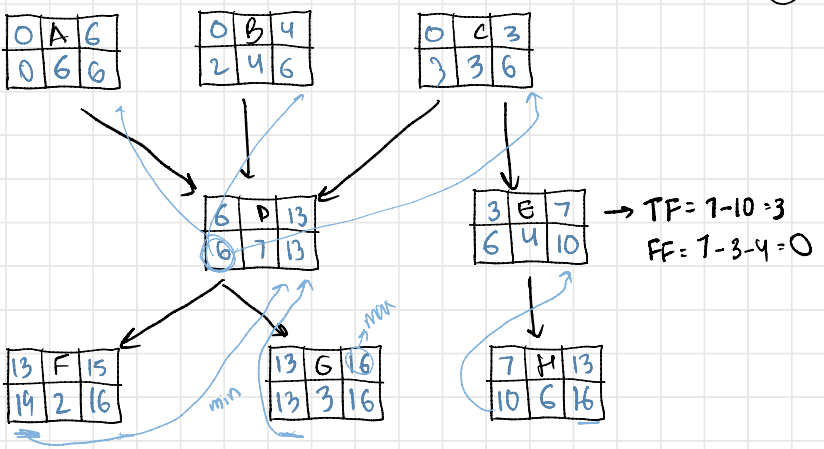
GANT CHART



Activity	Predecessor	Duration (Weeks)
A		6
B		4
C		3
D	A, B, C	7
E	C	4
F	D	2
G	D	3
H	E	6

a) Draw the network diagram to analyze your project.
b) How many paths are in the network, and what are they along with their duration? What is the critical path and its duration?
d) Do a forward and backward pass by showing all calculations (ES, EF, LS, LF).
e) What is the free float and total float on activity E?
f) What is the impact to the project if activity B takes three weeks longer than planned? Does this affect the critical path?





→ TF = 7-10 = 3
FF = 7-3-4 = 0

min
LF LS =

A D F = 15
A D G = 14
C E H = 13

B D F = 13
B D G = 14

C D F = 12
C D G = 13

Critical Path = 15

Total Float

$TF = LS - ES \mid TF = LF - EF$

$FF = \min ES_{Sucessor} - ES_{activity} - Duration_{activity}$

Free Float

T

Some terminologies

- Buffer also referred to as Schedule Margin, Schedule Buffer, Contingency, Reserve, etc... is an activity or period of time that is strategically placed on the Critical Path, typically prior to the agreed upon completion or delivery date.
- Lead
- The amount of time whereby a successor activity can be advanced with respect to a predecessor activity.
- Lag
- The amount of time whereby a successor activity is required to be delayed with respect to a predecessor activity.

RISK ANALYSIS

Reactive Risk Management

- project team reacts to risks when they occur
- Mitigation (avoid or reduce risk of loss)—plan for additional resources in anticipation of fire fighting
- fix on failure—resource are found and applied when the risk strikes
- crisis management—failure does not respond to applied resources and project is in jeopardy

Proactive Risk Management

- formal risk analysis is performed
- organization corrects the root causes of risk
  - TQM concepts and statistical SQA
  - examining risk sources that lie beyond the bounds of the software
  - developing the skill to manage change

Project Risks

- Identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, requirements problems and their impact on a software project, project complexity, size, and the degree of structural uncertainty

Risk Due to Product Size

Attributes that affect risk:

- **estimated size of the product in LOC or FP?**
- **estimated size of product in number of programs, files, transactions?**
- **percentage deviation in size of product from average for previous products?**
- **size of database created or used by the product?**
- **number of users of the product?**
- **number of projected changes to the requirements for the product? before delivery? after delivery?**
- **amount of reused software?**

Technical Risks

- Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than you thought it would be.

Business Risks

- Business risks threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are:
- (1) building an excellent product or system that no one really wants (market risk),
- (2) building a product that no longer fits into the overall business strategy for the company (strategic risk),
- (3) building a product that the sales force doesn't understand how to sell (sales risk),
- (4) losing the support of senior management due to a change in focus
- or a change in people (management risk), and
- (5) losing budgetary or personnel commitment (budget risks)

- **Known** risks are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment)
- **Predictable** risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced)
- **Unpredictable** risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance

Seven Principles

(a framework to accomplish effective risk management)

- **Maintain a global perspective**—view software risks within the context of system and the business problem
- **Take a forward-looking view**—think about the risks that may arise in the future; establish contingency plans
- **Encourage open communication**—if someone states a potential risk, don't discount it.
- **Integrate**—a consideration of risk must be integrated into the software process
- **Emphasize a continuous process**—the team must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved.
- **Develop a shared product vision**—if all stakeholders share the same vision of the software, it likely that better risk identification and assessment will occur.
- **Encourage teamwork**—the talents, skills and knowledge of all stakeholder should be pooled

Risk Management Paradigm

Risk Identification

- *Generic and Product specific*
- *Checklist:*
  - *Product size*—risks associated with the overall size of the software to be built or modified.
  - *Business impact*—risks associated with constraints imposed by management or the marketplace.
  - *Customer characteristics*—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
  - *Process definition*—risks associated with the degree to which the software process has been defined and is followed by the development organization.
  - *Development environment*—risks associated with the availability and quality of the tools to be used to build the product.
  - *Technology to be built*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
  - *Staff size and experience*—risks associated with the overall technical and project experience of the software engineers who will do the work.

Assessing Project Risk-I

- Have top software and customer managers formally committed to support the project?
- Are end-users enthusiastically committed to the project and the system/product to be built?
- Are requirements fully understood by the software engineering team and their customers?
- Have customers been involved fully in the definition of requirements?
- Do end-users have realistic expectations?

Assessing Project Risk-II

- Is project scope stable?
- Does the software engineering team have the right mix of skills?
- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?
- Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk Components

(U.S. Air Force [AFC88] risk drivers that affect software risk components)

- *performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- *cost risk*—the degree of uncertainty that the project budget will be maintained.
- *support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- *schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

Risk Projection (estimation)

- *Risk projection*, also called *risk estimation,* attempts to rate each risk in two ways
  - the likelihood or probability that the risk is real
  - the consequences of the problems associated with the risk, should it occur.
- The are four risk projection steps:
  - establish a scale that reflects the perceived likelihood of a risk
  - delineate the consequences of the risk
  - estimate the impact of the risk on the project and the product,
  - note the overall accuracy of the risk projection so that there will be no misunderstandings.

Building a Risk Table

Risk	Probability	Impact	RMMM
			Risk Mitigation Monitoring & Management

These slid

- Estimate the probability of occurrence
- Estimate the impact on the project on a scale of 1 to 5, where
  - Impact values:
    - 1—catastrophic
    - 2—critical
    - 3—marginal
    - 4—negligible
  - Sort the table by probability and impact

FIGURE 28.2 Sample risk table prior to sorting

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	

Risk Exposure (Impact)

The overall *risk exposure,* RE, is determined using the following relationship [Hal98]:

RE = *P* x *C*

where

P is the probability of occurrence for a risk, and

C is the cost to the project should the risk occur.

Risk Exposure Example

- **Risk identification.** Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.
- **Risk probability.** 80% (likely).
- **Risk impact.** 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is $14.00, the overall cost (impact) to develop the components would be 18 x 100 x 14 = $25,200.
- **Risk exposure.** RE = 0.80 x 25,200 ~ $20,200.

reusable components = 60

70% will be reused

Will be used = $60 \times \frac{70}{100} = 42$

Components to be developed from scratch = 60 - 42 = 18

P = 80

Avg component = 100 LOC

cost for each LOC = $14

impact = 18 x 100 x 14 = 25200

RE = 0.8 x 252000 = 20,2000

RE = P x C

(risk exposure, Probability, cost)

Risk Mitigation, Monitoring, and Management

- mitigation—how can we avoid the risk?
- monitoring—what factors can we track that will enable us to determine if the risk is becoming more or less likely?
- management—what contingency plans do we have if the risk becomes a reality?

Risk Due to Business Impact

Attributes that affect risk:

- affect of this product on company revenue?
- visibility of this product by senior management?
- reasonableness of delivery deadline?
- number of customers who will use this product
- interoperability constraints
- sophistication of end users?
- amount and quality of product documentation that must be produced and delivered to the customer?
- governmental constraints
- costs associated with late delivery?
- costs associated with a defective product?

Risks Due to the Customer

Questions that must be answered:

- Have you worked with the customer in the past?
- Does the customer have a solid idea of requirements?
- Has the customer agreed to spend time with you?
- Is the customer willing to participate in reviews?
- Is the customer technically sophisticated?
- Is the customer willing to let your people do their job—that is, will the customer resist looking over your shoulder during technically detailed work?
- Does the customer understand the software engineering process?

Risks Due to Process Maturity

Questions that must be answered:

- Have you established a common process framework?
- Is it followed by project teams?
- Do you have management support for software engineering
- Do you have a proactive approach to SQA?
- Do you conduct formal technical reviews?
- Are CASE tools used for analysis, design and testing?
- Are the tools integrated with one another?
- Have document formats been established?

Technology Risks

Questions that must be answered:

- Is the technology new to your organization?
- Are new algorithms, I/O technology required?
- Is new or unproven hardware involved?
- Does the application interface with new software?
- Is a specialized user interface required?
- Is the application radically different?
- Are you using new software engineering methods?
- Are you using unconventional software development methods, such as formal methods, AI-based approaches, artificial neural networks?
- Are there significant performance constraints?
- Is there doubt the functionality requested is "do-able?"

Staff/People Risks

Questions that must be answered:

- Are the best people available?
- Does staff have the right skills?
- Are enough people available?
- Are staff committed for entire duration?
- Will some people work part time?
- Do staff have the right expectations?
- Have staff received necessary training?
- Will turnover among staff be low?

Recording Risk Information

Project: Embedded software for XYZ system
Risk type: schedule risk
Priority (1 low ... 5 critical): 4
Risk factor: Project completion will depend on tests which require hardware component under development. Hardware component delivery may be delayed
Probability: 60 %
Impact: Project completion will be delayed for each day that hardware is unavailable for use in software testing
Monitoring approach:
Scheduled milestone reviews with hardware group
Contingency plan:
Modification of testing strategy to accommodate delay using software simulation
Estimated resources: 6 additional person months beginning in July

Models are used during the requirements engineering process to help derive the detailed requirements for a system, during the design process to describe the system to engineers implementing the system, and after implementation to document the system's structure and operation. You may develop models of both the existing system and the system to be developed:

1. Models of the existing system are used during requirements engineering. They help clarify what the existing system does, and they can be used to focus a stakeholder discussion on its strengths and weaknesses.

2. Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation. If you use a model-driven engineering process (Brambilla, Cabot, and Wimmer 2012), you can generate a complete or partial system implementation from system models.

Q1. You are about to start a software company. How will you make sure you and your software engineers adhere to the following principles?

i. CLIENT AND EMPLOYER
ii. PRODUCT

Write down a brief guide about how your organization will go about ensuring that these principles are followed. [5 marks]

6/9/22, 11:16 AM Software Engineering Code - ACM Ethics

2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.