Table of Contents

Midterm Tips

- ➔ White-box
- ➔ Black-Box
- ➔ Types of Black Box Testing
- ➔ Black Box Testing Techniques
- ➔ Cause-Effect Graphing
- ➔ Equivalence Partitioning
- ➔ Boundary Value Analysis
- ➔ Decision Table Technique
- ➔ Integration Testing
- ➔ And other type of testing
- ➔ Types of Review
  - ◆ Code Review
  - ◆ Desk Checking
  - ◆ Pair Programming
  - ◆ Code Walkthrough
  - ◆ Technical Review
  - ◆ Formal Inspection
- ➔ Quality
- ➔ Branches

Tutorial Question Appeared: 💚
PYQ Appeared: [🟥n] , n = times of appeared
PYQ Related: 🟠

C1: Software Testing Concepts

Software Testing

- Construction of industry products based on a set of requirements.
- Problems must be corrected if they exist.
- requires running the software on a computer and comparing it with the requirements.

💚Purposes of Software Testing

- Analyze programs and documentation to prevent failures.
- Ensure the system meets requirements and runs successfully in its intended environment.
- Verify the system is "fit for purpose."
- Confirm the system performs as expected.
- Identify and correct defects.
- Ensure user needs are satisfied.
- Prevent users from detecting issues.
- Deliver a quality product.

Roles in Testing

Test management role

- Takes overall responsibility for the test process, test team and leadership of the test activities.
- Focuses on test planning, test monitoring and control and test completion.

Testing role

- Takes overall responsibility for the engineering (technical) aspect of testing.
- Focuses on test analysis, test design, test implementation and test execution.

Who performs the test?

- Software test engineer/tester
- Programmers/Developers
- End-Users

💚Terminology (7)

Software error	• Incorrect internal state that is a sign / symptom of some fault. • May cause fault • Human mistake or error • Example: Bad programming/ Incorrect use of program statements
Software Fault (DEFECT) • A.k.a bug, defect or internal error	• Present since the software was developed. Only when the software is executed, it become visible as a failure. • Fault can cause zero to many failures for any number of users. • Example: Incorrect / forgotten statements in the program.

Defect Masking	• Fault is hidden by one or more other faults in other parts of the program. • Failure happens after the masking defects have been corrected. • This indicates / means that corrections can have side effects.
Software Failure	• indicates that the system cannot fulfill the user requirements • Difference between actual result / behavior and the expected result / behavior. • May be caused by faults. • Example: A software is too difficult to use or too slow but still fulfils the functional requirements.
Validation	• Process of evaluating software at the end of software development to ensure it has fulfilled with expected usage. • “Are we building the right system?”
Verification	• Process of checking if the products in a given software development phase fulfil the requirements set in previous phase. • “Are we building the system right?
Software Testing	• Process of evaluating a system or system component in manual or automated way to verify that it satisfies specified requirements. • Process of executing a program for finding errors.

* Not all software errors will turn into software faults. And, not all software faults will turn into software failure.

💚Principle of software testing (7)

Testing shows the presence of defects, not their absence.	• Only product failures can be shown. • Cannot prove that there are no defects in the product.
Exhaustive testing is impossible.	• Impossible to include all possible values in testing.
Testing activities should start as early as possible.	• Once all the requirements have been decided, testing activities can start. • Find defects early.
Defect clustering	• Defects are not evenly distributed, they cluster together. • When most defects are found in parts of a test object, it means that there will be more defects nearby.

The pesticide paradox.	• If the same tests are repeated over and over, they will lose their effectiveness because they cannot discover new defects. • New and modified test cases should be developed and added to the test.
Testing is context dependent.	• Every system should be tested in a different way. • Testing must be adapted to risk in the use and application environment.
No failures means the system is useful, is a fallacy.	• Finding and fixing defects does not ensure that the system meets user expectations.

Test Process (7)

Test Planning	• Define test objectives • Select approach that can achieve objectives within context constraints
Test Monitoring and Control	• Test monitoring: checking test activities and make comparison between actual progress and plan • Test control: taking necessary actions to meet testing objectives
Test Analysis	• Analyzing test basis • Identify testable features and prioritize associated test conditions with related risks
Test Design	• Elaborating test conditions into test cases and testing tools
Test Implementation	• Creating or acquiring the testing tools for test execution • Organizing test cases into test procedures • Building and verified test environment
Test Execution	• Running tests based on test execution schedule in manual or automated way
Test Completion	• Analyzing test activities • Identify lessons learned and improvements • Shut down the test environment to agreed state

	• Create test completion report and communicate to stakeholders

T1Q2. Software testing is unavoidable when developing the software. Some said software testing is an "umbrella activity". Analyse the reason behind.

- Software testing is a must in the whole development of the software.
- In each stage in software development, the software testing will be executed as an umbrella activity. Any outcome from each stage, they need to be tested.
- To ensure all the tasks are done in the right way, prevent failures.

T1Q3. Early testing is one of the principles of software testing, throughout the software development life cycle, identify what artifacts will be tested after the Analysis, Design, Coding and Implementation stage.

- Artifacts are the milestones for each stage (purpose).

Analysis stage

- Artifact: Requirement Document / Software Requirement Specification (SRS)
- Review whether all the requirements are there and they are correct.

Design stage

- Artifact: Software Design Document / Software Design Architecture Document (database design, UI design, prototype are parts of them)
- Test whether all the requirements are converted into a design document correctly.

Coding stage

- Artifact: Source code
- Test whether the function is working
- Test whether the modules are working when all the codes are combined together (compiler)

Implementation stage

- Artifact: Complete developed system
- Test the performance, usability, efficiency of the system and so on
- Test functional or non-functional requirements

T1Q7. Why some defects in the software will not turn into failure. Explain with one example.

- The defects are not executed yet or not happening yet so it would not turn into failure but it still might turn into failure in the future. (hiding effect)
- The defects might have a low severity effect on the system. For example, spelling mistakes, incorrect alignment or image display problems. They are observable by the users but it would not stop them from using the system.

T1Q8. When performing testing, some tests have a higher priority than others. What are the criteria to be considered in prioritizing the tests?

- Importance of function: Which function is more important
- Criticality of function: High critical functionalities and modules will be tested first
- Precondition: Consider the precondition test cases or function first, ensure that the precondition test cases have passed before continuing the subsequent functionalities. (e.g. login function is a must before any other functions such as create post function, search function)

C2: Level of Testing

Module (Unit) Testing

Process of testing individual sub-programs, subroutines, classes or procedures of a program.

	💚Incremental Approach...	💚Non-incremental Approach
Definition	Combine the next module to be tested with the set of previously tested modules before it is tested	Testing each module independently, then combining the modules to be tested as a whole program
Driver and Stub	Each module needs to have ONE driver module	Each module needs to have one driver module and one or more stubs modules
Advantages of Incremental Approach		
Work	Less work (only 5 drivers or 5 stubs required)	More work (5 drivers and 5 stubs required)
Error Detection	Programming errors can be detected earlier	Programming errors cannot be detected earlier (“do not see each other” until the end of testing)
Debugging	Easier for debugging	Difficult for debugging
Exposure	Actual modules receive more exposure by the completion of the last module test	Actual modules receive less exposure
Stub and Driver Generation Time	Less machine time used to generate stubs and drivers	More machine time used to generate stubs and drivers
Advantages of Non-Incremental Approach		
Execution Time	More machine time used to execute	Less machine time used to execute
Parallel Activities	Does not support parallel activities	Support parallel activities

Driver and Stub: Used to replace the missing software and simulate the interface between the software components in a simple manner.

Driver	(more on upper-level module but lower also can) used in Top-Down Integration Testing
Stub	(use when missing lower-level module) used in Bottom-Up Integration Testing

...Incremental Approach

Top-down strategy	Bottom-up strategy
Starts with the top, or initial module in the program	Begins with the terminal module(s) in the program
From main module to sub-modules	From sub-modules to main module
Stub is used to simulate sub-module	Driver is used to simulate main module
Advantageous when major flaws occur towards the top of the program	Advantageous when major flaws occur towards the bottom of the program
Once I/O functions are added, representation of test cases is easier	
Early program skeletons allow demonstrations and boosts morale.	The program as an entity does not exist until the last module (main module) is added.
Stub modules must be produced (more complicated than they first appear to be)	Driver modules must be produced.
Before the I/O functions are added, the representation of test cases in stubs can be difficult.	
Test conditions may be impossible or very difficult to create and observation of test output is more difficult.	Test conditions are easier to create with I/O functions.

Higher-order testing	System Testing
💚integration test • Prepare sufficient test data to see how the integrated system can work with the test data. • Done individual module testing before integrating them. Setup the integration environment (software, hardware, etc). • Consider the approaches to be used. • Prepare dummy code if there is a need to integrate a not fully completed individual module for testing purposes. Functional Testing • Process of attempting to find the difference between the program and external specifications.	Facility Testing • Determine whether the facility (function) mentioned in objectives was implemented. • Scan objectives one-by-one to produce the checklist for testing purposes. Volume Testing • Subject the program with a heavy volume of data. • To show that the program cannot handle the volume of data specified in its objectives. Stress Testing

- External specification: description of program behavior from end-users point of view. (purpose)
- Most of the functional testing is black-box technique.

System Testing

- Compare system / program to its original objectives.
- Formulate test case by analyzing user documentations.

Acceptance Testing

- Compare the program to its initial requirements and the current needs of its end users.
    - Alpha test
        - Versions of the complete software are tested by customers under the developer supervision at the developer site.
    - Beta test
        - Versions of the complete software are tested by customers without developer supervision at the customer's own site.

Installation Testing

- Find errors that occur during the installation process.
- Considerations:
    - Users must select a variety of options.
    - Files and libraries must be allocated and loaded. (installation file)
    - Valid hardware configurations must be present. (hardware)
    - Programs may need network connectivity to connect to other programs. (network)

User Interface Testing

- More focus on usability testing criteria.
- User Interface characteristics of the system are measured.
- Weaknesses of the system are identified for correction.
- Ensure that users of the system can carry out intended tasks efficiently, effectively and satisfactorily.

💚Regression Testing

- Retesting of a unit, integration and system after modification to ensure that the changes have been done.
- Subject the program with heavy load or stress (peak volume of data, or activity, encountered over a short-span of time)

Usability Testing

- Tasking the end user with testing the software in real time environment.

Security Testing

- Focus on security objectives.
- Process of attempting to design test case that subvert 顛覆 the program's security checks.
- To search security holes.

Performance Testing

- Focus on performance or efficiency objectives.
- Response time under certain workload and configuration conditions.

Storage Testing

- Focus on storage objectives.
- Amount of system memory usage and size of temporary or log files used by the program.
- To ensure that the program can control the use of system memory to not affect other processes running on the host.

Configuration Testing

- For programs that support various hardware configurations.
- Multiple web browsers or operating systems.

Compatibility / Conversion Testing

- Focus on compatibility and conversion procedures from existing system.
- Upgrading database system, new writing format of word files.

Installation Testing

- For systems that have complicated installation procedures.
- Important for packages which have an auto installation process.
- Vital part in system testing.

Reliability Testing

- Focus on specifying reliability of system
- Testing uptime requirements

Recovery Testing

- Focus on recovery objectives that state how the system is to recover from programming errors, hardware failures and data errors.

- Regression testing is applied to code immediately after changes are made.
- To ensure that there is no effect on the test object behavior after changes.
- Important way of monitoring the effects of change.

Critical Path Testing

- Define a critical path which represents the main function of the system.
- Critical paths must be covered first.
- Developer focus on design, implementation and testing of critical path first (skeleton development and testing)
- Normally implemented in large projects.

- To show that these recovery functions do not work correctly.

Serviceability / Maintenance Testing

- Focus on containing serviceability and maintenance characteristics.
- Including service aid provided to the system.

Documentation Testing

- To test accuracy and clarity of user documentations.

Procedure Testing

- Any prescribed human procedures should be tested during the system test.
- Database administrators should document procedures for backup and recovery of the database system.
- Then. A person not associated with database administration should test the procedures.

T2Q3. Compare Unit Test with Integration Test.

Unit Test	Integration Test
To test individual function / operation / component	To test the combined functionalities / operations / components in any level
Before integration	After integration
To test independent units / parts to check the functions within them	To see the compatibility of the combined units
Can test by using compiler / editor tools (more simple)	Require testing tools (more complex)

T2Q4. In some cases, Integration Test is performed but Unit Test is ignored. Explain the reasons.

- Focus on the overall outcome of the integrated system rather than individual unit performance. Unit tests may be ignored when: The unit is too simple to test/The unit cannot provide meaningful or accurate results in isolation.
- Some systems rely heavily on interconnected components. Testing a unit in isolation may not be feasible or meaningful.
- Components from previous projects are reused without modification. These components are considered already tested and reliable. Only integration testing is needed to verify compatibility with the current system.

T2Q5. Differentiate Regression Test with Re-Test.

Regression test

- Executed once there is any changes or modification on the source code.
- To compare with the precious version to ensure that there is no any defects caused after modification done.
- Might modify the test cases and requirements.

Re-test

- Repeat the same testing without any changes made
- To ensure the module is completely correct.
- Not modify any test case and requirements.

C3: Static Test

💚Concepts of Static Test

- Conducted before development of software
- Testing the system or objects without executing the code.
- Focuses on identifying defects in artifacts (e.g., requirements, design documents, test cases).
- Compares the test object against documentation to detect errors or faults.
- Examples:
  - Identifying incomplete requirements.
  - Checking for non-compliance with standards.

T3Q3. THREE advantages of Static Testing over Dynamic Testing?

- Can easily identify and fix the defects at earlier stages before source code is created.
- Save time and cost as the defects can be fixed immediately on documentation which wouldnt cause much time and cost.
- Offer a clear and better understanding of the project so the development time required can be shortened.

Types of Static Test (4)

1. Review

Perform different static analysis techniques to examine specific documents.

Similar to inspection, though inspections are typically more formal, involving specific rules and data collection.

Applicable Documents:

Contracts, requirements, design specifications, program code, test plans, and manuals.

Benefits:

💚Review Process

1. Planning
   - Decide which documents will undergo review and the technique to use.
   - Form a review team.
2. Kick-Off (Overview)
   - Provide necessary information about the document and review objectives to sharing information about the document reviewed.
3. Individual Preparation
   - Reviewers study the review object and check against the document, and note deficiencies(even any potential defects), or questions.
4. Review Meeting
   - Led by a review leader/moderator.
   - Participants should remain diplomatic and focus on contributing effectively.
   - Experts are encouraged to share opinions, as the focus is on evaluating the product, not the author.

• Early defect detection, reducing costs and development time. • Improved quality by finding inconsistencies and errors early. • Reduced failure rates through clarification and correction.	• Meetings usually have a set time limit and aim to verify if the document meets requirements, adheres遵守 to standards, and find defects. • Outcomes include recommendations to accept, revise, or rewrite the document, with all reviewers agreeing on the findings and overall result. 5. Rework • The manager decides whether to follow the recommendation or take another action. • The author typically addresses the defects identified in the review and reworks the document. 6. Follow-Up • The correction of defects is followed up, usually by the manager, moderator, or an assigned person. • If the first review result was unsatisfactory, a follow-up review is scheduled, focusing on the changed areas.

💚Types of Review(6)

A. Code Review	• Evaluates the code structure with the tester. • Checks for consistency, compliance with standards, and the quality of comments.
B. Desk Checking	• A primary static check(on the code) done by programmers before compilation or execution. • Identified errors will be checked and corrected by the author later. • The code is compared to the requirements specification or design to ensure it aligns with client requests.
C. Pair Programming	• Two programmers share a workstation to collaborate on writing and testing code. [from Extreme Programming (XP)] • One programmer (driver) controls the keyboard and mouse to write code, while the other (navigator) reviews the written code.
D. Code Walkthrough	• A less formal review is usually conducted in a single meeting. • Aims to find anomalies/improvement, consider alternative implementations, and evaluate conformance to standards and specifications. • Involves testers and programmers, with paper test cases used to mentally execute the program.
E. Technical Review	• A team of qualified personnel examines the software product's suitability and checks for discrepancies from the specification and standards point of view.
F. Formal Inspection	• Involves specialists, designers, authors, and a moderator. • The programmer describes the program logic, and questions are raised to identify potential problems.

	• The program is analyzed using a checklist to detect faults, and a report is produced with follow-up actions. • Defects may include logical errors, uninitialized variables, or non-compliance with standards. • Data collected during inspections helps assess the development process and optimize it for improvement.

Pros and Cons of code walkthrough and formal inspection:

Pros: more efficient than desk review, programming ability improves during discussion

Cons: More times, more man-power needed

2. Static Analysis

- Find defects in a document with support of tools
- Example:
  - Spelling checker
  - Compiler
  - Tools for checking programming standards

3. Control Flow Analysis

- Examine all possible sequence of a program (control flow)
- Performed using graphs or tables such as control flow graph
- Example: control flow of program affected by if, loop statements

4. Data Flow Analysis

- Detecting flow anomalies

T3Q4. Who are the participants in a Review and what are their roles?
• Project manager
  ○ Accept the final recommendation given or give a new suggestion on how to rework
  ○ Follow up of the rework
• Secretary
  ○ Share information of the meeting to all the members
  ○ Share document to the reviewers
• Author
  ○ Do the rework based on suggestion or recommendation
	T3Q5. Imagine that you are the chairperson of a formal review, if there is conflict happening among the reviewers, how do you solve the conflicts between the reviewers?
	• Request all the reviewers to express their understanding and opinion
	• Conduct a voting
	• Ask the reviewers to provide the pros and cons of their suggestion and justify their answer based on their opinion
	• Review leader decide his own recommendation himself or choose the suitable recommendation given by reviewers.

- Review leader
  - Lead the review meeting
  - Make the final recommendation
  - Follow up of the reworks
- Reviewers

T3Q7. Analyse the following **pseudo codes/codes and perform desk checking** on the changes of variable's value:

Test Data for number is 2, 3, 1, 8, 10

```
BEGIN
Count = 0
Total = 0
REPEAT
Count = count + 1
Get number
Total = Total + number
UNTIL count = 5
Display Total
END
```

Count	Number
0	0
1	2
2	3
3	1
4	8
5	10

C4: Dynamic Testing

💚Dynamic Testing

- Test the system and components by executing the source code.

• Provide required input values and observe the results. • Create test cases. For example, white-box testing (analyze the back-end source code to check how the code is running) and black-box testing (conducting testing without understanding how the source code works).	**Static test** • Testing without running the source code • Review the documentation.

💚Black Box Testing

- Conduct testing without understanding how the source code works.
- Usually conducted by testers and end-users.
- Not requiring programming techniques and knowledge.
- Create test cases from system specifications. (ignore design and code)
- Test functionality and features.

Goal: Find where the program fails to meet specifications.

Types of Black Box Testing (13)

Functional Testing	• Test the system against functional requirements/specifications. Input is fed into functions, and output is examined. Steps: 1. Identify expected functions of software to perform. 2. Create input data and determine output based on function's specifications. 3. Execute test cases. 4. Compare actual vs. expected output.
Smoke Test / Build Verification Testing	• Done by the development team. • Ensures critical functions work. Determines if a build is stable enough for further testing.

Regression Testing	• Repeated testing after modifications to find new defects. • Applied to functional and non-functional testing at all levels. • Ensures changes (enhancements or fixes) do not affect the software negatively.
Sanity Testing	• Subset of Regression Testing. Focuses on specific areas of functionality after minor changes. • Narrow and deep, often unscripted.
Load Testing	• Tests application behavior under expected load. Determines system behavior under normal and peak conditions. • Identifies maximum operating capacity and bottlenecks. • Example: Assess CPU, memory, and network usage as user numbers increase.
💚Stress Testing	• Tests beyond normal capacity to find breaking points to determine the stability of a given system. • Focuses on robustness, availability, and error handling under heavy load. • Ensures the system doesn't crash when there are insufficient resources.
Exploratory Testing	• Tester explores the software to find what it does/works and what doesn't does/works. • The tester is constantly making decisions about what to test next and where to spend the (limited) time. • Common in Agile development.
Performance Testing	• Measures system speed under specific workloads. Purposes: • Verify performance criteria. • Compare two systems to find which is better. • Identify causes of poor performance.
Usability Testing	• Tests how easily users can interact with the interface, usually involving real users. • Example: Evaluate website layout, navigation, and content.
💚Recovery Testing	• Tests recovery how fast and how well after crashes or hardware failures.

	• Forces software failures to verify recovery procedures.
User Acceptance Testing (UAT)	• Tests system for user acceptability and evaluates system compliance with business requirements. • Assesses if the system is ready for delivery. Often performed by clients or end-users.
Alpha Testing	• Conducted at the developer's site, developers observe users and record issues. • Performed when development is nearly complete; minor design changes are possible. • Done by a team independent of the design team, such as in-house test engineers or QA engineers.
Beta Testing / Field Testing	• Conducted at the customer's site, users test the system under real-world conditions. • Is the second phase of testing with intended audience feedback.

T4Q4: Compare **Volume Test, Stress Test, Load Test,** and **Performance Test**

Volume test	Stress test	Load test	Performance test
deal with huge amount of data To check whether the system can handle the large amount of data along with time without breaking down the system.	Similar to load tests, but increase the load to make them become a stress. To identify the breaking point / maximum limitation of the system can handle before breaking down.	deal with transactions, number of data, amount of devices and users and network bandwidth. To check whether the system can handle the given load without breaking down the system.	Deal with the response time, how fast the system can respond to its work. Check how the system perform, how fast is the response time. The shorter the response time, the shorter the waiting time.

Types of black box testing Techniques (3)

Exhaustive testing (all possible values) is impossible.Techniques applied to identify a right subset of test inputs:

1. Equivalence Partitioning Divide input data into equivalence classes (ECs). Test one representative input from each EC. Steps: • Identify equivalence classes (EC):	2. Boundary Value Analysis Tests the borders of equivalence classes. For each boundary: • Test the exact boundary value.

• Define test cases: Assign a unique number to each EC. Cover all valid and invalid ECs with test cases.	• Test the nearest adjacent values (inside and outside). Use the smallest possible increment (e.g., defined tolerance for floating-point data). Three test cases are created for each boundary. If adjacent equivalence classes share a boundary, the test cases overlap.

3. Logic-Based Testing (2)

3.1 Cause-Effect Graph...	3.2 Decision Table	3.3 Pairwise Testing
1. Identifies test cases based on dependencies. 2. Displays logical relationships between causes and effects in a cause-effect graph. 3. Requires identifying causes and effects from the specification. 4. Causes: Conditions defined by input values or combinations. 5. Logical Operators: Connect conditions (e.g., AND, OR, NOT). 6. Each cause can be True or False.	To transform a Cause-Effect Graph into a Decision Table and derive test cases, follow these steps: • Choose an effect • Add columns for cause-effect combinations • Remove duplicate entries • Check and delete for any repeated cause-effect	Combinatorial method used to test all possible discrete combinations of parameters. Ideal for situations where interactions between parameters are **unknown**. • Designed to explicitly cover known dependencies. • Aims to identify destructive interactions between parameters that are assumed to be independent or where no dependencies are specified. For each equivalence class, select a representative value. Combine every representative value from one class with every representative from all other classes.

Experience-Based Testing

- Performed by experienced testers, test analysts, and technical test analysts.
- Relies on the tester's experience with software testing and development.
- Examples:
    a. 💚Error guessing:
        i. Experienced testers guess which part of the system might contain errors.

ii. It is not a formal testing (experience based testing) as it is based on the experiences of the experienced testers.
iii. Experienced testers will provide the information before conducting the formal testing.

b. Checklist-based testing
c. Exploratory testing
d. Attack testing

Characteristics of Experience-Based Testing:

- Test cases are derived from the tester's skill, intuition, and past experience with similar applications or technologies.
- Helps identify tests that are not easily discovered by structured testing techniques.

When to Use Experience-Based Testing:

- When requirements and specifications are unavailable or inadequate.
- When there is limited knowledge of the software product.
- When time constraints prevent following a structured approach.

Examples for Equivalence Partitioning, Boundary Value Analysis

A web application is developed to determine the loyalty types of a club member. If the member stays in the club less than 12 months, the member will be awarded as Normal grade, VIP grade will be awarded to whom has become a member between 1 to 3 years. VVIP grade is for those members who stay more than 3 years.

Decide and apply appropriate Black Box Testing techniques to determine the number of test cases. You must show the detailed working.

Equivalence Partitioning

Parameter	Equivalence Partitioning	Representative	Expected Output
Duration, X	vEC1: 0 <= X < 12	10	Normal grade
	vEC2: 12 <= X <= 36	20	VIP grade
	vEC3: 37 <= X <= 120	100	VVIP grade

Parameter	Equivalence Partitioning	Representative	Expected Output
Duration, X	ivEC1: X < 0	-1	Error message:

			Duration cannot be in negative
	ivEC2: X > 120	200	Error message: Duration in exceeded the max allowed years

3 positive test cases (valid data) and 2 negative test cases (invalid data)

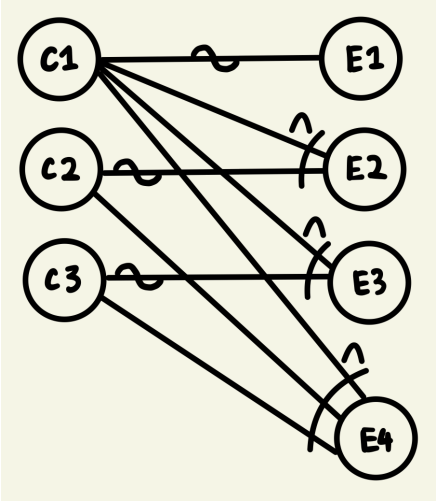
Boundary Value Analysis

Equivalence Partitioning	**Invalid (min-)**	**Valid (min, min+, max-, max)**	**Invalid (max+)**
vEC1: 0 <= X < 12	-1	0, 1, 10, 11	12
vEC2: 12 <= X <= 36	11	12, 13, 35, 36	37
vEC3: 37 <= X <= 120	36	37, 38, 119, 120	121

4 positive test cases (valid data) and 2 negative test cases (invalid data) for each partitioning.

Examples for Cause-Effect Graphing + Decision Table:

Causes	Effects
C1: Valid media file is selected.	E1: Error message "Invalid media file format, size, or resolution."
C2: Caption length is within 0–20 characters.	E2: Error message "Caption length exceeds the limit."
C3: Description length is within 0–1000 characters. Effects	E3: Error message "Description length exceeds the limit."
	E4: Post published successfully.



Conditions	TC1	TC2	TC3	TC4
C1	F	T	T	T
C2	-	F	-	T
C3	-	-	F	T
Actions				
E1	T	-	-	-
E2	-	T	-	-
E3	-	-	T	-
E4	-	-	-	T

Examples for Pairwise Testing:

Example Question:

Scenario: Consider an application with the following inputs:

- **List Box:** 10 elements (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- **Check Box:** Can be either checked or unchecked
- **Radio Button:** Can be either ON or OFF
- **Text Box:** Accepts values between 1 and 100 (only integers)

1. **Exhaustive Combination (Cartesian Product):** To calculate the exhaustive combinations, multiply the possible values of each parameter:
    - **List Box:** 10 possible values (0-9)
    - **Check Box:** 2 possible values (Checked, Unchecked)
    - **Radio Button:** 2 possible values (ON, OFF)
    - **Text Box:** 100 possible values (1-100)

    Total number of test cases using the Cartesian method:
    $10 \times 2 \times 2 \times 100 = 4000$ test cases
    This would include all combinations, including negative test cases.
2. **Reducing the Number of Test Cases (Using Conventional Testing):** Instead of testing every possible combination, you can simplify the test cases:
    - **List Box:** 2 combinations (positive and negative cases, e.g., valid and invalid selection)
    - **Check Box and Radio Button:** 2 combinations (ON/OFF)

	○ **Text Box:** 3 categories (valid integer, invalid integer, and alpha-special character) Total number of test cases using conventional software testing: 2×2×2×3=24 test cases This method reduces the number of test cases by focusing on the categories of inputs. 3. **Pairwise Testing:** Pairwise Testing further reduces the number of test cases by ensuring that all pairs of input values are covered. The technique involves: ○ **Ordering the values**: Start with the parameter having the most values, and order them accordingly. ○ **Verifying coverage**: Ensure that all pairs of values are tested, not higher-order combinations. After applying the Pairwise Testing technique, the number of test cases can be reduced to just **6 test cases**, covering all pairs of inputs. Result: ● **Exhaustive Combination:** 4000 test cases ● **Conventional Testing:** 24 test cases ● **Pairwise Testing:** 6 test cases

C5: White Box Testing

T5Q7. What are the challenges in White Box testing? Using your own words, list and explain four challenges in White Box testing.

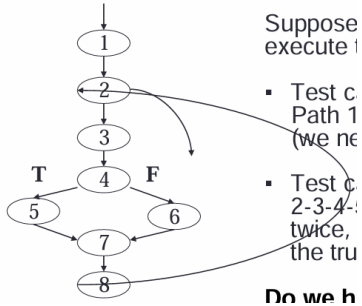
There are requirements of programming languages for understanding the source code.

White box testing can only test on the functionalities while black box testing is able to test on non-functionalities such as usability and performance.

White box testing will never be ended as it is required to execute once the program is updated.

It is time-consuming when converting the source code into control flow graph, calculating the complexity and testing all the functionalities.

Techniques (3)

Control flow / Coverage testing	**Statement/Node Coverage** ● Write test cases to cover ALL NODES in CFG(Control Flow Graph) ○ CFG ■ Representative of a program control flow or a program workflow or a program structure. ■ Translate the code into a graphical representation of a program structure. ● TRUE statement path will execute at least one *FALSE also need* ● Intuition直觉: code that never been executed during testing may contain errors (often is “low-probability” code) ●  Suppose execute ▪ Test c Path 1 (we ne ▪ Test c 2-3-4- twice, the tru **Do we h**（ANS= Test case 1: Path 1-2-3-4-5-7-8-2-3-4-6-7-8-exit） **Branch/Edge Coverage** ● Write test cases that executed EACH BRANCH, every program entry point will taken at least once ○ True and False for each IF-ELSE statement ○ Two branches correspond to the condition of loop ○ ALL alternatives (default) in SWITCH statement **Branch Condition Coverage** ● requires execute both branch and condition coverage (like draw truth table)

```
if A or B then
   s1
else
   s2
end_if_then_else
```

	A	B	Branch
test 1	T	F	true
test 2	F	F	false
test 3	T	T	true
test 4	F	T	true

-

Path Coverage

- Write test cases that execute ALL PROGRAM PATHS at least once

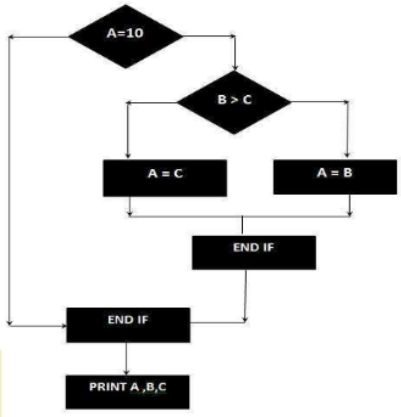
Path Coverage - Strategies to identify useful subsets when Path Coverage Impractical (3)

Basis Paths Coverage: C - Cyclomatic Complexity

Steps to arrive at Cyclomatic Complexity:

1. Draw a corresponding control flow graph
2. Determine Cyclomatic complexity
3. Determine independent paths
4. Prepare test cases

```
IF A = 10 THEN
  IF B > C THEN
      A = B
  ELSE
      A = C
  ENDIF
ENDIF
Print A
Print B
Print C
```



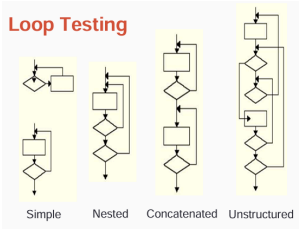
Edges(lines) E: 8
Nodes(shapes) N: 7

cyclomatic complexity: V(G)= E– N +2 --> 8- 7 +2 = 3

Loop Coverage

Dataflow Coverage

Loop testing



https://prnt.sc/lbTH82jBq40r

Ensure each path through the loop is tested at least once. Focus on the validity of loops.

Simple Loops 1. Skip the loop entirely. 2. Single pass through the loop. 3. Two passes through the loop. 4. M passes (where M<N, N = max allowable passes). 5. Test (N - 1),N, (N + 1) passes.	Nested Loops • Start testing from the innermost loop. • Apply simple loop testing to the inner loop. • Work outward until all loops are tested.
Concatenated Loops • Independent loops: Use simple loop testing. • Dependent loops: Treat as nested loops.	Unstructured Loops • Avoid testing; redesign the loop.

Data flow testing

- Tracks the lifecycle of data (variables): creation, usage, destruction.
- Focuses on the flow of data in the program.
- Identifies risky areas by analyzing data usage patterns.
- Uses control flow graphs to detect anomalies.

Data Usage

1. Defined (Def): Variable is created/assigned a value.
    - Examples:
        - int x;
        - x = a + b;
        - scanf(&x, &y);
2. Used (Use): Variable is used for computations or conditions.
    - Examples:
        - A = X + 2;
        - printf("value of x = ", x);
        - if (X < 10)

Testing Approach

- Select paths in the control flow to test the status of data.

	• Ensure: 1. All defined variables are used. 2. All used variables are initialized. • Cover all Def and Use criteria.

C5 – Sample Questions

process block: normal line (declaration, assign, calculation)
decision: switch case, if else, for , while, do while loop...
junction就是 (end switch, end if, end loop那些)

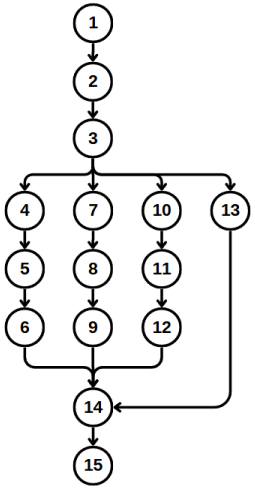
- 可以group 连续的process block画成一个圆圈（包括case里面）
  但是只可以全部group或者全部不group
- CFG no need label switch cases

T5Q8

```
x=0;
choice = input.next.charAt (0); // accept input from keyboard
switch (choice)
{
        case '1' :
                System.out.println("Addition !");
                x = x+1;
                break;
        case '2':
                System.out.println("Subtraction!");
                x = x-1;
                break;
        case '3' :
                System.out.println("Multiplication !");
                x = x * 1;
                break;
        default :
                System.out.println("Please enter a valid choice !");
}
return 0;
```

Control Flow Graph

line 1, 2 = process block
line 3 = decision
line 4-6, 7-9, 10-12, 13 = process block
line 14 = junction
line 15 = process block



Cyclomatic Complexity

V(G) = E - N + 2

V(G) = 10 - 8 + 2

V(G) = 4

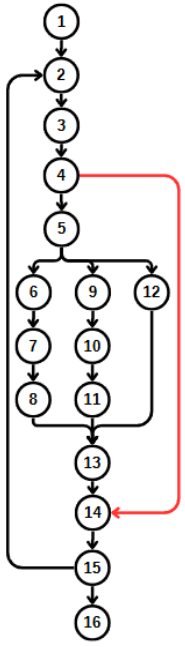
Basis Path. (independent path)

Branch Coverage

```
T5Q9
x=true;
do
{
        gender = input.next.charAt (0); // accept input from keyboard
if(Character.isDigit(gender) != True) //check whether the character is numeric digit
                {
                switch (gender)
                        {
                                case 'M' :
                                        System.out.println("Hello, Gentleman !");
                                        x = false;
                                        break;
                                case 'F' :
                                        System.out.println("Hello, Lady!");
                                        x = false;
                                        break;
                                default :
                                        System.out.println("Please enter a valid gender !");
                        }
        }
}while(x);
return 0;
```

* For Path 1, it would never happen logically, but we have to test it to make sure that it will not happen.

T5Q9



Cyclomatic Complexity

V(G) = E - N + 2

= 19 - 16 + 2

= 5

Basis Path

Path1: 1-2-3-4-5-**6**-7-8-13-14-15-16

Path2: 1-2-3-4-5-**9**-10-11-13-14-15-16

Path3: 1-2-3-4-5-**12**-13-14-15-16

Path4: 1-2-3-4-**14**-15-16

Path5: 1-2-3-4-14-15-**2**-3-4-14-15-16

Branch Coverage

Path1: 1-2-3-4-5-**6**-7-8-13-14-15-16

Path2: 1-2-3-4-5-**9**-10-11-13-14-15-16

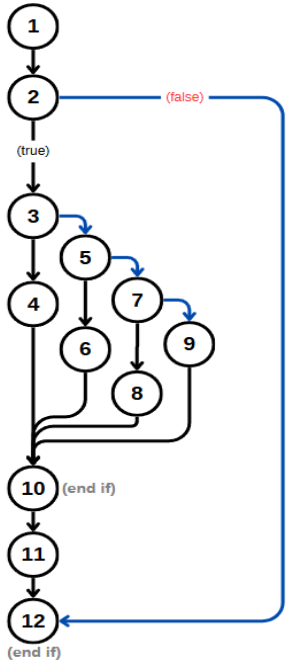
Path3: 1-2-3-4-5-**12**-13-14-15-16

Path4: 1-2-3-4-**14**-15-16

Path5: 1-2-3-4-14-15-**2**-3-4-14-15-16

```
passingMarks = 40;
Scanner input = new Scanner(System.in);
System.out.println("Input marks scored by you");
marksObtained = input.nextInt();
if (marksObtained >= passingMarks)
{
  if (marksObtained > 90)
    grade = 'A';
  else if (marksObtained > 75)
    grade = 'B';
  else if (marksObtained > 60)
    grade = 'C';
  else
    grade = 'D';

  System.out.println("You passed the exam and your grade is " +
}
```



15-12+2=5

List the path(s) of Statement coverage.

Path1: 1-2-3-5-7-9-10-11-12

Path2: 1-2-3-4-10-11-12

Path3: 1-2-3-5-6-10-11-12

Path4: 1-2-3-5-7-8-10-11-12

List the path of Branch Coverage.

Path1: 1-2-12

Path2: 1-2-3-4-10-11-12

Path3: 1-2-3-5-6-10-11-12

Path4: 1-2-3-5-7-8-10-11-12

Path5: 1-2-3-5-7-9-10-11-12

```
int number = 371, originalNumber, remainder, result = 0;

originalNumber = number;

while (originalNumber != 0)
{
    remainder = originalNumber % 10;
    result += Math.pow (remainder, 3);
    originalNumber /= 10;
}

if (result == number)
    System.out.println (number + "is an Armstrong number.");
else
    System.out.println (number + "is not an Armstrong number.");
```

while里面的最后的东西不要接去endwhile，dowhile要

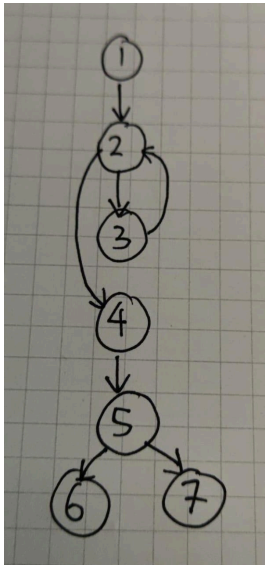
While从endwhile上一个回去

Do-while从endwhile回去，因为condition在最后

(i) Construct a Control Flow Graph (CFG) for the program shown in **Figure 1**. (6 marks)

(ii) Calculate the Cyclomatic Complexity from the CFG created in **Question 2 c) (i)**. (2 marks)

(iii) Provide any **THREE (3)** Basis Paths based on the result in **Question 2 c) (ii)**. (3 marks)

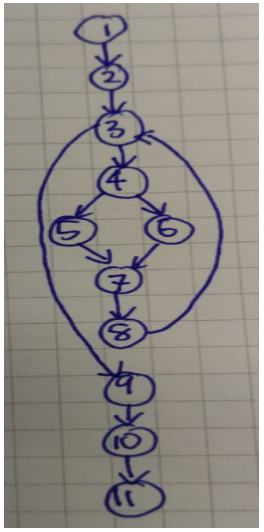


V(G) = E-N+2
= 7-7+2
=2

3 Basis Path
Path1: 1-2-4-5-6
Path2: 12-4-5-7
Path3: 1-2-3-2-4-5-7

2024 midterm

```
1. Start
2. Read X, Y
3. Z = 0
4. While X > 0 do
5.     If Y < 5 then
6.         Z = Z + X
7.     Else
8.         Z = Z + Y
9.     End If
10.    X = X - 1
11. End While
12. Print "Final value of Z:", Z
13. End
```



C6: Software Testing Life Cycle(6) - Part 1

The lecture notes for this few chapters are not so clearly structured based on actual STLC

(before STLC) Independent Test:

Check the errors that are difficult to find by the software developer.

Advantage	Disadvantages
• Reliable test report • Shorter turnaround time in testing • Independent organization have latest and specialize facilities that improve productivity of test cycle. • Test based on clear requirement without any assumption.	• Costly due to contract test. • Need time or difficult to understand the test object/product. • Independent tester may not communicate with developer, this communication gap may impact on software's quality. • May lose sense of responsibility on software quality.

Different Roles in Testing

- Primary Tester
- Secondary Tester
- Subject Matter Expert
- Functional Test Analyst
- Test Lead
- Test Manager

💚1. Requirement Analysis

- First stage in systems engineering process and software development process
- Define quality measure for requirement helps to rationalize unclear requirements
- Consider non-functional requirement like performance, security, etc
- Tester use checklist during requirement phase to verify quality of requirement

Correctness	• Judged based what user want • Can be judged based on standards
Completeness	• Ensure that no necessary element are missing from requirement

Consistency	• Verify that no internal or external contradictions among the element within the work product
Testability	• Verify that requirement is possible to create a test for requirement
Feasibility	• Ensure that it can be implement - given the available budget, schedules, technology and other resources
Traceability	• Identify each requirement in a way that it can be associated with all part of the system where it is used • Ability to review all defined test cases for each requirement specification • Ability to review any defects that have been associated with failed test cases • By reviewing traceability, we can ensure that each requirement has an adequate number (and level) of testing and determine which requirements caused the most defects during testing process

1.2 Requirement Review

- Team review of requirement specification
- Owner distributes requirement specification to all team members for review:
    - Project manager
    - Analyst that created the specification
    - Programmer(s) that perform coding
    - Tester(s) that responsible for testing new requirements
- After reviewing requirements, prepare comments for the review.

2. Test Planning

- Planning entire test efforts and individual test cases
- Define test scope, procedures, test strategy, resources and schedule of test

Test plan: overall planning

Test case: specific steps

💚Test Planning Activities

- Identify:
    - Test items, test levels

💚Consideration Factors When Planning Tests

- Infrastructure for test
    - Identify availability of infrastructure (e.g. testing tools, test harness (driver and stub), procedures, standards)
- Software
    - Can the software be tested?
    - Different test plan will be needed when dealing with software development which is nearly completed or completed.
- Development process

○ Test strategies ○ Test entry and exit criteria ● Manage resources to allocate resources for tests. ● Control, integrate and coordinate test activities (metric to monitor test execution) ● Describe test details (test specification description) ● Manage risk	○ Maturity of development process will affect how we should plan the testing ○ Example: ■ Waterfall model (completed the development of whole system before start testing) ■ Incremental model (deliver modules by modules) ● Human factor ○ What type of testers do we have? (qualification, skills, junior or senior)

2.1 Test Plan

Overview	● Introduce the plan of test project: ● What to test ● General approach taken in test ● Explanation of testing goal, methodologies and objectives ● Architecture of system
Bound	● What to test ● What not to test ● Important terminologies and acronyms ● Define the range of testing using Is/Is Not table ● Purpose: ○ Clarify terminology for those who are not in testing field ○ Ensure all test team member is operating from the same set of operations ○ Prepare Table of Definition ○ Describe where the testing is performed and how the works allocated among multiple organizations using table or diagram
Quality Risk	● Quality risk document ● Cross-reference the test strategy and test environments against risk categories

Transitions *	• System must satisfy a minimal set of qualifications for each test phase before test organization can effectively and efficiently run tests.
	• Specify the essential criteria for beginning and completing test phases
	💚**Test Entry Criteria:**
	What are the criterias we need to prepare or fulfil before we can enter into the testing base to start the testing.
	• This criteria addresses:
	  ○ Availability of necessary documentation, design and requirements information (allow testers to operate system and judge correct behavior)
	  ○ Is the system ready for delivery (which form is suitable for test phase)
	  ○ Availability of supporting utilities, accessories and prerequisites (which form can be used by testers)
	  ○ Is the system at the appropriate level of quality? (some or all of previous test phase has been completed / code review issues have been handled)
	  ○ Passing a smoke test (measure whether the quality is sufficient to enter test phase)
	  ○ Is the test environment ready? (lab, hardware, software, system administration support)
	💚**Test Continuation Criteria:**
	Decide either we are going to continue testing or put the testing on hold until the system is stable and appropriate for testing.
	• Define conditions that must be met during testing process to allow testing to continue effectively and efficiently
	  ○ Test environment must remain stable
	  ○ Manageable bug backlog
	  ○ Tests for the most part unblocked (by large bugs)
	  ○ Installable and stable test releases must be delivered regularly and properly
	  ○ Changes to system under test must be known and controlled
	💚**Test Exit Criteria:**
	The criterias to be met to prove that the testing has been completed.
	• Address the issue of how to determine when testing has been completed
	• Example:
	  ○ All test cases are executed at least once
	  ○ Min 95% of test cases pass
	  ○ Max 5% of test cases fails

Test Configurations and Environments	• Document which hardware, software, networks and lab space used to perform the testing
Test Developments **(3. Test Case Development?)**	• Define all your test cases using the testing approaches and techniques learned • How to design and develop test objects (test cases, test tools, test procedures, test suites, etc)

💚Test Environment
(4. Test Environment Setup?)
	• Plan a test bed (configured or setup environment) for carrying out effective test cases
	• Test environment will mimic the production environment
	• Budget limitations cause difficult to design a test bed with less equipment than used in real life
	
	💚Test Estimation
	• Project manager estimate testing activities, cost and schedule
	• Example:
	  ○ Effort estimation (cost of resources to complete the project)
	  ○ Number of tester required
	  ○ Cost of hiring testers
	  ○ Software cost for installation (software and testing tools)
	  ○ Task dependencies: relationship between two tasks, To do the right estimation in schedule by setting the timeline.
	Task dependency is the relationship between two tasks.
	Help us to do the right estimation in schedule by setting the timeline.
	• **Example:**
	  ○ **Finish-to-Start (task B cannot start until task A finished)**
	  ○ **Start-to-Start**
	  ○ **Finish-to-Finish**
	  ○ **Start-to-Finish**
	
	💚Task Effort Estimation
	• How much work is required to be done
	• 2 components:

	○ Number of test cases created by one person per day (creation) ○ Number of test cases executed by one person per day (execution) ● Other factors: ○ Effort needed to create a test environment (environment) ○ Effort needed to train test engineers on the project (training) ○ Availability of test engineers as when they are needed (availability)
Test Execution **(5. Test Executions?)**	● Define important factors affecting test executions ● Allocate resources and collecting gathered data ● Example: ○ Key participants ○ Test case and bug tracking ○ Bug isolation and classification ○ Test release management ○ Test cycles ○ Test hours 💚**Test Isolation and Classification** ● Requirements failure ○ The bug reported concerns a failure of system to meet its requirements ○ Appropriate party will solve the problem ● Non-requirements failure ○ The bug reported is not covered by system requirements ○ Significantly affects the quality of system in unacceptable ways ○ Appropriate party will solve the problem ● Waiver requested ○ Bugs that will not significantly affect the customers' and users' experiences of quality. ● External failure ○ The bug reported addresses a failure that caused by external factors or factors beyond the control of system under test ● Test failure ○ Defects happen during testing and it is caused by the testers.

	○ Not the development team is solving the problems, the testing team is solving it. ○ Might be due to the testing environment, testing tools, etc.

6. Test Cycle Closure

(content below from: Software Testing Life Cycle (STLC) - GeeksforGeeks)
main activities that take place during the test closure stage include:
Test Summary Report: Summarizes the testing process.
Defect Tracking: Tracks and manages defects until resolution.
Test Environment Clean-Up: Cleans up the test environment. Archives test data and artifacts.
Test Closure Report: Documents testing objectives, scope, schedule, and resources. Summarizes all testing activities.
Knowledge Transfer
Feedback and Improvements

C7: Software Testing life Cycle - Part 2

General Metrics 一般指标

Techniques utilized (either using template, figure or spreadsheet) are depending on:

- Preference of the testers and stakeholders,
- Needs and goals of the project,
- Regulatory requirements,
- Time and money constraints,

In general, the metrics considered include:

- The extent of completion of test environment preparation.
- The extent of test coverage achieved, measured against requirements, risks, codes, configurations and other areas of interest.
- The status of testing (including analysis, design and implementation) compared to various test milestones. . The economics of testing, such as the costs and benefits of continuing test execution in terms of finding the next defect, or running the next test.

Lastly, test summary report (next chapter).

💚Test Monitoring

to track testing works carry out

- Give feedbacks of ongoing test to test team and manager for guide or improve the testing
- Provide visibility of the test result to the project team
- Measure status against test exit criteria to determine whether the test is done
- Estimate future test efforts by gather data
- Small project done by test leader and can be done manually
- Large/complex/distributed project can use automated tools

Test Control

to guide and corrective actions that try to achieve the best possible project outcome

- Projects not always unfold as planned
- When plan and reality diverge, we have to bring it back under control

Test Data Management Process (6)

1. Analysis of data	Thorough analysis of all different kinds of data that may be required in the test case.
2. Data setup to mirror the production environment	Understand the real end user and production scenario Modify (add and remove) data from (1) based on the understanding.
3. Determination of the test data clean up	Deciding which data to be utilized in the current test cycle.

4. Identify sensitive data and protect it	Mechanism to shield sensitive data, especially when replicating real user environments.
5. Automation	Automate the creation of test data. Can be compared with manual data.
6. Effective data refresh using central repository	For future or next test cycle testing purposes.

T7Q1. Compare **Test Plan** with **Test Case**.
Similarity: Software testing related documentation that is prepared before test execution is started.
Test Plan: General documentation of entire planning of a testing (what to test, when to test, how to test, who will be testing the software)
Test Case: A detailed document which describes step by step how the testers should test the test item which has been mentioned in the test plan.

Test Case

Do at beginning of the testing process, we must understand each feature's required input and output and also how to integrate with the application.
It contains a set of test steps that include input, action and expected result.
It must exercise every feature of the application and we compare actual results to expected results at the end of the test.
Is to ensure defects can be found and prevented from being released.

Example of Test Case

Test Case Example1 (simple test)

Test Case #: 2.2	**Test Case Name:** Change PIN	**Page:** 1 of 1
System: ATM	**Subsystem:** PIN	
Designed by: ABC	**Design Date:** 28/11/2004	
Executed by:	**Execution Date:**	
Short Description: Test the ATM Change PIN service		

Pre-conditions
The user has a valid ATM card - The user has accessed the ATM by placing his ATM card in the machine
The current PIN is 1234
The system displays the main menu

Step	Action	Expected System Response	Pass/ Fail	Comment
1	Click the 'Change PIN' button	The system displays a message asking the user to enter the new PIN		
2	Enter '5555'	The system displays a message asking the user to confirm (re-enter) the new PIN		
3	Re-enter '5555'	The system displays a message of successful operation The system asks the user if he wants to perform other operations		
4	Click 'YES' button	The system displays the main menu		
5	Check post-condition 1			

Post-conditions
1. The new PIN '5555' is saved in the database

https://prnt.sc/KEHbf1UhVtH-

💚Characteristics of Good Test Case

a) Accurate:
Exacts the purpose. 明确目的
b) Economical:
No unnecessary steps or words.
c) Traceable:
Capable of being traced to requirements. 能追溯到需求
d) Repeatable:
Can be used to perform the test over and over. 可用于反复测试
e) Reusable:
Can be reused if necessary.

Creation of Test Cases

In **traditional development** testing and the writing of test cases;

- Requirements specifications are finished and the project is code complete before testing begins.

Recent software development methods:

- Require testing and test cases to be defined as the developers complete each part of the application.
- Example: upon adding a new feature to a GUI, the tester would then write and execute test cases to see if it's possible to break the application by using the new feature.

Techniques for Developing Test Cases

- There are many different ideas and techniques used for creating test cases
- Test cases are driven based on the testing techniques discussed in the previous chapter (White box testing, Black box testing, etc.)

Writing test cases

- System specifications and user documentation can provide excellent information for making test cases.
- Later, can write more test cases based on the function and flow of the application (white box)

- Once the requirement specification has been reviewed and approved by the reviewers, the test team should begin developing test cases.
- Once all test cases have been defined for the requirement specification, the tester should distribute the test cases to all team members for review (the same people that attended the requirements review).
- If test scenarios were all about, "What we are going to test" - the test cases are all about "How we are going to test a requirement".
    - Example: if the test scenario is "Validate the Admin login functionality"
    - This would yield in 3 test cases (or conditions)
    - Login (successful), Login-unsuccessful when incorrect username is entered, Login-unsuccessful when incorrect password is entered.

	• Each test case would in turn have steps to address how we can check if a particular test condition is satisfied or not.

Test Case Review

Below are the things that should be reviewed:

💚Positive Test Cases • Using valid data to do the testing. • Do the testing without breaking any rules in a normal environment. • The aim is to test the success of the functionalities. • Passed if it successfully performs the functions based on the valid data.	💚Negative Test Cases • Using invalid data to do the testing. • Conduct the testing by breaking some rules of how the system has been decided. • The aim is to test the error handling of the functionalities. • Passed if it successfully showed the error messages and vice versa.	Performance • Performance test cases ensure that the code will not become unusable with large amounts of data. Security Testing • If the feature is a secured feature, there should be security test cases to ensure that the correct rights are granted before specific actions can occur. Regression Testing

Test Readiness Review (2)

💚To provide management with the assurance that the software has undergone a thorough test process and is ready for turnover to the next test phase.

Scope: • To inspect the test products and test results from the completed test phase for completeness and accuracy. • To verify that the test cases, test scenarios, test scripts, test environment, and test data have been prepared for the next test phase.	💚TRR in Application(Software Company) Level (3) **1. Development Test Readiness Review** • **Informal** test readiness review conducted following successful completion of unit / module testing of a given application. **2. Project Test Readiness Review** • Formal test readiness review conducted following successful completion of the Software Integration Test (SIT) of a given application.

a) Informal review • May or may not include a formal meeting, but are subject to the same reporting requirements for formal reviews. b) Formal review • Formal Test Readiness Reviews will be held for each application of a release at the completion of the Software Integration Test, and at the completion of the Functional Validation Test • Implementation Readiness Review will be held by the Enterprise Test Organization at the completion of the Enterprise Acceptance Test and Operational Assessment.	**3. Enterprise Test Readiness Review** • Formal test readiness review conducted following successful completion of the Functional Validation Test (FVT) of a given application. 💚TRR in Enterprise(Client) Level (2) **1. Acceptance Test Readiness Review** • Formal test readiness review conducted following successful completion of the Enterprise Integration Test and Performance Test of each release. **2. Implementation Test Readiness Review** • Formal readiness review conducted following successful completion of the Enterprise Acceptance Test and portions, if not all of the Operational Assessment at the Enterprise level for a given release.

C8: Software Testing life Cycle - Part 3

Test Data Generation

Process of creating a set of data (random/structured/formatted data, maybe actual data or created artificial data) using a test data generator.
Use to test adequacy of new or revised software applications.

Use real data:	Generate random data:	Select data from a fixed pool:

• Production data that essential for day-to-day operations
• Large amount will not so good for unit/acceptance testing level
• Difficult to express test objective
• Data ends up completely externalized from the test
	• Can be very powerful
	• Large amount will useful for load and performance testing
	• Useful for flushing out edge case errors 用于清除边缘情况错误
	• Live data may look very different than randomly generated
		• Still keep intent close to the test objective
		• Could back this with real data
		  ○ Directly from service or database
		  ○ Fixture backed
		• Need a selection mechanism based on data characteristics.

💚Types/Approaches of Test Data Generator (4)

a) Random Test Data Generators	**b) Goal Oriented Test Data Generators**
• Randomly generate bit streams and convert into input type needed. • Simplest method can be used to test many programs. • Can be used to generate input for any type of program. Disadvantage: does not generate quality data (low coverage) - solely based on probability	• Using CFG to generate input for any path specified instead of just the usual way of generating input from the entry to the exit of the code. • Path selection is blind, can find any input for any path. Disadvantage: May generate infeasible paths (too many paths). Some improved approach in this classification: • Chaining approach (Extension of Goal Oriented) • Assertion oriented approach (Extension of Assertion Oriented) • to find paths effectively and automatically
c) Pathwise Test Data Generators	**d) Intelligent Test Data Generators**
• Similar to the Goal Oriented test data generator, but using a specific path (assigned specific path to follow) instead of giving a choice among many paths.	• Depend on sophisticated analysis of the code to be tested to search for test data. Generate test data quicker than other approaches

• Input needed:
  ○ The program
  ○ The testing criterion (path coverage, statement coverage, etc.)
• Better but harder to generate the needed test data due to greater path knowledge and prediction of coverage..
	Disadvantage: complex and need insights in order to anticipate the different problems.

Test Data Verification

How do we know all the data we use in testing is representative of data we'll receive?

Is there some way we can verify the data we generate?

Builders (to make it easy to create input data/object for unit tests) help here, they can collect all data they construct.

Always create fresh set of data based on testing needs - tester has big responsibility here to verify the correct data!

Test Environment Readiness

When enterprise applications move from development environment to testing environment, test environment readiness and availability will play a major role.

If the test environment is not equipped with required hardware, software, then it will have serious impact on system under test.

T8Q3. Explain the importance of test environment readiness review.

- Test lead or manager check whether all the hardware, software, network and other facilities are prepared using checklist.
- To ensure that all the facilities are setup and working properly before testing.
- Test entry criteria is not achieved and testing cannot be started if test environment readiness review is not executed.

💚Guidelines of Test Environment Readiness (4)

1. Stakeholder requirements collection and addressing:	Multiple teams will be working on different projects, testing and proper management of these stakeholders should happen.
2. Centralized way of managing environment and test data requirements:	Multiple teams would be working in multiple projects and at times, data requirements will be overlapped.
3. Test environment management strategy:	A clear-cut and effective way of managing and providing guidance to teams should be defined. As part of overall monitoring, periodic utilization data should be analyzed so that if a particular environment is not utilized effectively.

4. Multiple types/round/cycle of testing:	Definitely affect the testing environment. Therefore, the need is to identify which data can be shared, which data are exclusive in nature.

Test Deliverables

- Test Deliverables are the artifacts/documentation which are given to the stakeholders of a software project during the software development lifecycle. There are different test deliverables at every phase of the software development lifecycle.
- Some test deliverables are provided before the testing phase, some are provided during the testing phase and some after the testing cycles is over.

T8Q6. Analyse the effects of inappropriate setup of test environment.

- Software and hardware are not stable and are unable to continue testing.
- Time wastage as testing has to be on hold to check the test environment, so testing schedule will be delayed.
- Accuracy of test result will be affected.

C9: Test Management

Defect Management (4)

💚Defect Reporting	• Describe the defects in software, Serves as a communication tool between testers and developers. • Helps developers understand and fix the defect.
Defect Meeting	• Best way to disseminate info among testers, analysts and developer • Meeting is conducted end of every day • Defect are categorized by the type and severity
💚Defect Classification	• Help test planning and prioritization • Classification may vary depending upon requirement of project. 💚**Showstopper (X):** • Impact of the defect is severe. • System cannot be tested without resolving, internal solution may not be available. 💚**Critical (C):** • Impact of the defect is severe, BUT an internal solution is available. 💚**Non-critical (N):** • All defect not in the X and C, to be in N category. Can be solved via documentation or user training
💚Defect Priority	💚High: • further development or testing can't occur until defect repaired 💚Medium: • defect need to solve as soon as possible, it is impairing development and testing. 💚Low: • can be repair after more serious defect fixed

Test Report

💚Test Incident Report	• Defined as a written description of an incident (variation or deviation observed in system behavior from what is expected) observed during a testing.

	• Very often an incident is referred as a defect, but incident is basically any unexpected behavior or response from software that requires investigation IEEE 829 Standard Test Incident Report Template
	IEEE 829 Standard: TEST INCIDENT REPORT
	Test incident report identifier
	Summary
	• Summary of actual incident
	• References to:
	  ○ Test procedure used to discover incident
	  ○ Test logs showing actual execution of cases
	Incident description
	• Detailed explanation of the incident
	• Inputs / Expected results
	• Actual results
	• Anomalies
	• Procedure step
	Impact
	• Report actual damage caused by incident Severity/Priority assessment

Test Summary Report	• Created at a milestone or at the end of a test level. • Describe the results of a given level or phase of testing **IEEE 829 Standard Test Summary Report (8)** 1. Test summary report identifier • Unique ID to identify this summary report and indicate the level of testing. 2. Summary • Identify all relevant support materials so that the reader of the report knows which version and release of the project/software is being reported on. • Identify the specific version of an external package used in the testing, especially if a new release occurred during the test cycle and was not included. • Test items, environment, references 3. Variances • Document any changes or deviations from those areas agreed on in the reference documents.

	● Supporting documents for this deviation / change. 4. Comprehensive assessment ● Evaluation of the testing and test process in terms of the documented test objectives. ● Including: ○ Evaluation of test coverage ■ Total objectives ■ Objectives covered or Objectives omitted ○ Identification of uncovered attributes ■ Surprising trends and test process changes to cover them 5. Summary of results ● Report on the overall status of the incidents. ● Including good and bad incidents: ● Total Incidents ○ By Severity and Priority ○ By cost/failure impact ● Defect Patterns ● Open or Unresolved incidents 6. Evaluations ● An objective assessment of the failure likelihood and overall quality in terms of the criteria specified in the appropriate level test plan. ● Limitations ○ Incomplete or partial functions/features Dropped features (due to requirements change or defects) ● Failure Likelihood ○ High or Medium risk areas ○ Good quality areas or features 7. Summary of activities ● Cover the planned activities and the changes to those plans especially in areas where the amount of actual effort greatly exceeded the planned effort. Including: ● Staff time used: Hours per day/week ● Elapsed time versus staff time: Is staff working excess hours per week ● Costs - planned versus actual

	• Variances and the reasons for the change
	  ○ Changes to the project scope and direction
	  ○ Requirements and design changes
	  ○ Surprising defect trends
	  ○ Loss of personnel (development, test, etc.)
	  ○ Test environment availability and accuracy
	8. Approval
	• Provide a list and signature block for each approving authority. Have to match with the test plan.

💚Configuration Management

- A systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life. Determine the items that make up the software or system.
- Example; Test scripts, data, software, hardware.
- Establish and maintain consistency of product's performance, functional and physical attributes with its requirements, design and operational information.
- Help for storing, tracking and updating all system information daily.

💚Activities involved:

- Identifying and defining the items in the system,
- Controlling the change of these items throughout their lifecycle,
- Recording and reporting the status of items and change requests, and
- Verifying the completeness and correctness of items. Normally done during project planning phase.
- Often accompanied by a test item transmittal report or release notes.

Process of Configuration Management (5)

1. Planning and Management	Plans provide guidelines for the product design, allocation of responsibilities, tools and procedures to be applied. Identify third-party requirements and the audit and review process.
2. Identification	Setting of baselines and identifying the assets' configurations.
3. Control	Maintain configurations control, change requests, and its documentation

4. Status Accounting	Records and report the item configurations of a baseline over the item's life-cycle. Allows for a fast determination of the baseline configuration and changes that have taken place.
5. Verification/Audit	Verify and audit the implemented items, process, controls and maintain positive control of all managed product

Test Item Transmittal Report

Report which identifies test items by their current status and location information to be able to locate items based on specified dates.
IEEE 829 STANDARD: TEST ITEM TRANSMITTAL REPORT TEMPLATE
Transmittal report identifier
Transmitted items
Location
Status
Approvals

Tools Support Testing

Any suitable tool that supports one or more test activities, including;
• Test management
• Test automation (running tests)
• Bug/Defect Tracking (logging defects)
	• Manage requirements, configuration items, incidents, and testware (utilities, software)
	• Provide information on test activities and interface with test execution tools.
	• Maintain traceability between testware.
		Examples;
		• Test Management
		  ○ Requirement Management Tool
		  ○ Configuration Management Tool
		• Test Execution
		  ○ Static Test Tool/Static Analysis Tool Review Support Tool
		  ○ Tools Support Testing
		• Test Design Tool
		• Test Execution Tool
		• Tracking Tool
		  ○ Incident Management Tool
		  ○ Performance and Monitoring Tool

C10: Software Quality Assurance (SQA)

Concept of SQA

SQA:

- A planned and systematic pattern of all actions that provide adequate confidence to conforms established technical requirements for an item or product.
- Activities will be designed to evaluate the process of products developed/ manufactured, contrast with quality control.
- It does not include maintenance, scheduling, or budget concerns.

💚Software Quality Assurance vs. Software Quality Control

- While both have different objectives, Quality Control is a part of Quality Assurance.

Quality Control:	Quality Assurance:
• Focuses on evaluating the quality of a developed product. • Activities aim to identify and withhold products/ software that do not meet standards. • Conducted after development but before delivery to the client. • More to dynamic testing	• Focuses on minimizing quality costs through proactive measures. • Activities prevent, detect, and correct errors early in development. • Aims to reduce the rate of defects or failures before they occur. • More to static testing

Software Quality Factors

McCall's factor model(1977): 11 factors under 3 categories

💚T10Q2. List the structures (categories and factors) for McCall's classic factors model.

- Product operation: the product can be operated
- Product revision: the product has to be taken action such as maintenance
    - Maintainability:
        - Corrective Maintenance
        - Perfective Maintenance
        - Adaptive Maintenance
- Product transition: software need to work with another software
    - When the software needs to work on different operating systems, there will be a transition.
    - Collaborate with third party software like payment gateway in online shopping platform.

Product operation factors (5) - deal with requirements that directly affect daily operation of the software

1. Correctness	• define list of software system's required outputs • deals with multidimensional output specification (output mission, required accuracy, information completeness, is it up-to-date and how its availability, standards for coding and documenting)
2. Reliability	• deals with failures to provide service • determine maximum allowed failure rate (can refer to entire system/separate functions)
3. Efficiency	• deals with hardware resources need to perform all the functions and all other requirements • also deals with recharge time between system's portable units • includes computer processing, data storage, data maintenance capability
4. Integrity	• deals with software system security (authorization and access level requirements) • eg.: prevent access to unauthorized persons: read permit for majority personnel for viewing information, write permit for group allowed to add or modify data.
5. Usability	• deals with scope of staff resources needed to train a new employee to operate the software system

Product revision factors (3) - deal with requirements affect complete range of software maintenance activities 影响整个软件维护活动范围的那些要求

1. Corrective maintenance (correction of software faults and failures),
2. Adaptive maintenance (adapting the current software to additional circumstances and customers without changing the software,) and
3. Perfective maintenance (enhancement and improvement of existing software with respect to locally limited issues).

6. Maintainability	• determines efforts that needed by user and maintenance personnel to identify reason failures, correct failures and verify the success of corrections • Refer to the modular structure of software, the internal program documentation, and the programmer's manual, among other items
7. Flexibility	• Capabilities and efforts required to support adaptive maintenance activities (adapt software to variety customers of the same trade, different extent activities or different range of products) • Also support perfective maintenance activities

8. Testability	• deals with testing and operation of an information system (special features help tester, operation before start system, automatic diagnostic checks applied to detect cause of failures

Product transition factors (3) - adaption of software to other environments and interaction with other systems

9. Portability [developer additional requirements document(3)]	• deals with adaption to other environments (different hardware, os or so forth)
10. Reusability [developer additional requirements document(3)	• deals with utilizing software modules created for one project in the development of a new project.
11. Interoperability互操作性	• Focus on creating interfaces with other software systems or with other equipment firmware • can include software or firmware name(s) which requires interface, output structure accepted as standard in a specific industry or applications.

Other Factors(5)

- **Verifiability [developer additional requirements document(3)**
- Expandability
- Safety
- Manageability
- Survivability

💚SQA System(6)

Combines a wide range of SQA components, all of which are employed to challenge the multitude of sources of software errors and to achieve an acceptable level of software quality.

1. Pre-project components	• assure project commitments have been adequately defined (consider resources required, schedule and budget)
	• assure development and quality plans have been correctly determined
	• improve preparatory steps to initiate works on the project
	  ○ Contact review: detail examination for project proposal draft and contract drafts, development unit committed to functional specification, schedule and budgets
	  ○ Development and quality plan: prepare plan for project and its integrated quality assurance activities

2. Software project life cycle components	• Development life cycle stage: To detect design and programming errors, using review, expert opinion and software testing • Operation-maintenance stage: Specialized maintenance components, applied for functionality improving maintenance tasks

3. Infrastructure components for error prevention and improvement	• Prevention of software faults or lowering software fault rates, together with the improvement of productivity.
	• SQA components here including:
	  ○ Procedures and work instructions
	  ○ Templates and checklists
	  ○ Staff training, retraining, and certification
	  ○ Preventive and corrective actions
	  ○ Configuration management
	  ○ Documentation control

4. Management SQA components	• control of development and maintenance activities that mainly prevent or minimize schedule and budget failures
	• Support the managerial control of software development projects and maintenance services.
	• Control components including:
	  ○ Project progress control (including maintenance contract control)
	  ○ Software quality metrics
	  ○ Software quality costs

5. SQA standards, system certification, and assessment components	• To implement international professional and managerial standards within the organization. • Standards include quality management standards and project process standards
6. Human components	• Human components are the SQA organizational base includes managers, testing personnel, the SQA • unit and practitioners (SQA trustees, SQA committee members and SQA forum members). • to develop and support implementation of SQA components, detect deviations from SQA procedures and • methodology and suggest improvements to SQA components.

SQA System: Who and What Decide It?

a) Organizational Considerations:	b) Project and Maintenance Service Considerations:	c) Professional Staff Considerations:
• The type of software development clientele. • The type of software maintenance clientele. • The range of products. • The size of the organization. • The degree and nature of cooperation with other organizations carrying out related projects. • Optimization objectives	• The level of software complexity and difficulty. • The degree of staff experience with project technology. • The extent of software reuse in new projects	• Professional qualifications • Level of acquaintance with team members.

💚SQA Activities in Software Testing (6)

Plan Testing and SQA Processes	• Test processes should be planned, defined, and documented. • Quality Management Plan: Defines acceptable product quality and how to achieve it. • Test Plan: Describes what, when, how, and who will test. • Test Cases: Actions to verify the expected functionality of features.
Implement Test-Oriented Management	• Extreme Programming (XP): Aims for higher software quality with adaptability. • Test-Driven Development (TDD): Write tests before code, then write code to pass tests, and refactor after. • Pair Programming: Two developers work together on one computer, improving quality and reducing debugging costs.
Ensure Suitable Work Environment for SQA Team	• Involve the SQA team early for professional testing. • Build trust and respect between testers and developers. • Provide business training for the SQA team to enhance team performance. • Ensure good communication between testers and developers to avoid misunderstandings.
Optimize Use of Automated Tests	• Automated testing improves software quality. • Key trends: Increasing test automation and Agile adoption. • Use automation throughout the QA process.
Employ Code Quality Measurements	• Quality objectives should be measurable, documented, and tracked. • Use simple, effective metrics for workflow.

	• Key Aspects of Software Quality:
	  ○ Reliability: Minimize failures and downtime.
	  ○ Performance Efficiency: Resource use, scalability, and response times.
	  ○ Security: Protects information and handles vulnerabilities.
	  ○ Maintainability: Ease of modifying and adapting software.
	• expanded assessment:
	  ○ Rate of Delivery: Frequency of software updates.
	  ○ Testability: Ease of finding faults during testing.
	  ○ Product Usability: Quality of the user interface (UI).

Report Bugs Effectively	• A good bug report improves testing efficiency. • Clearly identify problems and guide engineers to solutions. • Poor reports can cause misunderstandings.

T10Q7. Discuss the benefits of applying SQA activities. *

- Improve software quality
- Customer satisfaction on the end product and customer retention
- Each team member's work would be more standardized.

T10Q8. Discuss the challenges in applying SQA activities. *

- Employees may not adapt to the SQA activities workflow or reject to apply it.
- Lack of knowledge of people who are not familiar with the SQA activities.
- Hire more employees or experts and lead to a higher cost to pay them.
- Time consuming on deploying the hardware and software needed during SQA activities.

C11: Software Quality Metrics

SQM:

- measurements of attributes about software quality along with its process of development
- to facilitate management control by planning and execution of the appropriate managerial interventions (based on calculate deviations of actual functional/ timetable/ budget performance from planned performance)
- to identify situations that require or enable development or maintenance process improvement (use preventive or corrective action)

IEEE Definitions:

1. A quantitative measure of the degree to which an item possesses a given quality attribute.

2. A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute.

Classifications (2)

1- First classification category distinguishes between the life cycle and other phases of the software system.

💚**Process metrics (4)** - related to the software development process

<table>
<tr>
<td>a) Software process quality metrics (2)</td>
<td>
<table>
<tr><td>i. Error density metrics.</td><td>ii. Error severity metrics.</td></tr>
<tr>
<td>

using error counted measures
Common applied: classification of the detected errors into severity classes, followed by weighting each class

<table>
<tr><th>Error severity class</th><th>Relative weight</th></tr>
<tr><td>Low severity</td><td>1</td></tr>
<tr><td>Medium severity</td><td>3</td></tr>
<tr><td>High severity</td><td>9</td></tr>
</table>
</td>
<td>

Used to detect adverse situations of increasing numbers of severe errors in situations where errors and weighted errors, as measured by error density metrics, are generally decreasing

</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>b) Software process timetable metrics</td>
<td>Based on accounts of success (completion of milestones per schedule) in addition to failure events (non-completion per schedule)
Or, calculates the average delay in completion of milestones</td>
</tr>
<tr>
<td>c) Error removal effectiveness metrics</td>
<td>Measure the effectiveness of error removal by the software quality assurance system after a period of regular operation (usually 6 or 12 months) of the system</td>
</tr>
</table>

d) Software process productivity metrics.	"direct" metrics that deal with a project's human resources productivity as well as "indirect" metrics that focus on the extent of software reuse

💚**Product metrics (4) -** related to software maintenance

Refer to the system's operational phase - year of regular use of the software system by "internal" or "external" customers who contracted for development or purchased.

💚Customer Services:

- **Help Desk (HD) services**
  - Based on customer calls
  - Software support by instructing customers
  - Depends on quality of user interface, user manual amd integrated help menus
- **Corrective maintenance services**
  - Based on failures reports
  - Correction of software failures identified by customers or customer service team

a) HD quality metrics	**i. HD calls density密度 metrics** • The extent of customer requests for HD services as measured by the number of calls. **ii. Severity严重性 of HD calls metrics** • Metrics of the severity of the HD issues raised. • detecting one type of adverse不利 situation: increasingly severe HD calls. **iii. Success of the HD services** • The level of success in responding to these calls. • capacity to solve problems raised by customer calls within the time determined in the service contract (availability) • A success is achieved by completing the required service within the time determined in the service contract.
b) HD productivity and effectiveness metrics	**i. HD Productivity Metrics**  **Ii, HD Effectiveness Metrics** • Refer to the resources invested in responding to customers' HD calls.

c) Corrective maintenance quality metrics	Deal with several aspects of the quality of maintenance services. **i. Software system failures density metrics** ● deal with the extent of demand for corrective maintenance ● Measure the number and/or weighted number of failures **ii. Software system failures severity metrics** ● deal with the severity of software system failures. ● Detect adverse situations of increasingly severe failures in the maintained software. **iii. Failures of maintenance services metrics** ● deal with cases where maintenance services were unable to complete on time or failed. ● Related to a software failure problem that was supposed to be solved after a previous call is commonly treated as a maintenance service failure **iv. Software system availability metrics** ● deal with the extent of disturbances where the services of the software system are unavailable or only partly available ● Source for all availability metrics is user failure records: ○ Full availability – where all software system functions perform properly ○ Vital availability – where no vital functions fail (but non-vital functions may fail) ○ Total unavailability – where all software system functions fail
d) Corrective maintenance productivity and effectiveness metrics	● Relates to the resources invested in correction of a single failure. ● A software maintenance system displaying higher productivity will require fewer resources for its maintenance task, while a more effective software maintenance system will require fewer resources, on average, for correcting one failure.

2- Second classification category refers to the subjects of the measurements (project metrics):

- Quality
- Timetable
- Effectiveness (of error removal and maintenance services)
- Productivity

Measurements of System Size (2)

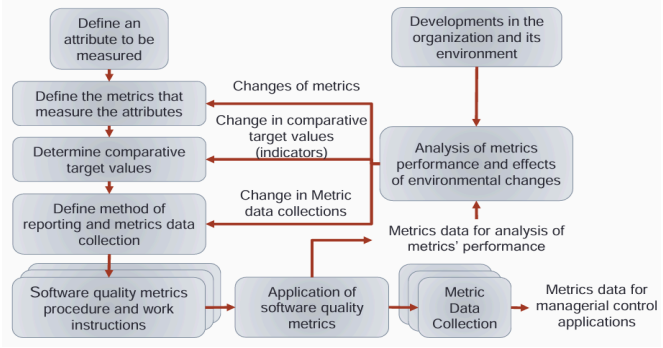
KLOC

- measures software size by number of code lines, specific to programming language or tool used

Function points

- measure of human resources required to development, based on functionality specified for the software system

Process of Defining Software Quality Metrics



https://prnt.sc/R-_JBKtYft5N

Requirements for successful Software Quality Metrics (2)

Applicability of quality metrics is determined by the degree to which the following general and operative requirements are fulfilled:

1. General requirements

- Relevant - measures an attribute of considerable importance
- Valid -measures the required attribute
- Reliable- produces similar results when applied in similar conditions
- Comprehensive - applicable to a large variety of situations
- Mutually exclusive - does not measure attributes already measured by other metrics.

2. Operative requirements

- Easy and simple – data collection is implemented with minimal resources
- Does not require independent data collection metrics data collection is based on currently employed data collection systems, e.g. employee attendance records, cost accounting methods
- Immune to biased interventions by interested parties (team members and others).

Limitations when Apply SQM (3)

a) Budget constraints 预算限制	in allocating the necessary resources (manpower, funds, etc.) for development of a quality metrics system and its regular application.
b) Human factors,	especially opposition反对 of employees to evaluation of their activities.
c) Uncertainty	regarding the data's validity, rooted in partial不完整 and biased偏见 reporting.

Benefits of Apply SQM (6)

1. Reduces the ambiguity歧义 that often surrounds complex and constrained software projects.
2. Helps managers to identify, prioritize, track and communicate project issues at all levels within the organisation.
3. Accurately describe the status of software project processes and products. Representing the quality of associated software products across the project life cycle.
4. Fosters the early discovery and correction of technical and management problems that can be more difficult or costly to resolve later.
5. Help to assess the impacts of decisions objectively and make informed trade-offs to best meet project objectives and to optimize software project and product performance.
6. Provides an effective rationale理由 for selecting the best alternatives.

Total Quality Management (TQM)

Is a holistically focused注重整体 quality management method and considers both corporate objectives and legal requirements at all levels of a company with the participation of all employees.

Also known as total productive maintenance

- Describes a management approach to long-term success through customer satisfaction.
- In a TQM effort, all members of an organization participate in improving processes, products, services, and the culture in which they work.

TQM can be split into two main sections:

Quality Assurance	Quality Control
Mainly concerned with documentation and reporting.It helps in obtaining and maintaining certifications in line with QM standards and regulations.	focused on process measurement and data analysis to guarantee quality.

Principles of TQM (8)

1. Customer-focused The customer ultimately determines the level of quality. No matter what an organization does to foster quality improvement—training employees, integrating quality into the design process, upgrading computers or software, or buying new measuring tools-the customer	**2. Total employee involvement** All employees participate in working toward common goals. Total employee commitment can only be obtained after fear has been driven from the workplace, when empowerment has occurred, and management has provided the proper environment.

determines whether the efforts were worthwhile是否值得.	
3. Process-centered A fundamental part of TQM is a focus on process thinking. A process is a series of steps that take inputs from suppliers (internal or external) and transform them into outputs that are delivered to customers (again, either internal or external). The steps required to carry out the process are defined, and performance measures are continuously monitored in order to detect unexpected variation.	**4. Integrated system** Micro-processes add up to larger processes, and all processes aggregate into the business processes required for defining and implementing strategy. ..... Thus, an integrated system connects business improvement elements in an attempt to continually improve and exceed the expectations of customers, employees, and other stakeholders.
5. Strategic and systematic approach This process, called strategic planning or strategic management, includes the formulation of a strategic plan that integrates quality as a core component. 这个过程称为战略规划或战略管理，包括制定将质量作为核心组成部分的战略计划	**6. Continual improvement** 持续改进 A major thrust of TQM is continual process improvement. Continual improvement drives an organization to be both analytical and creative in finding ways to become more competitive and more effective at meeting stakeholder expectations.
7. Fact-based decision making In order to know how well an organization is performing, data on performance measures are necessary. TQM requires that an organization continually collect and analyze data in order to improve decision making accuracy, achieve consensus, and allow prediction based on past history.	**8. Communications** During times of organizational change, as well as part of day-to-day operation, effective communications plays a large part in maintaining morale and in motivating employees at all levels. Communications involve strategies, method, and timeliness.

Strategy of Implementing TQM (12)

1. Top management learns about and decides to commit to TQM. TQM is identified as one of the organization's strategies.
2. The organization assesses current culture, customer satisfaction, and quality management systems.
3. Top management identifies core values and principles to be used, and communicates them.
4. A TQM master plan is developed on the basis of steps 1, 2, and 3.
5. The organization identifies and prioritizes customer demands and aligns products and services to meet those demands.
6. Management maps the critical processes through which the organization meets its customers' needs.
7. Management oversees the formation of teams for process improvement efforts.

8. The momentum of the TQM effort is managed by the steering committee.
9. Managers contribute individually to the effort through hoshin planning, training, coaching, or other methods.
10. Daily process management and standardization take place.
11. Progress is evaluated and the plan is revised as needed.
12. Constant employee awareness and feedback on status are provided and a reward/recognition process is established.

C11 – Sample Question in tutoirial

T11Q4. A software development department applies two alternative measures, NCE and WCE, to the code errors detected in its software development projects. Three classes of error severity and their relative weights are also defined:

Error Severity Class	Relative Weight
Low Severity	1
Medium Severity	3
High Severity	10

a. Draw the table as discussed in lecture, and calculate NCE and WCE when there are 59 low severity errors, 20 medium severity errors and 11 high severity errors: (density and severity)

Error Severity Class a	Calculation of NCE (number of errors) b	Calculation of WCE	
		Relative Weight c	Weighted errors D = b x c
Low Severity	59	1	59
Medium Severity	20	3	60
High Severity	11	10	110
Total	90		229

b. If KLOC = 40, calculate the CED and WCED.

CED = NCE/KLOC = 90/40 = 2.25 NCE per KLOC
WCED = WCE/KLOC = 229/40 = 5.725 WCE per KLOC

T11- Additional - 1.

(ii) The number of code error and the relative weight for each error severity class are shown in **Table 1** below:

Severity Class	Relative Weight	Number of Code Errors (NCE)
Low	2	250
Medium	5	150
High	11	75

Table 1

By referring to the figures given in the **Table 1**, compute the Total *NCE*, the *Weighted NCE (WCE)* for each severity class, the Total *WCE*, the *Code Error Density (CED)*, and the *Weighted CED (WCED)* if the KLOC is 65. (5 marks)

KLOC=65

Severity Class	Number of Code Errors (NCE)	Relative Weight	Weighted NCE (WCE)
Low	250	2	500
Medium	150	5	750
High	75	11	825
Total	475		2075

Total NCE = 250 + 150 + 75 = 475

WCE

- Low = 250 * 2 = 500
- Medium = 150 * 5 = 750
- High = 75 * 11 = 825

Total WCE = 500 + 750 + 825 = 2075

CED = 475/65 = 7.308

WCED = 2075/65 = 31.923

Average Severity of Code Errors (ASCE) = WCE/NCE = 2075/475 = 4.368

2. Calculate the **Development Errors Removal Effectiveness** (DERE) if the **total number of development errors** is 65 and **Total Number of Failure Detected** in a software of a year is 300.

DERE = NDE / (NDE + NYF)
= 65/(65+300) = 0.178

3. The following table shows the development milestone of a software.

Milestone	Hours of working
M1	240
M2	360
M3	160
M4	200
M5	60

DevH = 240+360+160+200+60 = 1020

Development Productivity (DevP) = DevH/KLOC
= 1020/250 = 4.08

ReKLOC = 0.2*250 = 50

Assumed that the program size is 250 KLOC, and 20% of the codes are reused from previous project. The new project has produced 580 pages of documents and 120 pages are reused from previous project. **Calculate DevP, Cre, DocRe**.	Code Reuse (CRe) = ReKLOC/KLOC = 50/250 = 0.2 ReDoc = 120 Documentation Reuse (DocRe) = 120/580 = 0.207
4. An online system is expected to operate 24 hours a year (365 days). However, the total time of the system that is not able to provide some services is 144 hours. From the 144 hours, 20 hours of total time where the system is not able to perform important e-commerce activities. In addition, a total of 15 hours that the system is totally failed. **Calculate Full Availability (FA), Vital Availability (VitA) and Total Unavailability (TUA) of the system**.	Number of hours software system is in service during one year (NYSerH) = 24 * 365 = 8760 Number of hours where at least one function is unavailable during one year (NYFH) = 144 **Full Availability =** (NYSerH - NYFH)/NYSerH = ((365 * 24) - 144) / (365 * 24) = 0.984 Num of hours when at least one vital function is unavailable during one year(NYVitFH) = 20 **Vital Availability (VitA) =** (NYSerH - NYVitFH) / NYSerH = (8760 - 20) / 8760 = 0.998 Number of hours of total failure (all system functions failed) during one year (NYTFH) = 15 **Total Unavailability (TUA) =** NYTFH / NYSerH = 15 / 8760 = 0.002

C11–PYQ

Question 3b)
In developing the proposed software as mentioned in Question 1 c), Warrior Tech Services applies software quality metrics in measuring the quality of their software product, software process, and software project.
(i)Warrior Tech Services uses software process development metrics for measuring the density of the code errors and its severity in the proposed mobile application. Table 1 shows the number of code errors and the relative weight for each error severity class.

Table 1

Severity Class	Relative Weight	Number of Code Error (NCE)
Low	3	250
Medium	6	100
High	9	50

i)

Severity Class	NCE	Relative Weight	WCE
Low	250	3	250*3=750
Medium	100	6	100*6=600
Hight	50	9	50*9=450
Total:	400		1800

KLOC=75
CED

Referring to the figures given in Table 1, calculate the Total NCE, the Weighted NCE (WCE) for each severity class, the Total WCE, the Code Error Density (CED), the Weighted CED (WCED), and the Average Severity of CED (ASCE) if the thousands of lines of code (KLOC) is 75. (7 marks) (ii) Table 2 shows the list of development milestones of the software proposed by Toyota Motor Corp. The project has produced 650 pages of documents and 145 pages are reused from a recent previous similar project. Out of 75 KLOC, 35% of the codes are reused from few previous projects. Calculate Development Productivity (DevP), Code Reuse (Cre), and Documentation Reuse (DocRe). (6 marks) **Table 2** Milestone / Hours of working M1 65 M2 73 M3 60 M4 135 M5 88	=400/75 =5.3333 **WCED** =1800/75 =24 **ASCE** =1800/400 =4.5 **ii)** 65+73+60+135+88=421 **Development Productivity (DevP)** 421/75 = 5.6133 **Code Reuse (Cre)** 0.35*75=26.25 26.25/75 = 0.35 **Documentation Reuse (DocRe)** 145/650 = 0.2231
Question 4a) Toyota Motor Corp expected that their own software as mentioned in Question 1 c) is to operate 24 hours daily in a year of services, however the total time of the software that not able to provide some functionalities is 155 hours. From the total hours, 45 hours of total time where the system not able to perform its vital functionalities. Additionally, for a total of 23 hours the software is totally failed. Calculate Full Availability (FA), Vital Availability (VitA), and Total Unavailability (TUA) of the software.(5 marks)	24*365=8760 **Full Availability (FA)** (8760-155) / 8760 = 0.9823 **Vital Availability (VitA)** (8760-45)/8760 = 0.9949 **Total Unavailability (TUA)** 23/8760 = 0.0026

C12: Software Quality Assurance Standard

Consists of monitoring the Software Engineering process/methods used to ensure quality.

This process/method must be ensured to follow one or more standards. Examples: ISO 9000, CMM/CMMI

Benefits of SQA standard

- The ability to apply software development and maintenance methodologies and procedures of the highest professional level.
- Better mutual understanding and coordination among development teams but especially between development and maintenance teams.
- Greater cooperation between the software developer and external participants in the project.
- Better understanding and cooperation between suppliers and customers, based on the adoption of known development and maintenance standards as part of the contract.

Organization Involved in SQA standard

- IEEE (Institute of Electrical and Electronics Engineers) Computer Society
- ISO (International Organization for Standardization) DOD (US Department of Defense)
- ANSI (American National Standards Institute)
- IEC (International Electrotechnical Commission)
- EIA (Electronic Industries Association).

Classifications of SQA standard

1. Quality management standards

- Focus on the organization's SQA system, infrastructure and requirements, while leaving the choice of methods and tools to the organization. (What process to perform)
- By complying with quality management standards, organizations can steadily assure that their software products achieve an acceptable level of quality.
- Includes certification and assessment methodologies
- Examples: ISO 9000-3, Capability Maturity Model (CMM)

2. Project process standards

- Focus on the methodologies for carrying out software development and maintenance projects. (How the process to be performed).
- Define the steps to be taken, design documentation requirements, the contents of design documents, design reviews and review issues, software testing to be performed and testing topics, and so forth.
- Many SQA standards in this class can serve as software engineering standards and vice versa.
- Examples: IEEE Std 1012-1998, ISO/IEC 12207, Capability Maturity Model Integration (CMMI)

Quality Management Standard (2)

Certification Standard (organization)

Objectives:

- Enable a software development organization to demonstrate consistent ability to assure that its software products or maintenance services comply with acceptable quality requirements. This is achieved by certification granted by an external body.

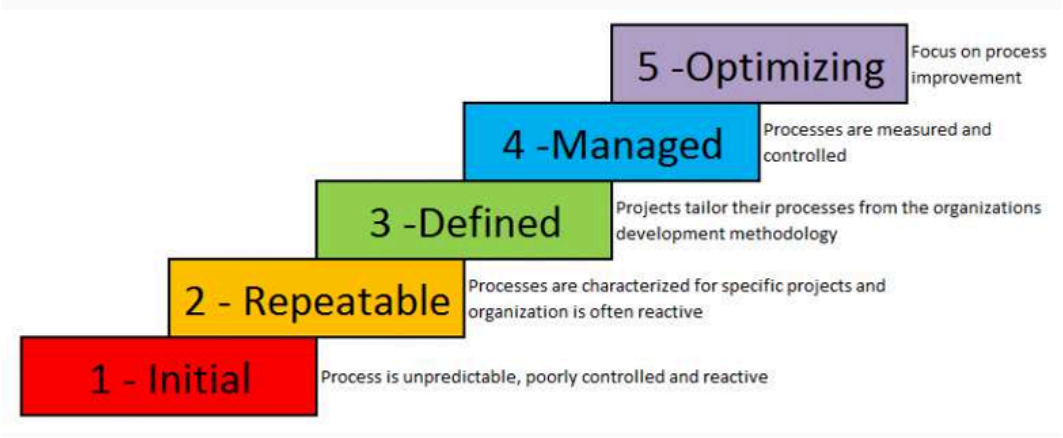
Assessment Standard (individual in organization)

Objectives:

- Serve software development and maintenance organizations as a tool for self-assessment of their ability to carry out software development projects.

• Serve as an agreed basis for customer and supplier evaluation of the supplier's quality management system. This may be accomplished by customer performance of a quality audit of the supplier's quality management system. The audit will be based on the certification standard's requirements. • Support the software development organization's efforts to improve quality management system performance and enhance customer satisfaction through compliance with the standard's requirements.	• Serve as a tool for improvement of development and maintenance processes. The standard indicates directions for process improvements. • Help purchasing organizations determine the capabilities of potential suppliers. • Guide training of assessors by delineating qualifications and training program curricula

Capability Maturity Model – CMM (5)

	• CMM was developed and promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DOD. • CMM is a methodology used to develop and refine an organization's software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes. • CMM establishes a framework for continuous process improvement. It covers practices for planning, engineering, and managing software development and maintenance.

1. Initial:

Processes are unpredictable, disorganized, ad hoc and even confused. Depends on individual efforts and is not considered to be repeatable. This is because processes are not sufficiently defined and documented to enable them to be replicated.

2. Repeatable:

Requisite processes are characterized, established, defined and documented. As a result, basic project management techniques are established, and successes in key process areas are able to be repeated.

3. Defined:

Organization develops its own standard software development process (methodology). These defined processes enable greater attention to documentation, standardization and integration.

4. Managed:

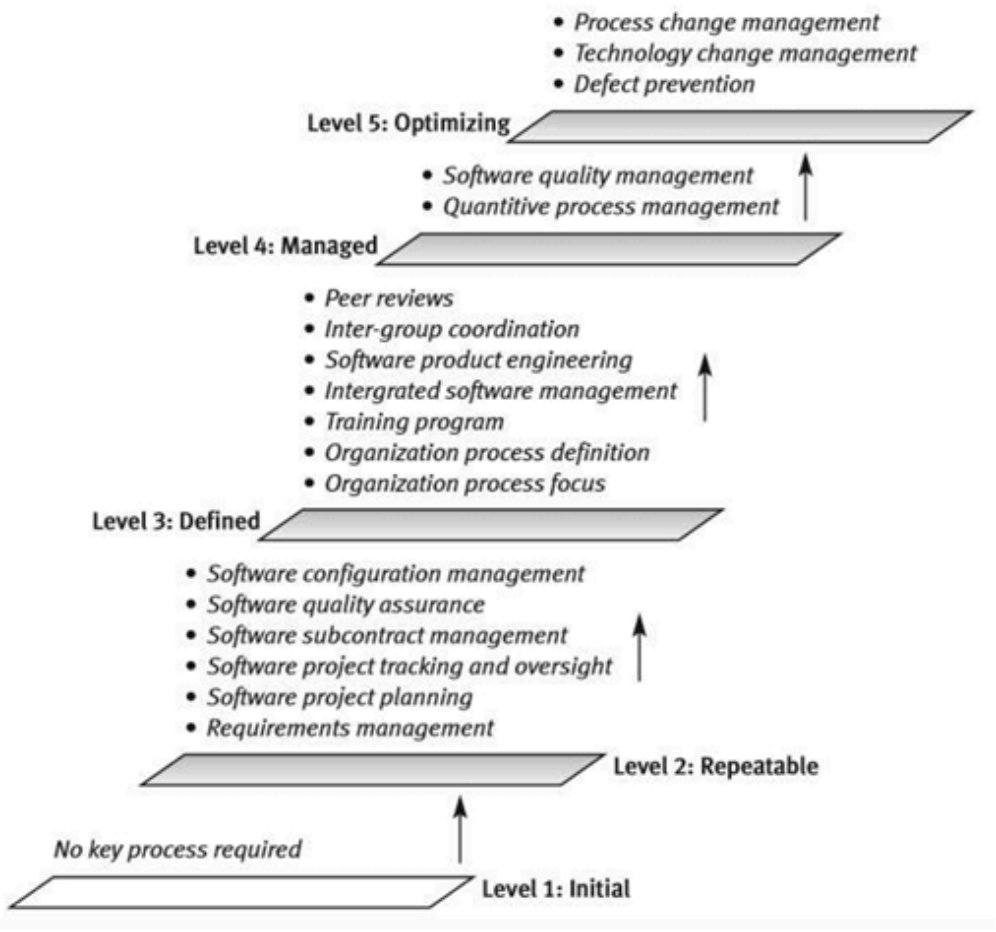
Organization monitors and controls its own processes through data collection and analysis.

5. Optimizing:

Processes are constantly improved through monitoring feedback from processes and introducing innovative processes and functionality.

Principle	Evolution
Application of more elaborate management methods based on quantitative approaches increases the organization's capability to control the quality and improve the productivity of the software development process. The vehicle for enhancement of software development is composed of the five-level capability maturity model. The model enables an organization to evaluate its achievements and determine the efforts needed to reach the next capability level by locating the process areas requiring improvement. Process areas are generic; they define the "what", not the "how". This approach enables the model to be applied to a wide range of implementation organizations because: It allows use of any life cycle model It allows use of any design methodology, software development tool and programming language It does not specify any particular documentation standard.	1. System Engineering CMM (SE-CMM) focuses on system engineering practices related to product-oriented customer requirements. It deals with product development: analysis of requirements, design of product systems, management and coordination of the product systems and their integration. In addition, it deals with the production of the developed product: planning production lines and their operation. 2. Trusted CMM (T-CMM) was developed to serve sensitive and classified software systems that require enhanced software quality assurance. 3. System Security Engineering CMM (SSE-CMM) focuses on security aspects of software engineering and deals with secured product secured product development processes, including security of development team members. 4. People CMM (P-CMM) deals with human resource development in software organizations: improvement of professional capacities, motivation, organizational structure, etc. 5. Software Acquisition CMM (SA-CMM) focuses on special aspects of software acquisition by treating issues contract tracking, acquisition risk management, quantitative acquisition management, contract performance management, etc. - that touch on software purchased from external organizations. 6. Integrated Product Development CMM (IPD-CMM) serves as a framework for integration of development efforts related to every aspect of the product throughout the product life cycle as invested by each department

The CMM model levels and key process areas (KPAs)	💚Disadvantages of CMM? • Lose perspective and forget that the real goal. Only focuses on the processes to be achieved. They don't consider whether the quality is implemented or not. So, it does not actually improve the process's quality. • Does not specify a particular way of achieving the goals.

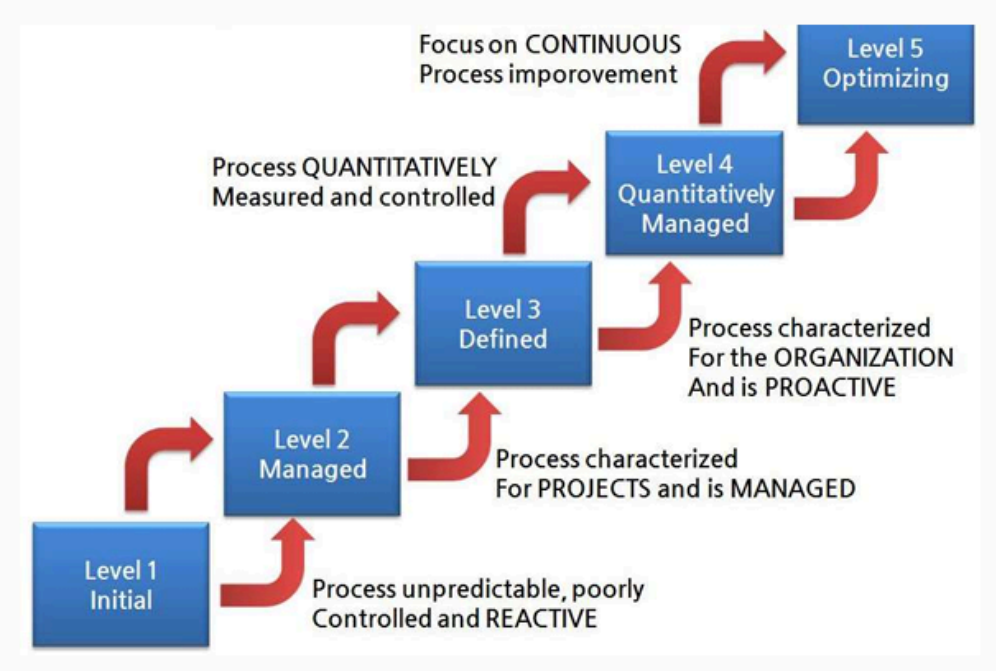


- Since the organization does not know the right way to do, CMM cannot be used for emergency fixing between the development of the project. Only suitable during the beginning of the project.
- 18 key process areas only focus on the improvement of management related activities but not with the context of software.

Capability Maturity Model Integration - CMMI (5)

💚How CMMI overcome Disadvantages of CMM

CMMI provides guidance on how to do it with the list of the processes to be done.
Clearly defined procedures on how to implement the processes.
Can improve the quality of the processes in the right way.

<table>
<tr>
<td> 
</td>
<td>In the first version, CMMI was tailored to software engineering. Following versions, it became more abstract and generalized (applied to hardware, software, and service development across every industry).
The CMMI capability levels are the same as CMM, apart from a minor change related to capability level 2 and level 4, namely:
Capability maturity level 1: Initial
Capability maturity level 2: Managed
Capability maturity level 3: Defined
Capability maturity level 4: Quantitatively managed
Capability maturity level 5: Optimizing

A substantial change has nonetheless evolved with respect to the processes included in the models. The 18 key process areas of CMM (frequently referred to as KPAs) were replaced by 25 process areas (PAs).
The PAs are classified by the capability maturity level that the organization is required to successfully perform. For each process area; objectives, specific practices and procedures are defined.</td>
</tr>
<tr>
<td>CMMI Process Areas
What are the CMMI Process Areas (PA) ?</td>
<td>Level 5: Optimizing
Organizational Innovation and Deployment (OID) - Process Management
Causal Analysis and Resolution (CAR) - Support
Level 4: Quantitatively Managed
Organizational Process Performance (OPP) - Process Management
Quantitative Project Management (QPM) - Project Management
Level 3: Defined
Engineering
Requirements Development (RD)
Technical Solution (TS)
Product Integration (PI)
Verification (VER)
Validation (VAL)
Process Management</td>
</tr>
</table>

CMMI[SM] Staged Representation			
Level	**Process Areas**		**Category**
5 Optimizing	Organizational Innovation and Deployment	OID	Process Mgt
	Causal Analysis and Resolution	CAR	Support
4 Quantitatively Managed	Organizational Process Performance	OPP	Process Mgt
	Quantitative Project Management	QPM	Project Mgt
3 Defined	Requirements Development	RD	Engineering
	Technical Solution	TS	Engineering
	Product Integration	PI	Engineering
	Verification	VER	Engineering
	Validation	VAL	Engineering
	Organizational Process Focus	OPF	Process Mgt
	Organizational Process Definition	OPD	Process Mgt
	Organizational Training	OT	Process Mgt
	Integrated Project Management	IPM	Project Mgt
	Risk Management	RSKM	Project Mgt
	Decision Analysis and Resolution	DAR	Support
	Integrated Supplier Management	ISM	Project Mgt
	Organizational Environment for Integration	OEI	Support
	Integrated Teaming	IT	Project Mgt
2 Managed	Requirements Management	REQM	Engineering
	Project Planning	PP	Project Mgt
	Project Monitoring and Control	PMC	Project Mgt
	Supplier Agreement Management	SAM	Project Mgt
	Measurement and Analysis	MA	Support
	Process and Prduct Quality Assurance	PPQA	Support
	Configuration Management	CM	Support
1 Initial			

https://prnt.sc/5z02JrPtIm5d

Organizational Process Focus (OPF)
Organizational Process Definition (OPD)
Organizational Training (OT)
Project Management
Integrated Project Management (IPM)
Risk Management (RSKM)
Integrated Supplier Management (ISM)
Integrated Teaming (IT)
Support
Decision Analysis and Resolution (DAR)
Organizational Environment for Integration (OEI)
Level 2: Managed
Project Management
Project Planning (PP)
Project Monitoring and Control (PMC)
Level 1: Initial
Support
Configuration Management (CM)
Measurement and Analysis (MA)
Process and Product Quality Assurance (PPQA)
Project Management
Supplier Agreement Management (SAM)