CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test - III

Sub:	Software Testing							Code:	10CS842
Date:	27/ 05 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	8-A,B	Branch:	CSE

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	Explain Test & Analysis strategies of Clean room process model	[10]	CO3	L3
2	Explain how to improve the process & ODC, classification of triggers, customer impact, defect types	[10]	CO2	L3
3 (a)	Explain Capture & Replay	[7]	CO2	L3
3 (b)	Define ODC, Root cross analysis	[3]	CO2	L1
4	Explain Test & Analysis strategies of SRET	[10]	CO3	L3
5 (a)	Explain Extreme programming strategy	[6]	CO3	L3
5 (b)	Define critical path, critical dependence	[4]	CO3	L3
6	Explain Risk planning	[10]	CO3	L3
7	Write the functional test specification for check configuration	[10]	CO3	L3
8	How documents are being organized? Give the sample naming conversion, compliant with IEEE standards	[10]	CO4	L3

Course Name / Code	Software Testing / 10CS842

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C842.1	Describe the Terminology & levels of testing	1	2	1	0	0	0	0	1	0	1	1	0
C842.1	Write the Test Document, Scenario, Case, Plan	1	1	2	1	0	0	0	1	2	1	0	0
C842.1	Explain the Software testing process, Techniques with examples	1	2	1	1	0	0	0	1	0	1	0	0
C842.1	Describe the process framework-Validation, Verification and Basic principles	1	1	1	1	0	0	0	1	0	1	0	0
C842.1	Explain the process by using Testing tools	1	1	1	1	2	0	0	1	1	1	2	0
C842.1	Demonstrate the process improvement in software testing	1	1	1	2	0	0	0	1	1	3	2	0

Revised Bloom's Taxonomy (RBT)	
Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

Programme Outcome
PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

1.Cleanroom process model

Clean room process model:

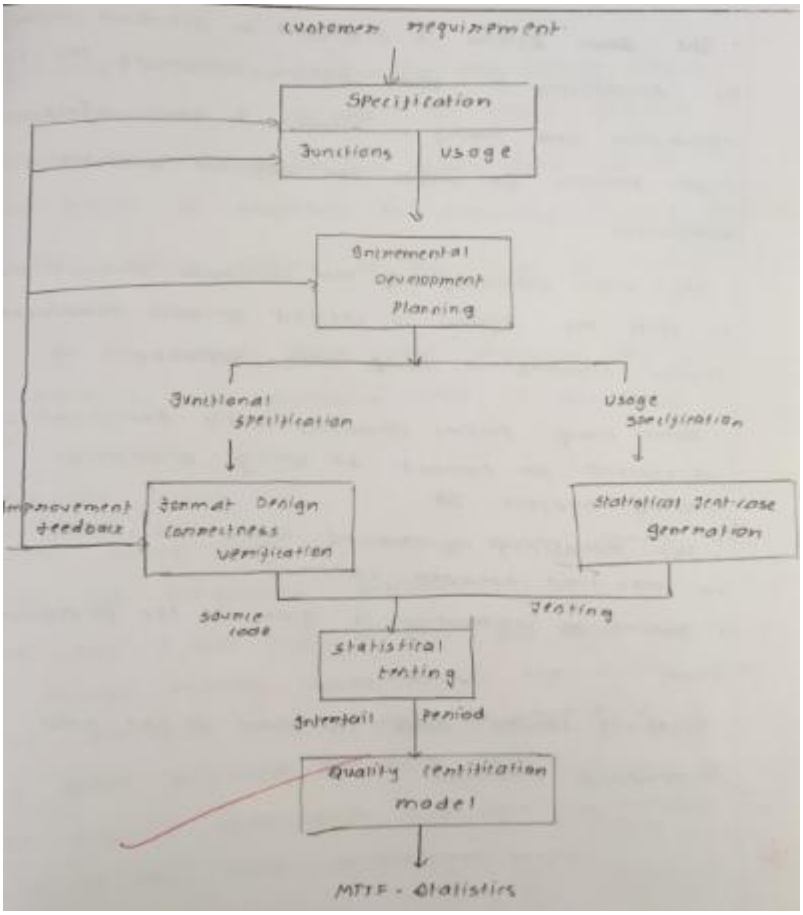
The clean room process model was introduced by IBM in early 1980's. Pairs development with V and V activities and stresses on analysis than testing in the early phases.

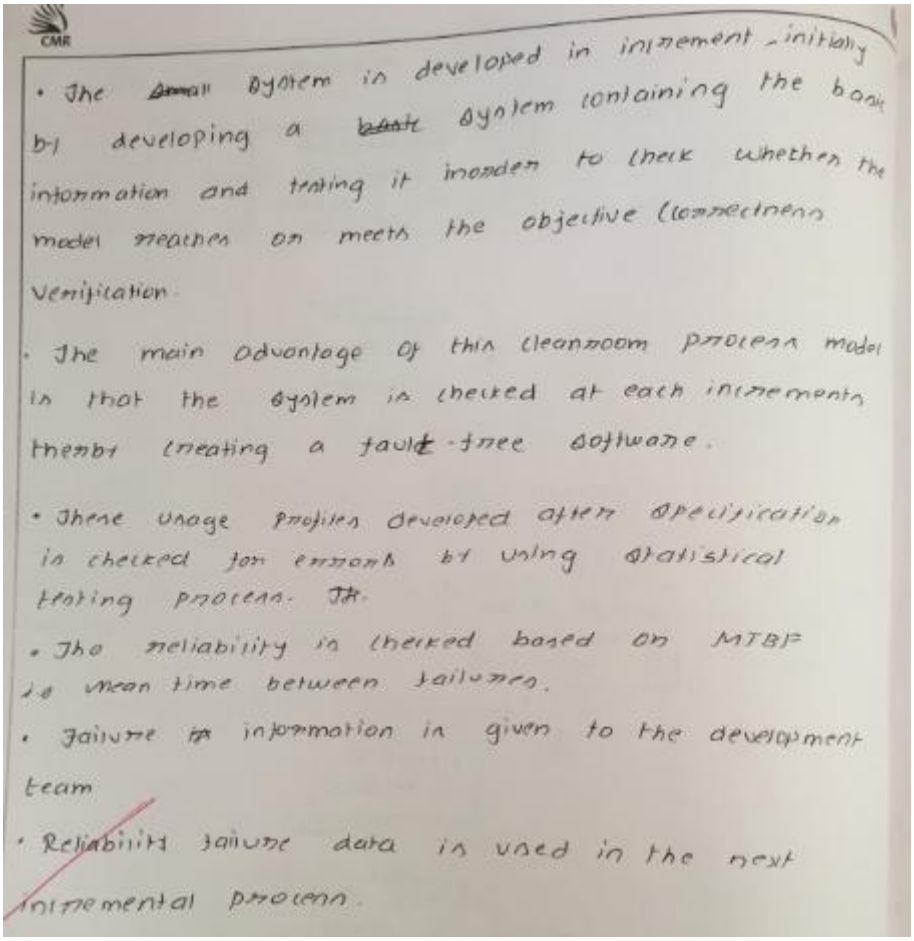
Testing is left for system certification. The cleanroom process model mainly involves two main cooperation team ie the development team and the quality team

It involves mainly on 5 activities like.

- Specifications
- Planning
- Design and verification
- Quality certification
- Feedback.

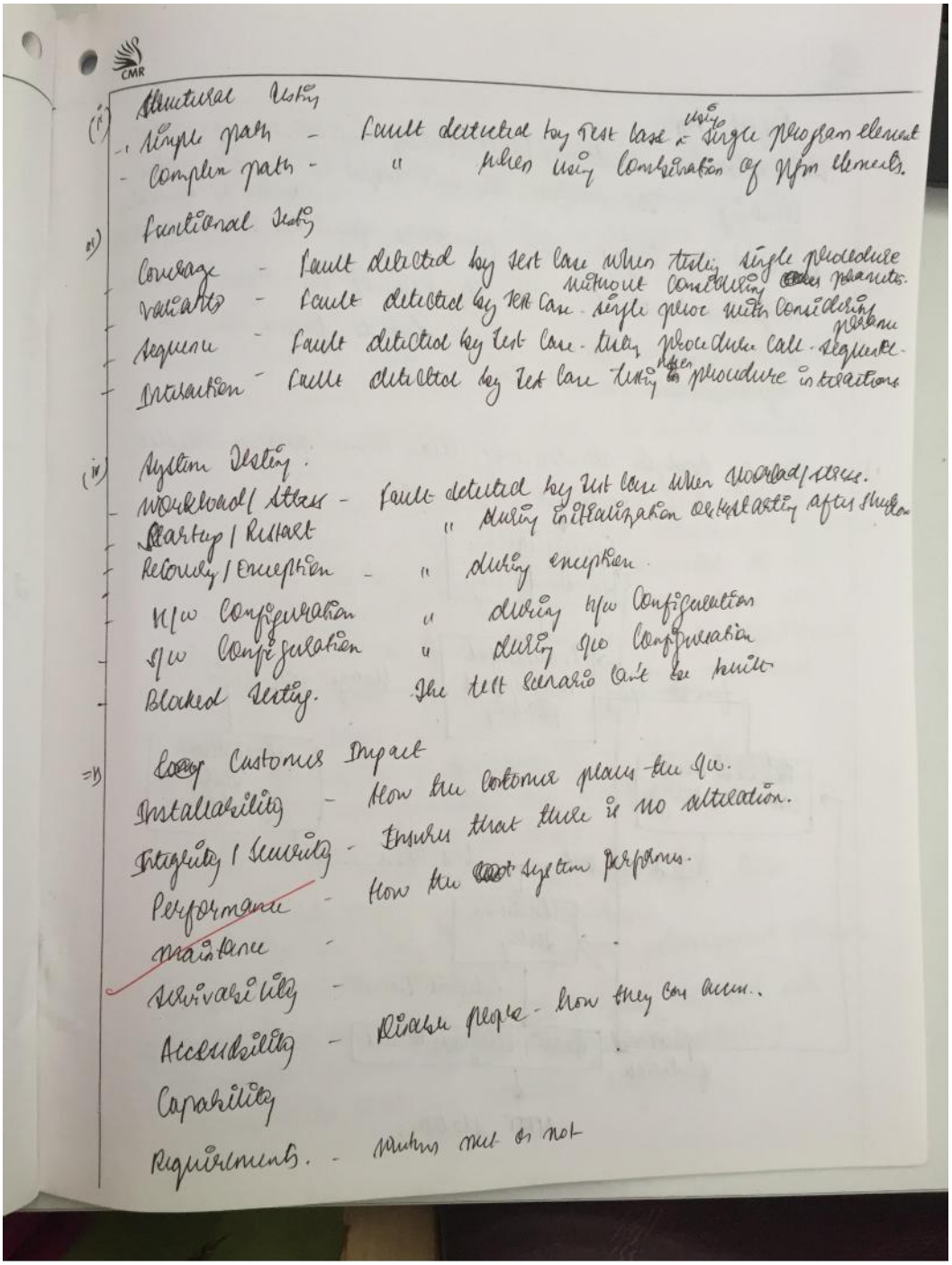
- In the specification activity, the development team defines and design the different behaviours of the system that should exhibit
- The quality team defines the usage scenarios that will be derived during the development of the entire system.
- The planning process mainly involves the certification method and the incremental develop





- The system is developed in increment - initially by developing a basic system containing the basic information and testing it in order to check whether the model matches or meets the objective (correctness verification.
- The main advantage of this cleanroom process model is that the system is checked at each increments thereby creating a fault-free software.
- These usage profiles developed after specification is checked for errors by using statistical testing process. It.
- The reliability is checked based on MTBF= i.e mean time between failures.
- Failure information is given to the development team
- Reliability failure data is used in the next incremental process.

2. ODC classification



- In the second situation - i.e fault due to inexperience can be overcomed by focused training.

ODC - Orthogonal Defect classification.

- This is helpful in improving the monitoring process, specially in large projects. in which the detailed summarization of the process is not possible.

This is classified into two - fault classification and fault Analysis.

The fault classification - done when fault is detected and when fault is fixed.

ODC classification → Reuse code testing, Structural functional & system testing.

(i) Reuse code testing.

- Design
- Logic / flow
- Backward Compatibility
- Internal document
- Natural compatibility
- Concurrency
- Side effect
- Logical dependency
- Rare situation.

- In the second situation - i.e fault due to inexperience can be overcomed by focused training.

ODC - Orthogonal Defect classification.

- This is helpful in improving the monitoring process, specially in large projects. in which the detailed summarization of the process is not possible.

This is classified into two - fault classification and fault Analysis.

The fault classification - done when fault is detected and when fault is fixed.

ODC classification → Reuse code Testing, Structural functional & System Testing.

(i) Reuse code Testing.

- Design
- Logic / flow
- Backward Compatibility
- Internal document
- Natural compatibility
- Concurrency
- Side effect
- Logical dependency
- Rare situation.

Defect Types

Initialization - Defect / fault during initialization of variable

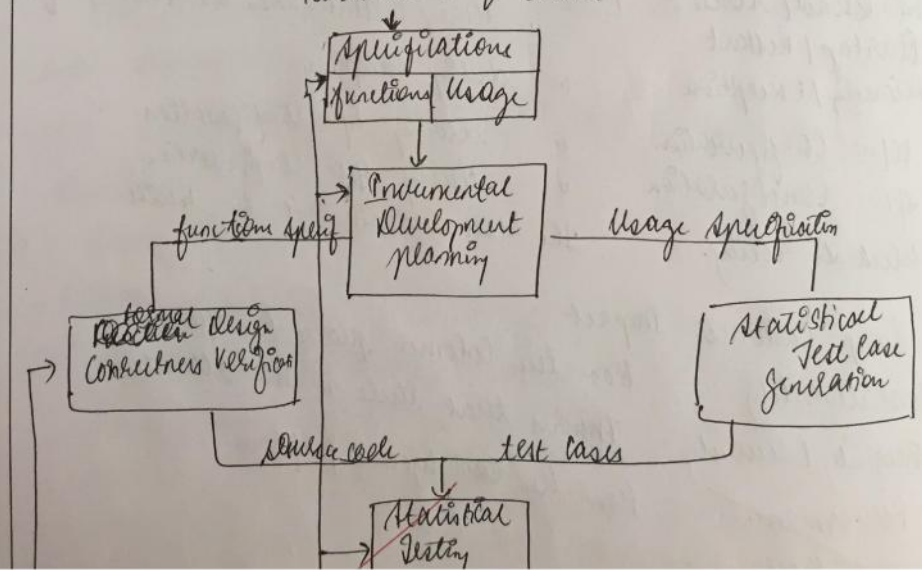
Checking - The variable are initialized to the proper value or not is checked.

Function / Class / Object - The procedure, functions solution approp

Algorithm / method - The efficiency of the algorithm checked

Timing / Synchronization - fault due to proper timing of synchronization is not done

Test by Analysis strategies of Clean Room process Model



SRET :

. Software reliability Engineered ~~Testing~~ Testing (SRET) approach developed by M and T in 1990's, assumes a spiral development model in which each coil of the spiral model is subjected to rigurous testing process

. SRET mainly consists of two types of testing i.e

*Development Testing -

~~In~~ this method, finds and removes the faults on errors in the software that is developed in-house.

* Certification Testing -

This method ~~either~~ of testing either accepts or rejects the outsourced software.

. There are 7 main steps in SRET. The initial two, the Quick decision-making which decides the system that requires seperate testing and ~~the~~ what kind of testing should be made for each system.

. There are 5 core steps that ~~are~~ run parallel in each coil of the spiral development model.

• The ~~small~~ system is developed in increment - initially by developing a ~~basic~~ system containing the basic information and testing it inorder to check whether the model reaches or meets the objective (correctness Verification.

• The main advantage of this cleanroom process model is that the system is checked at each increments thereby creating a fault-free software.

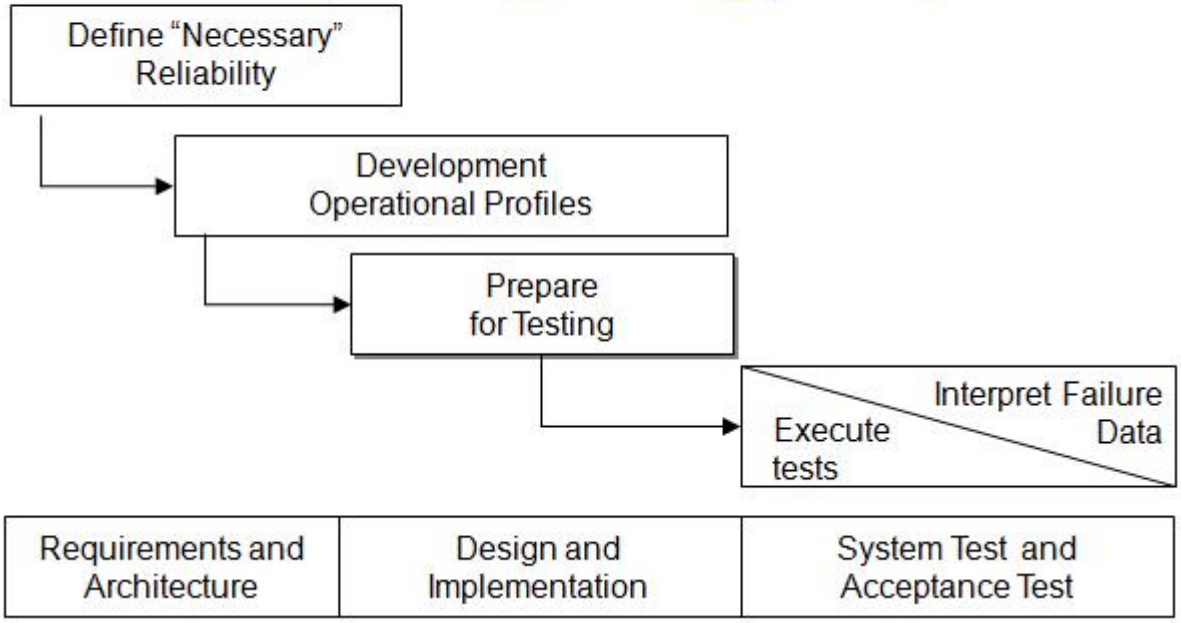
• These usage profiles developed after specification is checked for errors by using statistical testing process. IR.

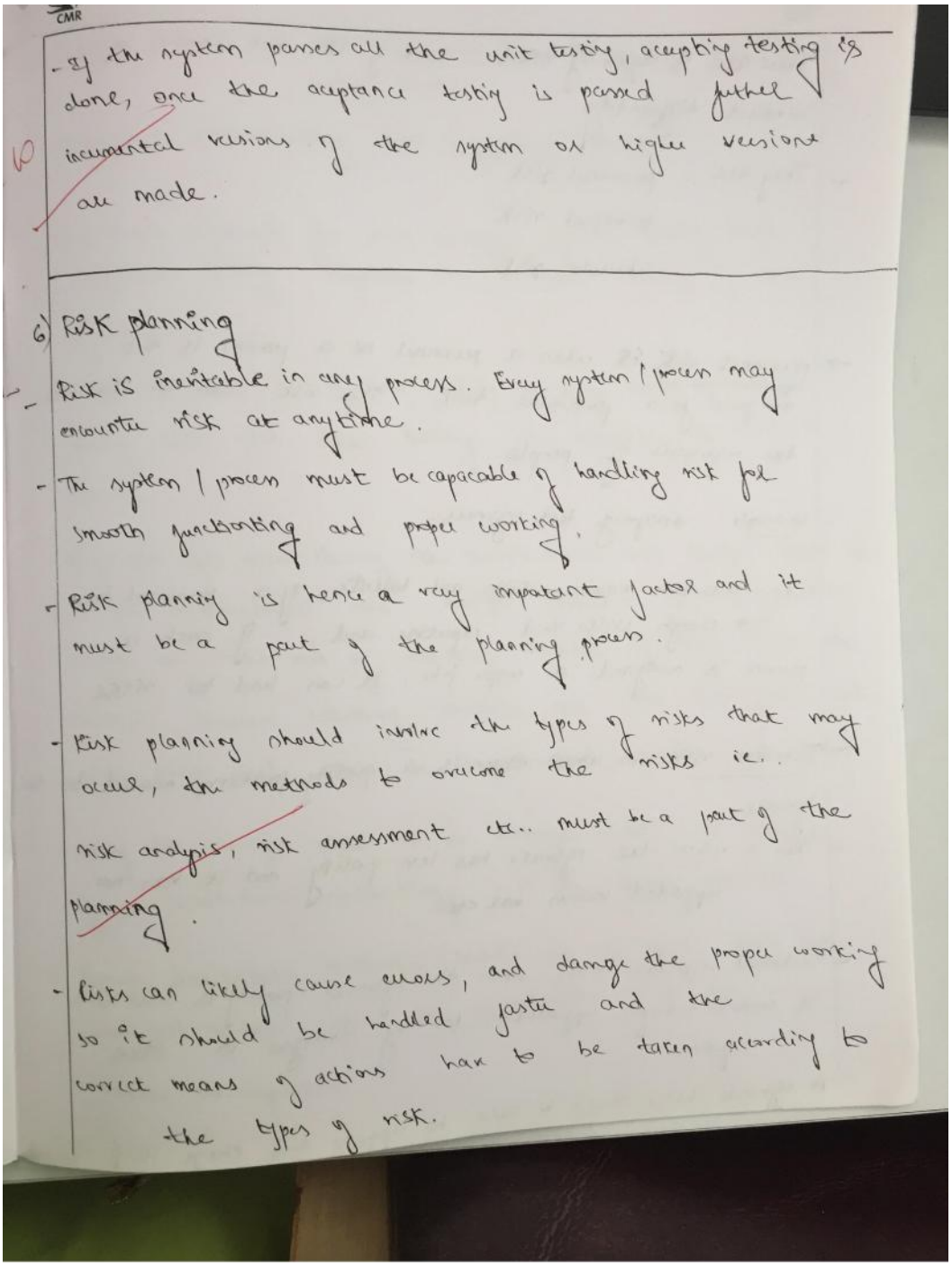
• The reliability is checked based on MTBF= i.e mean time between failures.

• Failure ~~is~~ information is given to the development team

• Reliability failure data is used in the next incremental process.

Example Process: Software Reliability Engineering Testing (SRET)

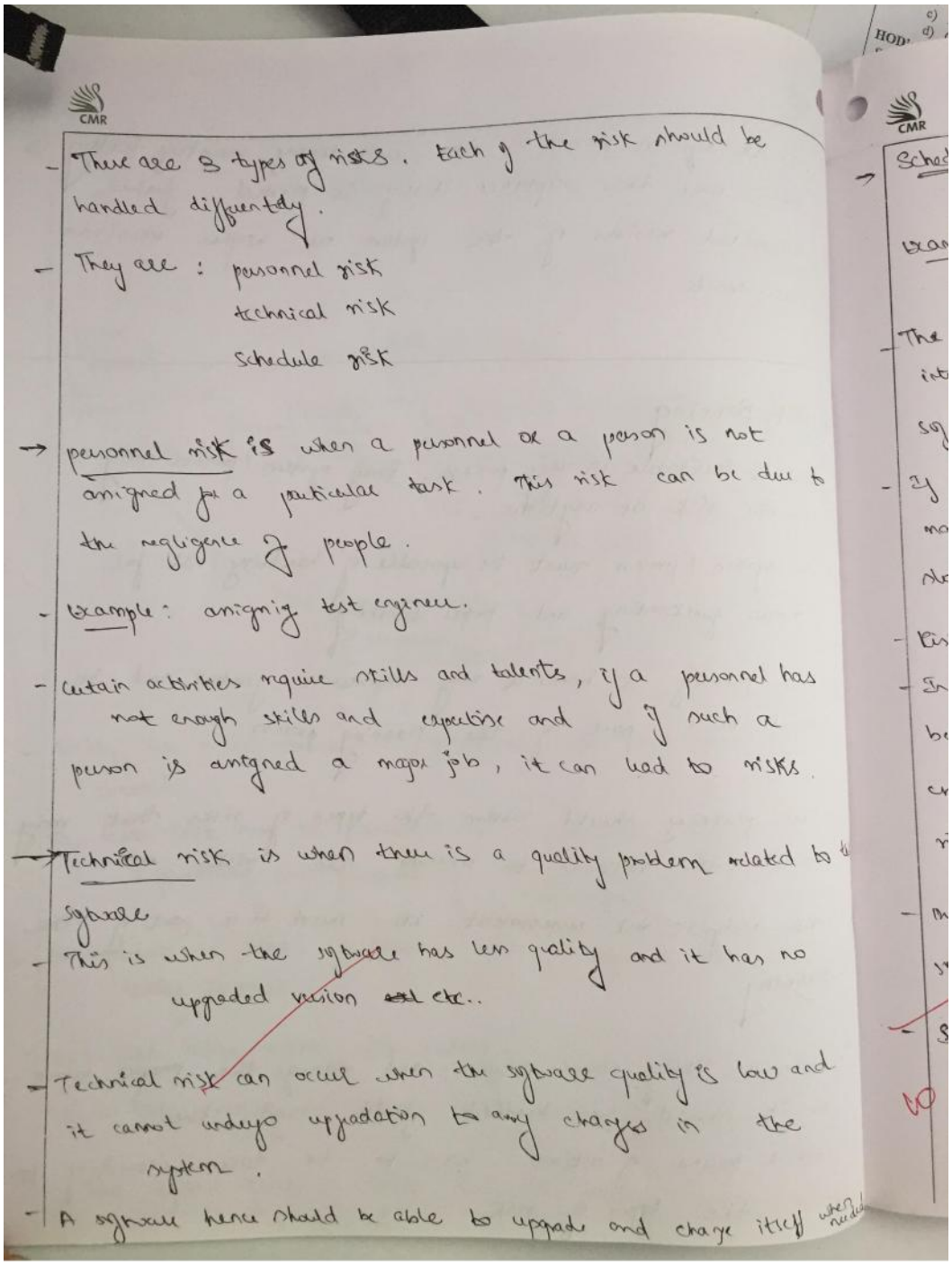




- If the system passes all the unit testing, accepting testing is done, once the acceptance testing is passed futher incremental versions of the system or higher versions are made.

6) Risk planning

- Risk is inevitable in any process. Every system / process may encounter risk at anytime.
- The system / process must be capacable of handling risk for smooth functioning and proper working.
- Risk planning is hence a very important factor and it must be a part of the planning process.
- Risk planning should involve the types of risks that may occur, the methods to overcome the risks ie.. risk analysis, risk assessment etc.. must be a part of the planning.
- Risks can likely cause errors, and damage the proper working so it should be handled faster and the correct means of actions have to be taken according to the types of risk.



- There are 3 types of risks. Each of the risk should be handled differently.
- They are : personnel risk
  technical risk
  Schedule risk

→ personnel risk is when a personnel or a person is not assigned for a particular task. This risk can be due to the negligence of people.

- Example: assigning test engineer.
- Certain activities require skills and talents, if a personnel has not enough skills and expertise and if such a person is assigned a major job, it can lead to risks.

→ Technical risk is when there is a quality problem related to the software

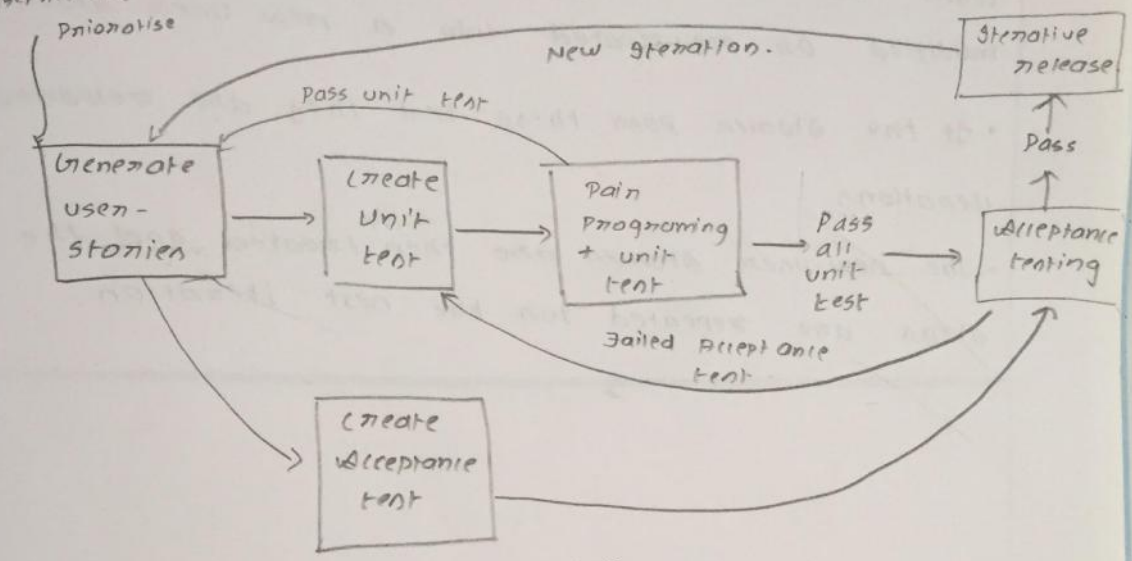
- This is when the software has less quality and it has no upgraded version etc..
- Technical risk can occur when the software quality is low and it cannot undergo upgradation to any changes in the system.
- A software hence should be able to upgrade and change itself when needed

Schedule risk : is based on the assumptions on the quality tasks.

example : underestimating the scaffolding process.

- The time required for unit testing, system testing and integration testing depends on the quality of the software
- If the software is of low quality, it may have to undergo many test and the testing is slower, it becomes slower.
- Risks can also occur during the architectural and design phase etc..
- In the architectural and design phase it is always better to make use of existing models if they match the criteria, because existing models are less error prone and risk prone and also be implemented faster.
- During the design phase it is always better to use a software architecture methodology, it is a good practice.
- Software architecture methods (models can be used such as waterfall model, etc.. to prevent risks and for better development.

The Test-cases which are defined denves on a major specification in this process.



XP

- The user-stories are generated by revising, refining and prioritizing them. ~~They are then~~
- The two tests are created to check whether there are any errors in the created stories.
- Unit testing is done ~~bened~~ along with pair programming. Pair programming reduces a error to a large extent where one programmer corrects or solves the error on bugs present in the other programmers code and thenby the speed of execution also increase.
- The other test is acceptance test which removes various error

Critical dependence:-

Critical dependence is a method that occurs when the activity in a critical path is scheduled soon after the other activities in the critical path, particularly its length ~~is~~ ~~do~~ on duration is same as the duration of the project.

It finds a link or relationship between the various activities in a project.

5a. Extreme Programming Strategy:-

Extreme Programming methodology (XP) emphasizes simplicity over generality, continuous vision and communication over structural organisation, frequent changes over big release, continuous testing and analysis over seperating roles and responsibilities, continuous ~~teeth~~ testing over traditional practices.

~~Involving customer into~~

Having customer involvement in the extreme programming results in requirement analysis (like revise, refine and prioratise user strategies (stories) and acceptance test to have iterative processes. Planning involves the early prioratination of the strategies and implement & release the small iterations.

2. Develop a "operational profile" -

Develop a overall ~~pro~~ model that contains both operational profile and operational model within a single operational model.

3. Prepare for testcase. -

Name and define various testcases and procedures.

4. Execute tests -

Based on the above test-case execute those tests and find results.

5. Interpret failure data -

The failure result is based on the different testing method. The development tests finds and removes the errors & faults present in the software developed in-house. The certification test either accepts or rejects the outsourced software.

10

5b. Critical Path :

Critical path specifies a set of activities that should be executed in sequence and it has the longest duration. It is a method to find the longest path ~~am~~ in the project that are present in the project. It also finds out the various activities and non-activities be executed in sequence otherwise results in failure of project.

(8) Organising Documents :

↳ Documentation is an important process in the software development cycle. Documentation allows us to easily reuse the project

↳ Documentaion of a project involves planning, specification and Reporting.

↳ Planning involves test and analysis strategy, test and analysis plan.

↳ Specification involves test design specification and, test plan, checklist

↳ Reports involve test and analysis report and other documents and references that are required.

↳ In small projects with sufficiently less number of documents, arranging other design & test documents together with standard content helps organising the documents is simple.

↳ But in large projects, the documents have to be organised properly. Regular update of a global guide regarding documentation is required.

↳ Mature projects involves metadata about the projects documents.

↳ Documents must contain the information required making the document self contained and, approvals and history.

Documentation template

Document Title

Approvals

Issued	Name	Signature	Date
Approved by	Name	Signature	Date
Distribution status	(Details regarding status)		
Distribution list	(List of users to whom it must be distributed)		

History:

Version	Description

Table of contents:-

Lists all the sections that are covered in the document.

Summary:

Summarizes the significance of the documents in brief.

Goals of the document:

It describes the goals of the document. It describes the relevance of documenting project

CMR

Required Documents & References :

Documentation also involves the list of documents that can be referred to get more information about the document and references.

Glossary :

Glossary includes all the terms that are used in the document and their references.

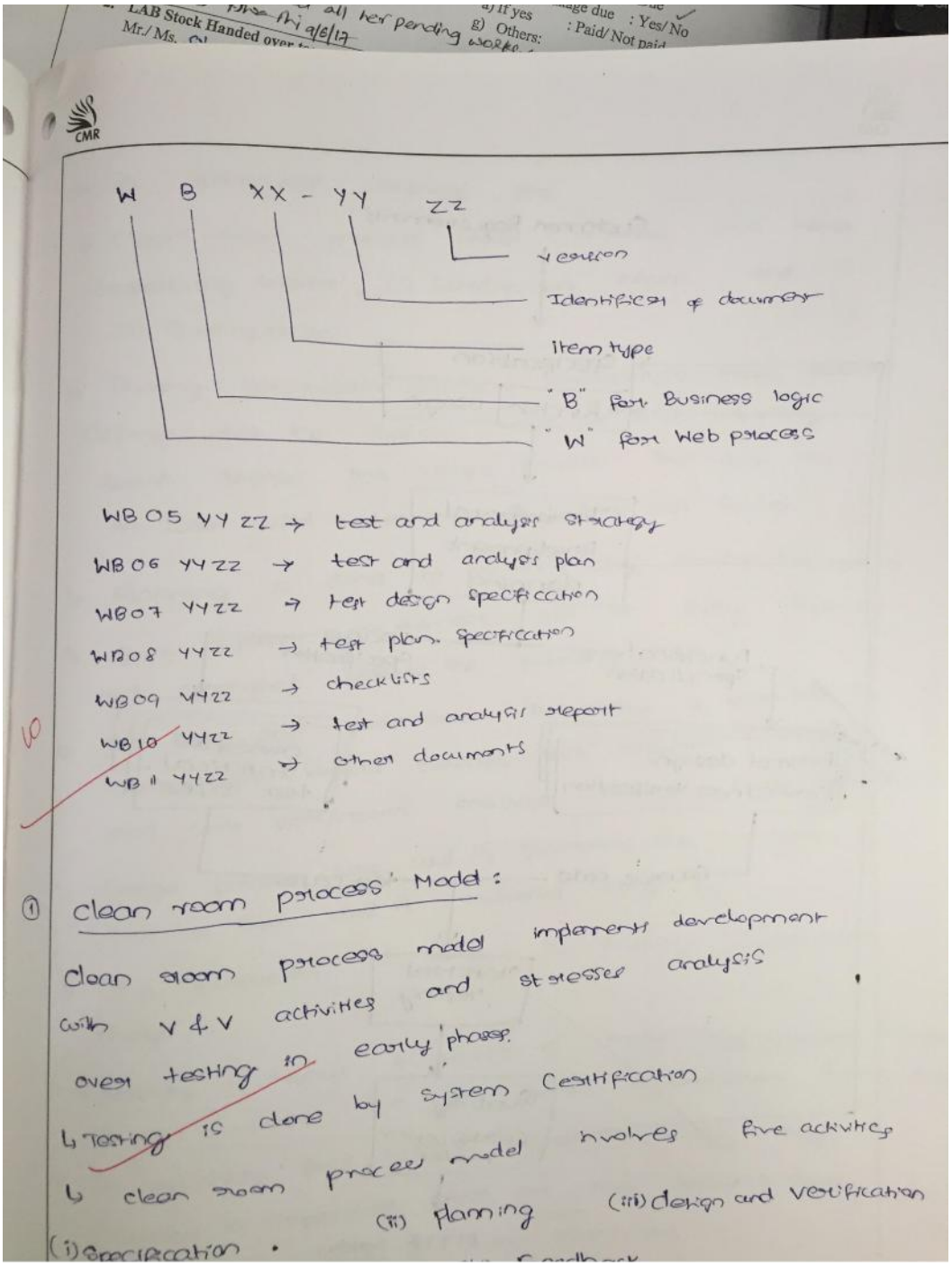
Section I :

⋮

Section N :

Above shown is a template of a Document.

- Documentation must follow proper Naming convention inorder to easily navigate.
- Names must be chosen based on the general documentation, purpose, specific document and version
- Sample naming convention, complaint with IEEE standards is shown in the next figure.



3. Capture & Replay

Sometimes it is difficult to either devise a precise description of expected behavior or adequately characterize correct behavior for effective self-checks. If one cannot completely avoid human involvement in test case execution, one can at least avoid unnecessary repetition of this cost and opportunity for error. The principle is simple. The first time such a test case is executed, the oracle function is carried out by a human, and the interaction sequence is captured. Provided the execution was judged to be correct, the captured log now forms an (input, predicted output) pair for subsequent automated retesting. Distinguishing between significant and insignificant variations from predicted behavior, in order to prolong the effective lifetime of a captured log, is a major challenge for capture/replay testing. Capturing events at a more abstract level suppresses insignificant changes.

Mapping from concrete state to an abstract model of interaction sequences is sometimes possible but is generally quite limited. A more fruitful approach is capturing input and output behavior at multiple levels of abstraction within the implementation. We have noted the usefulness of a layering which abstract input events are captured in place of concrete events. Typically, there is a similar abstract layer in graphical

output, and much of the capture/replay testing can work at this level. Small changes to a program can still invalidate a large number of execution logs.

the capture phase works by (1) identifying all the interactions between observed and external code, (2) suitably instrumenting the application code, and (3) efficiently capturing interactions at runtime. In the replay phase, our technique first performs two steps analogous in nature to the first two steps of the capture phase: it (1) identifies all the interactions between observed and external code, and (2) suitably instruments the application code. Then, the technique inputs an event log generated during capture and, for each event, either performs some action on the observed code or consumes some action coming from the observed code.

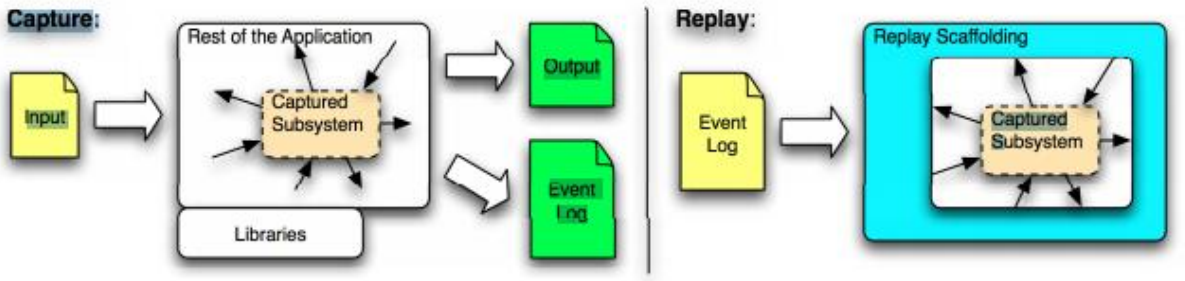


Figure 2: Overview of the capture-replay technique.

3(b) Root Cause Analysis

- Technique for identifying and eliminating process faults
  - First developed in the nuclear power industry; used in many fields.
- Four main steps
  - *What* are the faults?
  - *When* did faults occur? When, and when were they found?
  - *Why* did faults occur?
  - *How* could faults be prevented?

7. Change Configuration

- Same purpose as other software design documentation:
  - Guiding further development
  - Preparing for maintenance
- Test design specification documents:
  - describe complete test suites
  - may be divided into
    - unit, integration, system, acceptance suites (organize by granularity)
    - functional, structural, performance suites (organized by objectives)
    - ...
  - include all the information needed for
    - initial selection of test cases
    - maintenance of the test suite over time
  - identify features to be verified (cross-reference to specification or design document
  - include description of testing procedure and pass/fail criteria (references to scaffolding and oracles)
  - includes (logically) a list of test cases
- Complete test design for individual test case
- Defines
  - test inputs
  - required environmental conditions
  - procedures for test execution

    - expected outputs
- Indicates
    - item to be tested (reference to design document)
- Describes dependence on execution of other test cases

Is labeled with a unique identifier