Assignment - 1

1) Define the purpose of testing.

A: 1) To catch bugs:

- Bugs are due to imperfect communication among programmers.

2) Productivity Related reasons:

- Insufficient effort in QA ⇒ High Rejection Ratio ⇒ Higher Rework ⇒ Higher Net Costs.

3) Goals for testing:

- Primary goal is test prevention.
- If a bug is preveted, repetitive work can be avoided.
- If not possible, testing must reach its secondary goal of bud discovery.

4) 5 phases in tester's thinking:

Phase 0: says no difference between debugging & testing.

Phase 1: says testing is to show that the software works.

Phase 2: says software does not work.

Phase 3: says test for risk reduction.

Phase 4: A state of mind regarding, "What testing can do & cannot do? What makes software testable?"

5) Testing and Inspection :

- Inspection - also called static testing.

2) Explain Dichotomies.

A: . Dichotomy refers to the division into two especially mutually exclusive or contradictory groups or entities.

- There are six of them :

1. Testing vs Debugging.
2. Functional vs structural testing.
3. Designer vs Tester.
4. Modularity (Design) vs Efficiency.
5. Programming in SMALL vs Programming in BIG.
6. Buyer vs Builder.

1) Testing vs Debugging:

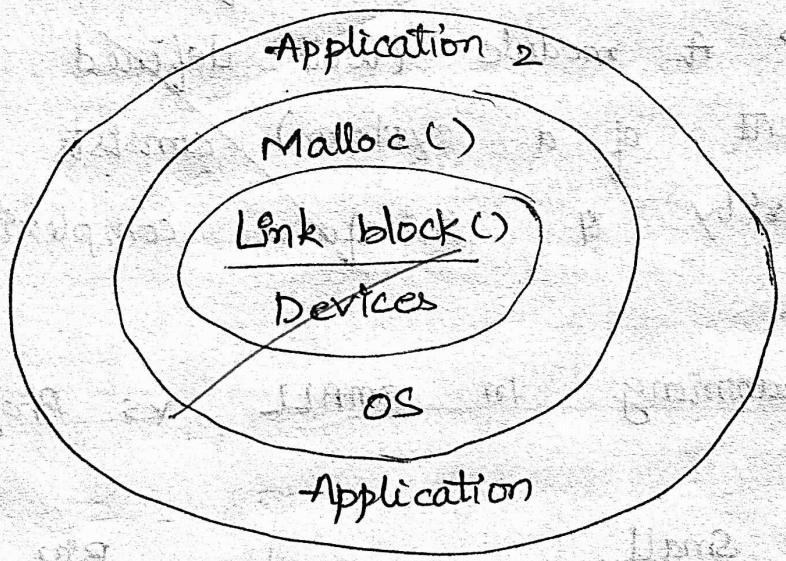
- Testing is to find bugs
- Debugging is to find the cause or misconception leading to the bug.

Testing	Debugging.
1. Starts with known conditions & uses predefined procedure.	1. Starts with possibly unknown initial conditions & end cannot be predicted.
2. Planned, designed and scheduled.	2. Procedures and duration are not constrained.

3. A demo of an error or apparent correctness.	3. A deductive process.
4. Proves programmer's success or failure.	4. It is programmer is vindication.

2) Functional vs structural Testing:

- Functional testing treats a program as a black box. Outputs are verified for conformance to specifications from user's point of view.
- Structural testing looks at the implementation details like programming style, control method, source language, database & coding details.



3) Designer vs Tester:

Programmer	Tester
1. Tests designed are more oriented towards structural testing	1. The tester can eliminate useless tests, optimize & do an efficient

	test design.
2. Likely to be biased.	2. Tests designed by independent test are bias free.
3. Tries to do the job in simplest & cleanest way trying to reduce the complexity.	3. Tester needs to be suspicious, uncompromising, hostile & obsessed with destroying the progra

4) Modularity vs Efficiency:

- System & Test design can both be modular.
- A module implies a size, an internal structure and an interface.
- A module (well defined discrete component of a system) consists of internal complexity & interface complexity & has a size.

5) Programming in SMALL vs Programming in BIG.

Small	Big
1. More efficiently done by informed, intuitive means and lack of formality.	1. A large no. of programmers & components.
2. Done for oneself	2. Program size implies non-linear effects.

6) Buyer vs Builder :

- Builder : Designs for & is accountable to the buyer.
- Buyer : Pays for the system.

3) Explain the model for testing:

A: 1) Project :

A model for project in a real world consists of the following 8 components :

- Application
- Staff
- Schedule
- Specifications
- Acceptance test
- Personnel
- Standards
- Source
- History

2) Roles of Models for Testing :

i) Overview :

- Testing starts with a program embedded in an environment.
- Human nature of susceptibility to error leads to 2 models.

ii) Environment :

- All hardware & s/w required to make the program run.
- Usually bugs do not result from the

environment but arise from our understanding of the environment.

3) Program :

- Complicated to understand in detail & deal with a simplified overall view.
- Focus on control structure ignoring processing & focus on processing ignoring control structure.

4) Bugs :

- Categorize the bugs as initialization, call sequence, wrong variable, etc.-

5) Testing & Levels :

- Unit & Component Testing.
- Integration Testing
- System Testing.

4) Explain the consequences of bugs.

A: Mild:

Aesthetic bug such as mispelled output.

Moderate:

Outputs are misleading or redundant impacting performance.

Annoying:

Systems behaviour is dehumanizing for ex. names are truncated / modified

arbitrarily, bills for $0 are sent.

Disturbing:

Legitimate transactions refused.

Serious:

Losing track of transactions & transaction events. Hence accountability is lost.

Very serious:

System does one transaction instead of another.

Extreme:

Frequent & Arbitrary.

Intolerable:

Long-term unrecoverable corruption of the database.

Catastrophic:

System fails & shuts down.

Infections:

Corrupts other systems.

5) Explain the taxonomy of bugs.

A: Adopt known taxonomy to use it as a statistical framework on which your testing strategy is based.

1. Requirements, Features, Functionality Bugs.

i) Requirements & specs:

→ Incompleteness, ambiguous or self-contradictory

→ Analyst's assumptions not known to the designer.

ii) Feature Bugs:

→ Specification problems create feature bugs

→ Wrong feature bug has design implications

iii) Feature Interaction Bugs:

→ Arise due to unpredictable interactions between feature groups or individual features

Testing techniques like transaction flow testing, syntax testing, domain testing, logic testing, etc., can eliminate requirements & specifications bugs.

2. Structural Bugs:

i) Control & Sequence Bugs:

→ Unit, structural, path & functional testing are used for prevention.

ii) Logic bugs:

→ Logic testing, careful checks & functional testing are used for prevention.

iii) Processing Bugs:

→ Arithmetic, algebraic, mathematical function evaluation, etc.

→ Unit testing & Domain testings are used for prevention.

3. Data Bugs:

i) Generic data bugs.

ii) Dynamic & static data.

iii) Info. parameter & control bugs.

iv) Contents, structure & attributes related bugs.

4. Coding bugs:

Bugs arise due to:

i) Coding errors.

ii) Syntax errors.

iii) Documentation Bugs.

5. Interface, Integration & system bugs.

i) External interfaces.

ii) Internal interfaces.

iii) Hardware Architecture Bugs.

iv) Operating System Bugs.

v) Software architecture bugs.

vi) Control & sequence bugs.

vii) Resource management bugs.

viii) Integration bugs.

ix) System bugs.