Week 10 Software Testing Tutorial

1. **Interface Errors + Component Testing**

   A development team integrates three modules into a component called BookingComponent.
   - DateValidator checks date formats and ranges
   - PaymentProcessor handles payment
   - TicketAllocator assigns seats

   During component testing, the following issues appear:
   1. When passing “2024-02-30” into BookingComponent.book(), the system crashes.
   2. PaymentProcessor sometimes returns “pending” instead of “success”, but the component assumes all payments are “success”.
   3. Under heavy load, two requests occasionally receive the same seat number.

   Task: For each issue, identify the type of interface error and explain why. Then explain why these problems appear at component testing rather than unit testing.

   Model Answer:
   1. Error type: Interface misuse — invalid date not handled. Crashes because the component does not validate inputs.
   2. Error type: Interface misunderstanding — component wrongly assumes payment always returns 'success'.
   3. Error type: Timing error — race condition in seat allocation.
   Why found in component testing: These errors occur when multiple units interact, not within a single isolated unit.

2. **Partition + Guideline-based Test Design**

   A function processUserAge(int age) returns:
   - “Child” for 0–12
   - “Teen” for 13–17
   - “Adult” for 18+
   - Throws an error for invalid ages

   Task: Using both equivalence partitioning and guideline-based testing, design a minimal but powerful set of test cases that is likely to reveal defects. Explain (1) which partitions you identified, (2) why each guideline-based case is important, and (3) which defects

your set would likely expose.

Model Answer:
Partitions:
- Invalid: <0, >100 (e.g., -1, 101)
- Child: 0–12 (e.g., 5)
- Teen: 13–17 (e.g., 15)
- Adult: 18+ (e.g., 25)

Guideline-based test cases:
- Empty/edge cases: 0, 12, 13, 17, 18
- Extreme values: very large age (99999)
- Null input (if allowed)

Defects exposed: boundary errors, incorrect comparisons, missing error handling.

3. **System Testing vs Release Testing**

A company completed development of an online shopping system. During system testing, developers confirmed that all integrated modules worked. However, during release testing, the independent team found issues:
- Users could not complete checkout when the network was slow
- Some workflows did not match the requirement document
- Error messages were confusing or missing

Task: Explain why these problems were not detected in system testing but were detected in release testing.

Model Answer:
- System testing focuses on integrated technical correctness under controlled conditions.
- Release testing evaluates requirement satisfaction, usability, and readiness for real users.
- Slow network issues appear only in realistic environments.
- Workflow mismatch is caught when testers validate against the requirement document.
- Poor error messages are user-facing quality issues — not developer-focused system tests.

4. **Performance Testing + Operational Profiles**

A streaming platform wants to conduct performance testing. Real usage statistics show:
- 65% watch videos
- 20% search
- 10% browse recommendations

- 5% manage account settings

Current testing uses equal load (25% each) and passes, but users still experience slow video loading.

Task: (1) Construct a correct operational profile, (2) explain why the current approach failed, and (3) describe a scenario where stress testing would be relevant.

Model Answer:
1. Correct operational profile (400 requests):
- Watch: 260
- Search: 80
- Browse: 40
- Account: 20

2. Why current approach failed:
- Equal 25% distribution does not reflect real workload.
- Watching videos is the heaviest operation; under-test → real bottleneck hidden.

3. Stress testing scenario:
- Sudden spike where thousands start streaming during a live event → test system limits.

5. **System Integration Failure**

A banking system consists of:
- LoginComponent
- BalanceComponent
- TransferComponent
- NotificationComponent

System testing passed, but during beta testing, issues occur:
- Transfers sometimes succeed but notifications are not sent
- Logging out immediately after transfer cancels the notification
- Two simultaneous transfers produce incorrect balances

Task: (1) Identify interface misuse, misunderstanding, or timing errors; (2) explain why these issues did not appear in system testing; (3) suggest which type of user testing is appropriate for revealing each issue.

Model Answer:
Error classification:
- Missing notifications → interface misunderstanding (component assumes notification

always succeeds).
- Logout cancelling notification → timing error (state update happens before async notification completes).
- Inconsistent balances → timing error (race condition between concurrent transfers).

Why not in system testing:
- System testing uses stable conditions.
- Beta testing reveals real-world timing, concurrency, and user workflow issues.

User testing types:
- Notification missing → Beta testing
- Logout timing issue → Alpha or Beta testing
- Concurrent balance errors → Beta testing with real users