Testing Tips and Tricks for Programmers

Pragmatic and Practical Testing

Yvonne Enselman, CTFL, CTAL-TA

Practical Testing

- Effective: producing a decided, decisive, or desired result
- Efficient: productive of the desired effect; especial to be productive without waste
- Avoid redundancy
- Reduce cost

What Testing Isn’t

- Proof that there are no bugs
- Proof that all defects have been found

Judy McKay:

“We have to be sure that we don’t compound this misconceptions that testing can prove the absence of defects or can discover all defects by publishing testing metrics with statements like ‘100 percent test coverage [achieved].’ Just because we ran 100 percent of our test cases doesn’t mean that we’ve covered 100 percent of what the code could do. We need to be sure our information is accurate, supportable, and understandable.”

We’re lost but we are making good time

- Test Oracle: What we expect, what should result, what we measure against

- Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining it meets required results

- A good tester expects to find bugs because all programs have bugs, a good tester expects to find defects because all systems contain them.

Risk Management

- Features: Will the system provide the right capabilities
- Schedule: Will you deploy or release the system soon enough
- Budget: Will the overall effort or project make financial sense
- Quality: Will the system you create satisfy the users, customers, Stakeholders

Mitigating and Identifying Risk

- Identification of Risk Categories
- Understanding Test Techniques
- Performing Risk Analysis

Test Techniques

- Static Testing: Testing the system without actually running the system, based on project artifacts.
- White-Box Testing: Testing how the system works internally, how it's built, and its structural characteristics.
- Black-Box Testing: Testing what the system does, particularly its behaviors and the business problems it solves.

Quality Risk Categories

Functionality

performance and reliability

Stress, capacity, and volume

States

Transitions

Installation and deinstallation

Operations

Maintenance and Maintainability

Regression

Usability and user interface

Data Quality

Errors and disaster handling and recovery

Date and time handling

Localization

Configuration and compatibility

Networked, inter networked, and distributed

Standards and regulatory compliance

Security

Timing and coordination

Documentation

Risk Analysis

- Determine what risks are evident
- Assign importance based on Business exposure and Technical exposure
- Decide what risks need which level of testing
- Use requirements and specifications
- Cross Team contributions and input are very valuable

In a Traditional Waterfall or V development methodology it might look like….

- Requirements Phase = Quality Risk Analysis
- After design specification update QRA
- Review the technical risks when implementation is complete
- Update ratings as Unit, Integration, and System testing begins

Equivalence Classes and Boundary Values

- Equivalence Class Partitioning: Identify the inputs, outputs, behaviors, environments, and/or any other testing items. Group all factors into classes that the systems should handle in a given way.

- Boundary Value Analysis: A boundary is where the test object's behavior will change. Especially when Greater Than, Less Than, Equal to are used in conjunction with each other or complicated algorithms.

Use Cases, Live Data, & Decision Tables

- Use case & Scenario Tests: Design cases that reflect the real world processing or anticipated performance of the test object.

- Live Data and Customer Testing: The above is what the system should do, this tests what the system actually does.

- Decision Tables: Translation of business rules into test cases, especially when multiple factors impact the behavior of the test object or interactions.

State Transition Diagrams

- Determine viewpoint, understand all states the test object can transition through, identify what can and can’t apply in each state, graph or model the system, determine the behavior of the test object in each state.

Structural Testing

- Control - Flow
- Data - Flow
- Integration Testing

Control Flow Testing

- Control Flow - the sequence of execution of statements
- Statement Coverage: You achieve 100% statement coverage when you have exercised every statement.
- Branch or Decision Coverage: You achieve 100% coverage when you have taken every branch each way.
- Condition Coverage: You achieve 100% coverage when the multiple conditions that direct behavior have been evaluated both true and false
- MultiCondition Coverage: When compound conditions apply to branching, you can move beyond condition coverage to multi condition coverage by requiring testing all possible combinations of conditions.
- Loop Coverage: In loop paths iterate each loop 0, 1, and multiple times. Ideally you can identify the maximum iterations and test to that boundary and past.
- Path Coverage: You achieve 100% coverage when you have taken all possible control paths

Data-Flow Testing

- Analyzation of the data progressing through the test object or process

Integration Testing

- Drivers and Stubs: Parts of the system that interacts isn't ready and needs to be simulated

- Integration Techniques: Isolating the components under test via a test environment can isolate the factors in the build.

- Backbone Integration: Using business based risk analysis processes are monitored through the entire system