Software Testing

INTRODUCTION

Intro

- Suppose you are being asked to lead the team to test the software that controls a new X-ray machine. Would you take that job?
- What if the contract says you'll be charged with murder in case a patient dies because of a mal-functioning of the software?

State-of-the-Art

- 30-85 errors are made per 1000 lines of source code.
- extensively tested software contains 0.5-3 errors per 1000 lines of source code.
- testing is postponed, as a consequence: the later an error is discovered, the more it costs to fix it.
- error distribution: 60% design, 40% implementation. 66% of the design errors are not discovered until the software has become operational.

Software testing

- Check software correctness.
- Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not.
- Who does testing?
  - Software Tester
  - Software Developer
  - Project Lead/Manager
  - End User

Error, fault, failure

- An **error** is a human activity resulting in software containing a fault
- A **fault** is the manifestation of an error
- A fault may result in a **failure**
- Error →Fault → Failure

Classification of testing techniques

- Classification based on the criterion to measure the adequacy of a set of test cases:
  - coverage-based testing
  - fault-based testing
  - error-based testing
- Classification based on the source of information to derive test cases:
  - black-box testing (functional, specification-based)
  - white-box testing (structural, program-based)
  - Grey-box testing

Comparison

	Black Box Testing	Grey Box Testing	White Box Testing
1.	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2.	Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3.	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4.	-Testing is based on external expectations -Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5.	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6.	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
7.	This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

Testing Myths

- **Myth:** Testing is too expensive.
- **Reality:** There is a saying, pay less for testing during software development or pay more for maintenance or correction later. Early testing saves both time and cost in many aspects however, reducing the cost without testing may result in the improper design of a software application rendering the product useless.
- **Myth:** Testing is time consuming.
- **Reality:** During the SDLC phases testing is never a time consuming process. However diagnosing and fixing the error which is identified during proper testing is a time consuming but productive activity.
- **Myth:** Testing cannot be started if the product is not fully developed.
- **Reality:** No doubt, testing depends on the source code but reviewing requirements and developing test cases is independent from the developed code. However iterative or incremental approach as a development life cycle model may reduce the dependency of testing on the fully developed software.

Testing Myths (2)

BRAC UNIVERSITY
Inspiring Excellence

- **Myth:** Complete Testing is Possible.
- **Reality:** It becomes an issue when a client or tester thinks that complete testing is possible. It is possible that all paths have been tested by the team but occurrence of complete testing is never possible. There might be some scenarios that are never executed by the test team or the client during the software development life cycle and may be executed once the project has been deployed
- **Myth:** If the software is tested then it must be bug free.
- **Reality:** This is a very common myth which clients, Project Managers and the management team believe in. No one can say with absolute certainty that a software application is 100% bug free even if a tester with superb testing skills has tested the application.

Testing Myths (3)

- **Myth:** Missed defects are due to Testers.
- **Reality:** It is not a correct approach to blame testers for bugs that remain in the application even after testing has been performed. This myth relates to Time, Cost, and Requirements changing Constraints. However the test strategy may also result in bugs being missed by the testing team.
- **Myth:** Testers should be responsible for the quality of a product.
- **Reality:** It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Tester's responsibilities include the identification of bugs to the stakeholders and then it is their decision whether they will fix the bug or release the software. Releasing the software at the time puts more pressure on the testers as they will be blamed for any error.

Testing Myths (4)

- **Myth:** Test Automation should be used wherever it is possible to use it and to reduce time.
- **Reality:** Yes it is true that Test Automation reduces the testing time but it is not possible to start Test Automation at any time during Software development. Test Automaton should be started when the software has been manually tested and is stable to some extent. Moreover, Test Automation can never be used if requirements keep changing.
- **Myth:** Any one can test a Software application.
- **Reality:** People outside the IT industry think and even believe that any one can test the software and testing is not a creative job. However testers know very well that this is myth. Thinking alternatives scenarios, try to crash the Software with the intent to explore potential bugs is not possible for the person who developed it.

Level of testing

- Functional Testing
  - Unit Testing
  - Integration Testing
  - System Testing
  - Regression Testing
  - Acceptance Testing
  - Alpha testing
  - Beta Testing
- Non-Functional Testing
  - Performance Testing (Load and Stress)
  - Usability Testing (Usability vs UI testing)
  - Security Testing
  - Portability Testing

Functional Testing

- **Unit Testing**
  - This type of testing is performed by the developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is separate from the test data of the quality assurance team.
  - The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.
- **Integration Testing**
  - The testing of combined parts of an application to determine if they function correctly together is Integration testing. There are two methods of doing Integration Testing Bottom-up Integration testing and Top Down Integration testing.
  - **Bottom-up integration** testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
  - **Top-Down integration** testing, the highest-level modules are tested first and progressively lower-level modules are tested after that. In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing.
- **System Testing**
  - This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.

Non-Functional Testing

- **Performance Testing**
  - It is mostly used to identify any bottlenecks or performance issues rather than finding the bugs in software. There are different causes which contribute in lowering the performance of software:
    - Network delay.
    - Client side processing.
    - Database transaction processing.
    - Load balancing between servers.
    - Data rendering.
- Performance testing is considered as one of the important and mandatory testing type in terms of following aspects:
  - Speed (i.e. Response Time, data rendering and accessing)
  - Capacity
  - Stability
  - Scalability
- It can be either qualitative or quantitative testing activity and can be divided into differe sub types such as Load testing and Stress testing.

Non-Functional Testing

- **Load Testing**
  - A process of testing the behavior of the Software by applying maximum load in terms of Software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of Software and its behavior at peak time.
  - Most of the time, Load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test etc.
- **Stress Testing**
  - This testing type includes the testing of Software behavior under abnormal conditions. Taking away the resources, applying load beyond the actual load limit is Stress testing.
  - The main intent is to test the Software by applying the load to the system and taking over the resources used by the Software to identify the breaking point. This testing can be performed by testing different scenarios such as:
  - Shutdown or restart of Network ports randomly.
  - Turning the database on or off.
  - Running different processes that consume resources such as CPU, Memory, server etc.

Test-Driven Development (TDD)

- First write the tests, then do the design/implementation
- Part of agile approaches like XP
- Supported by tools, eg. JUnit
- Is more than a mere test technique; it subsumes part of the design work

Steps of TDD

1. Add a test
2. Run all tests, and see that the system fails
3. Make a small change to make the test work
4. Run all tests again, and see they all run properly
5. Refactor the system to improve its design and remove redundancies

Thank you

BRAC UNIVERSITY
Inspiring Excellence