

Testing Concepts

This section contains the concepts that underlie the testing methodologies. It will contain certain perspectives about testing importance and requirements. It will not include the discussion of methodologies or techniques this is more about the basis of testing like the aims which the methodologies try to achieve.

Hitting the most critical areas of functionality - There are infinite tests in even the simplest program therefore when testing you must always give the highest priority to the most critical parts of the system. This naturally implies that the most time should be devoted to the most important parts.

Testing tools are just tools and should be treated as such - Testing tools are not a dogma and one would be a fool to blindly trust the results of a testing tool alone.

Most of the bugs are in the way data is handled - BJ Rollison when explaining BVA.

Structural Testing Drawback - When you are performing structural testing you are running a great risk of being biased by what the code does, therefore you're testing that the code works and not paying attention to possible invalid code. - Rob

Black Box Testing Drawback - When you're only testing from a black box perspective you can only achieve a certain limit of code coverage until you run into a plateau, that is when you must start using white box techniques to increase the code coverage. BJ Rollison.

System and Domain Knowledge are CRUCIAL - You must really know the system under test to be able to exercise it well. (Remember the example of the Gregorian Calendar with the missing days in October and the missed test cases of the Windows Keywords when creating file names) ¹

Quality is everybody's responsibility - Throughout the development of a project quality is important in all facets of the development and is not a task only pertaining to the 'QA team'.

Testing is all about asking questions - The testing perspective one must take when looking at the application is that of asking questions.

Unit testing is a MUST - Unit testing is a way for developers to test their work but they then to do it only from a developer's perspective, therefore it is usually not enough. Testers need to teach developers how to unit test.

¹ We were given a program to test the input for the Gregorian Calendar and from the tests we wrote we didn't pay attention to test 10 days that are missing from the Calendar because we didn't know about them. In a second exercise to test what are valid and invalid file names in a windows save as test box we missed a couple of test cases because of lack of domain knowledge such as trying to save a filename called COM1 or NUL which are reserved words.

Strive not to introduce bugs in the build - Try to catch bugs as quickly as possible and unit tests help in this.

A daily build is the absolute minimum - Even Microsoft runs daily build for Windows therefore there is no reason why you shouldn't (unless maybe yours system is more complex).

My opinions

On Bug Tracking - It is important to record and track bugs to help determine where the most effort needs to be placed in the test case design. If you know that most bugs are functional bugs in a particular section of the system then you need to identify these parameters (functional bugs and section in question) to create more test cases. This ties in nicely with Myers testing principle number 9 (pg 15).

Testing Techniques

This section contains the principle descriptions of what the techniques are about and when to apply them.

Template

Name:

Type: [Type of the technique amongst the different testing types]

Used For: [Usage of technique e.g. Data input testing]

Requires Coding Knowledge:

Domain Knowledge Required: [The domain knowledge required to apply this technique - Minimum, Medium, High]

Main Concept:

[The technique concept in a few easy words]

Notes:

[Personal or other notes on the technique]

Further Reading:

[Further Reading on the subject]

Next Actions:

[What are the next actions that need to be done to use/learn this technique]

Name: Boundary Value Analysis
Type: Functional Testing
Used For: Data input testing
Requires Coding Knowledge: No
Domain Knowledge Required: High
Main Concept:

For each input you need to test the min-1, min, min+1, nominal, max-1, max, max+1. For each parameter you have 6n+1 test cases. For example if you have a field that accepts integers between 0 and 10 these would be the tests.

min-1 = -1

min = 0

min+1 = 1

nominal = any number between 2 and 8

max-1 = 9

max = 10

max+1 = 11

You must also test the output boundaries which are boundaries that go out of range when the input is still in range. For example if there is a limit till end of 3000 in the "get next date" program you must test 31-12-3000 to see whether the output is returned. It is dependent on the context of the test whether the output should be considered valid or not.

[I'm not sure about this] In the case of non numeric fields boundary values might be illegal characters that might mess up with the string processing algorithms such as a ' when inputting characters to mix up a database.

Notes:

1. I believe developers are lazy people and they hate to do input validation therefore this is a good way to look for errors.
2. Even if the programmer didn't write constraints there are going to be limits to input imposed by the computer system, for example if you're using an integer you cannot exceed 32,767 depending on the architecture. Systems can be tested with these as maximum and minimum figures if no bounds are explicitly specified.
3. The source code is the best place to look for boundary values
4. Easy technique which can be quite effective especially when parameters are not dependent on each other.

Further Reading:

The Art of Software Testing - G Myers

Black Box Testing: Techniques for functional testing of software ... - B Beizer

Software Testing: The craftsman's approach - P Jorgensen

Next Actions:

- For the price market application try some boundary value analysis test cases.

- When testing using BVA try to find a way to bypass the validation procedures, example disabling JavaScript.
- Read from pg 59 of Myers book (some more detail stuff)
- Chapter 13 of Software Testing Fundamentals (M Hutcheson)

Name: Equivalence Class Partitioning

Type: Functional Testing

Used For: Data Input Testing

Domain Knowledge Required: High

Main Concept:

When testing input there are classes of values that have identical value when testing any input. You need only test one of these classes in order to test for this functionality as inputting more values will provide no additional benefit for finding bugs. When identifying equivalence classes you are grouping together the input values of the same equivalence class. Example of eq. partitioning of a month

Valid Class A: 1, 3, 5, 7, 8, 10, 12

Valid Class B: 4, 6, 9, 11

Valid Class C: 2

Invalid Class A: < 0

Invalid Class B: > 13

Invalid Class C: empty string

Invalid Class D: any non-integer value

These mean that for example if you test for -1 and you test for -100 you are not getting any benefit from running the second tests as it is within the same eq. class. February (Valid class C) is an example of a unique value in a class that is processed differently from the normal input, this is highly dependent on the domain knowledge one has.

Notes:

This technique is great for reducing redundant tests

Requires in depth knowledge to be able to identify specific and unique situations

Further Reading:

G Myers - The art of software testing

Next Actions:

- Find the equivalence class of office dialog box
- Need to search for information about fonts to be able to classify the fonts

Name: Pair wise testing

Type: Black box testing (Combinatorial Analysis)

Used For: Data input testing / Configuration and setup testing

Domain Knowledge Required: Medium

Main Concept:

Pairwise testing is used to tackle the problem of interrelated parameters. When parameters are affected by each other you need to test the effect of how these might effect each other therefore test the different combinations of the parameters.

For example if you are going to test the installation of a system on two different OS and 4 different browsers and 2 languages there are a total of 24 different ways to test the system. The problem is how are you going to choose which to test when this problem explodes.

There are tools that given the different combinations can try to determine a good set of test cases to test to increase the probability of finding an error.

Notes:

- The tool demonstrated in class are was easy to use

Further Reading:

<http://www.pairwise.org>

Next Actions:

- Read more about it

Web Application Security

- Security people are concentrating more on hardware and network security than Web Application Security - where the **real threat** lies. On the other hand developers tend not to give a damn about security which adds more problems to the equation.
- Security must be a **DESIGN CONSIDERATION**. We often tend to think about security in the last phases of the system but it really needs to start from the design to get embedded properly within the whole system.

Next Actions:

- Devote 3 hours a week to web application security testing
- Test the trading application for SQL Injection attacks.
- Search how you create bind queries in php (and how difficult it is) to identify the potential pitfalls in php developed applications.
- Download white papers from SPI Dynamics
- Search about other techniques listed above
- Get a book about web application security testing

The Testing Career

The ideal - These points are what testing should be about.

- If you're a developer you can speak as a peer to software developer
- Title should be test engineer not QA - quality is everybody's job
- The QA department is not a service to the development team.
- The development and testing team should be on the same location
- The software engineering team doesn't decide what to build but as a tester you can decide by deciding what to automate.
- People in the test team and development team can switch roles from time to time. (Do not ask in an interview)

Potential Pitfalls - There are many different ways companies look at a tester and a jump into this career should take into consideration several factors:-

- Are testers and developers treated as equals?
- Does the company employ testers who are not developers - use caution here because if non developers only are employed then it's likely they are not treated equal.
- Are developers and testers paid equal?
- What is the relationship between the testing team and the development team
- Any opposite point to the ideal list.
- Does the employer find value in a tester? What type of tester does he want a keyboard puncher or a test design creator? What is the ratio between testers and developers? In an interview you might try to fish for clues on this.

Other points and considerations

Think about the difference between writing code directly for the customer to develop custom solutions and writing code against a non customer generate spec for an off the shelf product. The type of development here varies considerably.

Team size - There are companies that actually have HUGE teams even with 100's of people.

It seems that the biggest companies like Microsoft and Cisco have the potential to employ test engineers as engineers (i.e. in the proper way) as they see value in doing so more than the lower end companies.

Many people identified with the fact that it is difficult to find people who want to be testers. Feldstein himself said that he tries to sell the position of tester to people on the phone and others commented on how difficult it is to find testers. Apart from the obvious advantage this presents I think I must be caution on some fronts:

1. Don't give the impression that you're into testing to find a job in another country.
2. Pay particular attention to the question - why do you want to be a tester.

3. Remember that employers don't want people who view testers as a transitional job.

Problems in the Testing World

We don't know what value the test team adds to the organisation.

- Usage of metrics to calculate value
- Quantifying the value of the test team is difficult

Appendix - Lists

Quality Objectives

- Functionality
- Usability
- Performance
- Security
- Compatibility
- Scalability
- Recovery

Bug type examples (Causes of error - Nguyen)

- Design error
- Coding standard or coding related error
- Functional error at run time
- UI design error
- Errors caused by environments
 - Configuration
 - Compatibility
- Performance error
- Functional error caused by heavy traffic
- Exception handling error
- Memory leak or clean up routines
- Errors in functional and data recovery after system failure
- Installation errors
- Vulnerabilities that expose security risks

Test types

- Design review
- Code inspection
- Code walkthrough
- Unit
- API
- External functional
- Usability
- Accessibility
- Configuration
- Compatibility
- Regression
- Performance

- Load
- Stress
- Failover / Recovery
- Installation
- Security
- Compliance

Test Approach Examples

- Requirements based
- Scenario based
- Soap Opera
- Model based
- Attack based
- Risk based
- Fault Injection
- DAST
- Exploratory