**QA Testing Basic Knowledge**

QA Basic Knowledge 3 (May 21)

* Agile
* Test Case: Login Window

QA Basic Knowledge 4 (May 21)

* Essential Software Testing
* Definition
  + IEEE829
    - “The process of analyzing a software item to detect the difference between existing and required condition (that is, bugs) and to evaluate the features of the software items”
    - Art -> Engineering
    - Waterfall Project Management -> Agile
  + ISTQB Glossary
    - “The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects”
  + By James
    - Match Requirement
    - Find more bugs
* Technically SQA and SQA Test are not the same, but we usually will not distinguish them much

QA Basic Knowledge 5 (May 21)

* Failures, Defects (bugs), and Mistakes
  + Mistake, made by a human being
  + Causes a defect to be placed in a product, the defect causes no harm as long as it is not encountered by anybody
  + End user use of the product “hit” the defect, it will give rise to failure

QA Basic Knowledge 6-7 (May 21)

* Testing psychology
  + Developers are viewed as “constructive”, testers as “destructive”, and the users just want to do their jobs
* Black box testing
  + Refers to testing a software item without knowing anything about its inner working, and the testing is based on the requirement specification of what the software should do, includes specified inputs, expected outputs, and steps
  + Requirements based testing
  + Do not know how code works
* White box testing
  + Internal structure-based testing, and requires knowledge of the programming language being used
* Gray box testing
  + Combination of white box testing and black box testing
  + Knows the internal structure using black box method

QA Basic Knowledge 8 (May 21)

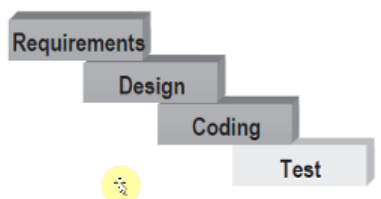
* Static Testing
  + Form of software testing where the software is not actually used
  + Check the design, documents, and read the code
* Dynamic Testing
  + Run the testing software
  + Giving input values
  + Check the output
  + Testing based on specific test cases by execution of the test object or running programs
* Functional Testing
* Non-functional Testing
* Regression testing
  + Re-run existing tests against the new release, to determine whether the changes break anything that worked prior to the change

Homework

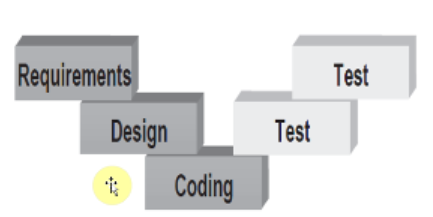
1. What is software testing?
   1. The process of analyzing a software item to find the difference between existing and requirement, and to evaluate the features of software items
2. Difference among failures, defects, and mistakes
   1. Mistakes are caused by human beings, and then a defect is to be placed in a product. The defect is harmless as long as it is not discovered by anyone. The end user might encounter the defect and cause the project to fail.
3. When do you start your software testing?
   1. Requirements and analysis, because if we find the bug earlier, then we can fix the bug earlier, it costs less.

**SDLC and Testing Process I**

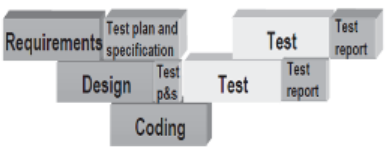
* SDLC stands for system development life cycle. The software project includes: software development, software testing, and project management. Specifically, project management is the most important and software development is the most well-structured. Disregard of quality testing and relatively few tools in testing were the 2 reasons that software testing was not popular, but this situation was getting better. Testing Process is part of SDLC.
* Basic lift-cycle models have three categories: requirement driven (waterfall) development model, iterative development model, and agile development model. Iterative development model was replaced by agile development. RDCT in development models: requirements engineering, design, coding, testing.
* The features of requirement driven model include: the customer knows what he or she wants, the requirements are frozen (changes are exceptions), and phase reviews are used as control and feedback points. The characteristics of a successful sequential development project are: stable requirements, stable environments, and focus on the big picture. According to statistic data, there is 30% change to have a successful project, due to overtime and high cost.
* **Waterfall model**
  + Waterfall model only has one RDCT (Requirement -> Design -> Coding -> Testing). The advantage is that process has been split into difference parts. The disadvantage is that mistakes from early state would affect significantly in the future because of later testing.



* **V-model**
  + V-model is a slightly improved version from waterfall model, there are related testing actives right after requirements and design, respectively. The main features: introducing more test levels, testing not only being performed at the end of the development life cycle, left side of the V is the processes to build the software product, and right side of the V is test levels to ensure that we have specified.



* **W-model**
  + W-model is a slightly improved version from V-model. Design and Test plan and specification are ongoing simultaneously after requirement. Test p&s and Coding are after Design. Tests are the last state with reports. Test plan and specification is the added feature from V-model. The main features: starts testing from the requirement, more time to plan and specify the test, extra test-related review of documents and code, more time to set up the test environments, and better chance of being ready for test execution as soon as something is ready to test.



**Test Levels**

* Acceptance testing: based on and testing of the user requirements. User is majorly involved. BA (business analyst), and technical people are involved to support users' testing.
* System testing: based on and testing of the software system requirements. Testing is based on the entire system, no individual testing for units internally. Blackbox testing.
* Component integration testing: based on and testing the implementation of the architectural design and user interface. Additional tests are required if any units are interacted with others.
* Unit testing (Component testing): based on and testing the implementation of the detailed design. For example, different developers will develop and test different units internally and individually.

**The Basic Software Testing Process (7-step)**

1. Requirement analysis
   1. Customer Requirement Document
   2. Software Requirement Specification (High level design document including DB design, UI design, etc.)
   3. Engineering Requirement Specification (Low level design document like coding class and function design)
2. Test plan and Strategy
   1. Test plan of table of content, including overview, features, strategies, methodologies, resources, team member, start and end, delivery, and schedule
3. Test Case design
   1. Many of many test cases based on SRS
4. Build Test Environment/lab (virtual machine)
   1. Check technique/lab to see if the computers are set up with right software app, like Java is ready to run
5. Execute Test cases
   1. Run the designed test cases.
6. Bug reporting/track/re-test
   1. According to the output, if the output is the same to the expectation, then pass. If bug is found, then report the bug, and keep tracking of it. Rerun the test case if the bugs are tested.
7. Final report

**Software R&D Team**

Department Header

QA Manager

Dev Manager

Tester

Developers

**Component Testing (Unit Testing)**

Outcome

Input

* Testing Object:
  + Individual components
* Requirements:
  + Code (Must be included)
  + Detail Designed Document
  + Some Software Requirement Specification
* Can be tested in isolation, typically done by developer who wrote the code, found defects, and fixed immediately
* Developer write a piece of code to test another piece of code
* May be done using a tool or framework like Java, JUNIT, MS.NET (MS VS UNIT, nunit)

**Integration Testing**

* Focus primarily on interfaces between components

Input

Output 1 = Input 2

Outcome

**System Testing**

* Testing a complete system as defined by project (Blackbox testing)
* Test Object:
  + Test the application from system level
* Requirements:
  + Software Requirement Specification
  + Product Requirement

Outcome

Input

**Acceptance Testing**

* Test Object:
  + The product is expected to be working and it is presented for acceptance
* Requirements:
  + User Requirement
  + User manual
* The customer and/or end users must be involved
* Test includes:
  + Business processes in the new system
  + Manual Operations
  + User Interface, reports, and so forth
  + Documents
  + Use Cases and/or scenarios

**Validation vs. Verification**

* Validation: “Are we building the right product”
  + Confirms that a required calculation of discount has been designed and coded in the product
* Verification: “Are we building the product correctly?”
  + Confirms that the implemented algorithm calculates the discount as it is supposed to un all details

**Test Inputs**

* Requirements and Design Documents
* Test application
* Project plan and management process
* Everything developer created
* Everything released to customers

**Test Outputs**

* Test plan
* Test design, test cases, test scripts (Automation)
* Test environments
* Test logs
* Bugs
* Progress reports
* Test summary report

**Test Plan**

1. Overview
   1. System Overview: Business introduction
   2. System Architecture: How we build system
2. Test Strategy and Test Methodologies
   1. Test Strategy: High level
   2. Test Methodologies: Different testing including web, security, etc.
3. Scope (The main part in test plan)
   1. Features I need to test. Ex. in “home feature model”, what kind of features are included? -- bestseller, featured product, shopping cart, product category & review …
   2. Features I don’t need to test
4. Test Procedure and Process
   1. Project Management Process (Agile)
   2. Bug management
   3. Review Process
   4. Test Case Process
5. Entry/ Exit Criteria
   1. Entry
      1. Test Case complete
      2. Application under testing
      3. Lab/ environment ready
   2. Exit (When should we stop?)
      1. All test cases are passed or 100% executed
      2. All test cases should 100% cover requirement or user story
      3. All high priority defects are closed, 90% low priority bugs are closed
6. Environment and Tools (Resources)
   1. Software
   2. Hardware
7. Role and Responsibilities
   1. Role and title
   2. Responsibilities (Manger, Project Manager, Leader, Tester)
8. Deliverable and Schedule
9. Risk
   1. Technique risk
   2. Project Management

**Test Case**

* Software Tester Designed
* To test a specific software function works correctly or not
* Includes input and expect results
* Includes a serial of test steps
* Execution preconditions and references

Requirements (3 levels)

* Business requirements
* User requirements
* System requirements

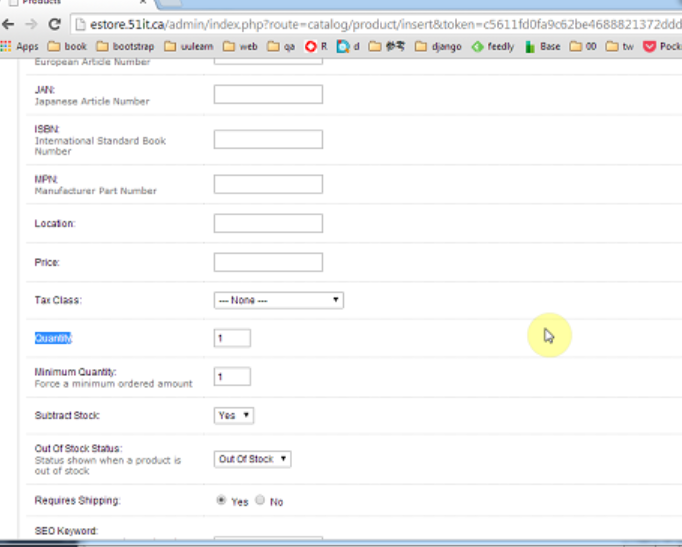
Test Case Design Techniques

* Test case design techniques are the heart of testing
* Types: Specification-Based Techniques
  + Design test cases based on an analysis of the description of the product without reference to its internal workings
  + Black-box tests
  + Focus on the functionality
  + Dependent on expectations to the product or system
* Equivalence portioning (valid and invalid): Portion the input or output into equivalence classes to create minimum number of black box tests. For example, if there are many groups of women and men, we just choose 1 woman and 1 man from each group as input to test.
  + Boundary value analysis (边界值)
* Types: Specification-Based Techniques
  + In defect-based testing, we are looking at the types of defects we might find in the product under testing
  + The techniques are therefore starting from previous experience, rather than the expected functionality or the structure of the test product

Equivalence Portioning (EP) with Test Case

* Analysis of requirements
* Identify equivalence partitioning to cover the requirement (Valid vs. Invalid)
* Write test data to cover the EP
* For example, a person who is younger than 19 cannot drink alcohol, we should have 8 cases, the last 4 are enhancements. The following are Test Data.
  + EP1: <= 0 -> -1
  + EP2: 1 to 18 -> 16
  + EP3: 19 to 150 -> 23
  + EP4: > 150 -> 189
  + EP5: age is blank
  + EP6: float number (10.6),
  + EP7: in English words (nineteen or ‘19’)
  + EP8: out of range number (999999999999999999999999999999)
* Test Case: can be combined if expected results and testing procedure are the same
  + Test age >= 19 can drink alcohol – EP3
    - Step 1: Enter age
    - Step 2: Press Check button
    - Expect result: get a message says “You can drink alcohol”
  + Verify age from 1- 18 cannot drink alcohol – EP2
    - Step 1: Enter age
    - Step 2: Press Check button
    - Expect result: get a message says “You cannot drink alcohol”
  + Verify the system can handle the age not in normal range – EP 1 & 4
    - Step 1: Enter age
    - Step 2: Press Check button
    - Expect result: get a message says “System cannot accept the age”
  + Verify the system can handle the invalid input – EP5 & 6 & 7
    - Step 1: Enter age
    - Step 2: Press Check button
    - Expect result: get a message says “Your input form is not valid”
  + Verify the system can handle integer overflow – EP8
    - Step 1: Enter age
    - Step 2: Press Check button
    - Expect result: get a message says “Integer Overflow”

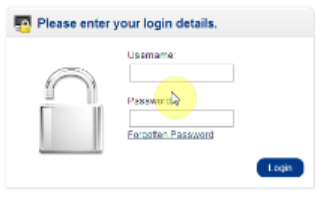
Write Test Case based on UI – input box



1. Integer
   1. Range (min - max)
      1. In range (min - max) + BV
      2. Out of range
      3. Analysis of result from business requirement, like -1 cannot be input as quantity
      4. 0 -> number you need to think of it
      5. Integer overflow (32 bit, 2^31 - 1)
   2. Format
      1. Blank
      2. String
      3. Float number
2. Float number
   1. Range (min - max)
      1. In range (min - max) + BV
      2. Out of range
      3. Analysis of result from business requirement, like money, 2 decimal digits only, 0.01 is the minimal amount of money, and lower & upper limit of transferring money
      4. 0 -> number you need to think of it
   2. Format
      1. Blank
      2. String
      3. How many “.”, like (“10.”, “.10”, “10..”)
3. String
   1. Range (min length – max length)
      1. In range (ex. username length is 4 - 20)
      2. Out of range
      3. Analysis of result from business requirement
   2. Format
      1. Blank
      2. Capital Letter or Lowercase, like [JAMES@gmail.com](mailto:JAMES@gmail.com) and [james@gmail.com](mailto:james@gmail.com) are the same, but passwords are different
      3. Special character, like `~!@#$%^&\*()\_+ and space “ ”
4. Date
   1. Range (Start date – End date)
      1. In range
      2. Out of range
      3. End date > Start date
      4. Today’s date should be considered as special case
      5. Analysis of result from business requirement
   2. Format
      1. US: mm/dd/yyyy
      2. CA: dd/mm/yyyy
      3. International: yyyy-mm-dd
      4. Blank
      5. Invalid date, like 13/13/2013, 12/32/2014, 2/29/2015

Practice: Write a Test Case for the following UI of login page (Should be finished within 5 minutes)

Req: Username is Email address (length: 6 - 20), Password (length: 6 - 40), and ignore the link of forget password.



Recall Test Case of String from past page to generate Equivalence Positioning

* Email
  + 6 – 20
    - Jamesl@gmail.com
* 1 – 5
  + a@b.c
* 21 – 256 (If no upper limit informed, 256 usually is chosen because of database design)
  + aaaaaaaaaaaaaa@gmail.com
* Blank
  + ‘ ‘
* Lower case and upper case are the same
  + JAMESL@gmail.com
  + Jamesl@gmail.com
  + jamesl@gmail.com
  + jamesl@gmail.COM
  + jamesl@GAMIL.com
* Special character (for security testing)
  + Not allowed: ‘ ! ~ % ^ & ( )
* Email format, like aaa@bbb.ccc
  + Allowed:
    - jamesl@gmail.com
    - jamesl@markham.on.ca
  + Not allowed:
    - jamesl@
    - jamesl@gmail.
    - jamesl@markham.on.
    - Jameslgmail.com
    - @gmail.com
    - jamesl@.com
* Password
  + 6 – 40
    - 123456
* 1 – 5
  + 12345
* 41 – 256
  + 1234567890123456789012345678901234567890
* Blank
* Lower case and upper case are different
  + Abcdefg
  + ABCDEFG
  + Abcdefg
* Special character (for security testing)
  + Not allowed: ‘ ! ~ % ^ & ( )
* Input should be shown like \*\* \*\*\*\*\*\*\*\*
  + Complex (not in example) -> Not considered for this case now

Test Case (data, steps, and expect result should be included for each)

1. Verify the user can login successfully with email address and password
   1. Data:
      1. jamesl@gmail.com
      2. 123456
   2. Steps:
      1. Input email address
      2. Input password
      3. Click “login” button
   3. Expect result:
      1. Login success to dashboard page
2. Verify user cannot login if email address and password mismatch
   1. Data:
      1. jamesl@gmail.com
      2. 12345
3. Verify password and email address cannot be empty
   1. Data:
      1. Email address = ‘ ‘, password = ‘123456’
      2. Email address = ‘jamesl@gmail.com’, password = ‘ ‘
4. Verify email address out of range
5. Verify password out of range
6. Verify email address is case-insensitive (lower = upper)
7. Verify email address is not case-insensitive (lower = upper)
8. Verify email address and password are not allowed special character
9. Verify email format
10. Verify password display “\*\*\*\*\*\*\*\*\*\*”
11. Verify user cannot access the resource protected by login page directly