INSTRUCTURE

INSTRUCTURE

Testing with Unknown Requirements

A Creative Adventure!

Presenter: Indira Pai , SET, Instructure

Warming up...

Requirements

Requirements:

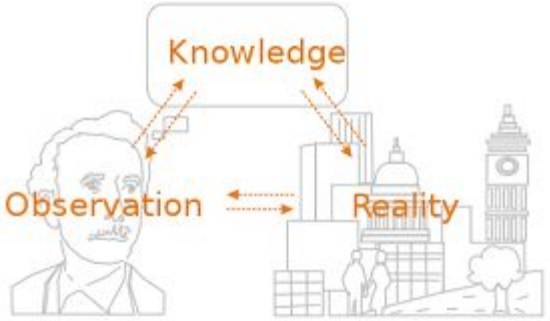
- Also known as
  - PRD: Product Requirements Document
  - SRS : Software Requirements Specification
- A comprehensive document that fully describes what the software will do and how it will be expected to perform.

Expected

- Expected Result:
  - Expected result is the functionality/result that is expected for the correct functioning of the application.
  - It is usually mentioned in the requirement specification documents.

Actual

- Actual Result:
  - Actual result is what we see in the application while testing.



Defect

- Defect:
  - The deviation between the expected and actual result, if any, is known as **defect.**

Without Requirements

- Without Requirements…
  - We don't really know what to test
  - We lose the comfort of having a point of reference, a result to compare against.

Why unknown requirements?

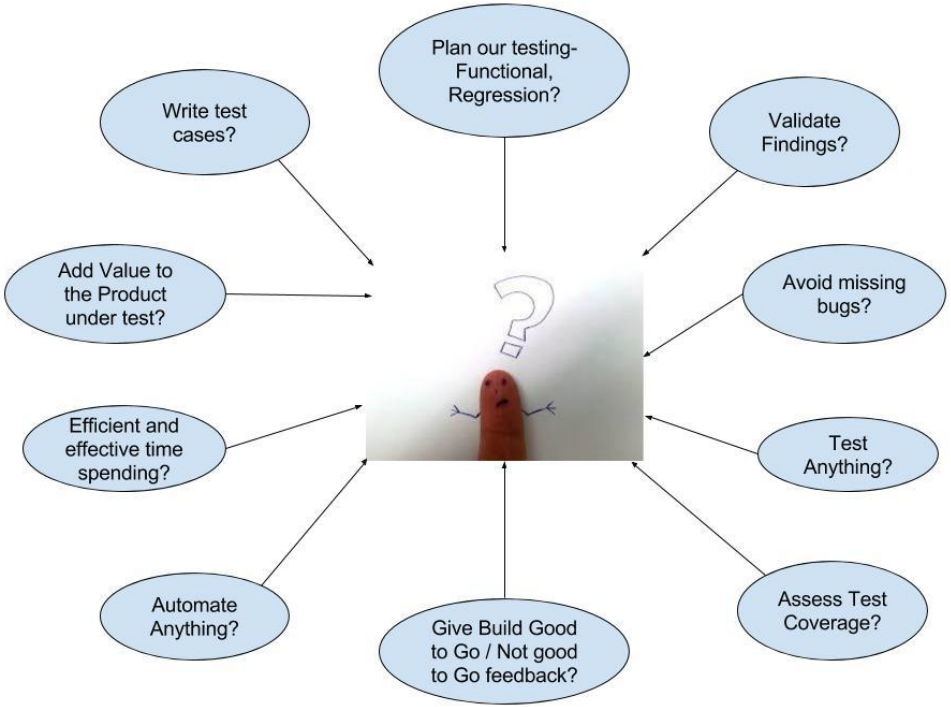
Lack of Documentation

- No documentation was not done at all
- Lots of documentation but none in sync with each other.
- General trend towards agile and scrum in the industry can often remove traditional checkpoints and artifacts for requirements

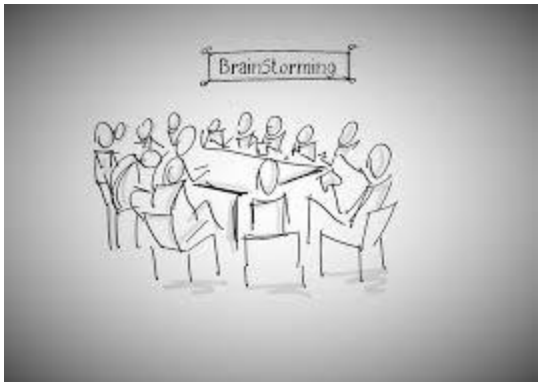
Lack of insight

- Complex computations which yield a result obtained by a complex formula/algorithm usage.
  - There are endless possibilities on what the input can be,
  - But no point of reference/ comparison for expected output,
  - Since it’s not possible to gauge that for all numbers/different ranges.

The Challenges of the UNKNOWN...



So what do we do?



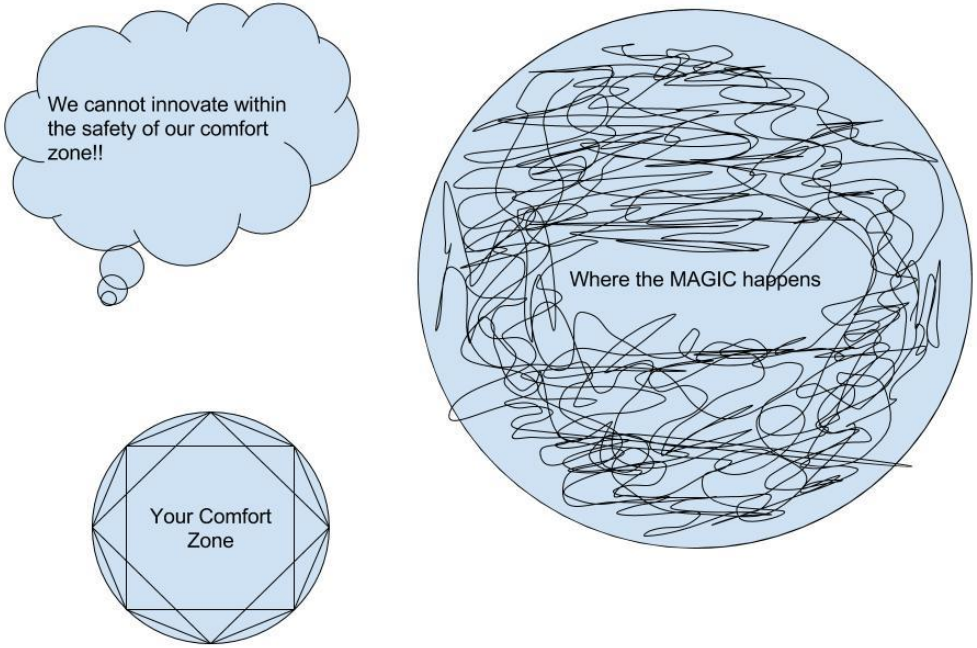
Gather Information

- Help Documents/ guides/ FAQs/Test cases/SRS of similar projects.
- Read up on news articles, white papers, blogs on the software.

Talk, Reach out

- Peers, Stakeholders, internal end users
- Requirements point of contact
- Projects with similar architecture to gain technical insight
- Projects with similar work methodologies to gain insight on how did they handle such situations.

Err… Why should we be excited/inspired/ :-))?



There are things known and there are things unknown, and in between are the doors of perception…

Aldous Huxley

Some other approaches

Comparative Study

- Compare the product with similar products and take a test run based on this comparison study
- Example: Online stores (Amazon/ Ebay)

Exploratory Testing

- Start exploring the software and check for ambiguous behaviour.
- Example: Browser Crashes on clicking Logout link.

Walkthrough Code

- Dig through the automation tests to figure out expected for scenarios which have been automated.
- **Dig through the Pseudo Code , and the code of the software under test and understand white-box analyse.**

UX Testing

- Explore the software as an end user
- List out pain points/annoyance which could possibly make it difficult to use
- Example: The “Logout” link is hidden in the current color scheme, not clearly visible.

I18N testing

- Internationalization is an often requested feature/support.
- Change browser default language and verify if software adapts to the change in language.
- Try this with different languages

A11y Testing

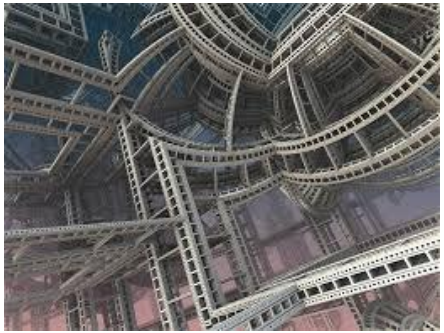
- **Accessibility Testing** deals with testing how accessible the software is to an end user with disabilities.
- Keyboard only navigation, Voice Over, Different Screen Readers are used to test this.
- We have a meetup in Dec where our accessibility expert QA will talk about this in more detail: https://goo.gl/icW1TI

Feedback

- **Propose/intuit what YOU think the requirements should be if you were the PM or dev based on your analysis.**

When the unknown is much more complex

- Examples:
  - Billing systems for 30 + million customers which have many offers, plans, freebies for rating and billing day and month wise
  - Mortgaging /Accounting portfolio systems with portfolio index
  - Health indexing systems with calculations based on different reports for million patients
  - Score calculations in various competitions/ sports tournaments



Introducing: Test Oracle

- A mechanism used for determining whether a test has passed or failed, where:
  - The system has a lot of complex computational logic
  - The expected results are not known.

Test Oracle : how it works

- Building/developing/coding
  - A computation engine exactly as the system under test
- Comparing
  - The output(s) of the system under test, for a given test case input,
  - To the outputs that the oracle determines that product should have.
  - If there is a deviation found, the problem is either in the SUT (Dev's code) or the Test Oracle (mechanism built to test this complex computation)
  - If there is no deviation found, revisit logic to make sure both pieces of code dont have the same bug.

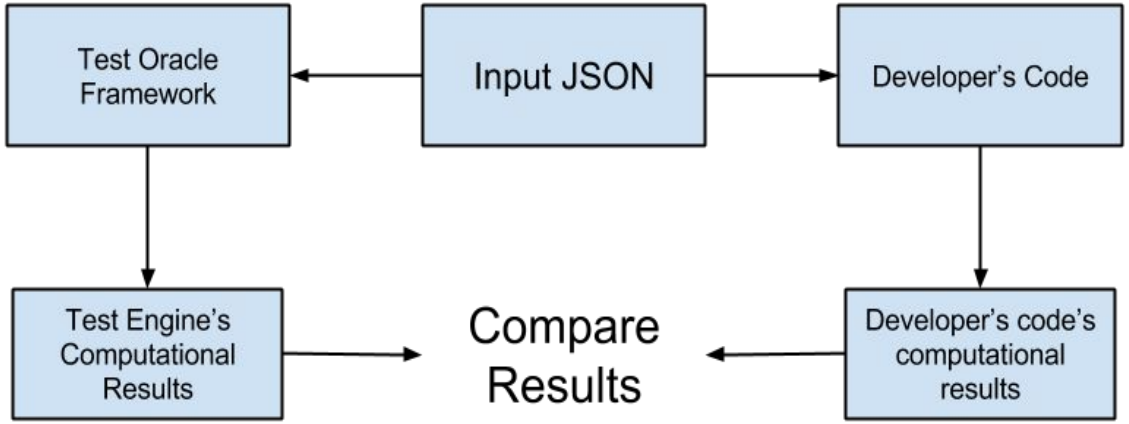
When Test Oracle?

- This is an ideal solution where system under test has the following:
  - Enormous amounts of input data processing for various ranges
  - Complex computations with input data with lot of conditions
  - Expected of the test is unknown
  - Computation algorithms of system under test is not subject to very frequent requirement changes
  - Input needs to be generated from the test oracle

Examples for Test Oracle usage

- Case Study: https://goo.gl/xP8wc9
- Some notes :
  - Application under test involves a computation engine.
  - The result of the engine is computed from a formula/algorithm post the input is provided.
  - A huge set of input data is involved, with multiple conditions:
    - First 50 inputs : Follow formula 1
    - From 51 to 500 : Follow Formula 2
    - From 500 to 5000 : Follow Formula 3 (and so on..)
  - Lots of permutations/combinations possible
  - No expected value to compare results for every condition.

Test Oracle: Testing Mechanism



Test Oracle: Pros

- Finding the following early in dev code:
  - Optimization issues
  - Functional issues
- Testing attained 100% coverage
  - Each and every calculation parameter for each and every condition was compared and tested thoroughly.
- Thorough understanding of the module and the business logic was achieved
  - Helped in debugging issues.
- Quicker turnaround of fixes and better code quality

Test Oracle: Cons

- Additional Time and Resource overhead for building test oracle framework
- Every change in Business Logic required test oracle as well as dev code to be changed.
  - Overhead of changing the code
  - Regression of test oracle as well as dev code needed

It’s time to give back! (Process Feedback)

- On the basis of test runs taken for different approaches discussed, submit feedback to your team, on what can be done to make the software much more awesome!!

Add to Automation

- Automate conditions which can be made out obviously :
  - Logout link color hex code should not be exactly same as the background.
  - Click on any visible element should not cause exceptions.
  - Use this analysis to build out on more automation

Conclusion

Getting Requirements documented is very important for team operation, as it helps everyone be on the same page.

However, depending on the given set of conditions, we have the power to evolve our ideas, and transform our challenges into a goldmine of opportunities!!

Thank You!
Happy Testing!!

Email me @ :
ipai@instructure.com