Noggin Tech Challenge

Contents

Agile Testing

Defect Priority and Severity

Question: What is the difference between defect **priority** and **severity**?

Answer:

Priority defines how quickly we need to resolve/fix a defect based on how often a defect may cause issues to the application. E.g's:

- **Low priority**: defect can be resolved later.
- **High priority**: defect should be resolved quickly.

Severity defines how much of an impact a defect is having on the application (how severe it is).

- **Low severity**: defect is not causing major disruption to core/key user journeys. User still able to use the application (low business impact).

- **High severity**: defect is causing major disruption to core/key user journeys and is potentially preventing user from using the application (high business impact).

Defect examples

Question: Assuming that you are testing Noggin Home Page, provide examples for:

- A high priority and high severity defect
- A high priority and low severity defect
- A high Severity and low priority defect
- A low Priority and low severity defect

Answers:

- **A critical priority and critical severity defect:**

  500 server error when loading the page

- **A high priority and high severity defect:**

  These issues are preventing the user from using the page i.e. stopping core user journeys e.g's:

  - Unable to login or access support
  - Learn More or Request a Demo links not redirecting user to appropriate pages
  - drop down of `.mega-menu-popout-container` elements not working (links not working on hover)

- **A high priority and low severity defect:**

  These defects are not breaking core user journeys but are seriously affecting the performance of the page e.g:

  - `.home-banner-cm` not loading correctly
  - Fonts/colours/images not loading correctly
  - typos in the one of the `.menu-container .hs-menu-item > a` elements
  - images in the header not loading

- **A high Severity and low priority defect** These defects are breaking non-core user journeys e.g:

  - Subscribe button not working
  - Marketing checkbox in the footer not clickable
  - White papers not readable (downloadable), links not working

- **A low Priority and low severity defect**

  - Typos on the page such as the 2nd `.three-col-container .hs_cos_wrapper p` element (Work Safety) has **induries** but it should be **injuries**
  - Padding issues, minor misalignment of elements, things that affect **a11y** of page

Testing new Release

Question: When testing a new release, if you have completed majority of testing for the release and you are on a tight deadline. What are the key factors you would consider if you are to stop further testing of the release?

Answer: So assuming I have completed > 75% tests completion:

- I would leave out **a11y** tests and **seo** testing.
- I would **not** leave out **UI** and **API** tests.
- Depending on the nature of the release, a decision would need to be made on **performance testing**.
- Questions to be be considered in making decsions re the above tests:
  - Have the main features for this release been built and tested successfully i.e have the core tickets for the release been marked as done?
  - Have we run regression tests to ensure the application is not in worse situation than before?
  - What is the risk of leaving the remainder testing for later? Could it have small or large implications?
  - Are minor issues that will arise because testing stops acceptable or not? Ensure that these minor issues are tracked down to be fixed at a later date.

Risk analysis during software testing

Question: What parameters should you consider if you are to perform a risk analysis during software testing and how you prioritise each of them?

Answer: I would consider the following:

- **Business value**: what is the business value will be added when what is being being built is implemented?
- **Business risk**: what is the business risk of what's being built not achieving the business value intended?
- **Regressions**: has regression testing been run to ensure that the stability of the site remains.
- **Severity of defects**: have severe defects been found which increase the risk of what is currently being built?
- **Time to build and test**: How much time has been spent and how much more time is required to build/test? Dev/testers time is an expensive resource.
- **In scope**: Is the outstanding work within the scope of our current focus (are we staying on track and keeping the main thing the main thing).
- **Security**: Have security implications been considered with what is being built? How will it affect the security of the site/application?

Agile testing Methodologies

Question: Name few Agile testing Methodologies and how you have adopted the right testing approach for your previous engagements?

Answer:

- **BDD (Behaviour Driven Development)**: Features are built to satisfy business needs, multiple stakeholders able to view and understand such features. Agile ceremonies are used such as 3 amigos (BA, Dev, Tester). At various clients (such as UsTwo and Pillar) I have been involved in BDD through 3 amigo sessions, building out tests prioritising behaviour and have discussed the desired outcome of features early on.

- **TDD (Test Driven Development):** (Red, Green, Refactor). I have used this to test my own work, by breaking it up into smaller pieces and ensure they are actually doing what I expect them to be doing.
- **CDCT (Consumer Driven Contract Testing)**. I have been involved with a client thinking through the use of CDCT and setting it up with PACT to help test various microservices and improve the stability of microservices.
- **AB Testing** - I have been part of a CRO (Conversation Rate Optimisation) team at a client that has built variants of a page (or pages) in an attempt to enhance a page/journey. In AB testing, a variant is built that changes/updates the look and feel of a page and it's then analysed for a period of time alongside the original page (called control). The experiment is to see if the variant is more successful that the control. It's all about optimising the rate of Users successfully completing journeys (on a page or across multiple pages).
- **ATDD Acceptance Test Driven Development:** User input is key, UAT tests are written, and then application is built to make it pass). As a manual QA at Zopa I was involved in such user input sessions.
- **Exploratory testing:** Manual testing where testing as a user but a very critical user.
- **Session-based testing:** Like exploratory testing but with specific mission in mind, hunting specific bugs in application.

  The Testing Pyramid should always be considered when thinking through the above methodologies 👆

Qualities of good Agile Tester?

Question: What qualities should a good Agile Tester have?

Answer: A tester in an Agile team should have the following competencies:

- Test automation skills (tool agnostic).
- Understand the difference between different testing methodologies and when and where to use each.
- Sharp eye for detail.
- Able to working closely with developers (not working in a silo).
- Not afraid to push back on developers.
- Not afraid to ask lots of questions to ensure acceptance/test requirements are clear.
- As Agile methodology depends heavily on collaboration, communication, and interaction between the team members as well as stakeholders outside the team, testers in an Agile team should have good interpersonal skills and communication.
- Be positive and solution-oriented with team members and stakeholders.
- Be quality-oriented (quality minded), think critically about feature/acceptance criteria and have a healthy skepticism for the product/feature.

- Actively acquire information from stakeholders (rather than relying entirely on written specifications)
- Accurately evaluate and report test results, test progress, and product quality
- Work effectively to define testable user stories, especially acceptance criteria, with customer representatives and stakeholders
- Collaborate within the team, working in pairs with programmers and other team members
- Respond to change quickly, including changing, adding, or improving test cases
- Plan and organise their own work

Scrum and Kanban

Question: What's the difference between Scrum and Kanban?

Answer: Both are agile (iterative and collaborative) strategies for how a team organises itself to manage its workflow. Basically, Kanban is more fluid and Scrum is more structured.

- **Kanban** relies on the team members to dictate velocity of moving tickets fromm **todo** to **done**. The flow of work is a continuous cycle of moving tickets from **todo** to **done**. Team members focus on speed and effieciency and can wear many hats in their goal to achieve this.
- **Scrum** relies on finite sprint cycles (1 or 2 week cycles) where team members commit to an amount of work to be achieved within that sprint. Team member roles are more defined and more agile ceremonies used to maintain structure and plan.

Imperative and Declarative tests

Questions: Give an example of an Imperative & Declarative test scenario in BDD format (Cucumber - Gherkin).

Imperative

Imperarive steps are very specific and describe the step-by-step process of a scenario. Steps are usually shorter and are named accordingly.

```
As a noggin user
I want to login into my acccount
So that I can see my my noggin profile

    GIVEN I navigate to the home page
    WHEN I click on the login link
    THEN I should see the noggin sign in page correctly displayed
    WHEN I input my domain
      AND I click continue
    THEN I should see my account page
```

Declarative

Declarative steps are "bigger picture thinking" and describes behaviour. Steps may contain other sub steps.

```
As a noggin user
I want to login into my acccount
So that I can see my my noggin profile

    GIVEN I’m a registered user
    WHEN I complete the login process
    THEN I should see my account page
```

Key Performance Metrics

Question: What are all the key performance metrics which needs to be captured during Performance testing of a Web Application?

- **Throughput (transactions/test duration):** how many requests/transactions a site/application can process per unit of time.
- **Response time (KB/sec):** Time from request -> server response completing.
- **Requests per second (RPS):** Total number of requests per second.
- **Virtual users (per unit of time):** Helps team estimate an average load as well as behaviour in different load conditions.
- **Error rate(%):** This is the ratio of invalid to valid answers over a period of time. This metric is usually seen when the load exceeds its capacity.
- **Wait time (KB/sec):** AKA average latency, this measures time from request -> the first byte is received (Not to confused with response time which is until last byte is received).
- **Average load time(sec):** Average time from initiation to page loading completely. Very important metric as users may abandon sites if pages takes too long to load.
- **Peak response time:** Max time it may take for a request to be fulfilled. If peak response time > average load time then part of the application can probably be improved (smaller image/library used etc)
- **Concurrent users:** AKA load size. Number of active users at any point.
- **Transactions passed/failed (%):** Percentage of passed or failed tests against the total number of tests.
- **CPU utilisation:** How much time a central processing unit uses to process request.
- **Memory utilisation:** How much resource (physical memory device) it takes to process a request.
- **Total user sessions:** This shows traffic intensity over a period of time e.g. 10 user sessions per hour. This can feature a number of page views and bytes transferred.

Test Automation

Test Automation Challenge:

Automate the following scenario using Selenium Java or JavaScript

Test Automation Requirement:

You should use Page object model framework, Cucumber (BDD framework), Maven (for Java), npm (for JavaScript),

Test Automation Steps:

- Go to https://www.noggin.io - DONE
- Click on 'RESOURCES' - Resource Centre link - DONE
- Validate the page title and few images - DONE
- Filter Resources by Emergency Management -DONE
- Validate if the filtered results are displayed - DONE
- Click on any of the download guide link and check if a new page is opened and validate for any content in that new page

Note: Share the code without the dependencies (dependencies should be mentioned in POM.xml for Java or package.json for JavaScript)

Test Automation Solution

I went with Cypress as my test automation tool of choice. It is a node application and a powerful testing tool. It has implicit waits, live reloading, a list of plugins to customise the application and a large community support base. It has been built with developers in mind so devs shouldn't take long to familiarise themselves with it, thus enabling greater collaboration with devs and testers and some cool shift-left. The feature files have been split according to resolution (@desktop, @desktopXL, @tablet, @mobile). Each feature file is a Scenario Outline with multiple Scenarios. I have been quite comprehensive in my testing of the resources page, shown by the many scenarios. Running all 56 tests across the 4 feature files takes approx 15min.

Running Cypress Tests

```
npm run cy:run:resources      // run all resources tests
npm run cy:open               // run cypress tests in UI mode
npm run cy:run                // run all cypress tests in HEADLESS mode
npm run cy:run:desktop        // run desktop tagged tests in HEADLESS mode
npm run cy:run:desktopXL      // run desktopXL tagged tests in HEADLESS
mode
npm run cy:run:tablet         // run tablet tagged tests in HEADLESS mode
npm run cy:run:mobile         // run mobile tagged tests in HEADLESS mode
npm run generate:html:report  // run after cypress tests (except for
cy:open) for html report to be generated
```

Performance Testing

Performance Testing Challenge:

Write a JMeter script for the below scenario and share the .jmx file

Performance Testing Steps:

- Go to https://www.noggin.io
- Click on 'RESOURCES' - Resource Centre link
- Add Assertions
- Filter Resources by Emergency Management
- Validate the if the filtered results are displayed and add assertions
- Click on any of the download guide link and check if a new page is opened and validate the for any content in that new page by adding an assertion

Performance Testing Conditions:

- Number of threads - 1
- Ramp up time - 5 seconds
- Iterations - 5
- Listener - Summary report

Performance Testing Solution

I created a JMeter script as per the requirements. Prior to this, I had only very briefly used JMeter. I enjoyed improving my skills using JMeter and have become aware at just how powerful this mature performance testing tool is. As an alernative, I created a script with K6, a highly rated Tool written in Go but which is implemented in JavaScript. I really enjoyed using K6! It has also been built with developers and shift-left in mind, with the hope to encourage devs to write Performance tests early on. I enjoyed writing the script in JS and am really excited to use it more going forward.

Running Performance Tests

```
npm run k6:run                  // run K6 performance tests in HEADLESS mode
jmeter -n -t perf/jmeter/Noggin-Users-Performance-Test.jmx -l
perf/jmeter/Noggin-Users-Performance-Results
```

Notes:

- After running the K6 script, an HTML report will be generated in the `perf/k6` directory in a file called `noggin-k6-performance-summary.html`. Open it up and take a look 👀
- After running the JMeter script, the results will be generated in the `perf/jmeter` directory in a file called `Noggin-Users-Performance-Results`

TODO

The following is a list of things I didn't get round to completeing in this Tech Challenge, but which I intend on delving into/setting up.

- **Performance Testing:** I would like to spend some more time playing with the following tools. I have seen great reviews and I feel like that could be extemely powerful and useful. I wonder if Noggin

have considered some of the following tools as an alternative to JMeter?
  - **K6:** I have really enjoyed using this tool and I will work on improve and build up my skills with K6. I found this to be a very user friendly tool and one that works very well with a node applications.
  - **Artillery:** I have used Artillery before and have been impresed with how user freindly it is. I really wanted to put together an Artillery Performance test for this challenge but I have run out of time. It's another popular performance testing tool that may prove very useful.
  - **Taurus:** (**T**est **A**utomation **R**unning **S**moothly) is an open source automation tool created by Blazemeter which provides an abstraction layer over existing Peformance Testing tools such as JMeter, Grinder, Gatling.
  - Install **cypress-audit** setup in the project for a11y and Lighthouse performance testing
- **Visual Regression Testing**: Install **cypress-image-screenshot** for regression testing