# 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 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

- **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:

  - Errors in the header or .home-banner-cm elements e.g.
  - Fonts/colours/images not loading correctly
  - drop down of `.mega-menu-popout-container` elements not working (links not working on hover)
  - 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 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 (core tickets for release are 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 arise because testing has not been completed acceptable or not?
  - As long as you can track them down and fix them 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
- Regressions
- Severity of issues
- Time to build
- Time to test
- Is the work within the scope of our current focus (are we staying on track and keeping the main thing the main thing).

**Agile testing Methodologies**

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

**Answer**:

- Test Driven Development TDD (Red, Green, Refactor)

- BDD Behaviour Driven Development (feature built to satisfy business need, multiple stakeholders able to view and understand features. 3 amigos)

- Contract Testing (CDCT Consumer Driven Contract Testing)

- AB Testing

- ATDD Acceptance Test Driven Development (User input is key, write UAT test, make it pass)

- 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 (and tool agnostic)
- Understand the difference between different testing methodologies and when and where to use each.
- Sharp eye for detail
- Working closely with developers (not working in silos)
- 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.
- Be positive and solution-oriented with team members and stakeholders
- Display critical, quality-oriented, skeptical thinking about the product
- 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 form todo to done. The flow of work is a continuous cycle of moving tickets from todo to done Team member roles are not clearly defined (members can wear many hats)
- **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**

Steps are very specific which describe the process. 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 (describes implementation)
    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 is bigger picture thinking which 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?

- **Response time (KB/sec):** Time from request -> server response completing
- **Requests per second (RPS):** Total number of requests per second
- **User transactions:** User transactions are a sequence of user actions via a software interface e.g.
- **Virtual users per unit of time:** Helps team estimate an average load as well as software 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.
- **Throughput (KB/sec):** bandwidth used during test i.e. max amount of data flowing through a particular network connection within a given amount of time. The number of concurrent users will affect this.
- **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.

This is an example from simple JMeter Test showing some of these key metrics:

- Max Users e.g. 5VU
- Avg. Throughput 11.98Hits/s
- Errors 0%
- Avg. Response Time 321.08ms
- 90% Response Time 436ms
- Avg. Bandwidth 3.28MiB/s

## Test Automation

**Challenge:** Automate the following scenario using Selenium Java or JavaScript

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

**Steps:**

- Go to https://www.noggin.io
- Click on 'RESOURCES' - Resource Centre link
- Validate the page title and few images
- Filter Resources by Emergency Management
- Validate if the filtered results are displayed
- 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)**

## Performance Testing

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

**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
- Conditions:
  - Number of threads - 1

    - Ramp up time - 5 seconds
    - Iterations - 5
    - Listener - Summary report

### Running Tests

Here are the test scripts for running the various types of testing:

```
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 k6:run            // run K6 performance tests in HEADLESS mode
```

### TODO**

- Upate Agile Testing especially Agile Testing Methodologies
- Cypress test suite completed but needs improving in the following ways:
    - Need to be able to navigate to another tab with Puppeteer
    - cypress-audit for a11y and performance testing
    - cypress-image-screenshot for regression testing
    - mochawesome reports
- Puppetteer test suite option
- JMeter:
    - Complete Blazemeter University
    - Update JMeter
- K6 POC
- Artillery POC

## Bugs found on the site

- induries instead of injuries