Piotr Owczarek

Odpowiedzi do zadania 1:

1. Testy funkcjonalne od niefunkcjonalnych przede wszystkim różnią się tym że:
    a. Testy **funkcjonalne** obejmują testowanie funkcjonalności w systemie lub działaniem użytkownika:
        i. „funkcja mnożenia dwóch liczb – zwraca złe wartości”
        ii. „Klikam przycisk ‘zapisz’ i program przestaje działać
    b. Testy **niefunkcjonalne** obejmują wszystko co nie jest działaniem użytkownika lub funkcjonalnością, np.:
        i. Bezpieczeństwo
        ii. Ergonomię
        iii. Konwersję danych
        iv. Testowanie instalacji
2.
    a. „Smoke testy” używane są jako typowe pierwsze testy, czy aplikacja zadziała. Czy do każdej funkcji – program ma dostęp. Jednakże ten test nie ma za zadania sprawdzić czy poszczególne funkcjonalności działają poprawnie – ma tylko sprawdzić czy coś się uruchomi. Czy program się zainstaluje itp.
        i. Tak jak napisane wyżej, przed oddaniem kodu do testów, lub już po stronie testera jako pierwszy test.
    b. „Test regresji” to testy, które mają za zadanie sprawdzić, czy kawałek naprawionego kodu nie wywołał błędu w innym miejscu, lub też wykrycia błędów, nie zostały odkryte podczas „naprawy”.
        i. Ten rodzaj testów powstaje na przetestowanym już raz kodzie, by móc znaleźć problemy, które mogą wyniknąć podczas „naprawy”.
3. Celem testowania jest sprawdzenie ilości błędów w kodzie zanim kod trafi na „produkcję” i wpędzi naszą firmę w kłopoty ☺. Poza tym dzięki testom, znacznie przyśpiesza się proces wypuszczenia aplikacji (oczywiście działając w Agile) pozbawionej większości błędów. Chodzi przede wszystkim o znalezienie błędów, które obniżałyby końcową jakość produktu – np.: „Kliknę klawisz ‘dodaj’ a usunęło mi wszystkie produkty, bo coś tam”. Dlatego Testy są ważne. By już na wczesnych etapach produkcji zminimalizować straty w czasie na potrzebę „naprawienia kodu” przez programistę.
4. Jak Tester może sprawdzić, czy błąd został wyeliminowany? Otóż na początek powinien zainstalować ‘łatkę’ od programisty. Następnie wykonać Re-Test, czyli test potwierdzający, że owa łatka załatwia problem, ale to nie wszystko. Następnym punktem jest wykonanie testów regresyjnych, które dadzą „100% pewności” (cudzysłów tutaj ma duże znaczenie, bo nigdy nie można być w pełni pewnym, czy w specyficznych warunkach 1+1 da zawsze dwa ☺, przede wszystkich chodzi o to, że zawsze mogą wystąpić błędy niezależnie od tego ile razy i jak bardzo to sprawdzaliśmy), że owa poprawka nie niszczy programu w innym miejscu.
5. Aby przetestować aplikację, którą mam w ręku użyłbym „atrapy” (*ang. mock*), którą to symulowałbym jakąś konkretną temperaturę. Symulator pozwala mi na dowolne ustawienie danych, jakie przyjmowałby czujnik. Czyli po prostu użyłbym systemu, który „podpiąłbym”

zamiast rzeczywistych danych. Daje mi to gwarancje, że uzyskam taką temperaturę jaką bym chciał, a nie szukał miejsca na świecie, gdzie osiągnę +200 i więcej, oraz nie będę musiał jechać w okolice Syberii, by zmierzyć zachowanie czujnika w temperaturach bardzo niskich. Poza tym istnieje jeszcze jedna rzecz. O ile samo „Mock'owanie" jest dobrym pomysłem, tak pozostaje jeszcze inna kwestia – oprogramowania. A więc tak:

a. Różne systemy mogą różnie reagować na zmianę temperatury (Android czy IOS mogą inaczej się zachowywać) dlatego warto zastanowić się nad Emulatorem lub symulatorem danego urządzenia.
b. O ile sam Emulator jest dobry dla androida, ponieważ jest to najbardziej core'owa rzecz łącząca większość androidów, tak dla IOS warto pomyśleć nad symolatorami, ponieważ niejednokrotnie będą dobrze działały te innych firm, a poza tym jest tylko kilka urządzeń spod marki Apple, więc niewiele różni się w nich system.

Najważniejsze jest samo przygotowanie, otóż zanim użyjemy mock'a – warto przygotować spreparowany mockiem symulator/emulator, na nim odpalić apkę, dla której środowisko będzie naturalnym, a to, że sami będziemy mogli zmienić temperaturę to już od nas zależy, a sama aplikacja będzie po prostu odbierała to jako gotowe paczki danych (jakby myślała to wpadłaby na to, że coś jest nie tak – więc cieszmy się, że aplikacje nie są tak mądre ☺). Temperatury w atrapie można ustawić dowolnie, ale skoro czujnik powinien działać -50 – 200 to wystarczą wartości brzegowe. Oraz przy okazji można sprawdzić odporność na błędy, gdyby paczki danych były niepoprawne, wysyłały by jakieś bzdury – to czy aplikacja sobie z tym poradzi.

6. Aby pokryć wszystkie możliwości potrzeba 4 testów:

SPOSÓB 1	**SPOSÓB 2**	**SPOSÓB 3**	**SPOSÓB 4**
A > 0	A > 0	A <= 0	A <= 0
B = 3	B != 3	B = 3	B != 3

7. 17, 18, 60, 61
8.
    a. Czy pole z hasłem ma być „gwiazdkowane" (<input type=password>) ?
    b. Czy pole z loginem może zawierać cyfry, znaki specjalne, spację ?
    c. Przykładowy login i hasło pozwalające na dostanie się do bazy
    d. Czy Hasło ma specjalne „wymagania", tj.: Duże/małe litery, cyfry, znaki specjalne
    e. Na wszelki wypadek – czy posiadają kopię bazy
    f. Czy Baza danych jest 'online' to znaczy działanie na żywym organizmie – aplikacji która aktywnie korzysta z tej bazy
9.
    a. GET:
        i. metoda pozwalająca na przesyłanie wartości parametrów w sposób widoczny dla użytkownika.
        ii. Posiada ograniczoną długość wartości parametrów

iii. Może być trzymana w pamięci

iv. Można odwołać się poprzez historię przeglądarki

v. Nie powinna być nigdy używana do przesyłania danych z formularza, jak również każdych innych danych, które muszę spełniać zakładane wymogi bezpieczeństwa

vi. Nie zmienia stanu danych po stronie serwera

vii. Każde zapytanie jest

b. POST:

i. Nie jest trzymana w pamięci

ii. Nie jest trzymana w historii przeglądarki

iii. Nie posiada ograniczenia co do długości danych

iv. Dane idą „pod spodem" przez co nie można jawnie powiedzieć co zostało przesłane, bez odpowiednich metod (np.: echo '$value1'; w php) przez co jest trochę bardziej bezpieczna.

10. Nie, jest protokołem bezstanowym, gdzie każde zapytanie to osobna transakcja nie mająca żadnych powiązań do poprzedniej.
11. W SQL różnica jest następująca:
    a. LEFT JOIN - Klucze A można powiązać (czyli dołączyć) z kluczami B, ale tylko z tymi, które posiada ją wspólne klucze z A, to znaczy:
        i. A: klucze(1,2,3,4,5), B:klucze(3,4,5,6,7,8) => left join połączy je w sposób następujący: AB:klucze(1,2,3,4,5)
    b. INNER JOIN – Zostaną wybrane tylko te klucze które występują w tabeli A i B:
        i. A: klucze(1,2,3,4,5), B:klucze(3,4,5,6,7,8) => inner left join połączy je w sposób następujący: AB:klucze(3,4,5)
12. /etc
13.
    a. Kopiowanie elementu (zwykłe tak obok siebie kopia leży)
    b. Kopiowanie elementu wraz z usuwaniem bazowego pliku (lub bezpieczniej – przeniesienie go do innego folderu)
    c. Kopiowanie folderu i sprawdzenie, czy zawartość pozostała taka sama w obu folderach
    d. Kopiowanie folderu wraz z usuwaniem bazowego folderu
    e. Kopiowanie elementu połączonego dowiązaniem symbolicznym z innym plikiem
    f. Kopiowanie elementu połączonego dowiązaniem symbolicznym z innym plikiem wraz z usunięciem bliku bazowego lub pliku z dowiązaniem
    g. Kopiowanie pliku z dowiązaniem stałym
    h. Kopiowanie pliku z dowiązaniem stałym oraz usunięcie pliku bazowego