Pytania:

1. Jakie są typowe cele testowania?

- zapobieganie defektom poprzez dokonywanie oceny produktów pracy, takich jak:
  wymagania, historyjki użytkownika, projekt i kod;
- weryfikacja, czy zostały spełnione wszystkie wyspecyfikowane wymagania
- sprawdzanie, czy przedmiot testów jest kompletny i walidowanie, czy działa zgodnie z oczekiwaniami użytkowników i innych interesariuszy;
- budowanie zaufania do poziomu jakości przedmiotu testów;
- wykrywanie defektów i awarii, a tym samym zmniejszenie poziomu ryzyka związanego z niedostateczną jakością oprogramowania;
- dostarczanie interesariuszom informacji niezbędnych do podejmowania świadomych decyzji
  (dotyczących zwłaszcza poziomu jakości przedmiotu testów);
- przestrzeganie wymagań wynikających z umów, przepisów prawa i norm/standardów i/lub sprawdzanie, czy obiekt testów jest zgodny z tymi wymaganiami lub standardami.

2. Czym się różni testowanie od debugowania?

- Testowanie pozwala ujawnić awarie, które są skutkiem defektów w oprogramowaniu. Następnie wykonywane jest testowanie potwierdzające (Retest), które pozwala sprawdzić, czy wprowadzone poprawki w rezultacie spowodowały usunięcie defektów.
- Debugowanie to czynność związana z wytwarzaniem oprogramowania, która polega na znajdowaniu, analizowaniu i usuwaniu tych defektów.

3. Dlaczego testowanie jest niezbędne?

- zmniejszenie ryzyka wystąpienia awarii podczas eksploatacji oprogramowania
- Wykrycie, a następnie usunięcie defektów, przyczynia się do podniesienia jakości modułów lub systemów.
- Testowanie oprogramowania może być niezbędne do spełnienia wymagań wynikających z umów, przepisów prawa bądź norm/standardów branżowych
- Nieprawidłowe funkcjonowanie oprogramowania może powodować wiele problemów, w tym straty finansowe, stratę czasu, utratę reputacji firmy, a nawet utratę zdrowia lub życia.

4. Jaka jest relacja między testowaniem a zapewnieniem jakości?

Zapewnienie jakości ma "wyższe cele od testowania". kiedy testowanie koncentruje się na pojedynczym produkcie, zapewnienie jakości dba o jakość wszystkich wytwarzanych produktów.

5. Jaka jest różnica między pomyłką, defektem a awarią?

Pomyłki mogą pojawiać się z wielu powodów, takich jak:

- presja czasu;
- omylność człowieka;
- brak doświadczenia lub niedostateczne umiejętności uczestników projektu;
- złożoność kodu, projektu, architektury, rozwiązywanego problemu i/lub wykorzystywanej technologii;
- nieporozumienia dotyczące interfejsów wewnątrz systemu i między systemami, zwłaszcza w przypadku dużej liczby tych systemów;
- stosowanie nowych, nieznanych technologii.

Defekt - skutek pomyłki (błędu) człowieka w kodzie oprogramowania lub w innym związanym z nim produkcie pracy.

Awaria – wynika z defektów w kodzie.

6. Jakie są zasady testowania?

- Testowanie ujawnia usterki, ale nie może dowieść ich braku
- Testowanie gruntowne jest niemożliwe
- Wczesne testowanie oszczędza czas i pieniądze
- Kumulowanie się defektów - przewidywane skupiska defektów
- Paradoks pestycydów - Ciągłe powtarzanie tych samych testów prowadzi do sytuacji, w której przestają one w pewnym momencie wykrywać nowe defekty
- Testowanie zależy od kontekstu
- Przekonanie o braku błędów jest błędem!!!

7. Jakie czynności są niezbędne w procesie testowym?

- **Planowanie testów** obejmuje czynności, których efektem jest zdefiniowanie celów testowania oraz określenie podejścia do osiągania celów testowania w granicach wyznaczonych przez kontekst
- **Monitorowanie testów** polega na ciągłym porównywaniu rzeczywistego z zaplanowanym postępem testowania przy użyciu miar specjalnie w tym celu zdefiniowanych w planie testów.
- **Nadzór nad testami** polega na podejmowaniu działań, które są niezbędne do osiągnięcia celów wyznaczonych w planie testów (z uwzględnieniem jego ewentualnych aktualizacji).
- **Analiza testów** służy do ustalenia tego, „co” należy przetestować.
- **Projektowanie testów** odpowiada na pytanie: „jak należy testować?”.
- **Implementacja testów** odpowiada na pytanie: „czy mamy wszystko, co jest potrzebne do uruchomienia testów?”.
- **Wykonywanie testów** polega na uruchamianiu są zestawów testowych, zgodnie z harmonogramem wykonywania testów.
- **Ukończenie testów** - tworzenie raportu z testów, przeanalizowanie wniosków

8. Jakie są czynniki psychologiczne wpływające na powodzenie testowania?

Identyfikowanie defektów podczas testowania statycznego, bądź identyfikowanie awarii w trakcie testowania dynamicznego może być odbierane jako krytyka produktu lub jego autora. Programistom trudno jest zaakceptować fakt, że ich kod jest błędny. Czynnik utrudniający przyjęcie informacji zwrotnej z testów stanowi skłonność wielu osób do obwiniania osoby przynoszącej złe wiadomości. Należy zacząć od współpracy, a nie od konfliktu. Wszystkich powinna łączyć świadomość wspólnego celu, jakim jest podnoszenie jakości systemów.

- Należy podkreślić korzyści wynikające z testowania.
- Należy wczuć się w sytuację drugiej osoby i zrozumieć, dlaczego negatywnie reaguje ona na podane informacje.
- Należy upewnić się, że rozmówca rozumie przekazywane informacje i vice versa.
- Należy zadbać o to, by testerzy przestrzegali przyjętych założeń, a ich osobiste nastawienie w jak najmniejszym stopniu wpływało na wykonywaną pracę.

9. Jakie są relacje między czynnościami związanymi z wytwarzaniem oprogramowania a czynnościami testowymi w cyklu życia oprogramowania?

- dla każdej czynności związanej z wytwarzaniem oprogramowania istnieje odpowiadająca jej czynność testowa;
- każdy poziom testów ma przypisane cele odpowiednie do tego poziomu;
- analizę i projektowanie testów na potrzeby danego poziomu testów należy rozpocząć podczas wykonywania odpowiadającej danemu poziomowi czynności związanej z wytwarzaniem oprogramowania;
- testerzy powinni uczestniczyć w dyskusjach dotyczących definiowania i doprecyzowywania wymagań i założeń projektu oraz w przeglądach produktów pracy (np. wymagań, założeń projektu czy też historyjek użytkownika) natychmiast po udostępnieniu wersji roboczych odpowiednich dokumentów.

10. Jakie znasz modele cyklu życia oprogramowania?

- **Waterfall** - w modelu kaskadowym czynności związane z wytwarzaniem oprogramowania (takie jak: analiza wymagań, projektowanie, tworzenie kodu czy testowanie) wykonuje się jedna po drugiej. Zgodnie z założeniami tego modelu czynności testowe występują dopiero wtedy, gdy wszystkie inne czynności wytwórcze zostaną ukończone.
- **Model V** - obejmuje poziomy testowania powiązane z poszczególnymi, odpowiadającymi im, fazami wytwarzania oprogramowania, co dodatkowo sprzyja wczesnemu testowaniu. Zakłada integrację procesu testowania z całym procesem wytwarzania oprogramowania, a tym samym wprowadza w życie zasadę wczesnego testowania.
- **Rational Unified Process** (RUP): poszczególne iteracje trwają zwykle stosunkowo długo (np. dwa lub trzy miesiące), a przyrosty funkcjonalności są odpowiednio duże, czyli obejmują np. dwie lub trzy grupy powiązanych funkcjonalności.
- **Scrum** - poszczególne iteracje trwają stosunkowo krótko (np. kilka godzin, dni lub tygodni), a przyrostowe części systemu są odpowiednio małe, czyli obejmują na przykład kilka udoskonaleń i dwie lub trzy nowe funkcjonalności.
- **Kanban** - wdrażany ze stałą lub zmienną długością iteracji umożliwia dostarczenie jednego udoskonalenia lub jednej funkcjonalności naraz (natychmiast po przygotowaniu) bądź zgrupowanie większej liczby funkcjonalności w celu równoczesnego przekazania na środowisko produkcyjne.
- **Model Spiralny** - tworzy się eksperymentalne elementy przyrostowe, które następnie mogą zostać gruntownie przebudowane, a nawet porzucone na dalszych etapach wytwarzania oprogramowania.

11. Jakie znasz poziomy testów?

- **Testowanie modułowe** (zwane także testowaniem jednostkowym, testowaniem komponentów lub testowaniem programu) skupia się na modułach, które można przetestować oddzielnie. Często wykonywane w izolacji od reszty systemu przy czym w takiej sytuacji może być konieczne użycie atrap obiektów (ang. mock object), wirtualizacji usług, jarzm testowych, zaślepek bądź sterowników.
- **Testowanie integracyjne** skupia się na interakcjach między modułami lub systemami. Skupia się na interakcjach i interfejsach między integrowanymi modułami.Testy tego typu mogą również obejmować interakcje z interfejsami dostarczanymi przez organizacje zewnętrzne.
- **Testowanie systemowe** skupia się na zachowaniu i możliwościach całego systemu lub produktu, często z uwzględnieniem całokształtu zadań, jakie może on wykonywać oraz zachowań niefunkcjonalnych, jakie można stwierdzić podczas wykonywania tych zadań.
- **Testowanie akceptacyjne** — podobnie jak testowanie systemowe — skupia się zwykle na zachowaniu i możliwościach całego systemu lub produktu. W wyniku testowania akceptacyjnego mogą powstawać informacje pozwalające ocenić gotowość systemu do wdrożenia i użytkowania przez klienta (użytkownika).

12. Jakie znasz typy testów?

- **Testowanie funkcjonalne** systemu polega na wykonaniu testów, które umożliwiają dokonanie oceny funkcji, jakie system ten powinien realizować. Wymagania funkcjonalne mogą być opisane w produktach pracy takich jak: specyfikacje wymagań biznesowych, opowieści, historyjki użytkownika, przypadki użycia lub specyfikacje funkcjonalne, ale zdarza się również, że występują one w postaci nieudokumentowanej. Funkcje opisują to, „co” powinien robić dany system.
- **Testowanie niefunkcjonalne** - celem jest dokonanie oceny charakterystyk systemów i oprogramowania, takich jak: użyteczność, wydajność, bezpieczeństwo itd. Testowanie niefunkcjonalne pozwala sprawdzić to, „jak dobrze” zachowuje się dany system. Testy powinny odbywać się na jak najwcześniejszym etapie, ponieważ zbyt późne wykrycie defektów niefunkcjonalnych może być bardzo dużym zagrożeniem dla powodzenia projektu.
- **Testowanie białoskrzynkowe** - warunki testowe wyprowadza się na podstawie struktury wewnętrznej lub implementacji danego systemu. Struktura wewnętrzna może obejmować kod, architekturę, przepływy pracy i/lub przepływy danych w obrębie systemu.

13. Jaka jest różnica pomiędzy testowaniem potwierdzającym a testowaniem regresji?

- **Testowanie potwierdzające (Retest)**. Po naprawieniu defektu można przetestować oprogramowanie przy użyciu wszystkich przypadków testowych, które wcześniej nie zostały zaliczone z powodu wystąpienia tego defektu, a powinny zostać ponownie wykonane w nowej wersji oprogramowania.
- **Testowanie regresji.** Istnieje ryzyko, że zmiana wprowadzona w jednej części kodu wpłynie przypadkowo na zachowanie innych części kodu w tym samym module, w innych modułach tego samego systemu, a nawet w innych systemach. Ponadto należy wziąć pod uwagę zmiany dotyczące środowiska, takie jak wprowadzenie nowej wersji systemu operacyjnego lub systemu zarządzania bazami danych.

14. Co to jest testowanie pielęgnacyjne?

- Po wdrożeniu w środowisku produkcyjnym oprogramowanie lub system wymaga dalszej pielęgnacji.
- Pielęgnacja jest niezbędna do utrzymania lub poprawy wymaganych niefunkcjonalnych charakterystyk jakościowych oprogramowania lub systemu przez cały cykl jego życia — zwłaszcza w zakresie takich parametrów jak: wydajność, kompatybilność, niezawodność, bezpieczeństwo i przenaszalność.
- Pielęgnacja może być wykonywana zarówno planowo (w związku z nowymi wersjami), jak i w sposób niezaplanowany (w związku z poprawkami doraźnymi — ang. hotfix).

15. Jakie mogą wystąpić zdarzenia wyzwalające testowanie pielęgnacyjne?

- **Modyfikacja.** Ta kategoria obejmuje między innymi: zaplanowane udoskonalenia, zmiany korekcyjne i awaryjne, zmiany środowiska operacyjnego (np. planowe uaktualnienia systemu operacyjnego lub bazy danych), uaktualnienia oprogramowania do powszechnej sprzedaży oraz poprawki usuwające defekty i słabe punkty zabezpieczeń.
- **Migracja.** Ta kategoria obejmuje między innymi przejście z jednej platformy na inną, co może wiązać się z koniecznością przeprowadzenia testów produkcyjnych nowego środowiska i zmienionego oprogramowania bądź testów konwersji danych
- **Wycofanie.** Ta kategoria dotyczy sytuacji, w której użycie aplikacji dobiega końca. Kiedy aplikacja lub system jest wycofywany, może to wymagać testowania migracji lub archiwizacji danych, jeśli zachodzi potrzeba ich przechowywania przez dłuższy czas;

16. Czym jest testowanie statyczne?

- Testowanie statyczne polega na sprawdzeniu ręcznym (przeglądy) lub analizie automatycznej (analiza statyczna) kodu lub innych dokumentów projektowych bez uruchamiania kodu.
- Analiza statyczna jest szczególnie ważna w przypadku systemów komputerowych krytycznych ze względów bezpieczeństwa (np. systemów stosowanych w lotnictwie, medycynie lub w energetyce jądrowej), ale staje się też coraz bardziej istotna i powszechnie stosowana w innych kontekstach (analiza taka jest np. ważnym elementem testowania zabezpieczeń).
- Pozwala wykryć defekty jeszcze przed rozpoczęciem testowania dynamicznego. Usunięcie defektów wykrytych w początkowym etapie cyklu życia oprogramowania jest często dużo tańsze niż usunięcie defektów wykrytych w kolejnych etapach — zwłaszcza po wdrożeniu oprogramowania i rozpoczęciu jego eksploatacji

17. Jakie są różnice między testowaniem statycznym a dynamicznym?

TESTOWANIE STATYCZNE	TESTOWANIE DYNAMICZNE
nie wymaga uruchomienia oprogramowania	wymaga uruchomienia oprogramowania
wspiera przede wszystkim weryfikację	wspiera przede wszystkim walidację
raczej wyszukuje defekty	raczej wyszukuje awarie
wymaga list kontrolnych i procesów	wymaga przypadków testowych
wykonywane przed kompilacją	wykonywane po kompilacji
koszt znajdywania i naprawy defektów jest niższy	koszt znajdywania i naprawy defektów jest wyższy

18. Jakie mogą zostać zidentyfikowane typowe defekty podczas testowania statycznego?

- defekty w wymaganiach (takie jak: niespójności, niejednoznaczności, wewnętrzne sprzeczności, opuszczenia, nieścisłości, przeoczenia czy elementy nadmiarowe w wymaganiach);
- defekty w projekcie (np. nieefektywne algorytmy lub struktury baz danych, wysoki stopień sprzężenia (ang. coupling) czy mała spójność (ang. cohesion);
- defekty w kodzie (np. zmienne z niezdefiniowanymi wartościami, zmienne zadeklarowane — lecz nigdy nie używane, niedostępny (martwy) kod, powielony kod);
- odchylenia od standardów (np. brak zgodności ze standardami tworzenia kodu);
- niepoprawne specyfikacje interfejsów (np. użycie różnych jednostek miary w systemie wywołującym i systemie wywoływanym);
- słabe punkty zabezpieczeń (np. podatność na przepełnienie bufora);
- luki lub nieścisłości w zakresie śledzenia powiązań między podstawą testów a produktami pracy związanymi z testowaniem lub luki pokrycia (np. brak testów odpowiadających kryteriom akceptacji).

19. Na czym koncentruje się proces przeglądu?

To, na czym koncentruje się przegląd, zależy od uzgodnionych celów przeglądu, takich jak: wykrywanie defektów, poszerzanie wiedzy, edukowanie uczestników (np. testerów i nowych członków zespołu) bądź omawianie określonych zagadnień i podejmowanie decyzji w drodze konsensusu.

Przeglądy mogą mieć różny charakter: od nieformalnego po formalny. Cechą charakterystyczną przeglądów nieformalnych jest to, że nie przebiegają one zgodnie ze zdefiniowanym procesem, a uzyskanych dzięki nim informacji nie trzeba formalnie dokumentować. Z kolei przeglądy formalne są przeprowadzane zgodnie z udokumentowanymi procedurami i z udziałem zespołu o ustalonym wcześniej składzie, a ich rezultaty muszą być obowiązkowo dokumentowane.

20. Jakie kroki zawiera proces przeglądu?

- **Planowanie:** określenie zakresu prac, w tym celu przeglądu i dokumentów (lub ich części) będących jego przedmiotem oraz ocenianych charakterystyk jakościowych.
- **Rozpoczęcie przeglądu** - rozesłanie produktu pracy (w formie papierowej lub elektronicznej) oraz innych materiałów (np.: formularzy dziennika problemów, list kontrolnych i innych powiązanych produktów pracy).
- **Przegląd indywidualny** (tj. indywidualne przygotowanie)
- **Przekazanie informacji o problemach i analiza problemów -** dokonanie oceny wniosków z przeglądu pod kątem kryteriów wyjścia w celu podjęcia decyzjI (odrzucenie produktu pracy, wymagane poważne zmiany, akceptacja - być może z niewielkimi zmianami).
- **Usunięcie defektów i raportowanie**

21. Jakie znasz rodzaje techniki testowania?

- **czarnoskrzynkowe** (zwane również technikami behawioralnymi lub opartymi na specyfikacji)bazują na analizie podstawy testów np. formalnych dokumentach zawierających wymagania, specyfikacje, przypadki użycia, historyjki użytkownika lub opis procesów biznesowych. Koncentrują się na danych wejściowych i wyjściowych przedmiotu testów, bez odwoływania się do jego struktury wewnętrznej.
- **białoskrzynkowe** (zwane także technikami strukturalnymi lub opartymi na strukturze) jest analiza architektury, szczegółowego projektu, struktury wewnętrznej lub kodu przedmiotu testów. W przeciwieństwie do technik czarnoskrzynkowych, białoskrzynkowe techniki testowania koncentrują się na strukturze i przetwarzaniu wewnątrz przedmiotu testów.
- **oparte na doświadczeniu** - pozwalają wykorzystać doświadczenie deweloperów, testerów i użytkowników do projektowania, implementowania i wykonywania testów.

22. Co to jest podział na klasy równoważności?

- Polega to na takim zorganizowaniu wejść do systemu, aby utworzyć grupy, które powodują podobne zachowanie oprogramowania. Dane znajdujące się w określonej klasie równoważności przetwarzane są w ten sam sposób. Technika ta stosowana jest zarówno dla wejść jak i wyjść, wartości wewnętrznych, wartości zależnych od czasu oraz parametrów interfejsów.
- klasy równoważności identyfikuje się dla danych poprawnych, czyli tych wartości, które powinny zostać zaakceptowane przez system
- klasy równoważności identyfikuje się dla danych niepoprawnych, czyli wartości, które system powinien odrzucić

23. Co to jest analiza wartości brzegowych?

- Typową wartością graniczną są wartości minimalna i maksymalna zakresu danych. Technika ta opiera się na założeniu, że istnieje większe prawdopodobieństwo zachowania oprogramowania dla wartości na krawędziach klas równoważności niż wewnątrz klas. Dlatego też warto wykonać testy, które weryfikują zachowanie systemu na krawędziach klas równoważności.
- wartość brzegowa poprawnego przedziału to poprawna wartość brzegowa
- wartość brzegowa niepoprawnego przedziału to niepoprawna wartość brzegowa

24. Czym jest testowanie w oparciu o tablicę decyzyjną?

Kolumny w tabeli – reprezentują reguły biznesowe (w postaci unikalnej kombinacji warunków) i skutki (tj. działanie systemu związane z daną regułą biznesową). Testy projektowane są tak, aby pokryć każdą kolumnę w tablicy, co oznacza wykorzystanie wszystkich kombinacji warunków uruchamiających.

25. Co to jest testowanie pomiędzy stanami?

- Ten rodzaj testów znajduje zastosowanie w przypadku systemów, w których istnieją obiekty mające różne stany lub sam system zachowuje się inaczej w zależności od aktualnych warunków oraz od historii (stanu) = takie zachowanie oprogramowania można opisać diagramem przejść stanów (automatem skończonym).
- Diagramy stanów opisują wszystkie możliwe stany, do których może przejść dany obiekt oraz pokazują jak zmienia się stan obiektu pod wpływem zdarzeń do niego docierających. W większości technik obiektowych diagramy stanu są rysowane dla pojedynczych klas, aby pokazać cały cykl życia pojedynczego obiektu.
- Kryteria warunkujące przejścia: czynności użytkownika, czynności systemu, zdarzenia czasowe.

26. Czym jest testowanie oparte na przypadkach użycia?

- Jest to dokumentacja wszystkich czynności wykonywanych przez Aktora (użytkownika) oraz odpowiadające im zachowania systemu (podmiotu). W takich przypadkach użycia odpowiemy sobie na pytanie, jak na dany scenariusz zareaguje system. Nacisk kładziony jest bardziej na użytkownika niż sam system.
- Przypadek użycia służy do określenia sposobu korzystania z systemu zaprojektowanego do wykonywania określonego zadania. Natomiast przypadek testowy to grupa danych wejściowych do testów, warunków wykonania i oczekiwanych wyników opracowanych dla określonego celu testu.
- Przypadek użycia nie jest uruchamiany ani wykonywany, ponieważ jest to najczęściej tekstowa czy schematyczna prezentacja dokumentu, która określa sposób wykonania określonych zadań.
- Przypadki testowe natomiast są przygotowywane przez testerów oprogramowania w celu walidacji oprogramowania oraz sprawdzenia, czy działa zgodnie on z wymaganiami klienta.

27. Czym jest testowanie instrukcji kodu?

Testowanie instrukcji służy do sprawdzania potencjalnie wykonywalnych instrukcji zawartych w kodzie. Przepływ powinien przejść przez wszystkie bloki instrukcji pseudokodu najkrótszą drogą.

28. Czym jest testowanie i pokrycie decyzji?

Podąża za konkretnym przepływem sterowania przez punkty decyzyjne. Pokrycie decyzji jest mocniejsze niż pokrycie instrukcji. 100% pokrycia decyzji gwarantuje 100% pokrycia instrukcji, ale nie odwrotnie. Podczas testowania decyzji brane są pod uwagę wszystkie bloki instrukcji.

29. Jakie możemy wyróżnić testy oparte na doświadczeniu?

- **Zgadywanie błędów** polega na stworzeniu listy potencjalnych pomyłek, defektów i awarii, a następnie zaprojektowaniu testów pozwalających uwidocznić te awarie i znaleźć pomyłki, które je spowodowały.
- **Testowanie eksploracyjne** polega na projektowaniu, wykonywaniu i rejestrowaniu testów nieformalnych (tj. testów, które nie zostały wcześniej zaprojektowane) oraz dokonywaniu ich oceny w sposób dynamiczny podczas ich wykonywania. Rezultaty testów dostarczają wiedzy na temat modułu lub systemu, a także pomagają tworzyć testy dotyczące obszarów wymagających dalszego przetestowania.
- W **testowaniu w oparciu o listę kontrolną** testerzy projektują, implementują i uruchamiają testy tak, aby pokryć warunki testowe wymienione w liście kontrolnej. Testerzy tworzą nowe listy kontrolne lub rozszerzają istniejące, ale mogą również korzystać z gotowych list kontrolnych bez ich modyfikacji.