Testowanie poziom podstawowy ISTQB by Wojciech Baczyński

1.1 Dlaczego testowanie jest niezbędne

1.1.1 Kandydat potrafi opisać - podając przykłady - w jaki sposób usterka w oprogramowaniu może wyrządzić szkodę osobie, środowisku lub firmie

Oprogramowanie, które działa niepoprawnie może spowodować wiele problemów, stratę pieniędzy, czasu lub utratę reputacji biznesowej. Może nawet spowodować utratę zdrowia lub życia.

1.1.2 Kandydat rozróżnia przyczynę usterki od jej skutków

Przyczyna usterki to najczęściej ludzki błąd, natomiast jej skutek to oprogramowanie działające nie tak jakbyśmy tego chcieli.

1.1.3 Kandydat potrafi podać powody tego, że testowanie jest niezbędne opierając się na przykładach

Rygorystyczne testy systemów i dokumentacji mogą pomóc w zmniejszeniu ryzyka wystąpienia problemów podczas użytkowania oprogramowania i przyczynić się do podniesienia jakości sytemu, jeżeli znalezione usterki zostaną poprawione przed wdrożeniem systemu do użytkowania.

1.1.4 Kandydat potrafi uzasadnić, że testowanie stanowi część zapewnienia jakości i podać przykłady tego, w jaki sposób testowanie przyczynia się do zwiększenia jakości

Za pomocą testów można zmierzyć jakość oprogramowania wyrażoną przez ilość znalezionych usterek zarówno dla funkcjonalnych jak i niefunkcjonalnych wymagań i atrybutów oprogramowania. Testowanie może budować zaufanie do jakości oprogramowania, jeżeli osoby testujące znajdują mało usterek lub nie znajdują ich wcale.

1.1.5 Kandydat potrafi wyjaśnić i porównać pojęcia pomyłka, usterka, awaria oraz odpowiadające im pojęcia błąd i pluskwa, podając przykłady

Człowiek może popełnić błąd (pomyłkę), która powoduje powstanie defektu (usterki, pluskwy) w kodzie programu lub dokumencie. Jeżeli kod zawierający defekt zostaje wykonany, system nie zrobi tego co powinien (lub wykona to czego nie powinien) powodując awarię. Usterki w oprogramowaniu, systemach lub dokumentach mogą prowadzić do wystąpienia awarii, ale nie wszystkie usterki powodują awarie.

1.2 Co to jest testowanie

1.2.1 Kandydat pamięta ogólne cele testowania

Istnieją różne cele testowania

- znajdowanie usterek
- nabieranie zaufania do poziomu jakości
- dostarczanie informacji potrzebnych do podejmowania decyzji

- zapobieganie defektom

1.2.2 Kandydat potrafi podać przykłady celów testowania w różnych fazach cyklu życia oprogramowania

Różne punkty widzenia pozwalają na wzięcie pod uwagę w testach różnych celów. Na przykład w testowaniu wytwórczym (np. testowaniu modułowym, integracyjnym lub systemowym) głównym celem może być wywołanie tylu awarii, ile się da, żeby zidentyfikować i poprawić usterki występujące w oprogramowaniu. W testach akceptacyjnych głównym celem może być potwierdzenie, że system działa tak jak powinien oraz nabranie pewności, że spełnia wymagania. W niektórych przypadkach celem testowania może być ocena jakości oprogramowania (bez intencji naprawiania defektów), by dostarczyć interesariuszom informacji o ryzyku związanym z wydaniem systemu w danej chwili. Testowanie pielęgnacyjne często zawiera testy sprawdzające, czy nie wprowadzono nowych usterek podczas wykonywania zmian. Głównym celem testowania produkcyjnego może być ocena atrybutów systemu takich jak niezawodność lub dostępność.

1.2.3 Kandydat rozróżnia testowanie od debagowania

Debagowanie różni się od testowania. Testowanie dynamiczne może pokazać awarie, których źródłem są usterki. Debagowanie jest czynnością programistyczną, która znajduje, analizuje i umożliwia usunięcie przyczyny awarii.

1.3 Siedem zasad testowania

1.3.1 Kandydat potrafi wyjaśnić siedem zasad testowania

1. Testowania ujawnia usterki
2. Testowanie gruntowne jest niewykonalne
3. Wczesne testowanie
4. Kumulowanie się błędów
5. Paradoks pestycydów - jeżeli te same testy są powtarzane bez zmian, to w końcu przestaną znajdować nowe usterki. Żeby przezwyciężyć "paradoks pestycydów", testy muszą być regularnie przeglądane i uaktualniane.
6. Testowanie zależy od kontekstu - testowanie wykonuje się w różny sposób w różnym kontekście. Na przykład oprogramowanie krytyczne ze względu na bezpieczeństwo testuje się inaczej niż sklep internetowy.
7. Mylne przekonanie o braku błędów - znajdowanie i naprawa usterek nie pomoże, jeżeli produkowany system nie nadaje się do użytkowania lub nie spełnia wymagań i oczekiwań użytkownika.

1.4 Podstawowy proces testowy

1.4.1 Kandydat pamięta pięć podstawowych czynności testowych i odpowiadające im zadania od planowania do zamknięcia testów

Podstawowy proces testowy składa się z następujących czynności

- planowanie i nadzór nad testami Planowanie testów polega na zdefiniowaniu celów testowania i określeniu czynności testowych potrzebnych do wypełnienia misji i celów testowania.
- analiza i projektowanie testów Podczas analizy i projektowania testów ogólne cele testowania przekształcane są w konkretne warunki testowe i przypadki testowe.

- implementacja i wykonanie testów Implementacja i wykonanie testów, to czynność, podczas której specyfikowane są procedury i skrypty testowe przez ustawienie przypadków testowych w konkretnej kolejności oraz dołączenie innych informacji potrzebnych do wykonania testów, konfigurowane jest środowisko testowe oraz wykonywane są testy.
- ocena kryteriów zakończenia i raportowanie Ocena spełnienia kryteriów zakończenia polega na ocenie wykonania testów zgodnie z przyjętymi celami testowania. Powinna ona być wykonywana dla każdego poziomu testowania.
- czynności zamykające test W ramach czynności zamykających testy zbierane są dane z zakończonych czynności testowych, żeby utrwalić doświadczenie, testalia, fakty i liczby. Czynności zamykające testy wykonywane są przy kamieniach milowych projektu takich jak: wydanie systemu, zakończenie (lub anulowanie) projektu testowego, osiągnięcie kamienia milowego, lub zakończenie wydania serwisowego.

1.5 Psychologia testowania

Można uniknąć spięć pomiędzy testerami, a analitykami, projektantami i programistami przez komunikowanie błędów, usterek i awarii w sposób konstruktywny. Ta zasada ma zastosowanie zarówno do defektów znalezionych podczas przeglądów, jak i w testowaniu.

Problemy z komunikacją mogą wystąpić jeżeli testerzy są postrzegani jedynie jako posłańcy przynoszący niechciane informacje o usterkach. Istnieje kilka sposobów na poprawienie komunikacji i relacji pomiędzy testerami i resztą zespołu:

- zacznij od współpracy, a nie od wojny - przypomnij wszystkim, że celem jest wyprodukowanie systemów o lepszej jakości
- komunikuj informacje na temat produktu w sposób neutralny, skoncentrowany na faktach bez krytykowania autora produktu, na przykład pisz obiektywne i konkretne raporty incydentów oraz uwagi z przeglądów
- spróbuj zrozumieć, co druga osoba czuje i dlaczego reaguje tak jak reaguje
- upewnij się, że druga strona zrozumiała, co powiedziałeś i upewnij się, że rozumiesz uwagi drugiej strony

1.5.1 Kandydat pamięta czynniki psychologiczne, od których zależy sukces testowania

Można uniknąć spięć pomiędzy testerami, a analitykami, projektantami i programistami przez komunikowanie błędów, usterek i awarii w sposób konstruktywny. Ta zasada ma zastosowanie zarówno do defektów znalezionych podczas przeglądów, jak i w testowaniu.

1.5.2 Kandydat potrafi pokazać różnice w nastawieniu testera i programisty

Sposób myślenia stosowany podczas testowania i przeglądania różni się od stosowanego podczas rozwoju oprogramowania. Programiści posiadający odpowiednie nastawienie są w stanie testować swój kod, ale zwykle odpowiedzialność za testowanie przekazuje się testerom żeby wzmocnić koncentrację wysiłków oraz uzyskać dodatkowe korzyści, takie jak niezależne spojrzenie wyszkolonych, zawodowych testerów. Testowanie niezależne może być wykonywane na dowolnym poziomie testów.

2.1 Modele wytwarzania oprogramowania

2.1.1 Kandydat potrafi wyjaśnić powiązania pomiędzy rozwojem oprogramowania, czynnościami testowymi oraz produktami w cyklu życia oprogramowania i podać przykłady na podstawie cech projektu i produktu oraz kontekstu

Cztery poziomy testowania to

- testy modułowe (jednostkowe)
- testy integracyjne
- testy systemowe
- testy akceptacyjne

Produkty związane z wytwarzaniem oprogramowania (takie jak scenariusze biznesowe lub przypadki użycia, wymagania, projekt i kod źródłowy) są często podstawą do testowania na jednym lub wielu poziomach.

2.1.2 Kandydat akceptuje fakt, że modele wytwarzania oprogramowania muszą zostać zaadaptowane do cech projektu i produktu

Iteracyjno-przyrostowe wytwarzanie oprogramowania to proces zbierania wymagań, projektowania, budowania oraz testowania systemu zorganizowany w krótsze cykle rozwojowe. System wyprodukowany według takiego modelu może być przetestowany na kilku poziomach w każdej iteracji. Każdy przyrost może podlegać zarówno weryfikacji jak i walidacji.

2.1.3 Kandydat pamięta atrybuty dobrego testowania mające zastosowanie w każdym z modeli życia oprogramowania

W każdym modelu rozwoju oprogramowania dobre testowanie posiada kilka niezmiennych cech

- dla każdej czynności związanej z wytworzeniem oprogramowania istnieją odpowiadające jej czynności związane z testowaniem
- każdy poziom testowania ma zdefiniowane cele
- analiza i projektowanie testów dla danego poziomu powinny rozpoczynać się już podczas odpowiadającej im fazy wytwarzania
- testerzy powinni uczestniczyć w przeglądach już od wczesnych wersji dokumentacji tworzonej podczas wytwarzania

2.2 Poziomy testów

2.2.1 Kandydat potrafi porównać różne poziomy testów: główne cele, typowe przedmioty testów, typowe cele testowania (np. strukturalne lub funkcjonalne) i związane z nimi produkty, testerów, typy usterek i awarii do znalezienia

Testy modułowe

Podstawa testów

- wymagania na moduły
- projekt szczegółowy
- kod

Typowe obiekty testów

- moduły
- programy

- programy do konwersji lub migracji danych
- moduły bazodanowe

Testy modułowe polegają na wyszukiwaniu błędów i weryfikacji funkcjonalności oprogramowania (np. modułów, programów, obiektów, klas), które można testować oddzielnie. Może być wykonywane w izolacji od reszty systemu, w zależności od kontekstu cyklu rozwoju oprogramowania i od samego systemu. Można podczas nich użyć zaślepek, sterowników testowych oraz symulatorów.

Testy integracyjne

Podstawa testów

- projekt oprogramowania i systemu
- architektura
- przepływy procesów
- przypadki użycia

Typowe obiekty testów

- implementacja baz danych podsystemów
- infrastruktura
- interfejsy
- konfiguracja systemu i dane konfiguracyjne

Testy integracyjne sprawdzają interfejsy pomiędzy modułami, interakcje z innymi częściami systemu (takimi jak system operacyjny, system plików i sprzęt) oraz interfejsy pomiędzy systemami.

Testy systemowe

Podstawa testów

- wymagania na system i oprogramowanie
- przypadki użycia
- specyfikacja funkcjonalna
- raporty z analizy ryzyka

Typowe obiekty testów

- podręczniki systemowe, użytkownika i operacyjne
- konfiguracje systemu i dane konfiguracyjne

Testy systemowe zajmują się zachowaniem systemu/produktu. Zakres testów powinien być jasno określony w głównym planie testów oraz w planach testów poszczególnych poziomów.

Testy akceptacyjne

Podstawa testów

- wymagania użytkownika

- wymagania systemowe
- przypadki użycia
- procesy biznesowe
- raporty z analizy ryzyka

Typowe obiekty testów

- proces biznesowy na systemie w pełni zintegrowanym
- procesy utrzymania i obsługi
- procedury pracy użytkowników
- formularze
- raporty
- dane konfiguracyjne

Odpowiedzialność za testy akceptacyjne leży często po stronie klientów lub użytkowników systemu. Mogą w nie być zaangażowani również inni interesariusze.

2.3 Typy testów

2.3.1 Kandydat potrafi porównać podając przykłady cztery typy testów (funkcjonalne, niefunkcjonalne, strukturalne oraz związane ze zmianami)

Testowanie funkcji (testowanie funkcjonalne)

Funkcje, jakie ma pełnić system, podsystem lub moduł, mogą być opisane w produktach takich jak specyfikacja wymagań, przypadki użycia lub specyfikacja funkcjonalna. Mogą też być nieudokumentowane. Funkcje są tym "co" system robi.

Testy funkcjonalne dotyczą funkcji lub innych cech(opisanych w dokumentach lub domniemanych przez testerów) oraz ich współdziałania z innymi systemami. Można je wykonywać na wszystkich poziomach (np. testy modułowe mogą bazować na specyfikacji modułów).

Testowanie atrybutów niefunkcjonalnych (testowanie niefunkcjonalne)

Testowanie niefunkcjonalne obejmuje następujące (ale nie tylko te) typy testów: testowanie wydajnościowe, testowanie obciążeniowe, testowanie przeciążeniowe, testowanie użyteczności, testowanie pielęgnowalności, testowanie niezawodności oraz testowanie przenaszalności. Testowanie niefunkcjonalne polega na sprawdzeniu "jak" system działa.

Testowanie niefunkcjonalne może być wykonywane na wszystkich poziomach testów. Termin testy niefunkcjonalne oznacza testy wymagane do zmierzenia właściwości systemów i oprogramowania, które mogą zostać określone ilościowo na zmiennej skali, takich jak czasy odpowiedzi w testach wydajnościowych.

Testowanie struktury/architektury oprogramowania (testowanie strukturalne)

Testy strukturalne (białoskrzynkowe) można wykonywać na każdym poziomie testowania. Technik strukturalnych najlepiej użyć po technikach opartych na specyfikacji, po to by zmierzyć dokładność testowania przez ocenę stopnia pokrycia wybranego typu struktury.

Pokrycie, to stopień, w jakim struktura została przetestowana przez zestaw testów wyrażony jako odsetek pokrytych elementów. Jeżeli pokrycie jest niższe niż 100%, można zaprojektować więcej testów, żeby przetestować te elementy, które zostały pominięte, i w ten sposób zwiększyć pokrycie.

Testowanie związane ze zmianami: testowanie potwierdzające oraz regresywne

Po wykryciu i naprawieniu defektu, powinien zostać wykonany retest, żeby potwierdzić usunięcie usterki. Takie testy nazywane są testami potwierdzającymi. Debagowanie (lokalizacja oraz poprawianie usterek) jest czynnością programistyczną, a nie testową.

Testowanie regresywne, to powtórzenie testów na już przetestowanym programie wykonywane po modyfikacjach żeby wykryć nowe usterki lub usterki odsłonięte na skutek wykonanych zmian. Usterki te mogą występować w testowanym oprogramowaniu, jak również w innych powiązanych lub niepowiązanych modułach. Testy regresywne wykonuje się po zmianach w oprogramowaniu, a także po zmianach w jego środowisku. Zakres testów regresywnych wynika z ryzyka nieznalezienia usterek w oprogramowaniu, które poprzednio działało poprawnie.

2.3.2 Kandydat uznaje, że testy funkcjonalne i strukturalne występują na każdym poziomie testów

Testy funkcjonalne dotyczą funkcji lub innych cech(opisanych w dokumentach lub domniemanych przez testerów) oraz ich współdziałania z innymi systemami. Można je wykonywać na wszystkich poziomach (np. testy modułowe mogą bazować na specyfikacji modułów). Testy strukturalne (białoskrzynkowe) można wykonywać na każdym poziomie testowania. Technik strukturalnych najlepiej użyć po technikach opartych na specyfikacji, po to by zmierzyć dokładność testowania przez ocenę stopnia pokrycia wybranego typu struktury.

2.3.3 Kandydat potrafi wymienić i opisać różne typy testów niefunkcjonalnych bazujących na wymaganiach niefunkcjonalnych

Termin testy niefunkcjonalne oznacza testy wymagane do zmierzenia właściwości systemów i oprogramowania, które mogą zostać określone ilościowo na zmiennej skali, takich jak czasy odpowiedzi w testach wydajnościowych. Testy te mogą zostać odniesione do modelu jakości oprogramowania. Testy niefunkcjonalne zajmują się zewnętrznym zachowaniem oprogramowania i w większości wypadków wykorzystują techniki czarnoskrzynkowe.

2.3.4 Kandydat potrafi wymienić i opisać typy testów bazujące na analizie struktury lub architektury systemu

Testy strukturalne (białoskrzynkowe) można wykonywać na każdym poziomie testowania. Technik strukturalnych najlepiej użyć po technikach opartych na specyfikacji, po to by zmierzyć dokładność testowania przez ocenę stopnia pokrycia wybranego typu struktury. Pokrycie, to stopień, w jakim struktura została przetestowana przez zestaw testów wyrażony jako odsetek pokrytych elementów. Jeżeli pokrycie jest niższe niż 100%, można zaprojektować więcej testów, żeby przetestować te elementy, które zostały pominięte, i w ten sposób zwiększyć pokrycie. Techniki oparte na pokryciu opisane są w rozdziale 4. Do pomiaru pokrycia (na przykład decyzji lub instrukcji) na wszystkich poziomach, ale szczególnie na poziomie testów modułowych i poziomie testów integracji modułów, mogą zostać użyte narzędzia. Testowanie strukturalne może zostać oparte na architekturze systemu, na przykład hierarchii wywołań.

2.3.5 Kandydat potrafi opisać cel wykonywania testów potwierdzających i regresywnych

Po wykryciu i naprawieniu defektu, powinien zostać wykonany retest, żeby potwierdzić usunięcie usterki. Takie testy nazywane są testami potwierdzającymi. Debagowanie (lokalizacja oraz poprawianie usterek) jest czynnością programistyczną, a nie testową. Testowanie regresywne, to powtórzenie testów na już przetestowanym programie wykonywane po modyfikacjach żeby wykryć nowe usterki lub usterki odsłonięte na skutek wykonanych zmian. Usterki te mogą występować w testowanym oprogramowaniu, jak również w innych powiązanych lub niepowiązanych modułach. Testy regresywne wykonuje się po zmianach w oprogramowaniu, a także po zmianach w jego środowisku. Zakres testów regresywnych wynika z ryzyka nieznalezienia usterek w oprogramowaniu, które poprzednio działało poprawnie. Testy, które mają być stosowane w testowaniu potwierdzającym i regresywnym muszą być powtarzalne.

2.4 Testowanie pielęgnacyjne

2.4.1 Kandydat potrafi porównać testy pielęgnacyjne (testowanie istniejącego systemu) do testowania nowej aplikacji uwzględniając typy testów, powody rozpoczęcia testowania oraz ilość testów

Testy pielęgnacyjne, oprócz przetestowania tego co zostało zmienione, zawierają testy regresywne tych części systemu, które się nie zmieniły. Zakres testów pielęgnacyjnych zależy od ryzyka zmian, wielkości istniejącego systemu oraz zakresu zmian. W zależności od zmian, testowanie pielęgnacyjne może być wykonane na niektórych lub wszystkich poziomach testowania oraz dla wybranych lub wszystkich typów testów.

2.4.2 Kandydat potrafi wymienić powody wykonywania testów pielęgnacyjnych (zmiany, migracja i wycofanie systemu)

Wdrożony system często musi działać przez lata lub dekady. W tym czasie system, jego dane konfiguracyjne i jego środowisko są często poprawianie, zmieniane i rozszerzane. Dla dobrego testowania pielęgnacyjnego krytyczne jest planowanie wydań z wyprzedzeniem. Konieczne jest rozróżnianie pomiędzy planowanymi wydaniami i szybkimi poprawkami. Testowanie pielęgnacyjne wykonuje się na działającym systemie na skutek modyfikacji, migracji lub złomowania oprogramowania lub systemu. Modyfikacjami mogą być planowane ulepszenia (np. według planowych wydań), poprawki lub naprawy awaryjne oraz zmiany środowiska, takie jak planowane podniesienia wersji systemu operacyjnego, bazy danych lub oprogramowania z półki albo łaty zabezpieczeń systemu operacyjnego. Testy pielęgnacyjne migracji oprogramowania (np. z jednej platformy na inną) powinny, oprócz testów zmian w oprogramowaniu, uwzględniać również testy produkcyjne nowego środowiska. Testy migracji (testowanie konwersji) są również potrzebne, gdy dane z innej aplikacji są migrowane do utrzymywanego systemu.

2.4.3 Kandydat potrafi opisać rolę testów regresywnych i analizy wpływu w utrzymaniu

Określenie, jaki wpływ na istniejący system mogą mieć zmiany, nazywamy analizą wpływu. Stosuje się ją, aby ustalić zakres testów regresywnych. Analiza wpływu może zostać użyta do wybrania zestawu testów regresyjnych. Testowanie pielęgnacyjne może być trudne do wykonania, jeżeli specyfikacja oprogramowania jest przestarzała lub gdy jej brak, lub gdy nie są dostępni testerzy posiadający wiedzę dziedzinową.

3.1 Techniki statyczne a proces testowania

3.1.1 Kandydat potrafi rozpoznać produkty, które mogą zostać sprawdzone przez różne techniki testowania statycznego

Do typowych usterek, które łatwiej wykryć w testach statycznych niż dynamicznych należą: odchylenia od standardów, usterki w wymaganiach, usterki w projekcie, niedostateczna pielęgnowalność oraz nieprawidłowe specyfikacje interfejsów.

3.1.2 Kandydat potrafi opisać znaczenie i wartość statycznych technik testowania w ocenie produktów procesu rozwoju oprogramowania

Główne korzyści z wykonywania przeglądów to: wczesne wykrycie i naprawa usterek, zwiększenie produktywności produkcji oprogramowania, redukcja czasu produkcji oprogramowania, zmniejszenie kosztów i czasu testowania, ogólne zmniejszenie kosztu wytwarzania i użytkowania oprogramowania, zmniejszenie liczby usterek oraz usprawnienie komunikacji. Przeglądy mogą wykrywać braki (na przykład w wymaganiach), które trudno jest wykryć w testach dynamicznych. Przeglądy, analiza statyczna oraz testy dynamiczne mają ten sam cel – znajdowanie usterek. Te trzy techniki uzupełniają się wzajemnie, mogą skutecznie i efektywnie znajdować różne typy błędów. Techniki statyczne, w odróżnieniu od testów dynamicznych, znajdują raczej przyczyny awarii (usterki), a nie same awarie.

3.1.3 Kandydat potrafi wyjaśnić różnice pomiędzy testami statycznymii dynamicznymi

W przeciwieństwie do technik dynamicznych, które wymagają uruchomienia oprogramowania, techniki statyczne polegają na sprawdzeniu ręcznym (przeglądy) lub analizie automatycznej (analiza statyczna) kodu lub innych dokumentów projektowych bez uruchamiania kodu.

3.2 Proces przeglądu

3.2.1 Kandydat pamięta kroki, role i odpowiedzialności związane z typowym przeglądem formalnym

Przegląd formalny zwykle składa się z następujących faz

1. planowanie
    - definiowanie kryteriów przeglądu
    - wybór uczestników przeglądu
    - przydział ról
    - ustalenie kryteriów wejścia i zakończenia dla bardziej formalnych typów przeglądów (np. dla inspekcji)
    - wybór fragmentów dokumentu do przejrzenia
2. rozpoczęcie
    - rozesłanie dokumentów
    - opisanie celów przeglądu, procesu i dokumentów uczestnikom przeglądu
    - sprawdzenie kryteriów wejścia (dla bardziej formalnych typów przeglądów)
3. przygotowanie indywidualne
    - przygotowanie przed spotkaniem przeglądowym przez przejrzenie dokumentów
    - zapisywanie potencjalnych defektów, pytań i komentarzy
4. kontrola/ocena/zapisanie wyników (spotkanie przeglądowe)

    - dyskusja lub spisywanie, z udokumentowaniem wyników i sporządzeniem protokołu (dla bardziej formalnych typów przeglądów)
    - zapisywanie defektów i rekomendacji dotyczących ich poprawiania, podejmowanie decyzji co do defektów

5. poprawki

    - naprawianie znalezionych defektów, zwykle wykonywane przez autora
    - uaktualnianie statusów defektów (w przeglądach formalnych)

6. zakończenie

    - sprawdzenie, że usterki zostały obsłużone
    - zbieranie metryk
    - sprawdzanie kryteriów zakończenia (dla bardziej formalnych typów przeglądów)

3.2.2 Kandydat potrafi wyjaśnić różnice pomiędzy różnymi typami przeglądów: przeglądem nieformalnym, przeglądem technicznym, przejrzeniem i inspekcją

Głównymi cechami, opcjami i celami powszechnie stosowanych typów przeglądów są:

Przegląd nieformalny

- brak formalnego procesu
- może przybrać formę programowania w parach oraz przegląd projektu lub kodu przez kierownika zespołu
- może być udokumentowany
- jego użyteczność może być różna w zależności od przeglądających
- główny cel: tani sposób na osiągnięcie niewielkich korzyści

Przejrzenie

- spotkanie jest prowadzone przez autora
- może przybrać formę scenariuszy, uruchamiania "na sucho", grupa kolegów
- sesje nie ograniczone czasowo o opcjonalnie przygotowanie przeglądających przed spotkaniem o opcjonalnie raport z przeglądu, lista uwag
- opcjonalnie protokólant (którym nie jest autor)
- w praktyce może być od całkiem nieformalnego do bardzo formalnego
- główne cele: uczenie się, zrozumienie, znajdowanie usterek

Przegląd techniczny

- posiada zdefiniowany proces wykrywania defektów między innymi przez kolegów i ekspertów technicznych z opcjonalnym udziałem kierownictwa
- może być organizowany jako przegląd koleżeński bez udziału kierownictwa
- w idealnej sytuacji prowadzony przez przeszkolonego moderatora (nie autora)
- przygotowanie przeglądających przed spotkaniem
- opcjonalnie z wykorzystaniem list kontrolnych
- przygotowanie raportu z przeglądu, który zawiera listę uwag, ocenę czy produkt programistyczny spełnia wymagania i, tam gdzie jest to potrzebne, rekomendacje związane z uwagami

- w praktyce może być wykonywany w sposób od całkiem nieformalnego do bardzo formalnego
- główne cele: przedyskutowanie, podjęcie decyzji, ocena alternatyw, wyszukanie usterek, rozwiązanie problemów technicznych oraz sprawdzenie zgodności ze specyfikacją i standardami

Inspekcja

- prowadzona przez przeszkolonego moderatora (nie autora)
- zwykle sprawdzenie przez kolegów
- posiada zdefiniowane role
- posiada metryki
- posiada formalny proces oparty na regułach i listach kontrolnych
- posiada zdefiniowane kryteria wejścia i zakończenia
- przygotowanie przed spotkaniem przeglądowym
- raport z inspekcji zawierający listę uwag
- formalny proces kontroli wykonania napraw o opcjonalnie elementy doskonalenia procesów
- opcjonalnie wykorzystanie czytającego
- główny cel: wyszukanie usterek

Przejrzenia, przeglądy techniczne oraz inspekcje można wykonywać w grupie osób równych rangą - kolegów z tego samego szczebla organizacji. Taki przegląd nazywa się przeglądem koleżeńskim.

3.2.3 Kandydat potrafi wskazać czynniki wpływające na skuteczne przeprowadzanie przeglądów

Następujące czynniki wpływają na sukces przeglądów

- każdy przegląd ma jasno zdefiniowany cel
- w przegląd zaangażowani są ludzie odpowiedni do jego celu
- testerzy są wartościowymi przeglądającymi, którzy przyczyniają się do sukcesu przeglądu oraz poznają produkt, co pozwala im przygotować testy wcześniej
- znalezione usterki są przyjmowane pozytywnie i wyrażane w sposób obiektywny
- rozwiązuje się problemy personalne i psychologiczne (np. jest to pozytywnym doświadczeniem dla autora)
- przegląd jest prowadzony w atmosferze zaufania; wyniki przeglądu nie zostają użyte do oceny uczestników przeglądu
- stosuje się techniki przeglądania adekwatne do celów przeglądu, do typu i poziomu produktu oraz przeglądających
- jeżeli jest to potrzebne, zostają użyte listy kontrolne lub role do podniesienia skuteczności znajdowania usterek
- organizuje się szkolenia, szczególnie z bardziej formalnych technik takich jak inspekcja
- kierownictwo wspiera dobry proces przeglądu (np. przez przeznaczenie odpowiedniej ilości czasu w harmonogramach na zadania związane z przeglądem)
- kładzie się nacisk na uczenie się oraz doskonalenie procesów

3.3 Analiza statyczna przy pomocy narzędzi

3.3.1 Kandydat pamięta typowe błędy wykrywane przez analizę statyczną i umie porównać je z przeglądami i testami dynamicznymi

Celem analizy statycznej jest wyszukanie usterek w kodzie programu lub w modelach bez uruchamiania oprogramowania, które jest sprawdzane przez narzędzie używane w tym procesie. Miejscem, w którym kod jest wykonywany, są testy dynamiczne. Analiza statyczna może znaleźć usterki, które są trudne do znalezienia podczas testowania dynamicznego. Tak jak to było w przypadku przeglądów, analiza statyczna znajduje usterki, a nie awarie. Narzędzia do analizy statycznej analizują kod oprogramowania (np. przepływ sterowania lub przepływ danych) jak również wygenerowane wyjście w postaci HTMLa lub XMLa.

3.3.2 Kandydat potrafi opisać używając przykładów typowe korzyści z analizy statycznej

O wartości analizy statycznej stanowią następujące jej cechy

- wczesne wykrywanie usterek, jeszcze przed wykonaniem testów wczesne wykrywanie podejrzanych aspektów kodu lub projektu przez wyliczenie miar, takich jak wysoki stopień złożoności
- identyfikacja defektów trudnych do wykrycia przez testowanie
- wykrywanie zależności i niespójności w modelach oprogramowania
- zwiększenie pielęgnowalności kodu i projektu
- zapobieganie defektom, jeżeli zastosowane zostają wnioski z analizy procesu rozwoju oprogramowania

3.3.3 Kandydat potrafi wymienić typowe defekty kodu i projektu, które mogą zostać wykryte przez narzędzia do analizy statycznej

Analiza statyczna zwykle wykrywa następujące typy usterek

- odwołanie do niezainicjalizowanej zmiennej
- niespójne interfejsy pomiędzy modułami
- niewykorzystywane lub niepoprawnie zadeklarowane zmienne
- martwy kod
- brakująca albo błędna logika (pętle potencjalnie nieskończone)
- zbyt skomplikowane konstrukcje
- naruszenie standardów kodowania
- słabe punkty zabezpieczeń
- naruszenie reguł składni kodu i modeli oprogramowania

4.1 Proces rozwoju testów

4.1.1 Kandydat odróżnia projekt testów od specyfikacji przypadków testowych oraz od procedury testowej

Podczas projektowania testów tworzy się i opisuje przypadki testowe i dane testowe. Przypadek testowy składa się ze zbioru wartości wejściowych, warunków wstępnych, oczekiwanych wyników oraz warunków zakończenia utworzonych żeby pokryć pewne cele testów lub warunki testowe.
Jako część specyfikacji przypadku testowego powinny zostać określone wyniki oczekiwane. Powinny one zawierać opis wyjść, zmian w danych i stanie oprogramowania oraz inne skutki testu. Gdy wyniki oczekiwane nie są zdefiniowane, może się zdarzyć, że wiarygodne, ale błędne, wyniki zostaną uznane za poprawne. Najlepiej jest, gdy wyniki oczekiwane zostaną zdefiniowane przed wykonaniem testów. Podczas implementacji testów przypadki testowe są rozwijane, implementowane, priorytetyzowane i układane w specyfikację procedur testowych. Procedura testowa zawiera kolejność działań podczas wykonania testu. Jeżeli testy są wykonywane

przy pomocy narzędzia do wykonywania testów, ciąg akcji jest zawarty w skrypcie testowym (który stanowi automatyczną procedurę testową).

4.1.2 Kandydat potrafi porównać następujące pojęcia: warunek testowy, przypadek testowy, procedura testowa

4.1.3 Kandydat potrafi ocenić jakość przypadków testowych, przez sprawdzenie czy mają jednoznaczne powiązania z wymaganiami i czy zawierają wynik oczekiwany

4.1.4 Kandydat potrafi utworzyć z przypadków testowych dobrze ustrukturalizowaną procedurę testową na poziomie szczegółowości odpowiadającym wiedzy testerów

Różne procedury testowe i automatyczne skrypty testowe są następnie układane w harmonogram wykonania testów, który definiuje porządek wykonania procedur testowych i automatycznych skryptów testowych (o ile są obecne). Przy tworzeniu harmonogramu wykonania testów bierze się pod uwagę takie czynniki jak testy regresywne, priorytety oraz zależności techniczne i logiczne.

4.2 Kategorie technik projektowania testów

4.2.1 Kandydat pamięta do czego przydatne są techniki projektowania testów oparte na specyfikacji (czarnoskrzynkowe) oraz techniki oparte na strukturze (białoskrzynkowe) oraz potrafi wymienić typowe dla nich techniki

Klasyczny podział wyróżnia techniki czarnoskrzynkowe oraz białoskrzynkowe. Czarnoskrzynkowe techniki projektowania testów (również nazywane technikami opartymi na specyfikacji) są sposobem na wywodzenie[1] oraz wybieranie warunków testowych, przypadków testowych i danych testowych bazującym na analizie podstawy testów. Można ich używać zarówno w testowaniu funkcjonalnym jak i niefunkcjonalnym. Techniki czarnoskrzynkowe z definicji nie wykorzystują żadnych informacji o strukturze wewnętrznej testowanego modułu lub systemu. Białoskrzynkowe techniki projektowania testów (również nazywane strukturalnymi lub technikami opartymi na strukturze) bazują na analizie struktury modułu lub systemu. Testy biało i czarnoskrzynkowe mogą również korzystać z doświadczenia programistów, testerów i użytkowników w celu określenia, co powinno zostać przetestowane. Niektóre techniki da się przyporządkować jednoznacznie do jednej kategorii, inne zawierają elementy więcej niż jednej kategorii. W tym sylabusie techniki projektowania testów oparte na specyfikacji nazywane są technikami czarnoskrzynkowymi, a techniki oparte na strukturze – białoskrzynkowymi. Techniki oparte na doświadczeniu omówione są oddzielnie.

4.2.2 Kandydat potrafi objaśnić cechy, podobieństwa oraz różnice pomiędzy technikami opartymi na specyfikacji, strukturze i doświadczeniu

Cechy wspólne technik projektowania testów opartych na specyfikacji, to między innymi

- w specyfikacji problemu do rozwiązania, oprogramowania lub jego komponentów używane są modele
- z tych modeli można, w sposób usystematyzowany, wywodzić przypadki testowe Cechy wspólne technik projektowania testów opartych na strukturze, to m.in.:
- do tworzenia przypadków testowych wykorzystywana jest wiedza o tym jak oprogramowanie jest skonstruowane, np. kod źródłowy lub szczegółowy projekt
- można mierzyć stopień pokrycia istniejących przypadków testowych, można też w sposób usystematyzowany tworzyć nowe przypadki testowe w celu zwiększenia pokrycia Cechy wspólne technik projektowania testów opartych na doświadczeniu, to m.in.:

- do tworzenia przypadków testowych wykorzystywane jest doświadczenie ludzi
    - wiedza testerów, programistów, użytkowników oraz innych interesariuszy o oprogramowaniu, jego środowisku i sposobie użytkowania
    - wiedza o prawdopodobnych defektach i ich położeniu

4.3 Techniki oparte na specyfikacji lub czarnoskrzynkowe

4.3.1 Kandydat potrafi napisać przypadki testowe na podstawie podanych modeli oprogramowania używając techniki klas równoważności, analizy wartości brzegowych, testowania w oparciu o tablicę decyzyjną, testowania przejść pomiędzy stanami

Podział na klasy równoważności

W technice podziału na klasy równoważności wejścia programu lub systemu są dzielone na grupy, które powodują podobne zachowanie oprogramowania, więc wysoce prawdopodobne jest to, że są przetwarzane w ten sam sposób. Klasy równoważności można znaleźć dla danych poprawnych (wartości, które powinny zostać zaakceptowane) oraz niepoprawnych (wartości, które powinny zostać odrzucone). Klasy równoważności można znaleźć również dla wyjść, wartości wewnętrznych, wartości zależnych od czasu (np. przed lub po zajściu jakiegoś zdarzenia) oraz parametrów interfejsów (np. komponentów integrowanych i testowanych podczas testów integracyjnych). Testy można tak zaprojektować, żeby pokrywały akceptowalne i nieakceptowalne klasy równoważności. Podział na klasy równoważności można zastosować na każdym poziomie testowania.

Analiza wartości brzegowych

Istnieje większe prawdopodobieństwo, że oprogramowanie będzie się błędnie zachowywać dla wartości na krawędziach klas równoważności niż w ich środku, więc testowanie tych obszarów najprawdopodobniej wykryje błędy. Minimum i maksimum klasy równoważności to jej wartości brzegowe. Wartość brzegowa poprawnego przedziału jest nazywana poprawną wartością brzegową, a wartość brzegowa niepoprawnego przedziału – niepoprawną wartością brzegową. Testy można zaprojektować tak, żeby pokrywały zarówno poprawne jak i niepoprawne wartości brzegowe. Podczas projektowania testów tworzy się przypadek testowy dla każdej wartości brzegowej

Testowanie w oparciu o tablicę decyzyjną

Tabele decyzyjne są dobrym sposobem na uchwycenie tych wymagań na system, które zawierają zależności logiczne, oraz na udokumentowanie wewnętrznej budowy systemu. Mogą być używane do zapisywania złożonych reguł biznesowych, które system ma obsługiwać. Podczas tworzenia tablic decyzyjnych analizuje się specyfikację systemu i wyszukuje warunki oraz wynikające z nich zachowanie systemu. Warunki wejściowe oraz zachowanie systemu często muszą być zapisane jako prawda lub fałsz (Boolean). Tabela decyzyjna zawiera warunki uruchamiające, często jako kombinacje prawdy i fałszu, dla wszystkich warunków wejściowych oraz działania wynikające z każdej z tych kombinacji. Każda kolumna tabeli odpowiada jednej regule biznesowej, która definiuje unikalną kombinację warunków i której skutkiem jest działanie związane z daną regułą biznesową. Standardowe pokrycie stosowane dla testowania w oparciu o tabelę decyzyjną wymaga zaprojektowania jednego testu dla każdej kolumny w tablicy, co zwykle oznacza wykorzystanie wszystkich kombinacji warunków uruchamiających.

Testowanie przejść między stanami

System może różnie odpowiadać w zależności od aktualnych warunków oraz od historii (od stanu). W takim przypadku zachowanie systemu można opisać diagramem przejść stanów (automatem skończonym). Pozwala to testerowi spojrzeć na oprogramowanie od strony stanów, przejść między stanami, wejść lub zdarzeń, które powodują zmiany stanów (przejścia) oraz na działania, które mogą wynikać z tych przejść. Stany testowanego systemu lub elementu są rozdzielne, zdefiniowane oraz ich liczba jest skończona. Tabela stanów pokazuje zależności pomiędzy stanami oraz wejściami i może uwypuklić przejścia nieprawidłowe. Testy można zaprojektować tak, żeby pokryły typowe sekwencje stanów, każdy stan, każde przejście, konkretny ciąg przejść lub żeby testowały przejścia nieprawidłowe.

4.3.2 Kandydat potrafi wyjaśnić główny cel każdej z czterech technik testowania, na jakim poziomie testowania mogą zostać użyte i jak można dla nich zmierzyć pokrycie

Podział na klasy równoważności

Technika podziału na klasy równoważności może zostać użyta do osiągnięcia celów pokrycia zarówno wejścia jak i wyjścia. Może zostać zastosowana do wartości wejściowych wprowadzanych ręcznie, przez interfejsy oraz do parametrów interfejsów podczas testów integracyjnych.

Analiza wartości brzegowych

Analiza wartości brzegowych może zostać zastosowana na każdym poziomie testowania. Jest ona stosunkowo łatwa w użyciu i daje duże możliwości wykrywania usterek. Szczegółowa specyfikacja jest pomocna w znajdowaniu interesujących wartości brzegowych. Technika ta jest często uważana za rozwinięcie techniki podziału na klasy równoważności lub innych technik czarnoskrzynkowych. Może być użyta dla klas równoważności wartości wprowadzanych przez interfejs użytkownika jak również, na przykład, dla przedziałów czasowych (np. time outów, wymagań na szybkość przetwarzania transakcji) lub zakresów tablic (np. rozmiar tablicy ma być 256x256).

Testowanie w oparciu o tablicę decyzyjną

Moc testowania w oparciu o tabelę decyzyjną leży w uwidocznieniu kombinacji warunków, które w innym przypadku mogłyby zostać pominięte w testach. Technika ta ma zastosowanie w każdej sytuacji, w której zachowanie oprogramowania zależy od kilku decyzji logicznych.

Testowanie przejść między stanami

Testowanie przejść między stanami jest często używane w testowaniu oprogramowania wbudowanego oraz ogólnie w automatyce. Jednakże technikę tę można zastosować do zamodelowania obiektu biznesowego posiadającego określone stany lub do testowania przejść między formatkami (np. w aplikacjach internetowych lub scenariuszach biznesowych).

4.3.3 Kandydat potrafi wyjaśnić na czym polega testowanie w oparciu o przypadki użycia i korzyści płynące z jego zastosowania

Testy można projektować na podstawie przypadków użycia. Przypadek użycia opisuje interakcje pomiędzy aktorami (użytkownikami lub systemami), które powodują powstanie wyniku wartościowego z punktu widzenia użytkownika lub klienta. Przypadek użycia może być opisany na wysokim poziomie abstrakcji (biznesowy przypadek użycia, poziom procesów biznesowych, niezawierający informacji o technologii) lub na poziomie

systemowym (systemowy przypadek użycia na poziomie funkcjonalności systemu). Każdy przypadek użycia posiada warunki wstępne, które muszą zostać spełnione, żeby przypadek użycia został wykonany. Każdy przypadek użycia kończy się warunkami końcowymi. Są nimi widoczne rezultaty jego wykonania oraz stan systemu po zakończeniu przypadku użycia. Przypadki użycia zwykle posiadają scenariusz główny (tj. najbardziej prawdopodobny) oraz czasami scenariusze poboczne.

Przypadki użycia opisują „przepływy" procesu przez system bazując na najbardziej prawdopodobnym rzeczywistym jego użyciu, co sprawia że przypadki testowe wywiedzione z przypadków użycia najbardziej przydają się wykrywaniu usterek w przepływach procesów z rzeczywistego użytkowania systemu. Przypadki użycia bardzo przydają się w projektowaniu testów akceptacyjnych, w których ma brać udział klient/użytkownik. Pomagają również wykrywać defekty integracji spowodowane interakcją i interfejsami różnych modułów, co nie byłoby widoczne w testach modułowych. Projektowanie przypadków testowych z przypadków użycia może zostać połączone z innymi technikami projektowania testów opartymi na specyfikacji.

4.4 Techniki oparte na strukturze lub białoskrzynkowe

4.4.1 Kandydat potrafi opisać pojęcie pokrycia kodu i jego znaczenie

Testowanie oparte na strukturze (białoskrzynkowe) bazuje na rozpoznanej strukturze oprogramowania lub systemu tak jak to widać w następujących przykładach:

- w testach modułowych: strukturą jest kod, to jest instrukcje, decyzje, rozgałęzienia lub nawet rozróżnialne ścieżki
- w testach integracyjnych: strukturą może być hierarchia wywołań (diagram, który pokazuje jak moduły wywołują inne moduły)
- w testach systemowych: strukturą może być budowa menu, proces biznesowy lub struktura strony webowej

Pokrycie instrukcji oblicza się przez podzielenie liczby wykonywalnych instrukcji pokrytych przez (zaprojektowane lub wykonane) przypadki testowe, przez liczbę wszystkich wykonywalnych instrukcji w testowanym kodzie.

4.4.2 Kandydat potrafi wyjaśnić pojęcia: pokrycia instrukcji i pokrycia decyzji oraz podać powody, dla których te pojęcia mogą zostać użyte nie tylko na poziomie testów modułowych, ale na przykład też dla procedur biznesowych na poziomie testów systemowych)

W testowaniu instrukcji stosuje się pokrycie instrukcji, które polega na zmierzeniu, jaki odsetek instrukcji wykonywalnych został przetestowany przez zestaw testów. W technice tej projektuje się przypadki testowe, tak by wykonać określone instrukcje, zwykle po to żeby zwiększyć pokrycie.

Pokrycie decyzji, spokrewnione z testowaniem gałęzi, polega na zmierzeniu jaki odsetek wyników decyzji (np. wyniku prawda lub fałsz instrukcji if) został przetestowany przez zestaw testów. W technice testowania decyzji projektuje się przypadki testowe tak aby pokryć określone wyniki decyzji. Gałęzie zaczynają się w punktach decyzyjnych i pokazują przekazanie sterowania do różnych miejsc w kodzie. Testowanie gałęzi różni się od testowania decyzji przez to, że skupia się na samych gałęziach.

4.4.3 Kandydat potrafi napisać przypadki testowe dla danych przepływów sterowania stosując techniki testowania instrukcji i testowania decyzji

4.4.4 Kandydat potrafi ocenić kompletność testów według zdefiniowanych kryteriów zakończenia na podstawie pokrycia instrukcji i decyzji

Testowanie decyzji jest jedną z form testowania przepływu sterowania, ponieważ 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.

4.5 Techniki oparte na doświadczeniu

4.5.1 Kandydat pamięta powody pisania przypadków testowych opierających się na intuicji, doświadczeniu oraz wiedzy na temat często spotykanych usterek

Z testowaniem opartym na doświadczeniu mamy do czynienia, gdy testy projektuje się na podstawie wiedzy i intuicji testerów oraz ich doświadczenia z podobnymi aplikacjami i technologiami. Jeżeli zostanie ono użyte jako uzupełnienie technik systematycznych (zwłaszcza gdy zostanie zastosowane po nich), może okazać się użyteczne w uchwyceniu specjalnych przypadków testowych, których nie da się łatwo zaprojektować używając technik formalnych. Niemniej jednak jego skuteczność może okazać się bardzo różna w zależności od doświadczenia testerów.

Szeroko wykorzystywaną techniką opartą na doświadczeniu jest zgadywanie błędów. Ogólnie rzecz biorąc testerzy przewidują usterki bazując na swoim doświadczeniu. Ustrukturalizowane podejście do zgadywania błędów polega na sporządzeniu listy możliwych defektów i na zaprojektowaniu testów, które atakują te defekty. To systematyczne podejście jest nazywane atakiem usterkowym. Listy defektów i awarii można budować na podstawie doświadczenia, dostępnych danych na temat usterek i awarii oraz na ogólnej wiedzy dlaczego oprogramowanie nie działa.

4.5.2 Kandydat potrafi porównać techniki oparte na doświadczeniu z technikami opartymi na specyfikacji

Testowanie eksploracyjne polega na równoległym projektowaniu testów, ich wykonywaniu, zapisywaniu wyników oraz uczeniu się, bazując na zamówieniu na testy zawierającym cele testów i trzymając się ram czasowych. To podejście okazuje się najpożyteczniejsze, gdy nie ma specyfikacji, albo jest ona niekompletna czy przestarzała, również w sytuacjach bardzo krótkiego czasu na testy. Przydaje się ono również do wzbogacenia i uzupełnienia bardziej formalnych testów. Może służyć też do kontroli procesu testowania w celu upewnienia się, że wszystkie najpoważniejsze usterki zostały wykryte.

4.6 Wybór technik testowania

4.6.1 Kandydat potrafi podzielić techniki projektowania testów według ich dopasowania do danego kontekstu, podstawy testowania, modeli oraz cech oprogramowania

Wybór, która technika testowania powinna zostać użyta, zależy od wielu czynników, m.in. typu systemu, regulacji prawnych, wymagań klienta lub kontraktu, poziomu ryzyka, typu ryzyka, celu testów, dostępnej dokumentacji, wiedzy testerów, czasu i budżetu, cyklu rozwoju oprogramowania, modelu przypadków użycia oraz doświadczenia związanego ze znajdowanymi typami usterek. Niektóre z technik można z większą korzyścią zastosować tylko na niektórych poziomach testowania. Inne techniki można z równym powodzeniem stosować na wszystkich poziomach testów. Podczas projektowania przypadków testowych testerzy zwykle stosują różne techniki projektowania testów włącznie z technikami opartymi na procesie, regułach oraz danych, po to żeby zapewnić odpowiednie pokrycie testowanego obiektu.

5.1 Organizacja testów

5.1.1 Kandydat uznaje ważność testowania niezależnego

Skuteczność wykrywania usterek w testach i przeglądach może zostać podniesiona przez zaangażowanie niezależnych testerów. Niezależność może występować w różnych wariantach, włączając w to następujące:

- brak niezależnych testerów, programiści testują swój własny kod
- niezależni testerzy wewnątrz zespołu projektowego
- niezależny zespół testowy lub grupa testerów wewnątrz organizacji podlegająca kierownikowi projektu lub zarządowi
- niezależni testerzy z departamentów biznesowych lub społeczności użytkowników
- niezależni specjaliści od określonych typów testów takich jak użyteczności, zabezpieczeń lub certyfikacji oprogramowania (którzy przeprowadzają certyfikację oprogramowania na zgodność z regulacjami prawnymi lub standardami)
- niezależni testerzy, którzy zostali wynajęci lub są na zewnątrz organizacji

5.1.2 Kandydat potrafi wyjaśnić korzyści i wady niezależnego testowania w organizacji

Korzyści z niezależności

- niezależni testerzy widzą inne i odmienne usterki niż twórcy oraz nie mają uprzedzeń
- niezależny tester może zweryfikować założenia poczynione podczas specyfikacji i implementacji systemu

Wady niezależności

- izolacja od zespołu deweloperskiego (jeżeli niezależność jest całkowita)
- programiści mogą utracić poczucie odpowiedzialności za jakość
- niezależni testerzy mogą być postrzegani jako wąskie gardło lub obwiniani za opóźnienia w wydaniach

5.1.3 Kandydat uznaje potrzebę włączenia różnych członków zespołu podczas tworzenia zespołu testerskiego

Zadania związane z testowaniem mogą być wykonywane przez ludzi pełniących konkretne role testerskie lub przez ludzi pełniących inne role np. kierownika projektu, menedżera jakości, programistę, eksperta biznesowego lub dziedzinowego, ludzi związanych z serwisem operacyjnym lub IT.

5.1.4 Kandydat pamięta zadania typowego lidera testów oraz testera

Typowe zadania lidera testów to

- koordynowanie strategii oraz planu testów z kierownikami projektu i innymi interesariuszami
- tworzenie lub przeglądanie strategii testów w projekcie oraz polityki testowania w organizacji
- przedstawianie perspektywy testowania w innych zadaniach projektowych takich jak planowanie integracji
- planowanie testów, uwzględniając ich kontekst oraz rozumiejąc cele testów i ryzyko, włącznie z wyborem podejścia do testów, szacowaniem czasu, pracochłonności i kosztów testowania, zdobywaniem zasobów, definiowaniem poziomów testów, cykli testowych oraz planowaniem zarządzania incydentami
- inicjowanie specyfikacji, przygotowania, implementacji i wykonania testów, monitorowanie ich wyników oraz sprawdzanie spełnienia kryteriów zakończenia

- zmiana planów z uwzględnieniem wyników oraz postępu testów (czasami udokumentowanego w raporcie statusu testów), a także podejmowanie działań koniecznych do rozwiązania problemów
- zorganizowanie odpowiedniego zarządzania konfiguracją testaliów w celu zwiększenia możliwości śledzenia powiązań
- wprowadzenie odpowiednich metryk do mierzenia postępu testów i oceny jakości testowania oraz produktu
- decydowanie co powinno zostać zautomatyzowane, w jakim stopniu i w jaki sposób
- wybór narzędzi wspierających testy oraz organizacja dla testerów szkoleń z użycia tych narzędzi
- decydowanie o implementacji środowiska testowego
- sporządzanie raportów podsumowujących testy bazując na informacjach zebranych podczas testów

Typowe zadania testera to

- przeglądanie i wnoszenie wkładu do planów testów
- analiza, przegląd oraz ocena wymagań użytkownika, specyfikacji oraz modeli ze szczególnym uwzględnieniem testowalności
- tworzenie specyfikacji testów
- współtworzenie środowiska testowego (często jest to koordynacja pracy administratorów oraz osób odpowiedzialnych za sieć)
- przygotowanie i pozyskanie danych testowych
- implementacja testów na wszystkich poziomach, wykonanie i logowanie testów, ocena wyników oraz dokumentowanie odchyleń od oczekiwanych wyników
- używanie narzędzi do administracji lub zarządzania testami oraz narzędzi do monitorowania testów, gdy jest to wymagane
- automatyzacja testów (może być wspierana przez programistę lub eksperta od automatyzacji testów)
- pomiar wydajności modułów i systemów (o ile ma zastosowanie)
- przeglądanie testów wytworzonych przez innych

5.2 Planowanie i szacowanie testów

5.2.1 Kandydat rozpoznaje różne poziomy i cele planowania testów

Na planowanie testów wpływ ma polityka testowania w organizacji, zakres testów, cele, ryzyko, ograniczenia, krytyczność, testowalność oraz dostępność zasobów. Im dłużej trwa planowanie projektu i testów, tym więcej informacji jest dostępnych i tym więcej szczegółów może być włączone do planowania. Planowanie testów jest czynnością stałą i wykonuje się je dla wszystkich procesów i zadań cyklu życia oprogramowania. Informacje zwrotne z czynności testowych są używane do wykrywania zmieniających się ryzyk, tak że planowanie może zostać dopasowane do bieżącej sytuacji w projekcie.

5.2.2 Kandydat potrafi omówić krótko cel i zawartość planu testów, projektu testów, procedury testowej zgodnie ze standardem dokumentacji testowania oprogramowania

W planowanie testów dla całego systemu lub jego części mogą wchodzić następujące czynności

- ustalenie zakresu i ryzyka oraz zidentyfikowanie celów testowania
- zdefiniowanie ogólnego podejścia do testowania włącznie z definicją poziomów testów oraz kryteriów wejścia i zakończenia

- integrowanie i koordynowanie zadań testowych z innymi zadaniami cyklu życia oprogramowania: zakupami, dostawami, rozwojem, działaniem produkcyjnym oraz pielęgnacją
- podejmowanie decyzji co testować, którym rolom będą przypisane które zadania testowe, jak zadania testowe powinny być wykonane oraz jak powinno się oceniać wyniki testów
- harmonogramowanie analizy i projektowania testów
- harmonogramowanie implementacji, wykonania i oceny testów
- przydzielanie zasobów do zadań testowych
- definiowanie ilości, poziomu szczegółowości, struktury oraz wzorców dokumentacji testowej
- wybór metryk do monitorowania i kontroli przygotowania i wykonania testów, naprawy defektów oraz elementów ryzyka
- decydowanie o poziomie szczegółowości procedur testowych, aby dostarczyć wystarczającą ilość informacji dla powtarzalnego przygotowania i wykonania testów

5.2.3 Kandydat rozróżnia odmienne podejścia do testowania, takie jak analityczne, oparte na modelach, metodyczne, zgodne z procesem lub standardem, dynamiczne/heurystyczne, konsultatywne oraz regresywne

Typowe podejścia do testów to

- podejścia analityczne, takie jak testy oparte na ryzyku, w którym testowanie jest kierowane na obszary o największym ryzyku
- podejścia oparte na modelach, takie jak testowanie stochastyczne wykorzystujące informacje statystyczne na temat współczynników awarii (takich jak modele wzrostu niezawodności oprogramowania) lub wykorzystania oprogramowania (takich jak profile operacyjne)
- podejścia metodyczne, takie jak podejścia oparte na awariach (włącznie ze zgadywaniem błędów i atakami usterkowymi), oparte na doświadczeniu, na listach kontrolnych lub na atrybutach jakościowych
- podejścia zgodne ze standardem lub procesem, takie jak te określone przez standardy przemysłowe lub metodyki zwinne
- podejścia dynamiczne i heurystyczne, takie jak testowanie eksploracyjne, w którym testowanie bardziej reaguje na zdarzenia podczas testów niż jest wykonywane według planu i w którym wykonywanie testów i ocena wyników dzieją się równolegle
- podejścia konsultatywne, w których pokrycie testowe jest sterowane głównie przez wskazówki i porady ekspertów technologicznych lub biznesowych z zewnątrz zespołu testowego
- podejścia regresywne, w których używa się powtórnie istniejących materiałów testowych, rozbudowanej automatyzacji regresywnych testów funkcjonalnych oraz standardowych zestawów testów

5.2.4 Kandydat odróżnia planowanie testów systemu od harmonogramowania ich wykonania

5.2.5 Kandydat potrafi napisać harmonogram wykonania testów dla danego zbioru przypadków testowych, uwzględniając ich priorytety oraz zależności logiczne i techniczne

5.2.6 Kandydat potrafi wymienić czynności przygotowania i wykonania testów, które należy wziąć pod uwagę przy planowaniu

Podejście do testów jest implementacją strategii testów w konkretnym projekcie. Podejście do testów jest definiowane i uszczegóławiane w planach testów oraz projektach testów. Zwykle zawiera decyzje podejmowane na podstawie celów projektu (testowego) oraz oceny ryzyka. Stanowi ono punkt wyjściowy do planowania procesu testowania, wyboru technik projektowania testów i stosowanych typów testów, a także do definiowania

kryteriów wejścia i zakończenia. Wybrane podejście zależy od kontekstu i może uwzględniać ryzyka, niebezpieczeństwa oraz wymagane bezpieczeństwo, dostępne zasoby oraz umiejętności, technologię, naturę systemu (np. system niestandardowy vs. z półki), cele testów i uregulowania prawne.

5.2.7 Kandydat pamięta typowe czynniki, które mają wpływ na pracochłonność testowania

Pracochłonność testów może zależeć od wielu czynników, a w tym

- cech produktu:
  - jakości specyfikacji oraz innych informacji używanych w modelach testowych (tj. w podstawie testów), wielkości produktu, złożoności dziedziny problemu, wymagań na niezawodność oraz zabezpieczenie oraz wymagań na dokumentację
- cech procesu produkcyjnego:
  - stabilności organizacji, użytych narzędzi, procesu testowego, umiejętności ludzi oraz presji czasu
- wyników testów:
  - liczby usterek oraz pracochłonności napraw i przeróbek

5.2.8 Kandydat odróżnia od siebie dwa koncepcyjnie odmienne podejścia do szacowania: podejścia bazujące na metrykach i podejścia wykorzystujące ekspertów

Istnieją dwa podejścia do szacowania pracochłonności testów

- podejście oparte na metrykach
  - szacowanie pracochłonności testów bazując na pomiarach minionych lub podobnych projektów lub bazujące na typowych wartościach
- podejście oparte na ekspertach
  - szacowanie zadań przez ich przyszłych wykonawców lub przez ekspertów

5.2.9 Kandydat potrafi rozpoznać i uzasadnić odpowiednie kryteria wejścia oraz zakończenia konkretnych poziomów testowania oraz grup przypadków testowych (np. testów integracyjnych, testów akceptacyjnych lub przypadków testowych w testach użyteczności)

Kryteria wejścia zwykle mogą zawierać następujące zagadnienia

- dostępność i gotowość środowiska testowego
- gotowość narzędzi testowych w środowisku testowym
- dostępność testowalnego kodu
- dostępność danych testowych

Typowo kryteria zakończenia mogą składać się z

- miar staranności, takich jak pokrycie kodu, funkcjonalności lub ryzyka
- estymat gęstości błędów lub miar niezawodności
- kosztu
- istniejącego ryzyka, takiego jak niepoprawione usterki lub brak pokrycia pewnych obszarów
- harmonogramów np. zdefiniowanych na podstawie czasu do wypuszczenia produktu na rynek.

5.3 Monitorowanie postępu testów i nadzór

5.3.1 Kandydat potrafi wskazać najczęściej używane metryki do monitorowania przygotowania i wykonania testów

Często wykorzystywanymi metrykami są

- procent pracy wykonanej przy przygotowywaniu przypadków testowych lub odsetek przygotowanych przypadków testowych
- procent prac wykonanych przy przygotowywaniu środowiska testowego
- wykonanie przypadków testowych (np. liczba wykonanych/nie wykonanych przypadków testowych oraz liczba przypadków testowych zaliczonych/niezaliczonych
- informacje o usterkach (np. gęstość błędów, defekty znalezione i poprawione, współczynnik awarii oraz wyniki testów)
- pokrycie testami wymagań, ryzyka i kodu
- subiektywne zaufanie testerów do produktu
- daty kamieni milowych
- koszt testowania, włączając w to porównanie kosztu do korzyści dla znalezienia kolejnego defektu lub wykonania kolejnego testu

5.3.2 Kandydat potrafi wyjaśnić i porównać metryki stosowane w raportowaniu i kontroli testów (np. znalezione i poprawione usterki, zaliczone i niezaliczone testy)

Przykładami działań związanych z kierowaniem testami są

- podejmowanie decyzji na podstawie informacji uzyskanych z monitorowania testów
- zmiana priorytetów testów, kiedy zmaterializuje się ryzyko (np. oprogramowanie zostanie dostarczone z opóźnieniem)
- zmiana harmonogramu testów związana z dostępnością lub niedostępnością środowiska testowego
- ustanowienie kryteriów wejścia wymagających, aby programista zretestował poprawki (wykonał testy potwierdzające) zanim włączy je do wydania.

5.3.3 Kandydat potrafi omówić krótko cel i zawartość raportu podsumowującego testy zgodnie ze standardem dokumentacji testowania oprogramowania

Raportowanie testów podaje podsumowanie projektu testowego, a w tym

- co się zdarzyło w czasie testowania, np. daty spełnienia kryteriów zakończenia
- analizę informacji oraz metryk wspierającą rekomendację oraz decyzje co do przyszłych działań, takich jak pozostałe usterki, opłacalność dalszego testowania, pozostałe obszary ryzyka oraz poziom zaufania do testowanego oprogramowania

Pomiary powinny być wykonywane w trakcie oraz na koniec poziomu testów, żeby ocenić

- dopasowanie celów testów do poziomu testów
- adekwatność wybranego podejścia do testów
- skuteczność testów w odniesieniu do celów testowania

5.4 Zarządzanie konfiguracją

5.4.1 Kandydat potrafi opisać krótko, w jaki sposób zarządzanie konfiguracją wspiera testowanie

Celem zarządzania konfiguracją jest ustanowienie i utrzymanie integralności produktów (modułów, danych i dokumentacji) związanych z oprogramowaniem lub systemem przez cały projekt lub cykl życia produktu. Zarządzanie konfiguracją pomaga testerom jednoznacznie wskazać (i odtworzyć) testowany element, dokumenty testowe, testy oraz jarzmo testowe.

5.5 Ryzyka a testowanie

5.5.1 Kandydat potrafi opisać ryzyko jako potencjalny problem, który zagrażałby osiągnięciu celów projektowych jednego lub kilku interesariuszy projektu

Ryzyko projektowe, to obszary ryzyka otaczające zdolność projektu do osiągnięcia postawionych przed nim celów, takie jak

- czynniki organizacyjne
    - braki w umiejętnościach, szkoleniach lub personelu
    - problemy kadrowe
    - problemy polityczne takie jak:
        - problemy z testerami komunikującymi swoje potrzeby oraz wyniki testów
        - brak reakcji zespołu w związku z informacjami pozyskanymi podczas testów i przeglądów (np. brak doskonalenia procesów produkcji i testowania)
    - nieprawidłowe nastawienie i oczekiwania od testowania (np. niedocenianie wartości znajdowania błędów podczas testowania)
- problemy techniczne
    - problemy ze zdefiniowaniem poprawnych wymagań
    - stopień, w jakim wymagania mogą zostać spełnione przy istniejących ograniczeniach
    - środowisko testowe niegotowe na czas
    - spóźniona konwersja danych, planowanie migracji oraz rozwój i testowanie narzędzi do migracji i konwersji danych
    - niska jakość projektu, kodu, danych konfiguracyjnych, danych testowych i testów
- problemy z dostawcami
    - niewywiązywanie się dostawców ze zobowiązań
    - problemy z kontraktami

5.5.2 Kandydat pamięta, że poziom ryzyka jest określany przez prawdopodobieństwo wystąpienia oraz wpływ (potencjalną szkodę, jaką może uczynić gdy wystąpi)

Ryzyka używa się do określenia, kiedy rozpocząć testowanie oraz co powinno zostać dokładniej przetestowane. Testowanie jest wykorzystywane do zredukowania ryzyka wystąpienia niekorzystnych zdarzeń lub do zredukowania ich wpływu.

5.5.3 Kandydat rozróżnia elementy ryzyka projektowego i produktowego

5.5.4 Kandydat rozpoznaje typowe elementy ryzyka projektowego i produktowego

Potencjalne obszary wystąpienia awarii (przyszłych niekorzystnych zdarzeń lub niebezpieczeństw) w oprogramowaniu lub systemie nazywane są ryzykiem produktowym, ponieważ stanowią ryzyko dla jakości

produktu. Mogą to być:

- dostarczanie awaryjnego oprogramowania
- możliwość wyrządzenia szkody człowiekowi lub firmie przez oprogramowanie lub sprzęt
- niedostateczne atrybuty oprogramowania (np. funkcjonalność, niezawodność, użyteczność lub wydajność)
- niska jakość lub brak spójności danych (np. problemy z migracją danych, problemy z konwersją danych, problemy z przekazywaniem danych, naruszenie standardów danych)
- oprogramowanie, które nie spełnia założonych funkcji

Oparte na ryzyku podejście do testowania umożliwia w sposób proaktywny obniżanie ryzyka produktowego rozpoczynając już od wstępnej fazy projektu. Zawiera ono identyfikację obszarów ryzyka produktowego, co umożliwia użycie ich jako wskazówek w planowaniu i kontroli testów, specyfikacji, przygotowaniu oraz wykonaniu testów. W podejściu do testów opartym na ryzyku zidentyfikowane obszary ryzyka mogą zostać wykorzystane do:

- określenia technik testowania, które powinny zostać użyte
- określenia zakresu testów
- priorytetyzacji testów w celu znalezienia usterek krytycznych tak wcześnie jak to tylko możliwe
- określenia, czy nie można użyć działań niezwiązanych z testowaniem w celu redukcji ryzyka (np. przeszkolenie niedoświadczonych projektantów)

5.5.5 Kandydat potrafi opisać przy użyciu przykładów jak analiza ryzyka i zarządzanie ryzykiem mogą zostać wykorzystane przy planowaniu testów

Testowanie oparte na ryzyku bazuje na grupowej wiedzy i obserwacjach interesariuszy projektu w celu określenia obszarów ryzyka oraz poziomów testowania wymaganych, żeby odnieść się do nich.

W celu zapewnienia minimalizacji szansy wystąpienia awarii produktu, zarządzanie ryzykiem daje zdyscyplinowane podejście do:

- oceny (powtarzanej regularnie), co może pójść nie tak (ryzyka)
- ustalenia, którymi obszarami ryzyka należy się zająć
- wdrożenia działań zarządzających tymi obszarami ryzyka

5.6 Zarządzanie incydentami

5.6.1 Kandydat zna zawartość raportu incydentu zgodnie ze standardem dokumentacji testowania oprogramowania

Raport incydentów może zawierać następujące szczegóły

- datę zgłoszenia, zgłaszającą organizację oraz autora
- wyniki oczekiwane oraz rzeczywiste
- wskazanie na element testowy (element konfiguracji) oraz na środowisko
- proces cyklu życia oprogramowania lub systemu w którym incydent został zaobserwowany
- opis incydentu, w celu umożliwienia odtworzenia i rozwiązania, włącznie z logami, zrzutami baz danych oraz zrzutami ekranu
- obszar lub stopień wpływu na interesy interesariuszy

- stopień wpływu na system
- pilność, priorytet naprawy
- status incydentu (np. otwarty, odłożony, duplikat, oczekujący na naprawę, naprawiony i oczekujący na retest, zamknięty)
- podsumowania, rekomendacje oraz zgody
- zagadnienia globalne, takie jak inne obszary, na które mogą mieć wpływ zmiany związane z incydentem
- historia zmian, np. ciąg czynności podjętych przez członków zespołu w celu wyizolowania incydentu, jego naprawy oraz potwierdzenia tej naprawy
- referencje, włączając w to identyfikator specyfikacji przypadku testowego, który wykrył problem

5.6.2 Kandydat potrafi napisać raport incydentu zawierający opis awarii zaobserwowanej podczas testowania

Raporty incydentów istnieją w celu

- dostarczenia programistom i innym stronom informacji na temat problemów, aby umożliwić identyfikację, wyodrębnienie i naprawę, jeżeli okaże się to konieczne
- dostarczenia liderom testów środków do śledzenia jakości testowanego systemu oraz postępu testów
- dostarczenia pomysłów na doskonalenie procesu testowania

6.1 Typy narzędzi testowych

6.1.1 Kandydat potrafi podzielić różne typy narzędzi testowych według ich celów, według czynności w podstawowym procesie testowym oraz w cyklu życia oprogramowania

Narzędzia testowe mogą być wykorzystywane w jednej lub wielu czynnościach wspierających testowanie. Mogą to być

- narzędzia używane wprost w testach takie jak narzędzia wspierające wykonanie testów, narzędzia generujące dane testowe i narzędzia porównujące wyniki
- narzędzia wspomagające zarządzanie procesem testowym takie jak narzędzia do zarządzania testami, wynikami testów, danymi, wymaganiami, incydentami, defektami itd. oraz narzędzia raportujące i monitorujące wykonanie testów
- narzędzia używane w rozpoznaniu (eksploracji), np. narzędzia monitorujące dostęp do plików przez aplikację
- dowolne narzędzie wspierające testy (w takim znaczeniu arkusz kalkulacyjny jest również narzędziem testowym)

W zależności od kontekstu wsparcie narzędziowe dla testów może mieć jeden lub kilka z poniższych celów

- zwiększenie efektywności czynności testowych przez zautomatyzowanie powtarzających się zadań lub wsparcie dla czynności testowych wykonywanych ręcznie takich jak planowanie testów, projektowanie testów, raportowanie i monitorowanie testów
- automatyzacja czynności, które wymagałyby wielkich nakładów, gdyby były wykonywane ręcznie (np. testy statyczne)
- automatyzacja czynności, które nie mogą być wykonane ręcznie (np. testy aplikacji klient-serwer na wielką skalę)

- zwiększenie niezawodności testów (np. przez automatyzację porównywania dużej ilości danych lub symulacje)

6.1.2 Kandydat potrafi wyjaśnić pojęcie "narzędzie testowe" oraz cel wsparcia narzędziowego dla testów

Istnieje wiele narzędzi wspierających różne aspekty testowania. Narzędzia testowe można podzielić na wiele sposobów: według celów, na komercyjne, darmowe, o otwartym kodzie, shareware, według wykorzystywanej technologii i tak dalej. W tym sylabusie narzędzia są podzielone na grupy według wspieranych przez nie czynności testowych. Niektóre narzędzia wspierają jeden rodzaj czynności. Inne mogą być przydatne w różnych czynnościach testowych, ale zostały zaklasyfikowane do tej grupy, z którą są najmocniej związane. Narzędzia od jednego dostawcy, a szczególnie narzędzia, które zostały zaprojektowane do współdziałania, mogą zostać powiązane w jeden pakiet. Niektóre typy narządzi są inwazyjne w tym sensie, że samo narzędzie może wpływać na wynik rzeczywisty testu. Na przykład czasy uzyskane podczas testów wydajnościowych mogą się różnić, ponieważ narzędzie wykonuje dodatkowe instrukcje, albo można uzyskać różne wyniki pokrycia kodu. Zjawisko to jest nazywane efektem próbnika.

6.2 Skuteczne użycie narzędzi, potencjalne korzyści i ryzyko

6.2.1 Kandydat potrafi opisać krótko potencjalne korzyści oraz ryzyko automatyzacji testów oraz wspierania testów narzędziami

Potencjalnymi korzyściami z używania narzędzi są

- zredukowana zostaje powtarzająca się praca (np. uruchamianie testów regresywnych, powtórne wprowadzanie tych samych danych testowych oraz sprawdzanie zgodności ze standardami kodowania)
- zwiększa się spójność i powtarzalność (np. testy wykonywane przez narzędzie w tej samej kolejności i z tą samą częstością oraz testy wywiedzione z wymagań)
- ocena jest obiektywna (np. miary statyczne, pokrycie)
- łatwiejszy jest dostęp do danych o testach i testowaniu (np. statystyki i wykresy obrazujące postęp testów, współczynniki występowania incydentów oraz wydajność)

Ryzyko związane z narzędziami do testowania obejmuje

- nierealistyczne oczekiwania od narzędzia (co do funkcjonalności lub łatwości użycia)
- niedoszacowanie czasu, kosztów oraz pracochłonności wstępnego wdrożenia narzędzia (włączając w to szkolenia oraz ekspertów zewnętrznych)
- niedoszacowanie czasu i pracochłonności potrzebnych do osiągnięcia znaczących i trwałych korzyści z narzędzia (włączając w to potrzebę wykonania zmian w procesie testowym oraz ciągłe doskonalenie sposobu wykorzystania narzędzia)
- niedoszacowanie pracochłonności utrzymania artefaktów testowych wygenerowanych przez narzędzie zbytnie poleganie na narzędziu (zastąpienie narzędziem projektowania testów lub wykorzystywanie narzędzia, gdy testowanie ręczne byłoby lepsze)
- niewykorzystywanie kontroli wersji testaliów w narzędziu
- lekceważenie zależności i problemów z współpracą krytycznych narzędzi takich jak, narzędzia do zarządzania wymaganiami, narzędzia do kontroli wersji, narzędzia do zarządzania incydentami, narzędzia do śledzenia defektów oraz narzędzia od różnych dostawców
- słaba reakcja dostawcy w ramach wsparcia, nowych wersji oraz poprawiania usterek

- ryzyko wstrzymania projektu dla narzędzia darmowego / open-source
- nieprzewidziane, takie jak niezdolność do wspierania nowej platformy

6.2.2 Kandydat pamięta specjalne uwarunkowania dla narzędzi wspierających wykonywanie testów, analizę statyczną oraz zarządzanie testami

Narzędzia do wykonywania testów

Narzędzia do wykonywania testów uruchamiają skrypty zaprojektowane tak, aby zaimplementować testy przechowywane elektronicznie. Ten typ narzędzi często wymaga wykonania znaczącej ilości pracy, zanim zostaną osiągnięte istotne korzyści.

Narzędzia do analizy statycznej

Gdy narzędzia do analizy statycznej zostają użyte na kodzie źródłowym mogą wymusić stosowanie się do standardów kodowania, ale takie narzędzie zostanie użyte na kodzie już istniejącym może wygenerować dużą liczbę komunikatów. Ostrzeżenia nie powodują zatrzymania kompilacji kodu, ale powinny być zaadresowane, tak żeby utrzymanie kodu było w przyszłości łatwiejsze. Najwłaściwszym podejściem do wdrażania tych narzędzi jest wyłączenie niektórych komunikatów na początku, a następnie stopniowe ich włączanie.

Narzędzia do zarządzania testami

Narzędzia do zarządzania testami muszą się komunikować z innymi narzędziami lub arkuszami kalkulacyjnymi w celu dostarczenia użytecznej informacji w formacie, który najbardziej pasuje do potrzeb organizacji.

6.3 Wdrażanie narzędzi w organizacji

6.3.1 Kandydat potrafi wymienić główne zasady wprowadzania narzędzi do organizacji

Główne aspekty do wzięcia pod uwagę podczas wyboru narzędzia dla organizacji to

- ocena dojrzałości organizacji, mocnych i słabych stron oraz identyfikacja możliwości doskonalenia procesu testowania wspieranego narzędziami
- ocena według jasnych wymagań oraz obiektywnych kryteriów
- wykonanie dowodu słuszności pomysłu (proof-of-concept) z użyciem narzędzia testowego, po to żeby zbadać, czy jest ono skuteczne dla danego testowanego oprogramowania w ramach istniejącej infrastruktury lub po to, żeby określić jakie zmiany w infrastrukturze są potrzebne do skutecznego użycia narzędzia
- ocena dostawcy (włącznie ze szkoleniami, wsparciem oraz aspektami komercyjnymi) lub firm udzielających wsparcia w przypadku narzędzi niekomercyjnych
- identyfikacja wymagań wewnętrznych na doradztwo i szkolenia w użyciu narzędzia
- ocena potrzeb szkoleniowych z uwzględnieniem obecnych umiejętności automatyzacji testów przez zespół testowy
- szacowanie stosunku korzyści do kosztów na podstawie konkretnego przypadku biznesowego

6.3.2 Kandydat potrafi wymienić cele dowodu słuszności pomysłu (proof-ofconcept) w ocenie narzędzia oraz cele fazy pilotażowej we wdrażaniu tego narzędzia

Wdrażanie wybranego narzędzia w organizacji zaczyna się od projektu pilotażowego, który ma następujące cele

- szczegółowe zapoznanie się z narzędziem
- ocena czy i jak narzędzie pasuje do obowiązujących procesów i praktyk oraz ustalenie, co ewentualnie należałoby zmienić ustalenie standardów użycia, zarządzania, przechowywania oraz pielęgnacji narzędzia oraz artefaktów testowych (np. wypracowanie konwencji nazewnictwa plików i testów, stworzenie bibliotek oraz zdefiniowanie modularyzacji zestawów testów)
- ocena, czy korzyści zostaną osiągnięte przy rozsądnych kosztach

6.3.3 Kandydat uznaje, że poza samym zakupem narzędzia potrzebne jest też dobre wsparcie w jego użyciu

Czynnikami wpływającymi na sukces wdrożenia narzędzia w organizacji są

- stopniowe wdrażanie narzędzia w pozostałej części organizacji
- adaptacja i udoskonalenie procesu tak, aby pasował do sposobu używania narzędzia
- zapewnienie szkoleń oraz doradztwa nowym użytkownikom
- zdefiniowanie wytycznych co do użycia narzędzia
- wdrożenie sposobu na zbieranie użytecznych informacji z wykorzystania narzędzia
- monitorowanie wykorzystania narzędzia oraz osiąganych korzyści
- zapewnienie wsparcia dla zespołu testowego w użyciu danego narzędzia
- zbieranie wniosków z wykorzystania narzędzia przez wszystkie zespoły