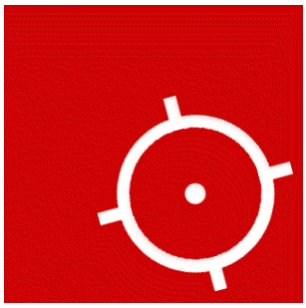

Podstawą tego rozdziału jest Foundation Level Syllabus wydany przez ISTQB.

Techniki projektowania testów

Identyfikowanie warunków testów i projektowanie przypadków testowych

W zakresie identyfikowania warunków testów i projektowania przypadków testowych możemy wyróżnić trzy kroki:
+ projektowanie testów poprzez identyfikację warunków testowych
+ tworzenie przypadków testowych
+ tworzenie procedur testowych.

Proces może być wykonany w różny sposób, od bardzo nieformalnego z niewielką ilością dokumentacji, po bardzo formalny. Poziom formalizmu zależy od kontekstu testowania, zawiera organizację, dojrzałość procesów testowania i programowania, ramy czasowe i zaangażowanych ludzi.

Podczas projektowania testów, dokumenty będące ich podstawą zostają przeanalizowane w celu zdeterminowania, co przetestować np. zidentyfikowanie warunków testowych. Warunki testowe definiuje się jako pojedyncze czynniki lub wydarzenie, które może być weryfikowane przez jeden lub więcej przypadków testowych (np. funkcja, transakcja, parametry jakości lub element struktury). Ustalenie zdolności śledzenia warunków testowych poprzez specyfikację i wymagania umożliwia zarówno analizę wpływu, gdy zmieniają się wymagania oraz pokrycie wymagań, które zostaje określone dla zbioru testów. Podczas projektowania testów szczegółowe metody testowe zostają wprowadzone w życie, bazując na identyfikacji ryzyka.

Podczas specyfikowania przypadków testowych są one wraz z danymi testowymi rozwijane i opisywane w szczegółach poprzez techniki tworzenia testów. Przypadki testowe zawierają zestaw wartości wejściowych, warunki wykonania testów, oczekiwany rezultat i warunki po wykonaniu testu, stworzone dla pokrycia konkretnych warunków testowych. Norma IEE 829 opisuje zawartość specyfikacji testowej oraz specyfikacji przypadków testowych.
Oczekiwany rezultat powinien być częścią specyfikacji przypadku testowego i zawierać wyniki, zmiany danych lub stanów i/lub inne konsekwencje testu. W przypadku, gdy oczekiwany rezultat nie został zdefiniowany wtedy pożądany, ale potencjalnie niosący błędy, rezultat może być interpretowany jako poprawny. Oczekiwany rezultat powinien w najlepszym przypadku być zdefiniowany przed wykonaniem testu.
Przypadki testowe ułożone są w porządku ułatwiającym ich wykonanie, zgodnie z wyspecyfikowaną procedurą testową. Taka procedura (lub manualny skrypt testowy) definiuje sekwencje akcji dla wykonania testu. W przypadku, jeśli test wykonywany jest automatycznie, sekwencja akcji zostaje opisana w skrypcie testowym (który jest zautomatyzowaną procedurą testową).
Różne procedury testowe i zautomatyzowane skrypty testowe są formowane w plan wykonania testów, definiujący kolejność ich wykonania. Określa on, kiedy i przez kogo procedury testowe i ewentualnie zautomatyzowane skrypty testowe są wykonane. Plan wykonania testów rozważa różne czynniki takie jak testy regresyjne, priorytety oraz techniki i logiczne zależności.

Kategorie technik projektowania testów

Celem technik projektowania testów jest identyfikacja warunków testowych i przypadków testowych.
Jest to klasyczne rozróżnienie, pokazać techniki testów jako czarnoskrzynkowe lub białoskrzynkowe. Czarnoskrzynkowe techniki (nazywane również technikami bazowanymi na specyfikacji) są metodą definiowania i wybierania warunków testowych lub przypadków testowych bazującą na analizie podstawowych dokumentów opisujących funkcjonalność oraz czynniki niefunkcjonalne, dla komponentu lub systemu bez odwołań do wewnętrznej struktury. Białoskrzynkowe techniki (nazywane strukturalnymi lub technikami bazowanymi na strukturze) są oparte na analizie wewnętrznej struktury komponentu lub systemu.

Niektóre techniki mogą być łatwo przypisane do odpowiedniej kategorii, inne mają elementy więcej niż jednej kategorii. Można się wtedy spotkać z pojęciem szaroskrzynkowej metody, czyli czymś pomiędzy białą i czarną skrzynka.
Ta publikacja odnosi się do metod opartych na specyfikacji lub doświadczeniu, takich jak czarnoskrzynkowe lub opartych na strukturze, takich jak białoskrzynkowa.

Ogólne właściwości technik opartych na specyfikacji:
+ modele oprogramowania lub jego komponentów, formalne i nieformalne, są używane dla specyfikacji problemu, który musi zostać rozwiązany
+ z tych modeli mogą być systematycznie tworzone przypadki testowe.

Ogólne właściwości technik opartych na strukturze:
+ informacja o tym jak oprogramowanie jest skonstruowane (kod, struktura) jest używane do tworzenia przypadków testowych
+ rozmiar pokrycia oprogramowania może zostać zmierzony dla istniejących przypadków testowych i dzięki ich dalszemu tworzeniu można systematycznie zwiększać pokrycie kodu.

Właściwością techniki bazowanej na doświadczeniu jest wiedza i doświadczenie ludzi. Używamy jej do stworzenia przypadków testowych np. wiedza testerów, programistów, użytkowników lub innych udziałowców na temat oprogramowania, jego użycia i środowiska lub wiedza o możliwych defektach i ich efektach.

Specyfikacyjne lub czarnoskrzynkowe
Partycje równoważne

Dane wejściowe do oprogramowania lub systemu są podzielone na grupy, dla których, można zdefiniować oczekiwane, podobne zachowanie systemu. Równoważne partycje (lub klasy) mogą być określone dla poprawnych i niepoprawnych danych np. danych, które powinny być odrzucone. Partycje mogą zostać zidentyfikowane dla danych wyjściowych, wewnętrznych wartości, wartości powiązanych z czasem (np. przed lub po zdarzeniu) i dla parametrów interfejsu (np. podczas testów integracji). Testy są zaprojektowane w celu jak najpełniejszego pokrycia partycji. Równoważne partycje mogą występować na wszystkich poziomach testowych.

Równoważna partycja jako technika może być użyta dla osiągnięcia pokrycia wejść i wyjść. Może zostać zastosowane dla danych wprowadzanych przez użytkownika, danych wejściowych podawanych poprzez interfejsy do systemu, lub parametry interfejsów podczas testowania integracji.

Analiza wartości granicznych

Zachowanie na każdej z krawędzi partycji w wielu przypadkach jest niepoprawne, więc granice są obszarami gdzie testowanie może wykryć defekty. Wartości minimalna i maksymalna partycji są jego wartościami granicznymi. Wartość graniczna dla poprawnej partycji jest poprawną wartością graniczną i odpowiednio dla niepoprawnej jest niepoprawną wartością graniczną. Testy mogą być zaprojektowane dla pokrycia obu wartości. Podczas tworzenia przypadku testowego, wybierana jest wartość dla każdej granicy.
Analiza tego rodzaju może pojawić się na wszystkich poziomach testowych. Jest relatywnie prosta do zastosowania, a jej zdolność do znajdywania defektów jest bardzo wysoka. Pomaga w tym szczegółowa specyfikacja.
Technika ta jest często postrzegana jako rozszerzenie równoważnych partycji i może zostać użyta w celu zdefiniowana danych wprowadzanych przez człowieka jak i dla czasów odpowiedzi czy granicznych wartości tabeli. Analiza pomaga również dobierać dane testowe.

Analiza tablicy decyzji

Tablice decyzji są dobrym rozwiązaniem w określaniu wymagań systemu zawierających warunki logiczne i dla udokumentowania wewnętrznej struktury systemu. Mogą zostać użyte do zapisu skomplikowanych zasad biznesowych systemu do stworzenia. Specyfikacja jest analizowana, a warunki i akcje systemu są identyfikowane. Warunki wejściowe i akcje są często deklarowane jako prawda lub fałsz. Tablica decyzji zawiera warunki rozpoczęcia, często kombinację prawdy i fałszu dla konkretnych danych wejściowych i reakcję dla każdej kombinacji warunków. Każda kolumna tabeli związana jest z zasadą biznesową, definiującą unikalną kombinację warunków, jakie wynikają z wykonania akcji powiązanej z zasadą. Standardowym pokryciem używanym w testowaniu za pomocą tabeli decyzji jest osiągnąć przynajmniej jeden test w kolumnie, co zasadniczo zawiera pokrycie wszystkich kombinacji dla warunków uruchomienia.
Zaletą testowania z użyciem tabeli decyzji jest to, że tworzy ona kombinację warunków, które nie mogą w inny sposób zostać wykonane podczas testów. Może być zastosowana do wszystkich sytuacji, kiedy akcja oprogramowania zależy od wielu logicznych decyzji.

Analiza przejścia stanów

System może przedstawiać różne odpowiedzi w zależności od aktualnych warunków oraz wcześniejszych zdarzeń. W tym przypadku, aspekty systemu mogą zostać pokazane jako diagram przejścia stanów. Umożliwia to testerowi zobaczyć oprogramowanie pod kątem jego stanów, przejść między stanami, danymi wejściowymi zdarzenia powodującego wykonanie zmiany stanu i akcji, które mogą skutkować z tych zmian. Stany oprogramowania lub obiektu poddawanego testom są oddzielane, identyfikowane i grupowane w numery. Tablica pokazuje relacje pomiędzy stanami i danymi wejściowymi i może podkreślać możliwe niepoprawne przejścia. Testy mogą być projektowane dla pokrycia typowych sekwencji stanów, dla pokrycia wszystkich stanów, dla sprawdzenia każdego przejścia, dla przetestowania specyficznych sekwencji przejść lub do testowania nieprawidłowych przejść.
Testowanie zmiany stanów jest bardziej użyteczne w testach przemysłu z zagnieżdżonym oprogramowaniem i techniczną automatyzacją. Jednakże, technika ta jest również odpowiednia dla modelowania celów biznesowych określonych poprzez stany (np. aplikacje internetowe, scenariusze biznesowe).

Analiza przypadków użycia

Test może być stworzony na bazie przypadków użycia lub scenariuszy biznesowych. Przypadki użycia opisują interakcje między użytkownikiem i systemem, które mogą prowadzić do rezultatu w postaci wartości dla użytkownika systemu. Każdy przypadek użycia ma warunki poprzedzające, które muszą zostać osiągnięte by przypadek użycia zadziałał prawidłowo. Każdy przypadek użycia kończy się warunkami końcowymi, które są obserwowalnymi wynikami i końcowym stanem systemu po zakończeniu przypadku użycia. Przypadek użycia ma zazwyczaj swój główny scenariusz, a czasami alternatywne odgałęzienia.
Przypadek użycia opisuje procesy wewnątrz systemu bazując na im najprawdopodobniejszym jego użyciu, tak więc przypadki testowe utworzone z przypadków użycia są bardzo użyteczne w odnajdywaniu defektów w procesach podczas realnego użycia systemu. Przypadki użycia, często odnoszą się do scenariuszy i są bardzo przydatne dla projektowania testów akceptacyjnych z udziałem klienta. Pomagają odkryć defekty integracji spowodowane przez interakcje i interferencje różnych komponentów, dla których pojedyncze testy komponentów nie wykryły błędów.

Strukturalne/biała skrzynka

Testowanie białoskrzynkowe jest bazowane na identyfikowalnej strukturze oprogramowania i systemu, jak w poniższych przykładach:
+ poziom komponentów - struktura będąca kodem np. deklaracje, decyzje i gałęzie
+ poziom integracji - struktura może być nazwana drzewkiem (diagram, w którym moduły wywołują inne moduły)
+ poziom systemu - struktura będąca strukturą menu, procesów biznesowych lub struktury strony internetowej.

Przedyskutujemy w tej części dwie powiązane z kodem techniki strukturalne używane dla osiągnięcia pokrycia kodu, oparte na analizie deklaracji i decyzji. Dla testowania decyzji, diagram kontroli stanów może być użyty dla lepszej wizualizacji alternatyw dla każdej decyzji.

Analiza pokrycia poleceń kodu źródłowego

W testowaniu komponentów, pokrywamy deklaracje, czyli oceniamy, w jakim procencie zostały one wykonane/sprawdzone przez zbiory testów. Testowanie decyzji tworzy przypadki testowe dla wykonania konkretnych decyzji, w normalnym przypadku by podnieść pokrycie decyzji.

Analiza pokrycia decyzji w kodzie źródłowym

Pokrycie decyzji, powiązane z testowaniem gałęzi jest ocenianiem procentu wyników decyzji (np. opcje prawda, fałsz dla deklaracji IF), które zostały wykonane przez zbiory przypadków testowych. Testowanie decyzji tworzy przypadki testowe dla wykonania konkretnych wyników decyzji, dla zwiększenia pokrycia decyzji.
100% pokrycia decyzji gwarantuje 100% pokrycia deklaracji, ale nie odwrotnie.

Inne

Możemy wyróżnić również inne analizy takie jak pokrycie warunków i pokrycie wielu warunków.
Koncept pokrycia może zostać zastosowany do innych poziomów testowych (np. poziomu integracji) gdzie procent modułów, komponentów lub klas, które zostały sprawdzone przez zbiory przypadków testowych mogą być przedstawione jako pokrycie modułów, komponentów lub klas.
Wsparcie narzędziowe automatyzacji testowania kodu jest bardzo użyteczne.

Doświadczeniowe

Prawdopodobnie najszerzej używane techniki to zgadywanie błędów. Testy są tworzone z umiejętności testera, jego intuicji i doświadczenia z podobnymi aplikacjami i technologiami. Używana zazwyczaj jako dodatek do systematycznych technik. Testowanie intuicyjne może być użyteczne dla identyfikacji specjalnych testów nieuchwytnych dla formalnych technik., głównie używana po metodach bardziej formalnych. Jednakże ta technika może przedstawiać różne poziomy efektywności, w zależności od doświadczenia testera. Metoda strukturalna, dla techniki zgadywania błędów, jest wyliczeniem możliwych błędów i zaprojektowanie testów w ten sposób by sprawdzały one te błędy. Defekty te oraz lista powodowanych przez nie problemów może być zbudowana na doświadczaniu, dostępnych danych na temat defektów i problemów, z ogólnie dostępnej wiedzy na temat problemów oprogramowania.
Testowanie uczące jest równoległym projektowaniem, wykonaniem i zapisywaniem testów, opartym na opisie celów testów i prowadzonym w określonych ramach czasowych. Jest to najbardziej przydatna metoda, gdy mamy niedostateczną lub nieadekwatną dokumentację i negatywną presję czasu, lub, gdy chcemy uzupełnić formalne testowanie. Może służyć jako sprawdzenie procesu testowego, dla wsparcie twierdzenia, że najważniejsze błędy zostały znalezione.

Dobór technik

Wybór, którą technikę wybrać zależy od kilku czynników:
+ typ systemu
+ standardy dla systemu
+ wymagania kontraktu lub klienta
+ poziom ryzyka
+ cele testów
+ dostępna dokumentacja
+ wiedza testerów
+ czas i budżet
+ typ cyklu tworzenia oprogramowania
+ model przypadków użycia
+ poprzednie doświadczenia
+ typy znalezionych defektów.

Niektóre techniki są bardziej odpowiednie do konkretnej sytuacji i poziomu testów; inne są bardziej odpowiednia dla innych poziomów.