WYKŁAD 4

Spis treści

4 Techniki testowania

4.1 Kategorie technik testowania

Każda technika testowania, włączając te omówione w bieżącym rozdziale, ma z założenia pomagać w identyfikowaniu warunków testowych, przypadków testowych i danych testowych.

Przy wyborze technik testowania, które będą stosowane, należy wziąć pod uwagę wiele czynników takich jak:

- złożoność modułu lub systemu;
- obowiązujące przepisy i normy;
- wymagania klienta lub wymagania wynikające z umów;
- poziomy ryzyka i jego rodzaj;
- dostępną dokumentację;
- wiedzę i umiejętności testerów;
- dostępne narzędzia;
- czas i budżet;
- model cyklu życia oprogramowania;
- typy defektów spodziewane w modułach i systemach.

Niektóre techniki testowania lepiej sprawdzają się w określonych sytuacjach lub na określonych poziomach testów, inne zaś można z powodzeniem stosować na każdym poziomie testów. Aby uzyskać najlepsze rezultaty, testerzy zazwyczaj tworzą przypadki testowe łącząc ze sobą różne techniki.

Użycie technik testowania w analizie, projektowaniu i implementacji testów może mieć zróżnicowany charakter: od bardzo nieformalnego (wymagającego minimalnej lub niewymagającego żadnej dokumentacji) po bardzo formalny. Właściwy stopień sformalizowania zależy od kontekstu testowania, w tym od: dojrzałości procesów testowych i procesów wytwarzania oprogramowania, ograniczeń czasowych, norm bezpieczeństwa i wymogów prawnych, wiedzy i umiejętności zaangażowanych osób oraz przyjętego modelu cyklu życia oprogramowania.

4.1.1. Kategorie technik testowania i ich cechy charakterystyczne

Techniki testowania podzielono na: czarnoskrzynkowe, białoskrzynkowe i oparte na doświadczeniu.

Techniki czarnoskrzynkowe (zwane również technikami behawioralnymi lub opartymi na specyfikacji) bazują na analizie podstawy testów np. formalnych dokumentach

zawierającychwymagania, specyfikacje, przypadki użycia, historyjki użytkownika lub opis procesów biznesowych. Techniki z tej grupy można stosować zarówno w testowaniu funkcjonalnym, jak i w testowaniu niefunkcjonalnym. Techniki czarnoskrzynkowe koncentrują się na danych wejściowych i wyjściowych przedmiotu testów, bez odwoływania się do jego struktury wewnętrznej.

Podstawą białoskrzynkowych technik testowania (zwanych 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.

Techniki testowania oparte na doświadczeniu pozwalają wykorzystać doświadczenie deweloperów, testerów i użytkowników do projektowania, implementowania i wykonywania testów. Techniki te są często stosowane razem z technikami czarnoskrzynkowymi i białoskrzynkowymi.

Najważniejsze cechy charakterystyczne czarnoskrzynkowych technik testowania:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, którą mogą stanowić wymagania na oprogramowanie, specyfikacje, przypadki użycia i historyjki użytkownika;
- przypadki testowe mogą być wykorzystywane do wykrywania rozbieżności między wymaganiami a ich implementacją bądź odstępstw od wymagań;
- pokrycie mierzy się biorąc pod uwagę przetestowane elementy podstawy testów i technikę zastosowaną do podstawy testów.

Najważniejsze cechy charakterystyczne białoskrzynkowych technik testowania:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, która może obejmować: kod, architekturę oprogramowania, szczegółowy projekt bądź dowolne inne źródło informacji o strukturze oprogramowania;
- pokrycie mierzy się biorąc pod uwagę przetestowane elementów wybranej struktury (np. kod lub interfejsy) oraz zastosowaną technikę do podstawy testów.

Najważniejsze cechy charakterystyczne technik testowania opartych na doświadczeniu:

- warunki testowe, przypadki testowe i dane testowe wyprowadza się z podstawy testów, która może obejmować wiedzę i doświadczenie testerów, deweloperów, użytkowników oraz innych interesariuszy;

- wiedza i doświadczenie mogą dotyczyć między innymi przewidywanego sposobu korzystania z oprogramowania, środowiska pracy oprogramowania oraz prawdopodobnych defektów i ich rozkładu.

4.2 Czarnostrzynkowe techniki testowania

4.2.1. Podział na klasy równoważności

Technika podziału na klasy równoważności polega na dzieleniu danych na grupy (zwane klasami równoważności lub klasami abstrakcji) w taki sposób, aby każda grupa zawierała elementy, które z założenia mają być przetwarzane w ten sam sposób.

Klasy równoważności można wyznaczać zarówno dla wartości poprawnych, jak i dla wartości niepoprawnych:

- wartości poprawne to wartości, które powinny zostać zaakceptowane przez moduł lub system, a zawierająca je klasa równoważności nosi nazwę „poprawnej klasy równoważności”;
- wartości niepoprawne to wartości, które moduł lub system powinien odrzucić, a zawierająca je klasa równoważności to „niepoprawna klasa równoważności”;
- klasy można identyfikować w odniesieniu do wszelkich elementów danych, które są związane z przedmiotem testów, takich jak: dane wejściowe, dane wyjściowe, wartości wewnętrzne bądź wartości zależne od czasu (np. występujące przed zdarzeniem lub po zdarzeniu), a także w odniesieniu do parametrów interfejsu (np. integrowanych modułów testowanych w ramach testowania integracyjnego);
- każdą klasę można w razie potrzeby podzielić na podklasy;
- każda wartość musi należeć do jednej i tylko jednej klasy równoważności;
- jeśli w przypadkach testowych są stosowane niepoprawne klasy równoważności, należy je testować indywidualnie (tzn. nie należy ich łączyć z innymi niepoprawnymi klasami równoważności), aby uniknąć zamaskowania awarii. Maskowanie awarii może mieć miejsce, gdy w tej samej chwili występuje kilka awarii, ale tylko jedna jest widoczna, przez co pozostałe awarie pozostają niewykryte;

Warunkiem uzyskania stuprocentowego pokrycia przy korzystaniu z tej techniki jest pokrycie przez przypadki testowe wszystkich zidentyfikowanych klas równoważności (włączając niepoprawne klasy równoważności), co wymaga wybrania do testów co najmniej jednej wartości z każdej klasy. Pokrycie mierzy się jako iloraz liczby klas równoważności przetestowanych przy użyciu co najmniej jednej wartości przez łączną liczbę zdefiniowanych klas równoważności i zazwyczaj wyrażane jest w procentach. Podział na klasy równoważności można stosować na wszystkich poziomach testów.

4.2.2. Analiza wartości brzegowych

Technika analizy wartości brzegowych jest rozszerzeniem techniki podziału na klasy równoważności, ale może być stosowana tylko w przypadku uporządkowanych klas zawierających dane liczbowe lub sekwencyjne. Wartościami brzegowymi klasy równoważności są jej wartość minimalna i maksymalna (lub pierwsza i ostatnia wartość).

Rozważmy następujący przykład. Załóżmy, że pole umożliwia akceptację jednoznakowej wartości całkowitej wprowadzanej z klawiatury numerycznej (tym samym zakładamy, że niemożliwe jest wprowadzenie wartości nienumerycznych). Akceptowalny zakres wartości całkowitych mieści się przedziale od 1 do 5 włącznie. W tym przypadku możemy więc wyróżnić trzy klasy równoważności: niepoprawna (wartości za niskie), poprawna, niepoprawna (wartości za wysokie). Dla poprawnej klasy równoważności wartości brzegowe to 1 i 5. Dla niepoprawnej klasy równoważności (za wysokie) wartość brzegowa to 6. Dla niepoprawnej klasy równoważności (za niskie) jest tylko jedna wartość brzegowa 0, ponieważ jest to klasa z jednym tylko przedstawicielem.

W powyższym przykładzie identyfikujemy dwie wartości brzegowe do przetestowania dla każdej granicy pomiędzy sąsiadującymi klasami. Granica pomiędzy klasą niepoprawną (za niskie) a poprawną daje dane testowe 0 i 1. Granica pomiędzy klasą poprawną i niepoprawną (za wysokie) daje dane testowe 5 i 6 Pewna odmiana tej techniki identyfikuje trzy wartości do przetestowania dla zadanej wartości brzegowej: wartość leżącą tuż przed wartością brzegową, wartość brzegową i wartość leżącą tuż za wartością brzegową. Jeśli rozpatrujemy poprawną klasę równoważności z poprzedniego przykładu, wartości testowe dla dolnej wartości brzegowej wynoszą 0, 1 i 2, a wartości testowe dla górnej wartości brzegowej — 4, 5 i 6.

Niepoprawne zachowanie jest bardziej prawdopodobne dla wartości brzegowych klas równoważności niż dla wartości z wnętrza klasy. Należy pamiętać, że zarówno wyspecyfikowane, jak i zaimplementowane wartości brzegowe, mogą zostać przesunięte powyżej lub poniżej zamierzonego położenia lub całkowicie pominięte, a ponadto mogą wystąpić niechciane nadmiarowe wartości brzegowe. Analiza wartości brzegowych i ich testowanie pozwala odkryć niemal wszystkie takie defekty, ponieważ pozwala na wykazanie w oprogramowaniu zachowań z klasy równoważności innej niż klasa, do której powinna należeć wartość brzegowa.

Analizę wartości brzegowych można stosować na wszystkich poziomach testów. Technika ta służy zwykle do testowania wymagań, które odwołują się do przedziału liczb (włączając w to daty i godziny). Pokrycie wartości brzegowych danej klasy mierzy się jako iloraz liczby przetestowanych wartości brzegowych przez łączną liczbę zidentyfikowanych wartości brzegowych, zazwyczaj wyrażony w procentach.

4.2.3. Testowanie w oparciu o tablicę decyzyjną

Tablice decyzyjne są dobrym sposobem na modelowanie złożonych reguł biznesowych, które muszą zostać zaimplementowane w systemie. Opracowując tablice decyzyjne, tester identyfikuje warunki (zazwyczaj dane wejściowe) i wynikające z nich akcje (zazwyczaj dane wyjściowe) systemu. Tworzą one wiersze tablicy, przy czym warunki znajdują się zwykle u góry, a akcje — na dole. Każda kolumna odpowiada regule decyzyjnej, która określa unikatową kombinację warunków powodującą wykonanie akcji związanych z tą regułą. Wartości warunków i akcji przedstawia się zwykle jako wartości logiczne (prawda/fałsz) lub dyskretne (np. czerwony, zielony, niebieski), ale mogą one mieć również postać liczb lub przedziałów liczb. W tej samej tablicy mogą występować różne typy warunków i akcji.

Poniżej przedstawiono typową notację stosowaną w tablicach decyzyjnych. Warunki:

- „T” oznacza, że warunek został spełniony (spotykane są również oznaczenia „P” i „1”).
- „N” oznacza, że warunek nie został spełniony (spotykane są również oznaczenia „F” i „0”).
- „—” oznacza, że wartość warunku nie ma znaczenia (spotykane jest również oznaczenie „nd”).

Akcje:

- „X” oznacza, że akcja powinna zostać wykonana (spotykane są również oznaczenia „T”, „P” i „1”).
- Puste pole oznacza, że akcja nie powinna zostać wykonana (spotykane są również oznaczenia „N”, „F” i „0”).

Pełna tablica decyzyjna zawiera tyle kolumn (przypadków testowych), ile jest niezbędne do pokrycia wszystkich kombinacji warunków. Usuwając kolumny, które nie wpływają na wynik testów, liczbę przypadków testowych można znacznie zmniejszyć, na przykład poprzez usunięcie niemożliwych do spełnienia kombinacji warunków.

Minimalnym wymogiem dotyczącym pokrycia w przypadku testowania w oparciu o tablicę decyzyjną jest zazwyczaj utworzenie co najmniej jednego przypadku testowego dla każdej reguły decyzyjnej w tablicy. Zwykle wiąże się to z koniecznością pokrycia wszystkich kombinacji warunków. Pokrycie jest mierzone jako iloraz liczby reguł decyzyjnych przetestowanych przez przynajmniej jeden przypadek testowy przez łączną liczbę reguł decyzyjnych, zazwyczaj wyrażony w procentach.

Zaletą testowania w oparciu o tablicę decyzyjną jest możliwość zidentyfikowania wszystkich ważnych, istotnych kombinacji warunków, które w innym przypadku mogłyby zostać przeoczone. Ponadto metoda ta pomaga znaleźć ewentualne luki w wymaganiach. Można ją stosować we

wszystkich sytuacjach, w których zachowanie oprogramowania zależy od kombinacji warunków oraz na dowolnym poziomie testów.

4.2.4. Testowanie przejść pomiędzy stanami

Moduł lub system mogą różnie reagować na dane wejściowe w zależności od warunków bieżących lub historycznych (np. zdarzeń, które wystąpiły od momentu uruchomienia systemu). Historię dotychczasowych zdarzeń można przedstawiać przy pomocy pojęcia stanu. Do zobrazowania możliwych stanów oprogramowania oraz przejść między nimi służy diagram przejść między stanami. Przejście inicjowane jest przez zdarzenie (np. wprowadzenie przez użytkownika wartości w pole). Wynikiem zdarzenia jest przejście ze stanu do stanu. To samo zdarzenie może skutkować dwoma lub więcej przejściami z tego samego stanu. Rezultatem zmiany stanu może być akcja oprogramowania (np. wyświetlenie wyniku lub komunikatu błędu).

Tablica przejść między stanami zawiera wszystkie poprawne przejścia między stanami (a potencjalnie również przejścia niepoprawne) oraz zdarzenia i akcje związane z poprawnymi przejściami. Diagramy przejść między stanami przedstawiają zwykle tylko poprawne przejścia, nie zawierają natomiast przejść niepoprawnych.

Testy przejść między stanami można zaprojektować w sposób zapewniający pokrycie typowej sekwencji stanów, przetestowanie wszystkich stanów bądź przetestowanie wszystkich przejść, konkretnych sekwencji przejść lub przejść niepoprawnych.

Technikę testowania przejść między stanami stosuje się w przypadku aplikacji wyposażonych w menu. Jest ona również rozpowszechniona w branży oprogramowania wbudowanego. Ponadto technika ta nadaje się do modelowania scenariuszy biznesowych zawierających określone stany oraz do testowania sposobu poruszania się (nawigacji) po ekranach. Pojęcie stanu ma charakter abstrakcyjny i może odpowiadać zarówno kilku wierszom kodu, jak i całemu procesowi biznesowemu.

Pokrycie zwykle mierzy się jako iloraz liczby przetestowanych stanów lub przejść między stanami przez łączną liczbę zidentyfikowanych stanów lub przejść wprzedmiocie testów, zazwyczaj wyrażony w procentach.

4.2.5. Testowanie oparte na przypadkach użycia

Testy można wyprowadzać zprzypadków użycia. Opisują one interakcje z elementami oprogramowania, weryfikując wymagania dla funkcjonalności. Z przypadkami użycia związane są pojęcia „aktorów” (tj. użytkowników, urządzeń zewnętrznych bądź innych modułów lub systemów) oraz „podmiotów” (moduł lub system, do którego przypadek użycia jest zastosowany).

Każdy przypadek użycia określa konkretne działanie (zachowanie), które podmiot może wykonywać we współpracy z jednym lub kilkoma aktorami (UML 2.5.1 2017). Przypadki użycia można opisywać w kategoriach interakcji i działań, warunków wstępnych bądź warunków wyjściowych, a jeśli wymagają tego okoliczności — również w języku naturalnym. Interakcje między aktorami a podmiotem mogą powodować zmianę stanu podmiotu. Do odwzorowywania takich interakcji można użyć graficznych modeli przepływów pracy (ang. workflow), diagramów aktywności lub modeli procesów biznesowych.

Przypadek użycia może obejmować potencjalne odmiany podstawowego zachowania, zachowania wyjątkowe i mechanizmy obsługi błędów (odpowiedź systemu i ewentualnie odtworzenie go po wystąpieniu awarii wynikającej z pomyłki, defektu w aplikacji lub błędu komunikacji np. skutkującego komunikatem błędu). Testy projektuje się tak, aby umożliwiały sprawdzenie wszystkich zdefiniowanych zachowań (tj. zachowania podstawowego, wyjątkowego — zwanego też alternatywnym — oraz obsługę błędów). Pokrycie można mierzyć jako iloraz liczby przetestowanych zachowań określonych przez przypadki użycia przez łączną liczbę zachowań określonych przez przypadki użycia, zwykle wyrażony w procentach.

4.3. Białoskrzynkowe techniki testowania

Testowanie białoskrzynkowe opiera się na strukturze wewnętrznej przedmiotu testów. Techniki testowania białoskrzynkowego mogą być stosowane na wszystkich poziomach testów, jednakże dwie techniki związane z kodem, które omówiono w tym podrozdziale, najczęściej stosuje się na poziomie testów modułowych. Istnieją również bardziej zaawansowane techniki, których używa się w systemach krytycznych ze względów bezpieczeństwa, w systemach o newralgicznym znaczeniu dla działalności przedsiębiorstwa oraz w środowiskach wymagających wysokiego poziomu integralności (wcelu uzyskania pełniejszego pokrycia), ale nie są one przedmiotem tego opracowania.

4.3.1. Testowanie i pokrycie instrukcji

Testowanie instrukcji służy do sprawdzania potencjalnie wykonywalnych instrukcji zawartych w kodzie. Pokrycie mierzy się procentowo jako iloraz liczby instrukcji wykonanych przez testy przez łączną liczbę instrukcji wykonywalnych w przedmiocie testów, zazwyczaj wyrażany w procentach.

4.3.2. Testowanie i pokrycie decyzji

Testowanie decyzji służy do sprawdzania decyzji zawartych w kodzie oraz kodu wykonywanego na podstawie wyników decyzji. W tym celu tworzy się przypadki testowe, które odzwierciedlają przepływy sterowania występujące po punkcie decyzyjnym. W przypadku instrukcji IF tworzy się jeden przypadek dotyczący spełnienia warunku i jeden przypadek dotyczący niespełnienia

warunku, a w przypadku instrukcji CASE niezbędne są przypadki testowe odpowiadające wszystkim możliwym wynikom, włącznie z wynikiem domyślnym.

Pokrycie mierzy się jako iloraz liczby wyników decyzji wykonanych przez testy przez łączną liczbę możliwych wyników decyzji w przedmiocie testów, zwykle wyrażony w procentach.

4.3.3. Korzyści wynikające z testowania instrukcji i testowania decyzji

Uzyskanie stuprocentowego pokrycia instrukcji kodu gwarantuje, że wszystkie instrukcje wykonywalne zawarte w kodzie zostały przetestowane co najmniej raz, nie gwarantuje natomiast, że przetestowana została cała logika decyzyjna. Jeśli chodzi o techniki białoskrzynkowe omówione w tym sylabusie, testowanie instrukcji kodu może zapewnić mniejsze pokrycie niż testowanie decyzji.

Uzyskanie stuprocentowego pokrycia decyzji oznacza, że wykonano wszystkie wyniki decyzji, czyli przetestowano zarówno wyniki „prawda”, jak i wyniki „fałsz” — nawet jeśli instrukcja warunkowa nie zawiera bloku instrukcji odpowiadającego decyzji „fałsz" (np. IF bez bloku ELSE). Pokrycie instrukcji kodu pomaga znaleźć defekty w kodzie, który nie został przetestowany przy użyciu innych testów, a pokrycie decyzji — w kodzie, w przypadku którego w innych testach nie uwzględniono wszystkich możliwych wyników decyzji.

Uzyskanie stuprocentowego pokrycia decyzji gwarantuje stuprocentowe pokrycie instrukcji kodu (ale nie odwrotnie).

4.4 Techniki testowania oparte na doświadczeniu

W ramach technik opartych na doświadczeniu przypadki testowe projektuje się z wykorzystaniem umiejętności i intuicji testerów oraz ich doświadczenia z podobnymi aplikacjami i technologiami. Techniki te pomagają w identyfikowaniu przypadków testowych, które trudno jest zidentyfikować przy użyciu innych, bardziej usystematyzowanych technik. Z drugiej strony uzyskany poziom pokrycia i skuteczność mogą bardzo różnić się w zależności od podejścia i doświadczenia testera — w pewnych przypadkach pokrycie może być trudne do oszacowania, a nawet niemożliwe do zmierzenia.

W kolejnych punktach omówiono najczęściej stosowane techniki oparte na doświadczeniu.

4.4.1. Zgadywanie błędów

Zgadywanie błędów to technika pozwalająca przewidywać wystąpienie błędów, defektów i awarii na podstawie wiedzy testera dotyczącej między innymi:

- dotychczasowego działania aplikacji;
- typowych popełnianych pomyłek;
- awarii, które wystąpiły w innych aplikacjach.

Metodyczne podejście do zgadywania błędów polega na stworzeniu listy potencjalnych pomyłek, defektów i awarii, a następnie zaprojektowaniu testów pozwalających uwidocznić te awarie i znaleźć pomyłki, które je spowodowały. Listy pomyłek, defektów i awarii można opracowywać na podstawie własnego doświadczenia, danych dotyczących defektów i awarii oraz powszechnej wiedzy na temat przyczyn awarii oprogramowania.

4.4.2. Testowanie eksploracyjne

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.

Testowanie eksploracyjne jest czasami przeprowadzane w formie tzw. testowania w sesjach, na potrzeby ustrukturyzowania aktywności. W testowaniu w sesjach, testowanie eksploracyjne odbywa się wściśle określonym przedziale czasu, a tester prowadzi testy zgodnie z kartą sesji testowej (zawierającą cele testu) do przeprowadzenia testu. Co więcej, tester może udokumentować wykonane kroki i uzyskane informacje w karcie sesji testowej.

Testowanie eksploracyjne jest najbardziej przydatne w przypadku niepełnych lub niewłaściwie sporządzonych specyfikacji bądź w przypadku testowania pod presją czasu. Ponadto może być uzupełnieniem innych, bardziej formalnych technik testowania.

Testowanie eksploracyjne jest mocno związane z reaktywną strategią testowania. W ramach testowania eksploracyjnego można korzystać także z innych technik (czarnoskrzynkowych, białoskrzynkowych i opartych na doświadczeniu).

4.4.3. Testowanie w oparciu o listę kontrolną

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. Listy kontrolne można opracowywać na podstawie własnego doświadczenia, danych dotyczących potencjalnych defektów i awarii, znajomości oczekiwań użytkowników lub wiedzy na temat przyczyn i objawów awarii oprogramowania.

Listy kontrolne można tworzyć na potrzeby różnych typów testów, w tym na potrzeby testów funkcjonalnych iniefunkcjonalnych. W przypadku braku szczegółowych przypadków testowych, testowanie w oparciu o listę kontrolną zapewnia niezbędne wytyczne i pozwala uzyskać pewien stopień spójności procesu testowania. Listy mają charakter wysokopoziomowy, w związku z czym podczas faktycznego testowania może występować pewna zmienność przekładająca się na większe pokrycie, ale kosztem mniejszej powtarzalności.