25. TESTOWANIE OPROGRAMOWANIA – IDEA, CELE, ZAKRES, RODZAJE, PODEJŚCIA/FILOZOFIA, DANE WEJŚCIOWE

SPIS TREŚCI

Testowanie oprogramowania 1
Cel testowania 2
Przebieg testowania 2
Podział testów 2
Pięć poziomów testów 2
Testy statyczne 2
Testy WhiteBox (białoskrzynkowe) 3
Testy BlackBox (czarnoskrzynkowe) 4
Testowanie przyrostowe 5
Inne metody testowania 7
Zaślepki 7
Obwody sterujące 7
Podział danych testowych ze względu na ich pochodzenie 7
Podział danych testowych ze względu na ich dziedzinę (klasę) 8
Diagramy przepływu sterowania 8

TESTOWANIE OPROGRAMOWANIA – proces związany z wytwarzaniem oprogramowania. Jest on jednym z procesów kontroli jakości oprogramowania. Testowanie ma dwa główne cele:

- Weryfikację oprogramowania
- Walidację oprogramowania.

Weryfikacja oprogramowania ma na celu sprawdzenie, czy wytwarzane oprogramowanie jest zgodne ze specyfikacją. Walidacja sprawdza, czy oprogramowanie jest zgodne z oczekiwaniami użytkownika.

CEL TESTOWANIA

Celem testowania jest zapewnienie, że program w pełni realizuje zadanie, do którego został przygotowany i w każdym przypadku generuje poprawne wyniki.

PRZEBIEG TESTOWANIA

Główną i podstawową operacją testowania jest kontrola zgodności generowanych przez program wyników z przygotowanymi wcześniej poprawnymi wynikami odniesienia. W czasie trwania procesu testowania dąży się do sprowokowania błędu – stworzenia sytuacji, w której błąd może się uwidocznić, a następnie sporządza się dokładny opis okoliczności, które doprowadziły do powstania błędu.

PODZIAŁ TESTÓW

Testy można podzielić na kilka sposobów:

- na poziomy (fazy)
- na białoskrzynkowe (strukturalne; Analiza kodu) oraz czarnoskrzynkowe (funkcjonalne; Analiza specyfikacji)
- na testy funkcjonalne i testy właściwości.

PIĘĆ POZIOMÓW TESTÓW:

1. testy jednostkowe
2. testy modułów
3. testy podsystemów
4. testy systemu
5. testy akceptacyjne albo testy alfa i testy beta

TESTY STATYCZNE

Testy statyczne są formą testowania oprogramowania bez uruchamiania programu podczas testów. Test polega na automatycznym i ręcznym sprawdzaniu kodu w celu znalezienia błędów. Najczęściej wykonywany jest przez twórców kodu, jako pierwsze i podstawowe sprawdzenie każdego programu. Testowanie statyczne sprawdza podstawową poprawność kodu i pozwala ocenić, czy program jest gotowy na bardziej szczegółowe testowanie.

W testowaniu statycznym można wyróżnić podstawową analizę kodu i precyzyjne wyszukiwanie typowych błędów.

- Analiza kodu polega na badaniu kodu poprzez czytanie i wyszukiwanie w składni błędów takich jak niezakończone pętle, złe odwołania do pamięci czy też niezainicjowane zmienne.
- Wyszukiwanie typowych błędów jest najczęściej dokonywane przez programistów poprzez dokładne czytanie ze zrozumieniem całego kodu programu w celu wykrycia typowych błędów, które nie są wykrywane przez narzędzia programistyczne, natomiast mogą doprowadzić do błędnych wykonań w programie.

WYBRANE PUNKTY TYPOWEJ LISTY KONTROLNEJ

- Błędy danych:
  - czy używa się zmiennych przed ich zainicjowaniem ?
  - czy używa się stałych?
  - czy poprawnie określono górny zakres tablic?
- Błędy sterowania
  - czy warunki w instrukcjach warunkowych i pętlach są poprawne?
  - czy wykonanie każdej pętli się zakończy?
  - czy w instrukcjach wyboru uwzględniono wszystkie przypadki?
  - czy w instrukcjach wyboru należy zawrzeć instrukcję <break>, jeśli tak, to czy zastosowano ?
- Błędy wejścia/wyjścia:
  - czy używa się wszystkich zmiennych wejściowych?
  - czy przypisano wartości zmiennym wyjściowym przed ich zwróceniem?
- Błędy interfejsu
  - czy każda funkcja i procedura wywołana jest z poprawną liczba parametrów?
  - czy typy parametrów są zgodne?
- Błędy składowania danych
  - dla struktur dynamicznych: czy przydzielana jest pamięć?
  - dla struktur dynamicznych: czy zwalniana jest pamięć?
- Wyjątki
  - czy uwzględniono wszystkie możliwe sytuacje błędne?

TESTY WHITEBOX (BIAŁOSKRZYNKOWE)

Testy strukturalne – polegają na testowaniu programu poprzez podawanie na wejściu takich danych, aby program przeszedł przez każdą zaimplementowaną ścieżkę. Zasady te są definiowane przez kryteria pokrycia wszystkich pętli oraz wszystkich warunków. Testy białej skrzynki nie są w stanie wykazać braku implementacji funkcji, którą powinien posiadać system docelowy. Sprawdzają jednak dokładnie operacje wykonywane w zaimplementowanych metodach.

Nierzadko w trakcie testowania programu techniką szklanej skrzynki wprowadzane są do wnętrza programu sztucznie specjalnie spreparowane dane w celu dokładniejszego przetestowania reakcji. Ten sposób nazywamy metodą „Słonia w Kairze".

ZALETY I WADY TESTOWANIA WZGLĘDEM KODU:

Zalety:
+ zabezpiecza przed błędami wykonania,
+ zapewnia, że wszystkie instrukcje będą się wykonywały poprawnie,
+ łatwo wybrać dane testowe (znany typ danych wejściowych i wyjściowych),
+ pomaga zoptymalizować kod aplikacji,
+ umożliwia dokładne zlokalizowanie przyczyny i miejsca występowania błędu.
Wady:
- konieczna jest znajomość tekstu źródłowego programu,
- może być bardzo żmudne i długotrwałe (trzeba kilkakrotnie przeanalizować każdy warunek i pętlę),
- trudno określić ile razy trzeba powtórzyć wykonanie programu,
- nie pozwala wykryć niezgodności ze specyfikacją (czy np. realizowane jest właściwe zadanie),
- przy dużej ilości kodu łatwo jest „przegapić" jakieś błędy (pominąć błędną linię kodu itp.).

TESTY BLACKBOX (CZARNOSKRZYNKOWE)

Testy funkcjonalne znane są także, jako testy czarnej skrzynki, ponieważ osoba testująca nie ma dostępu do informacji na temat budowy programu, który testuje. Często testy takie są wykonywane przez inne osoby niż programiści tworzący program. Nierzadko są to osoby nieposiadające wiedzy z zakresu programowania. Osoba testująca program nie opiera danych testowych na budowie wewnętrznej programu, lecz na założeniach funkcjonalnych, jakie powinien spełniać program zgodnie z dokumentacją.

W przypadku braku implementacji funkcji wymaganych przez założenia, testy funkcjonalne wykryją błąd. Zakres badanych wartości jest zwykle inny niż w przypadku testów strukturalnych.

Testy czarnej skrzynki posiadają większą szansę wykrycia błędnych wykonań, ale jednocześnie nie dostarczają zazwyczaj precyzyjnej informacji na temat przyczyny wystąpienia błędu w programie. Ze względu na szeroki zakres do przetestowania w zaawansowanych systemach informacyjnych często określa się dane testowe na podstawie względnego podobieństwa danych (klas podobnych). Dzięki temu możliwe jest przetestowanie większego zakresu danych przy jednoczesnym zmniejszeniu liczby testów o bardzo podobnych przejściach przez program.

ZALETY I WADY TESTOWANIA WZGLĘDEM SPECYFIKACJI:

Zalety:
+ testowane jest środowisko, w którym testy są przeprowadzane,
+ odpowiednie dobranie danych testowych pozwala wykryć wiele różnych błędów logicznych,
+ pozwala na uniknięcie poważniejszych błędów wynikających z niejasności sformułowań w specyfikacji
Wady:
- wyniki testów mogą być szacowane zbyt optymistycznie,
- nie wszystkie właściwości systemu mogą być przetestowane,
- nie można dokładnie określić przyczyny powstania danego błędu,
- nie daje pewności, że program nie zakończy się błędem wykonania,
- powodzenie i przebieg tego testowania w dużym stopniu zależy od doświadczenia i intuicji testującego.

TESTOWANIE PRZYROSTOWE

Testowanie przyrostowe - testowanie, podczas którego moduły lub systemy są integrowane i testowane po jednym lub kilka jednocześnie, dopóki wszystkie elementy nie zostaną zintegrowane i przetestowane. Realizacja:

- Testowanie zstępujące (z góry na dół)
- Testowanie wstępujące (z dołu do góry)

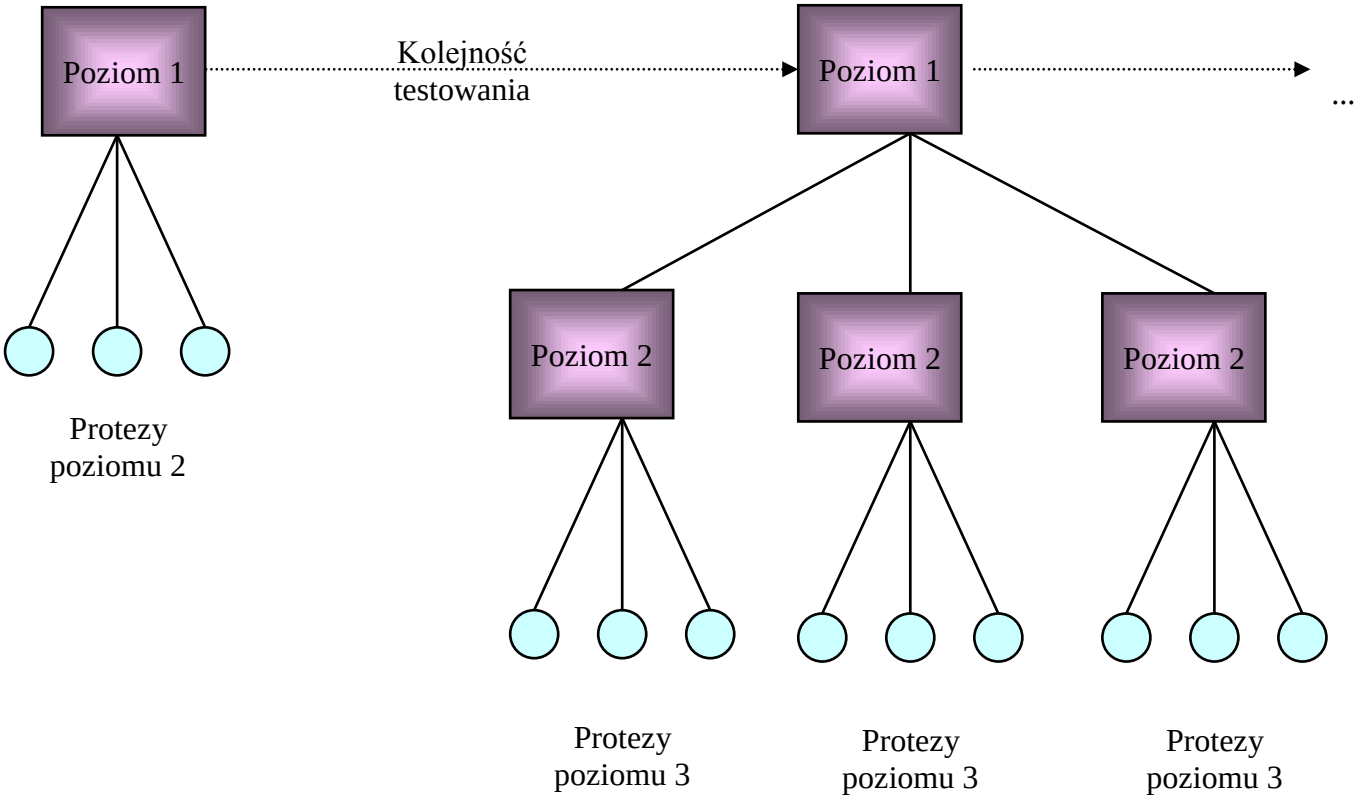
TESTOWANIE PRZYROSTOWE – STRATEGIA ZSTĘPUJĄCA

Testowanie zstępujące opiera się na zasadzie łączenia i sprawdzania fragmentów programu w kolejności z góry na dół. Testuje się najpierw program główny, używając zaślepek zamiast dotychczas niezakodowanych modułów niższego poziomu. Kolejne fazy testowania polegają na wykonywaniu programu przy stopniowym dołączaniu pełnych wersji wywoływanych modułów.

Proces jest powtarzany do momentu złożenia i przetestowania wszystkich modułów.

Zaleta: *nie trzeba tworzyć programów testujących, ani przygotowywać nowych danych testowych dla poszczególnych modułów. Umożliwia szybkie wykrycie poważnych błędów kompozycyjnych projektu, skraca czas powstawania projektu (przez łączenie elementów różnych typów testowania)*

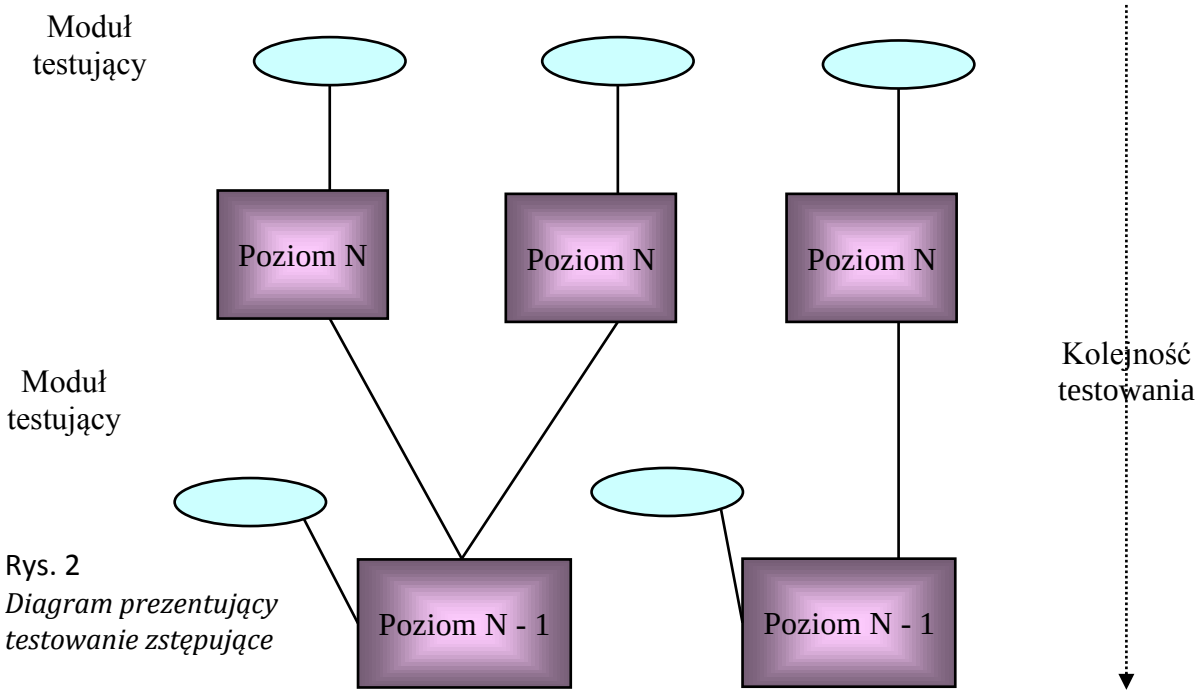
Wada: *nie ma bezpośredniego wpływu na dane przekazywane między modułami co uniemożliwia wywołanie procedur ze wszystkimi istotnymi danymi. Może się też okazać (po pewnym czasie prac), że napisanie któregoś z niższych modułów nie jest możliwe, co owocuje koniecznością zmian całej kompozycji.*



Rys. 1 Diagram prezentujący testowanie zstępujące

TESTOWANIE PRZYROSTOWE – STRATEGIA WSTĘPUJĄCA

Metoda polega na sukcesywnym pisaniu i testowaniu fragmentów programu w kolejności od dołu do góry. Najpierw pisane i testowane są moduły najniższego poziomu, następnie moduły poziomu bezpośrednio wyższego. Proces powtarza się do momentu złożenia i przetestowania głównego modułu programu.



Rys. 2
Diagram prezentujący testowanie zstępujące

***Wada**: zachodzi ryzyko niewykrycia we wczesnym etapie poważnego błędu (gdy staje się on widoczny dopiero w końcowej fazie projektu). W wyniku tego konieczne może być wprowadzanie gruntownych zmian w momencie, gdy całe prace zmierzały już ku końcowi; Trzeba też tworzyć obwody sterujące i zestawy danych dla każdego testowanego modułu.*

INNE METODY TESTOWANIA

Poza testowaniem wstępującym i zstępującym (a także ich modyfikacjami), możemy także wyróżnić testowanie mieszane (stanowiące pewne połączenie, lub bardziej kompromis pomiędzy powyższymi sposobami testowania), zmodyfikowane testowanie mieszane (połączenie testowania wstępującego i zmodyfikowanej wersji testowania zstępującego), oraz testowanie zwane big-bang (początkowe testowanie odizolowanych modułów, a następnie scalonej całości). Testowanie to jednak ma niewiele zalet, a bardzo dużo wad.

ZAŚLEPKI

Zaślepka (namiastka) modułu jest zastępczym modułem realizującym funkcje modułu właściwego w sposób uproszczony. Uproszczenie to może być mniejsze lub większe. Najbardziej uproszczony kod zaślepki może się ograniczać tylko do dostarczenia pewnych prawidłowych, ustalonych lub łatwo wyznaczalnych wartości, lub wypisywania komunikatu. Zaślepki występują przy testowaniu zstępującym.

OBWODY STERUJĄCE

Obwodem sterującym nazywamy krótki program, który pozwala na przetestowanie modułu przez zadanie określonych danych wejściowych, wywołanie wszystkich procedur modułu i obserwację dostarczanych przez nie wyników. Tworzenie obwodów sterujących powoduje wzrost kosztów projektu, jednak w przypadku elementów najniższego poziomu – procedur – trudno sobie wyobrazić rezygnację z ich przetestowania w krótkim programie przed włączeniem do dużego projektu. Obwody sterujące występują w metodzie testowania zwanej metodą wstępującą.

PODZIAŁ DANYCH TESTOWYCH ZE WZGLĘDU NA ICH POCHODZENIE

- Dane spreparowane (tworzone przez programistę lub zewnętrzny program):
  - Dane sterowane(Łatwe wyznaczenie poprawnego wyniku);
  - Dane losowe(Brak możliwości szybkiego wyznaczenia poprawnego wyniku);
- Zmodyfikowane dane faktyczne (selektywnie zmodyfikowane dane rzeczywiste, wprowadzają element realności, równocześnie umożliwiając uproszczenie wyznaczenia poprawnego wyniku końcowego);

- Dane rzeczywiste(pozwalają na przetestowanie programu w rzeczywistym środowisku, trudne wyznaczenie poprawnego wyniku końcowego);

PODZIAŁ DANYCH TESTOWYCH ZE WZGLĘDU NA ICH DZIEDZINĘ (KLASĘ)

Jako, że nie istnieje możliwość wykonania testów dla wszystkich możliwych danych wejściowych, dobrą metodą jest podzielenie danych na określone klasy. Jako kryterium stosuje się rodzaj przynależności do dziedziny danych. Do różnych klas należeć, więc będą dane:

- Z dziedziny (z uwzględnieniem przypadków szczególnych)
- Spoza tej dziedziny (z lewej i z prawej strony / z góry i z dołu)
- Z krańców dziedziny

Każdy test powinien reprezentować jedną z wyodrębnionych klas. Jeśli program prawidłowo obsłuży sprawdzane dane z jakiejś klasy, możemy uważać, że podobnie zachowa się w stosunku do pozostałych danych w tej klasie.

DIAGRAMY PRZEPŁYWU STEROWANIA

Za pomocą diagramów przepływu sterowania możemy dobrać odpowiednio dane testowe, tak, aby spowodować przynajmniej jednokrotne wywołanie każdej instrukcji w module, przy ograniczeniu ilości danych testowych. Dane testowe należy, zatem przygotować tak, aby nastąpiło wykonanie każdego rozgałęzienia w schemacie modułu. DPSy obrazują przepływ sterowania programu. Na ich podstawie można utworzyć macierz pokrycia gałęzi, co z kolei umożliwia dobór odpowiednich danych testowych dla danego modułu. Dzięki DPSom możemy znacznie zredukować ilość danych testowych, przy równoczesnej pewności, że każde rozgałęzienie w module zostało sprawdzone przynajmniej jeden raz. Zatem bez wahania można stwierdzić, iż są one nieodłącznym elementem efektywnego testowania.