Testowanie oprogramowania

Informatyka, sem.5

Rok akademicki 2018/2019

Dr inż. Dominik Olszewski

Testowanie – pojęcia podstawowe (1)

- **Pomyłka (error):** Działanie człowieka powodujące powstanie nieprawidłowego wyniku.
- **Usterka (fault):** Skutek pomyłki (błędu) twórcy oprogramowania. Usterka może, ale nie musi spowodować awarii.
- **Awaria (failure):** Odchylenie od spodziewanego zachowania albo wyniku działania oprogramowania. Skutki awarii mogą być różnej wagi.

Testowanie – pojęcia podstawowe (2)

- **Weryfikacja**
  - **“Proces kontroli systemu lub modułu polegający na** sprawdzeniu, czy produkty danego etapu produkcji spełniają warunki zadane na początku tego etapu”**.**
    - czyli
  - Sprawdzenie czy poprawnie **zbudowano aplikację**
    - czyli
  - Aplikacja jest **zgodna z jej specyfikacją.**

Testowanie – pojęcia podstawowe (3)

- **Walidacja**
  - Określenie poprawności produktów procesu tworzenia oprogramowania pod względem potrzeb i wymagań użytkownika
    - czyli
  - Sprawdzenie czy aplikacja jest **poprawna**
    - czyli
  - **Spełnione są życzenia klienta.**

Testowanie – cel działania

- Określenie poziomu jakości produktu i jego rzetelności (reliability).
- **Rzetelność (Reliability): "prawdopodobieństwo, że** aplikacja nie spowoduje awarii w **określonym czasie przy zadanych warunkach"**
- **Walidacja** i **weryfikacja** to **cele** procesu testowania, a nie jego etapy.

Testowanie – różne definicje i opisy (1)

- **Testowanie**
  - Testowanie to proces sprawdzania oprogramowania w celu zweryfikowania czy spełnia ono wyspecyfikowane wymagania. Jest to również działanie zmierzające do wykrywania błędów i określenia poziomu ryzyka po wdrożeniu.
  - Testowanie to proces wykonywania programu z intencją wykrycia błędu. To implikuje, że musi być pewien kod do wykonania.
  - Dobry przypadek testowy to taki, który ze znacznym prawdopodobieństwem spowoduje wykrycie nieznanego jeszcze błędu. Przypadki testowe (dane wejściowe) muszą być wybierane systematycznie dla poprawnego i błędnego zachowania się testowanego systemu.
  - Pomyślnie przeprowadzony test to taki, który wykrywa jeszcze nie znaleziony błąd, ma więc naturę destrukcyjną, bo niszczy skonstruowane oprogramowanie.

Testowanie – różne definicje i opisy (2)

- Testowanie
  - Testowanie nie może wykazać, że system jest bez błędów, może jedynie pokazać, że błędy istnieją (Dijkstra). {Testowanie nie jest formalną weryfikacją}.
  - Testowanie jest mało efektywnym sposobem zapewnienia jakości. {Lecz najczęściej stosowanym}.
  - Sesja testowa, zakończona sukcesem, prowadzi do fazy poprawiania błędów (*debugging*).
  - Testowanie jest procesem, więc musi być systematycznie kierowane, nadzorowanie i wspomagane przez odpowiednie narzędzia.
  - Testowanie występuje w <u>każdej</u> stosowanej metodyce tworzenia oprogramowania (kaskadowa, RUP, XP, TDD, SCRUM, inne metodyki zwinne).

Testowanie - wnioski

- Wnioski z poprzednich rozważań.
  - Trzeba wiedzieć, co jest poprawnym, a co nie poprawnym działaniem systemu – wyrocznia (ang. oracle).
  - Musi istnieć formalny zapis, do którego porównywany jest rezultat testu.
  - Pełna automatyzacja procesu testowania jest niemożliwa:
    - Teoretycznie zachowanie całego systemu (sprzęt + oprogramowanie systemowe + aplikacje) jest nieprzewidywalne.
    - W praktyce testowanie kompletne (wszystkie możliwe kombinacje wejść) jest niewykonalne.
    - Błędy użytkownika są trudne do odtworzenia (kot na klawiaturze).
    - Nie ma pewności, czy program testujący działa poprawnie.

Testowanie - uwagi

- Dlaczego należy testować systematycznie?
  - Komercyjnie produkowane oprogramowanie zawiera 3 – 30 defektów w 1000 linii kodu.
  - Przeciętny czas usunięcia jednego błędu, to do 12 godzin pracy (analityk + projektant + ponowne testowanie).
  - Utrzymanie oprogramowania to ~50% kosztów jego wytworzenia.

Przykład (książka Myers) (1)

- Program wczytuje trzy liczby typu całkowitego. Są one interpretowane jako boki trójkąta. Program ma podać, czy podane wejście odpowiada trójkątowi równobocznemu, równoramiennemu, dowolnemu, bądź trójkąt nie istnieje. Należy wybrać zbiór wejść, by być przekonanym, że program działa poprawnie.
  - Warunek istnienia trójkąta: żaden bok nie może być mniejszy lub równy zeru oraz każdy bok musi być mniejszy niż suma wszystkich dzielona przez dwa.
  - Trójkąt równoboczny ma wszystkie boki równe.
  - Trójkąt równoramienny ma dwa boki równe.
  - Trójkąt dowolny ma trzy boki różne i spełnia warunek istnienia.

Przykład (książka Myers) (2)

- W matematyce, liczb całkowitych jest nieskończenie wiele. Komputery maja określone zakresy liczb całkowitych. Niech rozpatrywany zakres zawiera 10^4 wartości.
- Liczba możliwych kombinacji dla trzech wielkości wejściowych to $10^4 \times 10^4 \times 10^4 = 10^{12}$ możliwości.
- Niech na sekundę sprawdzamy 1000 = 10^3 przypadków. Działamy 24 godziny, 365 dni w roku (rok ma 31 536 000 sekund).
- Potrzebny czas, to ponad 31,7 lat.

Przykład (książka Myers) – testy (3)

- 1. (5,4,3) – dowolny
- 2. (3,3,4) – równoramienny
- 3. (3,3,3) – równoboczny
- 4. (50,50,25) – równoramienny
- 5. (50,25,50) – równoramienny (perm)
- 6. (25,50,50) – równoramienny (perm)
- 7. (10,10,0) – nie istnieje
- 8. (3,3,-4) – nie istnieje
- 9. (5,5,10) – nie istnieje
- 10. (5,10,5) – nie istnieje (perm)
- 11. (10,5,5) – nie istnieje (perm)
- 12. (8,2,5) – nie istnieje
- 13. (8,5,2) – nie istnieje (perm)
- 14. (2,8,5) – nie istnieje (perm)
- 15. (2,5,8) – nie istnieje (perm)
- 16. (5,2,8) – nie istnieje (perm)
- 17. (5,8,2) – nie istnieje (perm)
- 18. (0,0,0) – nie istnieje
- 19. (#,4,5) - błędne dane
- 20. (3,$,5) – błędne dane
- 21. (3,4,%) – błędne dane
- 22. (,4,5) – brak danych
- 23. (4, , 5) – brak danych
- 24. (4,5,) – brak danych

Przykład (książka Myers) (4)

- Uwagi do przykładu:
  - Większość przypadków reprezentuje nie istniejące trójkąty,
  - Każdy istniejący rodzaj trójkąta testowany jest przynajmniej jeden raz,
  - Permutacje są stosowane dla sprawdzenia, czy kolejność danych nie wpływa na wynik obliczeń,
  - Wielkości graniczne są badane (zerowa długość boku, długość boku równa połowie sumy boków),
  - Wprowadzenie danych nienumerycznych zamiast numerycznych (złe typy),
  - Liczba sugerowanych testów jest niewielka w porównaniu ze wszystkimi możliwościami.

Testowanie – przykłady awarii i usterek

- Usterki w specyfikacji interfejsu
  - Rozbieżność między potrzebami klienta a usługą serwera,
  - Rozbieżności między wymaganiami a implementacją
- Usterki algorytmiczne
  - Brak inicjalizacji,
  - Rozgałęzienie (za wcześnie, za późno)
  - Brak testów nil, null.
- Usterki dokumentacji
  - Dokumentacja nie precyzuje wszystkich warunków.
- Awarie
  - Awarie wynikające z przeciążenia systemu,
  - Awarie przekroczenia warunków granicznych,
  - Awarie przekroczenia czasu odpowiedzi,
  - Awarie wydajności.

Postępowanie - awarie

- Zapobieganie awariom (zanim system zostanie wdrożony)
  - Dobre metody zmniejszające złożoność systemu,
  - Kontrola wersji zapewniająca spójność systemu,
  - Systematyczna weryfikacja zapobiegająca błędom w algorytmach.
- Wykrywanie i obsługa awarii (system działa)
  - Testowanie: wykrywanie zaplanowanych awarii,
  - Debugging: usuwanie awarii i jej przyczyn,
  - Monitoring: śledzenie stanu aplikacji, poszukiwanie usterek zmniejszających wydajność.
- Podniesienie po wykrytej usterce (awarii) działającego systemu
  - Systemy bazodanowe – poprawienie baz (transakcje atomowe).
  - Wprowadzenie nadmiarowości (redundancji).

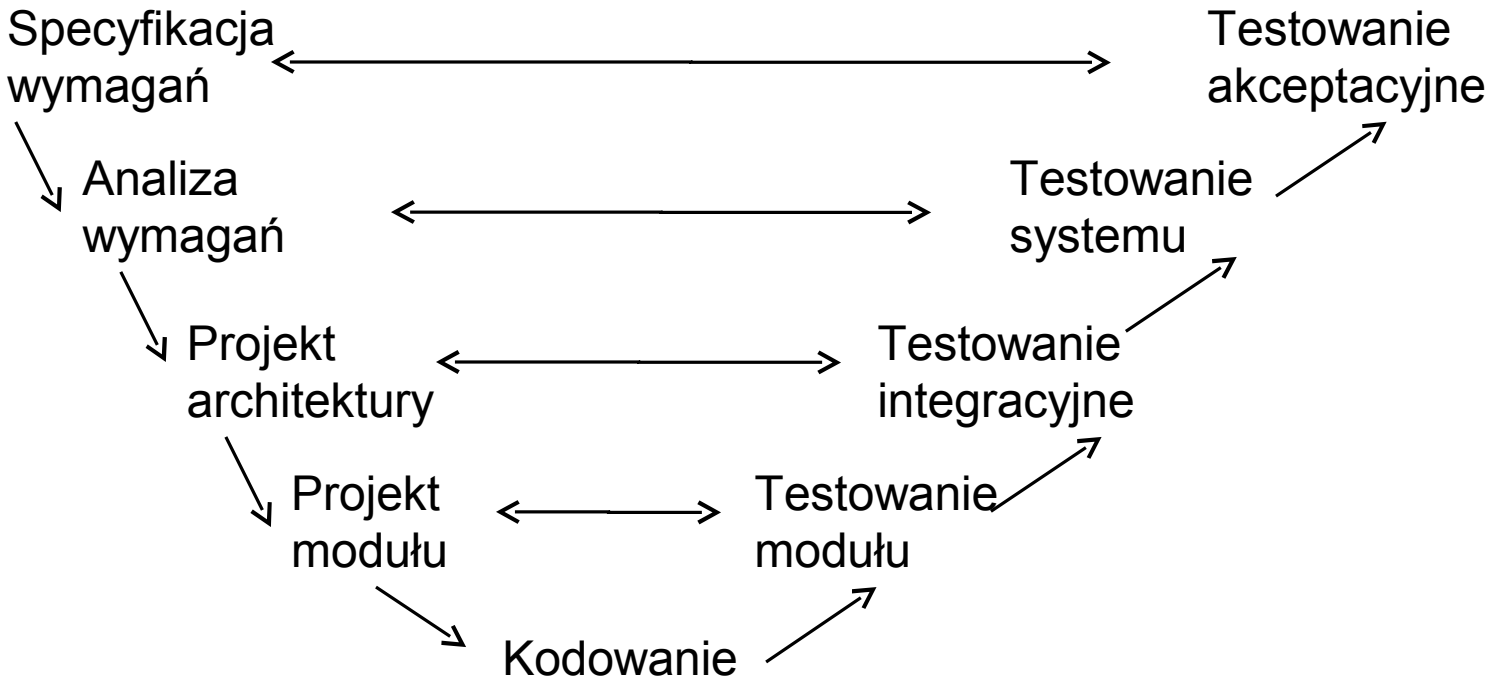
Testowanie – pewne uwagi

- Nie ma możliwości przetestowania kompletnego (gruntownego) nietrywialnego modułu oprogramowania.
  - Teoretyczny problem stopu (kiedy zatrzymać obliczenia),
  - Praktyczny problem – nie wystarczająca moc obliczeniowa i czas.
- Testowanie może tylko ujawnić usterki, lecz nie udowodni ich braku (Dijkstra).

Testowanie wymaga kreatywności

- Testowanie jest uznawane za „gorsze” zajęcie.
- By utworzyć efektywny test potrzeba:
  - Dokładnie poznać i zrozumieć badany system,
  - Znać odpowiednie techniki testowania,
  - Posiadać umiejętności do sprawnego i efektywnego zastosowania tych technik.
- Testowanie powinno być dokonywane przez niezależnych testerów
  - Programista często „widzi” to, co powinno być, a nie to, co jest w rzeczywistości.
  - Programiści często wybierają tylko takie zbiory danych, dla których system działa.
- Oprogramowanie często nie działa, gdy jest uruchamiane przez kogoś innego (nie twórcę),
  - Oby to nie był użytkownik końcowy!!

Testowanie oprogramowania - model V



Specyfikacje *Planowanie testów* *Poziom testów*

Testowanie modułu (ang. unit testing)

- Każdy moduł (klasa, metoda) testowany jest oddzielnie,
- Poziom: kod źródłowy,
- Zwykle wymagane są drajwery (ang. drivers) i zaślepki (ang. stubs).

Testowanie integracyjne

- Moduły pogrupowane w podsystemy,
- Podejście „big bang”: wszystkie moduły (z podsystemu) połączone i testowane jednocześnie.
- Podejście przyrostowe: moduły pogrupowane w podsystemy w sposób warstwowy
  Wymagane warstwowe drajwery i zaślepki.
- Interfejsy między warstwami.

Testowanie systemowe

- Testowanie kompletnego systemu (ze sprzętem, systemem operacyjnym, bazami danych, czujnikami,…),
- Cel testowania: wydajność, pojemność, odporność na błędy, bezpieczeństwo, konfiguracje sprzętu.
- Poziom: zewnętrzny interfejs systemu.

Specjalne rodzaje testów

- Testowanie pojemności systemu (ograniczenia narzędzi),
- Testowanie obciążalności systemu,
- Testowanie bezpieczeństwa,
- Testowanie wydajności,
- Testowanie konfiguracji systemu,
- Testowanie sposobów instalacji systemu,
- Testowanie procedur podnoszenia po awarii,
- Testowanie sposobów wprowadzania poprawek.
- Testowanie zgodności protokołów z innymi systemami.

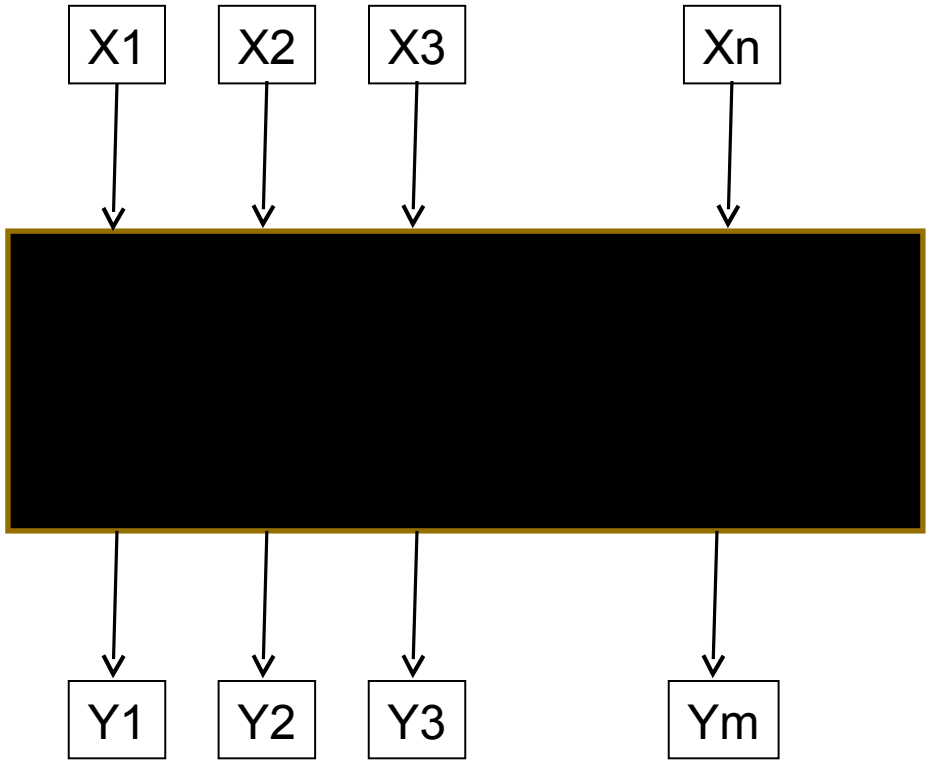
Testowanie akceptacyjne

- Testy z udziałem użytkownika (testy alpha, testy beta),
- Testy „aktualnych potrzeb”,
- Testy użyteczności w środowisku wytwórcy:
  - Testy interfejsów przez zespół wytwórcy w środowisku rozwojowym,
  - Testy interfejsów przez użytkownika w środowisku rozwojowym (kontrolowane, zapisywane, analizowane przez ekspertów).

Testowanie czarno-skrzynkowe (ang. black box testing) – testowanie funkcjonalne.

- Szczegóły implementacyjne modułów (kod) są niedostępne,
- Uwaga skoncentrowana na interfejsach użytkownika (dotyczy to modułów, podsystemów),
- Funkcjonalność obserwowana z zewnątrz, badanie zachowania wejście – wyjście.
- Bazuje na klasyfikacjach wejścia.
- Zwykle stosowane przy testowaniu systemu, testowaniu integracyjnym i testowaniu akceptacyjnym.

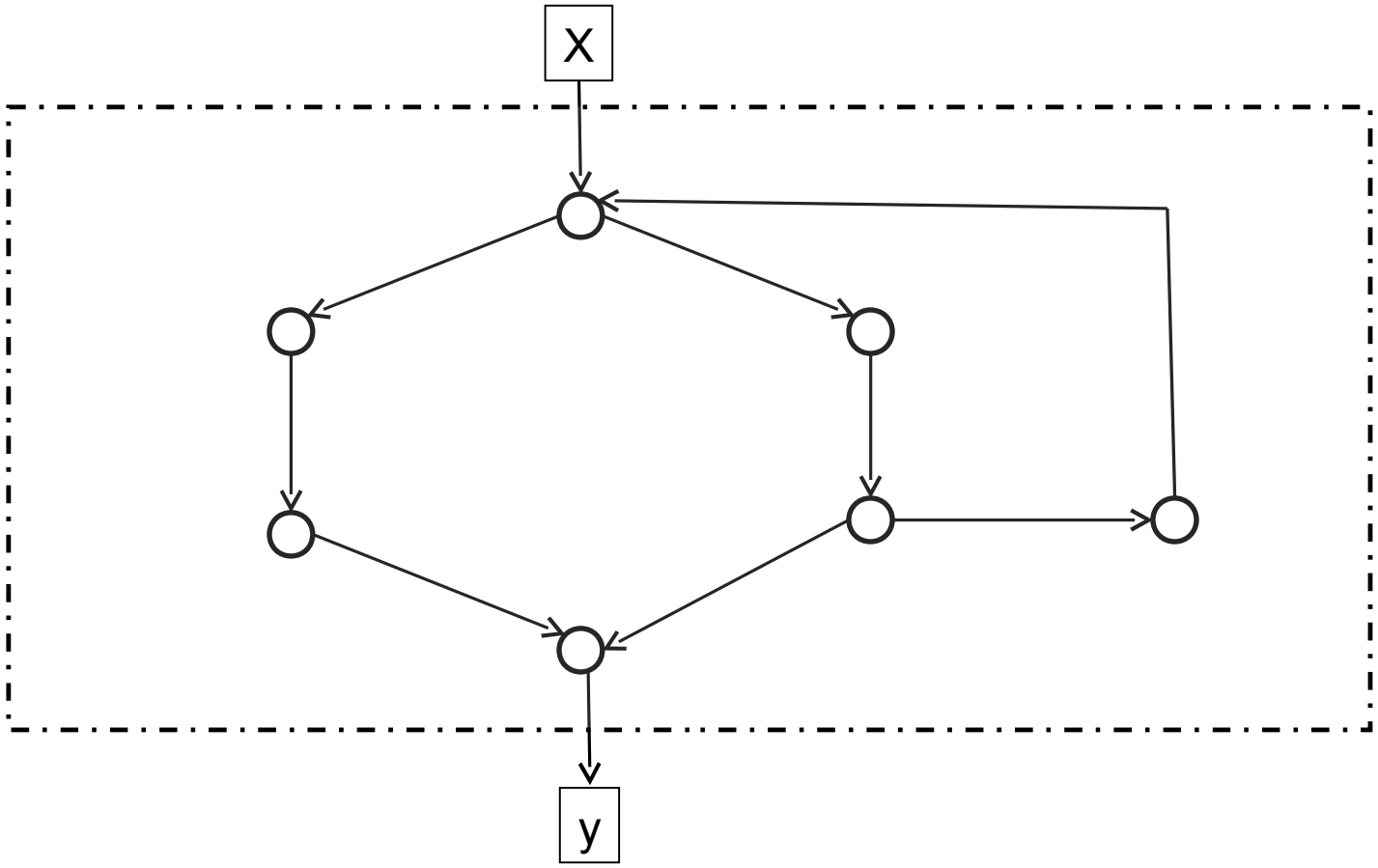
Testowanie czarno-skrzynkowe



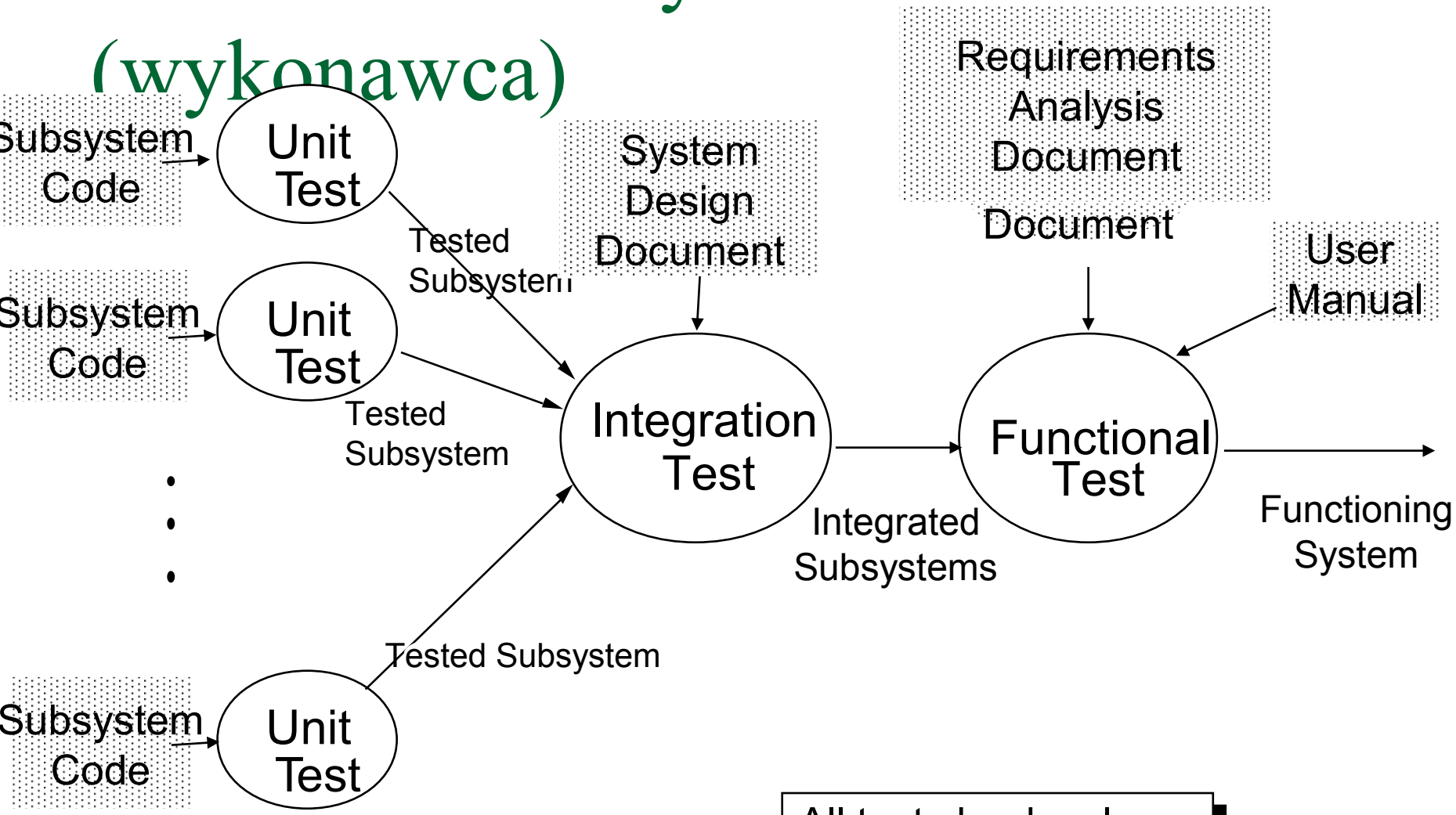
Testowanie biało-skrzynkowe – testowanie strukturalne

- Struktura modułów badana na poziomie kodu źródłowego,
- Celem jest wykonanie niezbędnej liczby wykonań modułu (funkcji) – różnych ścieżek wykonań,
- Bazuje na przepływie sterowania (ang. control flow) i przepływie danych (ang. data flow),
- Różne miary ilościowe *pokrycia* (ścieżek, gałęzi, warunków),
- Szczególnie przydatne do testowania modułów (ang. unit testing).

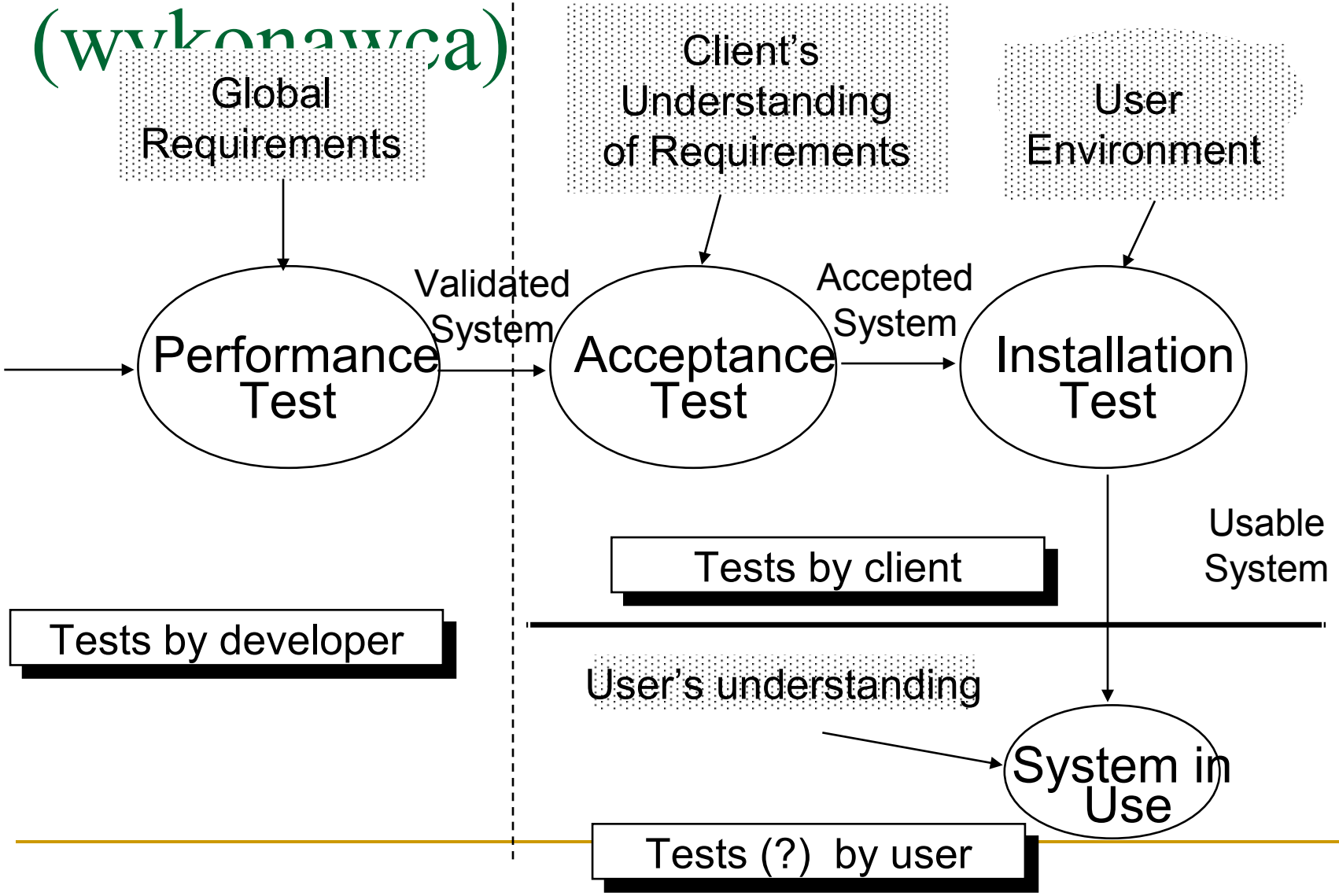
Testowanie biało-skrzynkowe



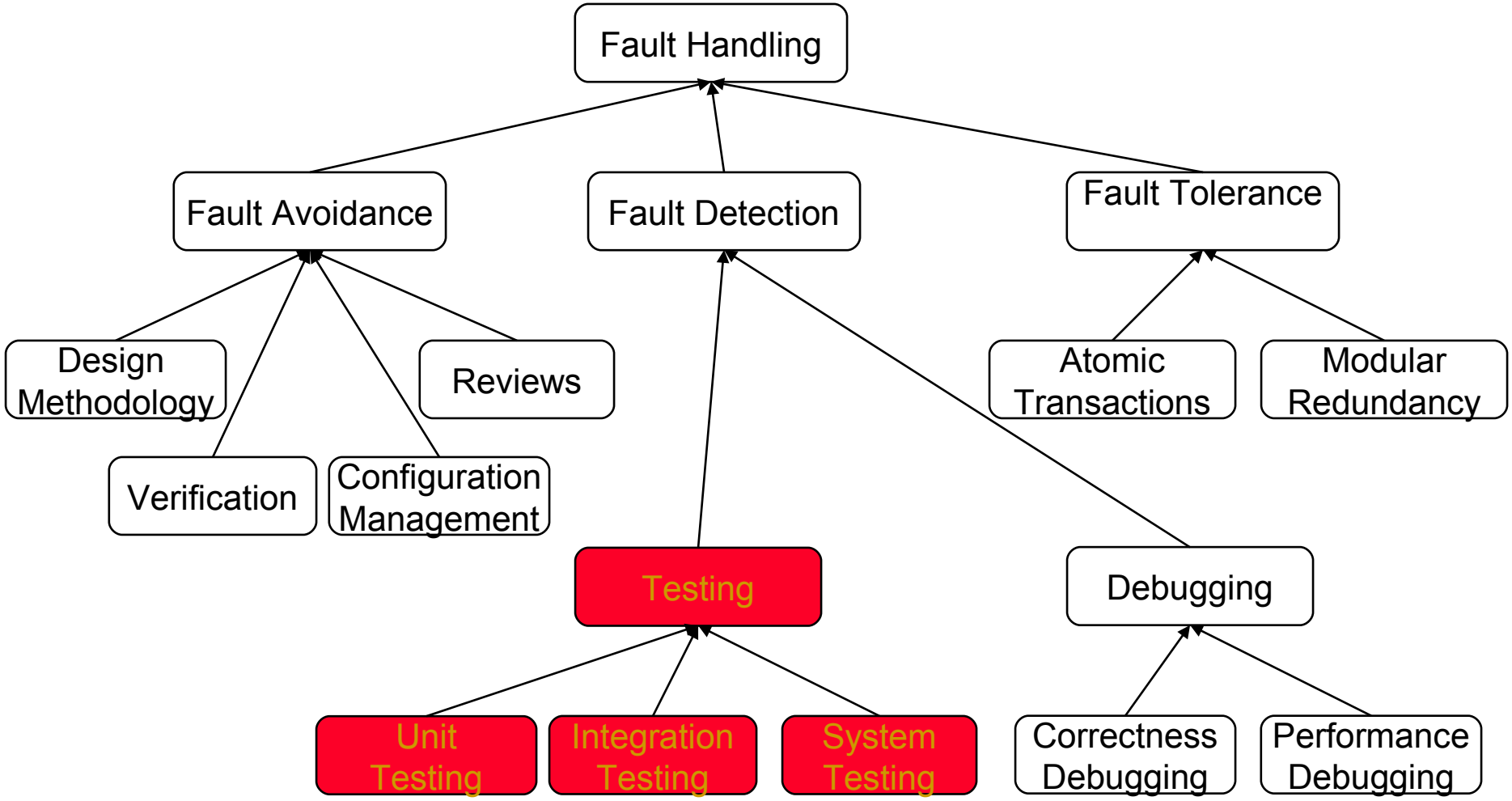
Testowanie – czynności (wykonawca)



Testowanie – czynności (wykonawca)



Testowanie – unikanie usterek



Testowanie – element procedur zapewnienia jakości (QA)