

Weryfikacja i zatwierdzanie oprogramowania

Testowanie oprogramowania

Zagadnienia

- Weryfikacja i zatwierdzanie oprogramowania
- Testowanie w trakcie rozwoju/wytwarzania
- Wytwarzanie sterowane testami
- Testy wydania
- Testy użytkowników



Weryfikacja a zatwierdzanie

verification vs validation



■ Weryfikacja

- Czy odpowiednio budujemy system
- Oprogramowanie powinno być zgodne ze specyfikacją

■ Zatwierdzanie

- Czy budujemy odpowiedni produkt
- Oprogramowanie powinno być zgodne z rzeczywistymi potrzebami użytkowników.

Względność weryfikacji i zatwierdzania



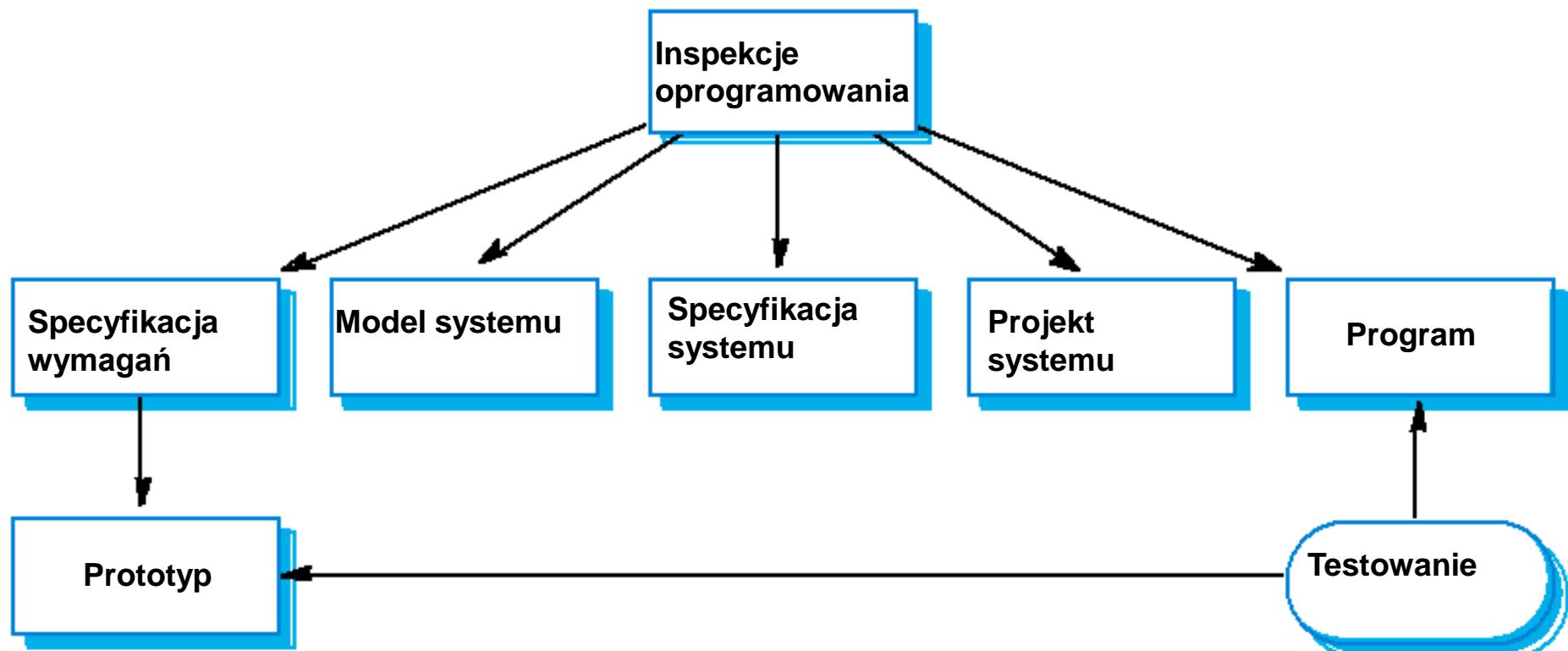
- Celem procesu weryfikacji i zatwierdzania jest osiągnięcie przeświadczenia, że system „pasuje do założeń,,.
- Zależny od:
 1. **Celu oprogramowania**
 - Wymagany poziom zaufania zależy od tego jak oprogramowanie jest krytyczne (z punktu widzenia organizacji).
 2. **Oczekiwania użytkowników**
 - Użytkownicy mogą mieć niskie oczekiwania w stosunku do pewnych rodzajów oprogramowania.
 3. **Otoczenia marketingowego**
 - Wczesne dostarczenie produktu na rynek może mieć większe znaczenie od znalezienia w nim usterek.

Inspekcje i testowanie



- **Inspekcje oprogramowania**
 - Analiza statycznej reprezentacji systemu w celu wykrycia problemów (statyczna weryfikacja)
 - Jej uzupełnieniem może być wykorzystanie narzędzi do automatyzacji dokumentowania i analizy kodu źródłowego.
- **Testowanie oprogramowania**
 - Dynamiczna analiza poprzez obserwacje zachowania systemu
 - System jest uruchamiany z danymi testowymi i dokonywana jest obserwacja jego zachowania.

Statyczna i dynamiczna weryfikacja



Inspekcje oprogramowania



- Analiza źródłowej reprezentacji systemu mająca na celu wykrycie defektów i anomalii (wymagania, kod źródłowy, projekt, dane konfiguracyjne)
- Nie wymagają działającego systemu (może być wykonana przed implementacją)
- Mogą być zastosowane do dowolnej reprezentacji systemu (wymagania, projekt, dane konfiguracyjne, dane testowe, itp.)
- Praktycznie stosowana i efektywna technika wykrywanie błędów.

Zalety inspekcji



- W czasie pojedynczej inspekcji może zostać wykrytych wiele błędów systemu.
 - w testowaniu jeden defekt może maskować inny więc wymagane jest wielokrotne uruchamianie
- Wiele usterek jest powtarzalnych co powoduje, że doświadczeni „inspektorzy” są w stanie łatwo je zlokalizować

Inspekcje a testowanie



- Techniki komplementarne a nie wykluczające
- Inspekcje nie są w stanie:
 - Zweryfikować czy system odpowiada użytkownikom
 - Sprawdzić realizacji wymagań niefunkcjonalnych
- Powinny więc być uzupełnieniem procesu testowania.

Testowanie

- **Uruchomienie** programu w kontekście sztucznych danych i wnioskowanie na podstawie rezultatów działania

Co to znaczy, że test zakończył się powodzeniem?



Testowanie oprogramowania



- Celem testowania jest wykazanie, że oprogramowanie:
 - wykonuje to czego od niego oczekujemy
 - wykrycie defektów zanim zostanie przekazane do użycia.
- Testowanie oprogramowania wymaga jego uruchomienia z wykorzystaniem sztucznych danych.
- Rezultaty wykonanego testu sprawdzane są pod kątem pojawienia się błędów, anomalii czy też informacji o niefunkcjonalnych atrybutach systemu.
- Testowanie może wykazać istnienie błędów a nie ich brak.
- Testowanie jest częścią procesu weryfikacji i zatwierdzania oprogramowania, na który składają się również inne techniki.

Cele testowania



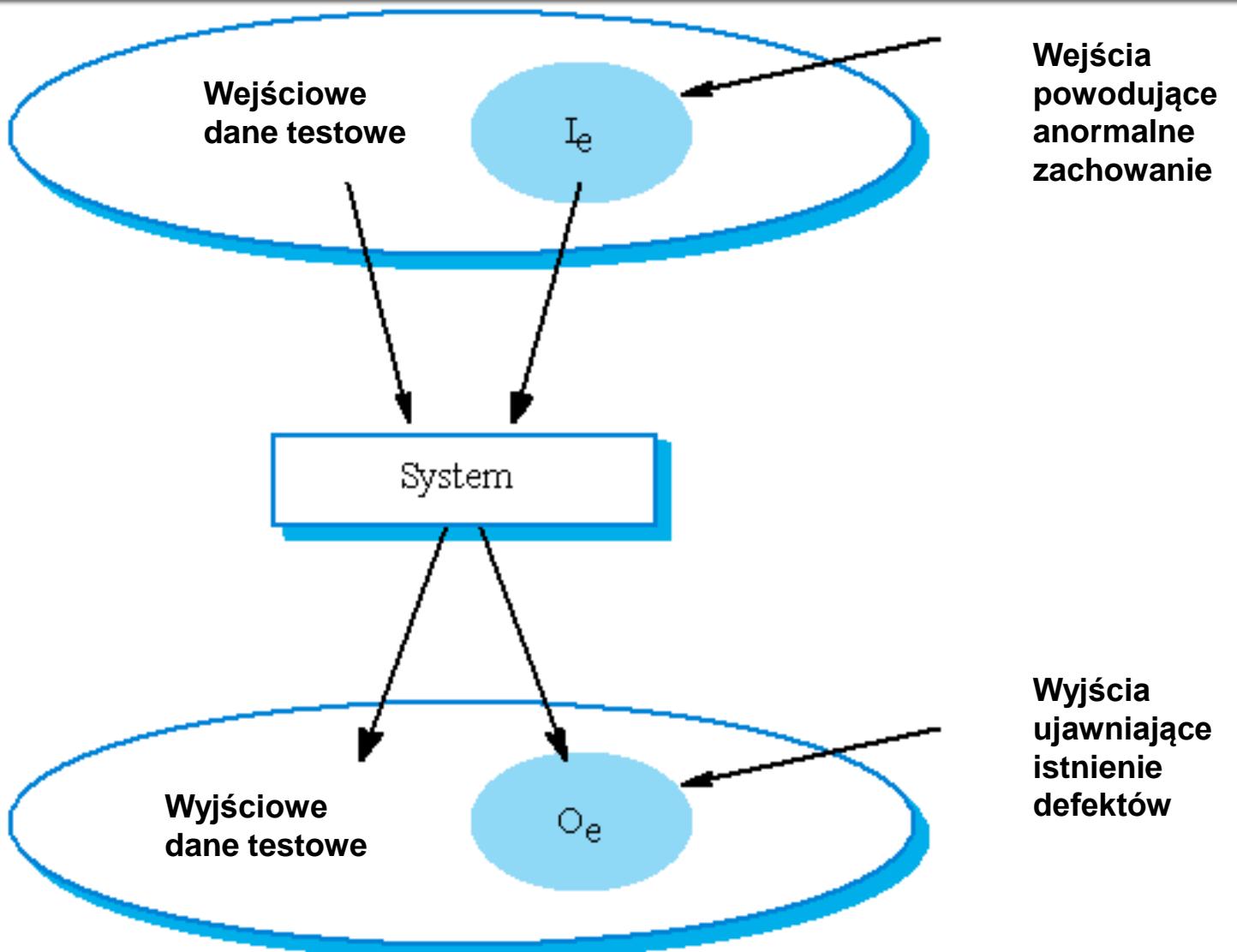
1. Wykazanie twórcom i klientom systemu, że spełnia on wymagania
 - Z punktu oprogramowania dopasowanego oznacza to, że powinien istnieć przynajmniej jeden test dla każdego wymagania. Z punktu widzenia oprogramowania generycznego oznacza to, że powinny istnieć testy dla wszystkich cech systemu oraz dla ich kombinacji (włączone do wydania).
2. Wykrycie sytuacji niepoprawnego lub nieoczekiwanej zachowania systemu oraz jego niezgodności ze specyfikacją
 - Działania podejmowane w celu wykorzenienia przypadków niepożądanego zachowania systemu (awarie, nieoczekiwana interakcja z otoczeniem, błędy w obliczeniach, uszkodzenia danych).

Testy zatwierdzające a testy usterek

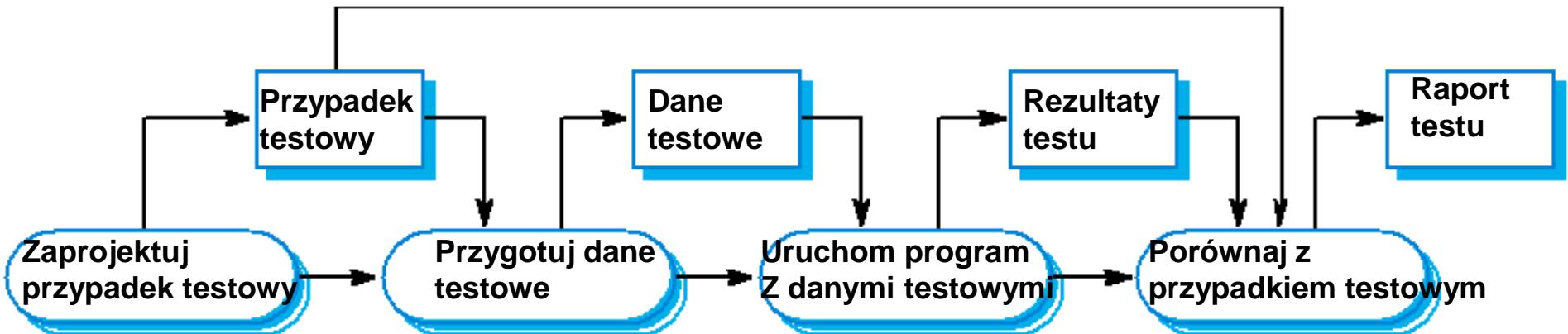


- Testy zatwierdzające (ang. validation testing)
 - Oczekujemy poprawnego działania systemu w kontekście wykonania zestawu przypadków testowych odzwierciedlających pożądane wykorzystanie systemu.
 - Sukces testu - wykazanie, że system działa zgodnie z założeniami
- Testy usterek (ang. defect testing)
 - Przypadki testowe są zaprojektowane w celu ujawnienia usterek. Przypadki testowe w trakcie testów usterek mogą być celowo komplikowane i nie jest wymagane aby odzwierciedlały normalne wykorzystanie systemu.
 - Sukces testu – zmuszenie do niepoprawnego zachowania, które eksponuje usterkę systemu.

Model testowania wejście-wyjście



Model procesu testowania oprogramowania



Etapy testowania

- 
1. Testy w trakcie rozwoju (ang. Development tests) – system jest testowany podczas produkcji w celu wykrycia błędów oraz usterek.
 2. Testy wydania (ang. Release testing) – odrębny zespół testuje kompletną wersję systemu przed jej udostępnieniem użytkownikom.
 3. Testy użytkowników (ang. User testing) – (potencjalni) użytkownicy testują system w swoim środowisku.

Testy w trakcie rozwoju



- Wszystkie aktywności związane z testowaniem wykonywane przez zespół rozwijający system.
 1. **Testy jednostkowe** (ang. Unit tests) – testowane są pojedyncze jednostki oprogramowania (np. klasy obiektów). Skupiają się na testowaniu funkcjonalności implementacji (obiektów, metod).
 2. **Testy komponentów** (ang. Component tests) – testowaniu podlega kilka pojedynczych jednostek zintegrowanych do postaci złożonego komponentu. Koncentrują się na testowaniu interfejsów komponentu.
 3. **Testy systemowe** (ang. System testing) – wybrane lub wszystkie komponenty systemu są integrowane i system jest testowany jako całość. Skupiają się na testowaniu interakcji pomiędzy komponentami.

Testy jednostkowe (1)



- Testowanie pojedynczych jednostek w **izolacji**.
 - Zależności zewnętrzne muszą być dostarczone w sposób kontrolowany
 - Kod symulujący, mock objects
- Są to testy usterek.
- Jednostką oprogramowania może być:
 - Pojedyncza funkcja lub metoda obiektu.
 - Klasa obiektów z kilkoma atrybutami i metodami.
 - Złożony komponent posiadający interfejs pozwalający na dostęp do jego funkcjonalności (np. kilka klas w pakiecie).
 - Test jednostkowy dotyczy implementacji komponentu.

Testy jednostkowe (2)



- Automatyzacja wykonania
- Pokrycie kodu testem (ang. test coverage)
- Zależności zewnętrzne:
 - mogą:
 - dostarczać niedeterministycznych rezultatów; działać wolno; mieć trudne do odtworzenia stany; jeszcze nie istnieć ...
 - ... muszą mieć „dublerów”

Testy jednostkowe a zależności zewnętrzne



- Dubler jest dla widza (testowanej jednostki) „przezroczysty”
- Rodzaje dublerów zależności zewnętrznych
 - **Dummy objects**
 - Obiekty „wypełniacze”
 - **Fake objects**
 - „Nieprodukcyjna” implementacja
 - **Stubs**
 - Dostarczają ustalonych odpowiedzi na wybrane wywołania
 - **Mocks**
 - Obiekty budowane przez specyfikowanie sposobu w jakie powinny być wywoływane (weryfikacja zachowania)

Stub vs Mock



```
public interface MailService {  
    public void send (Message msg);  
}  
  
public class MailServiceStub implements MailService {  
    private List<Message> messages = new ArrayList<Message>();  
    public void send (Message msg) {  
        messages.add(msg);  
    }  
    public int numberSent() {  
        return messages.size();  
    }  
}
```

Stub

```
class OrderStateTester...  
public void testOrderSendsMailIfUnfilled() {  
    Order order = new Order(TALISKER, 51);  
    MailServiceStub mailer = new MailServiceStub();  
    order.setMailer(mailer);  
    order.fill(warehouse);  
    assertEquals(1, mailer.numberSent());  
}
```

Test

Mock & test

```
class OrderInteractionTester...  
public void testOrderSendsMailIfUnfilled() {  
    Order order = new Order(TALISKER, 51);  
    Mock warehouse = mock(Warehouse.class);  
    Mock mailer = mock(MailService.class);  
    order.setMailer((MailService) mailer.proxy());  
  
    mailer.expects(once()).method("send");  
    warehouse.expects(once()).method("hasInventory")  
        .withAnyArguments()  
        .will(returnValue(false));  
  
    order.fill((Warehouse) warehouse.proxy());  
}
```

Testowanie klas obiektów



- Pokrycie testem
 - Przetestowanie wszystkich operacji obiektu
 - Ustawienie i odczytanie wartości wszystkich atrybutów obiektu
 - „Przećwiczenia” obiektu we wszystkich możliwych stanach.
- Testowanie klas obiektów komplikuje się w kontekście dziedziczenia (informacja, którą należy przetestować nie jest zlokalizowana w jednej klasie).

Interfejs obiektu „stacji pogodowej”



WeatherStation

identifier

reportWeather ()

reportStatus ()

powerSave (instruments)

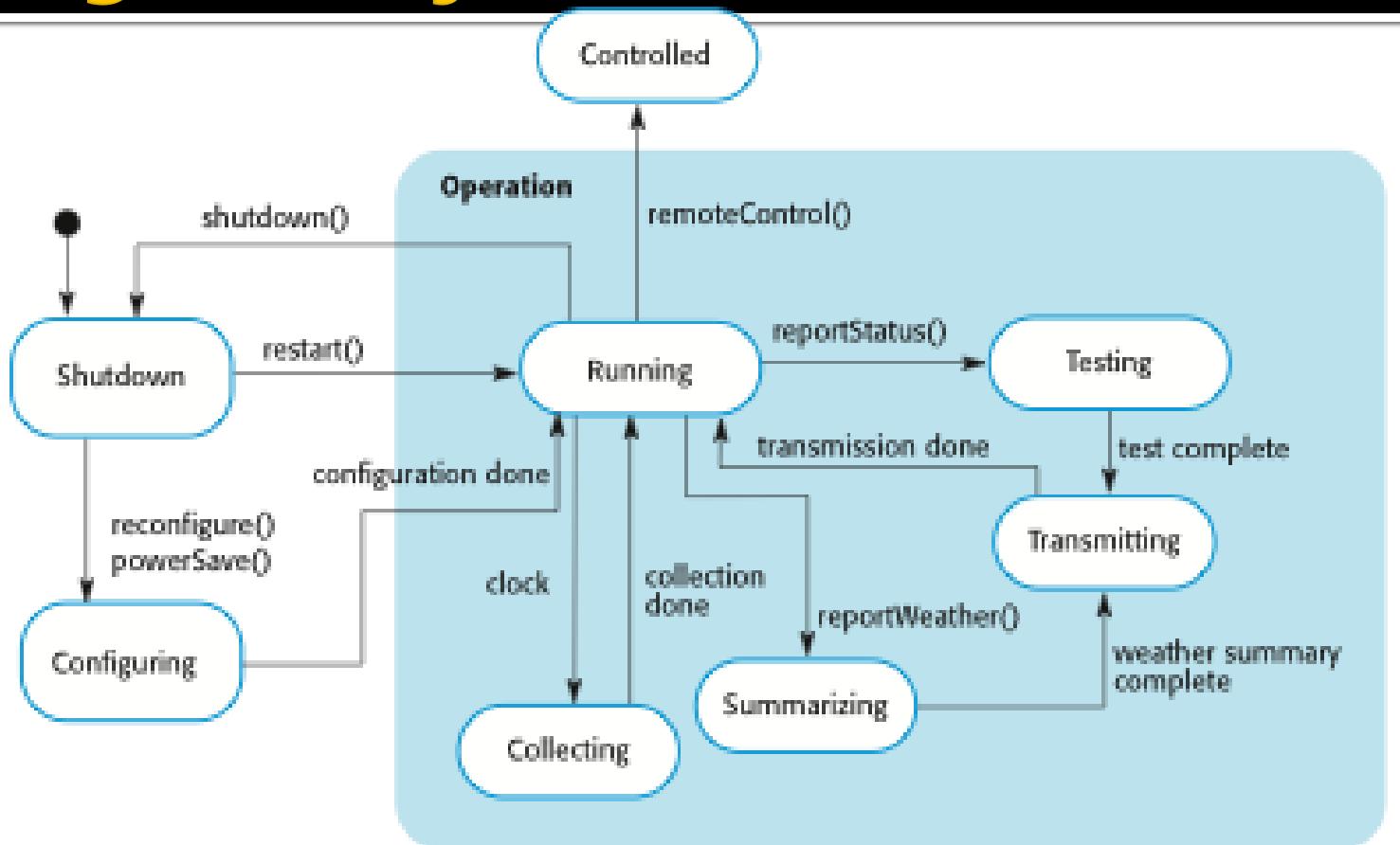
remoteControl (commands)

reconfigure (commands)

restart (instruments)

shutdown (instruments)

Diagram stanów „stacji pogodowej”



1. Shutdown -> Running-> Shutdown
2. Configuring-> Running-> Testing -> Transmitting -> Running
3. Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Testowanie implementacji „stacji pogodowej”



- Należy zdefiniować przypadki testowe dla scenariuszy użycia obiektu: reportWeather, calibrate, test, startup oraz shutdown.
- Wykorzystując model maszyny stanów należy zidentyfikować sekwencje stanów które powinny zostać przetestowane oraz sekwencję zdarzeń, która powoduje przejścia pomiędzy stanami.
- Np:
 - Shutdown -> Running-> Shutdown
 - Configuring-> Running-> Testing -> Transmitting -> Running
 - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Automatyzacja testowania

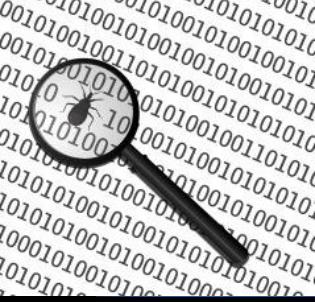


- Tam gdzie jest to możliwe, testy jednostkowe powinny być automatyczne - test jest wykonywany bez potrzeby ręcznej interwencji
- Automatyzacja wymaga wykorzystania odpowiednich narzędzi (ang. test automation framework) (np. JUnit) pozwalających na uruchamianie odpowiednio opracowanych testów.
- Narzędzia do automatyzacji testów udostępniają infrastrukturę, która pozwala na opracowywanie oraz uruchamianie przypadków testowych (ang. test cases). Przypadki testowe mogą być łączone w zestawy (ang. test suites) i uruchamiane na żądanie lub automatycznie (np. przez system automatyzacji budowania).

Kanoniczna struktura testu



1. **Inicjalizacja** (ang. setup) - inicjalizacja danych wejściowych, środowiska oraz oczekiwania w stosunku do rezultatów.
2. **Wykonanie** (ang. call) – wykonanie przypadku testowego – wywołanie metod podlegających testowaniu.
3. **Weryfikacja** (ang. assertion) **założeń** – porównanie rezultatów wywołania z oczekiwaniemi. Jeżeli założenia są spełnione (asercja ewaluuje do wartości prawda) – test zakończy się sukcesem. Jeżeli asercja ewaluuje do wartości fałsz – test zawiódł.
4. **Wyburzanie** – (ang. tear down) – sprzątanie po teście (np. usuwanie pozostawionego stanu, który mógłby mieć wpływ na działanie innych testów).



Skuteczność testu jednostkowego

- Przypadki testowe:
 1. Powinny wykazać, że jeżeli jednostka jest użyta zgodnie z oczekiwaniami wykonuje to co powinna.
 2. Jeżeli komponent posiada usterki – przypadki testowe powinny je ujawnić.
- Prowadzi to do 2 typów przypadków testowych:
 1. Przypadki testowe odzwierciedlające normalne operacje – wykazujące działanie zgodne z oczekiwaniami.
 2. Przypadki testowe wykorzystujące nieprawidłowe dane wejściowe w celu kontroli czy komponent odpowiednio na nie reaguje. Tego typu testy bazują na wiedzy na temat najczęściej pojawiających się problemów.

Strategie testowania



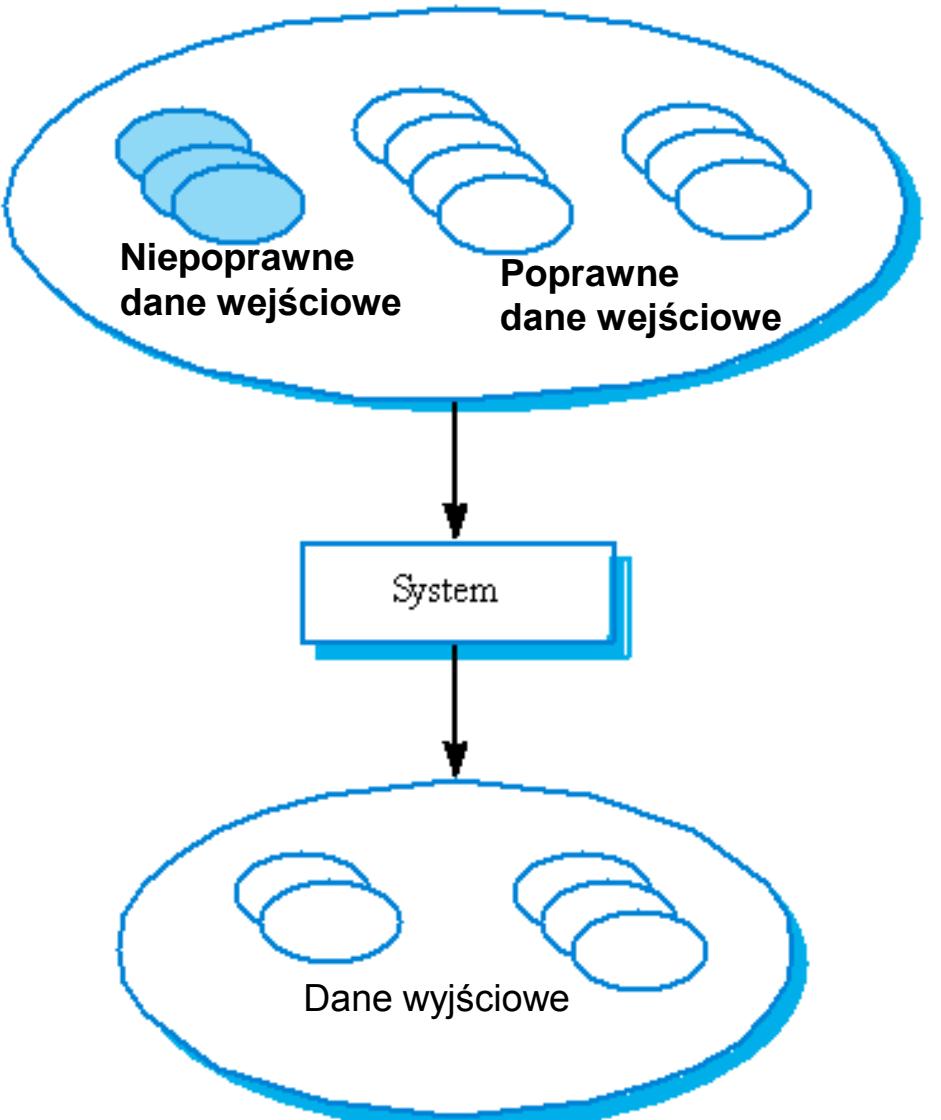
1. Testowanie z wykorzystaniem klas równoważności
– identyfikacja grup danych wejściowych posiadających tę samą charakterystykę, które powinny być przetwarzane w taki sam sposób.
 - Testy powinny dotyczyć każdej grupy.
2. Testy bazujące na doświadczeniu – dobór przypadków testowych na podstawie wiedzy (np. najczęściej popełniane przez programistów błędy).

Klasy równoważności



- Dane wejściowe oraz rezultaty można często podzielić na klasy (dziedziny).
 - Liczby dodatnie, liczby ujemne, napisy bez białych znaków
- W ramach wartości należących do danej klasy program zachowuje się w porównywalny sposób.
 - Przypadki testowe powinny być określone ta, aby uwzględniały wszystkie klasy.

Klasy równoważności



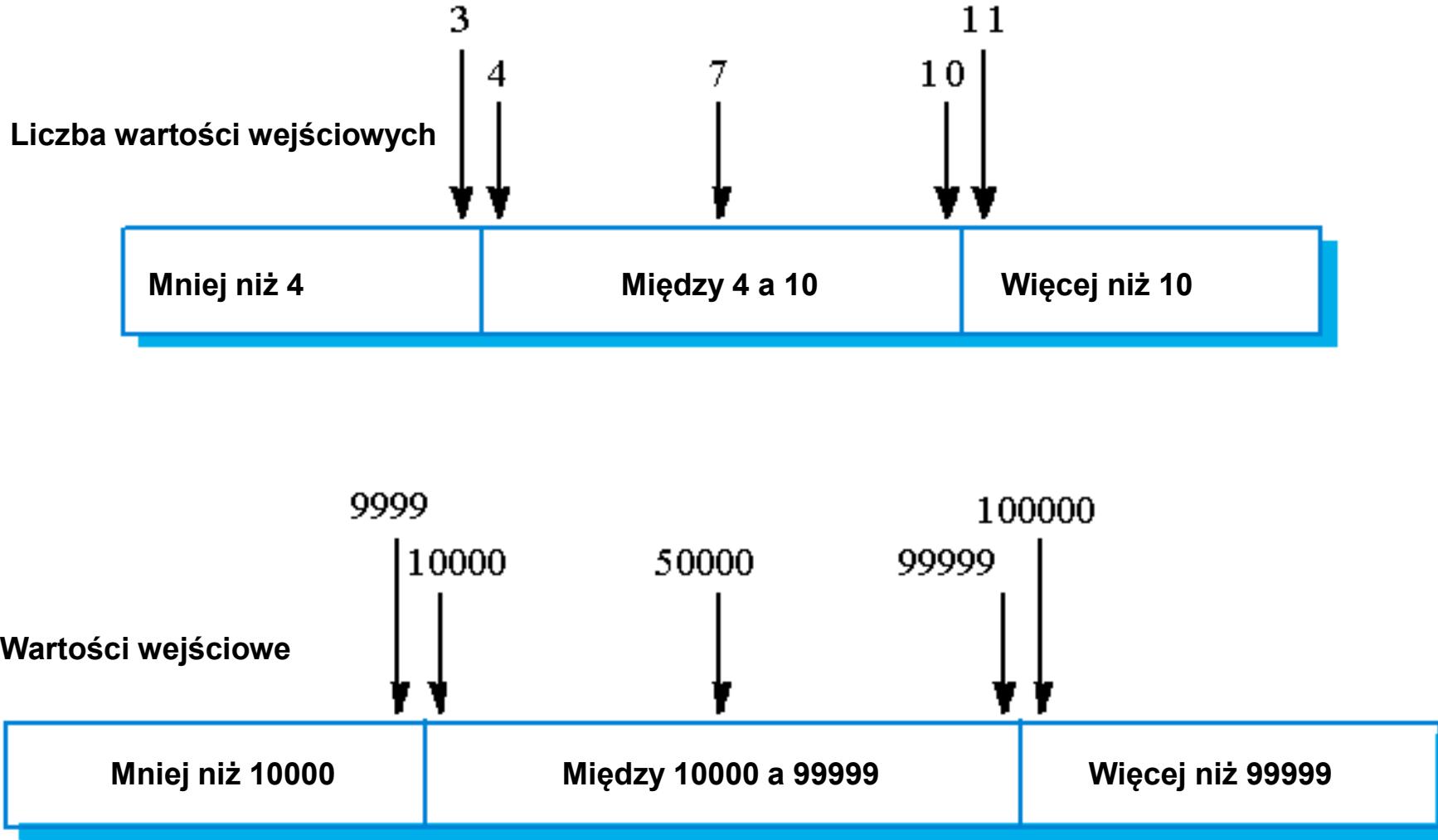
Pomysły na testy - Burza mózgów



- Pole przyjmuje wartości całkowite z przedziału <20,50>
- Jakie testy powinniśmy przeprowadzić?



Wybieranie wartości testowych dla klas równoważności



Przykład – procedura wyszukująca



```
procedure Search (K : ELEM ; T: SEQ of ELEM;  
    Found : in out BOOLEAN; L: in out ELEM_INDEX);
```

Pre-condition

-- ciąg ma przynajmniej jeden element
 $T'FIRST \leq T'LAST$

Post-condition

-- element został znaleziony i L jest jego referencją
(Found **and** $T(L) = K$)

or

-- elementu nie ma w ciągu
(**not** Found **and**
not (**exists** i, $T'FIRST \geq i \leq T'LAST, T(i) = K$))

Procedura wyszukiwania – klasy równoważności (1)



- Dane wejściowe, w których element kluczowy znajduje się w ciągu (Found = true)
- Dane wejściowe, w których element kluczowy nie występuje w ciągu
- ...

Dodatkowe założenia – projektowanie testów na podstawie doświadczenia



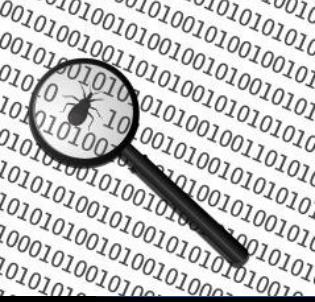
- Testuj na ciągach, które składają się tylko z jednej wartości
- Użyj ciągów rozmaitych rozmiarów w różnych testach
- Opracuj testy, aby można było odczytać pierwszy, środkowy i ostatni element ciągu
- Testuj z wykorzystaniem sekwencji o zerowej długości

Procedura wyszukiwania – klasy równoważności (2)



- Dane wejściowe, w których element kluczowy znajduje się w ciągu (Found = true)
- Dane wejściowe, w których element kluczowy nie występuje w ciągu
- Ciąg wejściowy zawiera jedną wartość
- Liczba elementów ciągu wejściowego jest większa niż 1

Ogólne wskazówki dotyczące testowania

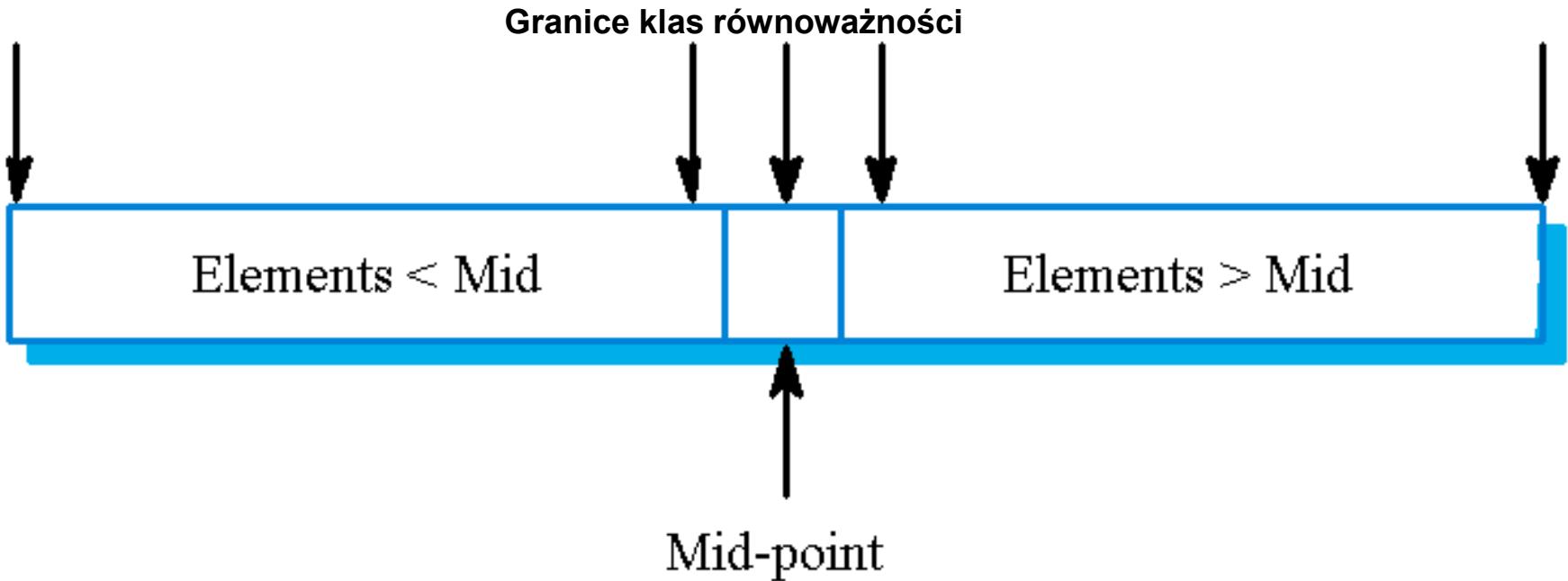


- Wybierz takie dane wejściowe, które spowodują wygenerowanie wszystkich komunikatów o błędach.
- Zaprojektuj wejścia w taki sposób, aby spowodowały przepełnienie bufora.
- Wielokrotnie powtarzaj te same dane wejściowe lub ich serie.
- Wymuś wygenerowanie niepoprawnych danych wyjściowych.
- Wymuś wykonanie obliczeń, których rezultat będzie zbyt mały lub zbyt duży.

Testy strukturalne

- Nazywane również testami białej skrzynki (ang. white-box testing)
- Opracowywane na podstawie wiedzy o strukturze i implementacji oprogramowania
- Celem jest zapewnienie aby każde instrukcja programu była wykonana co najmniej raz (nie wszystkie możliwe ścieżki programu).
- Możliwe do opracowania dla stosunkowo niewielkich komponentów

Dodatkowe klasy równoważności



Wyszukiwanie binarne – klasy równoważności

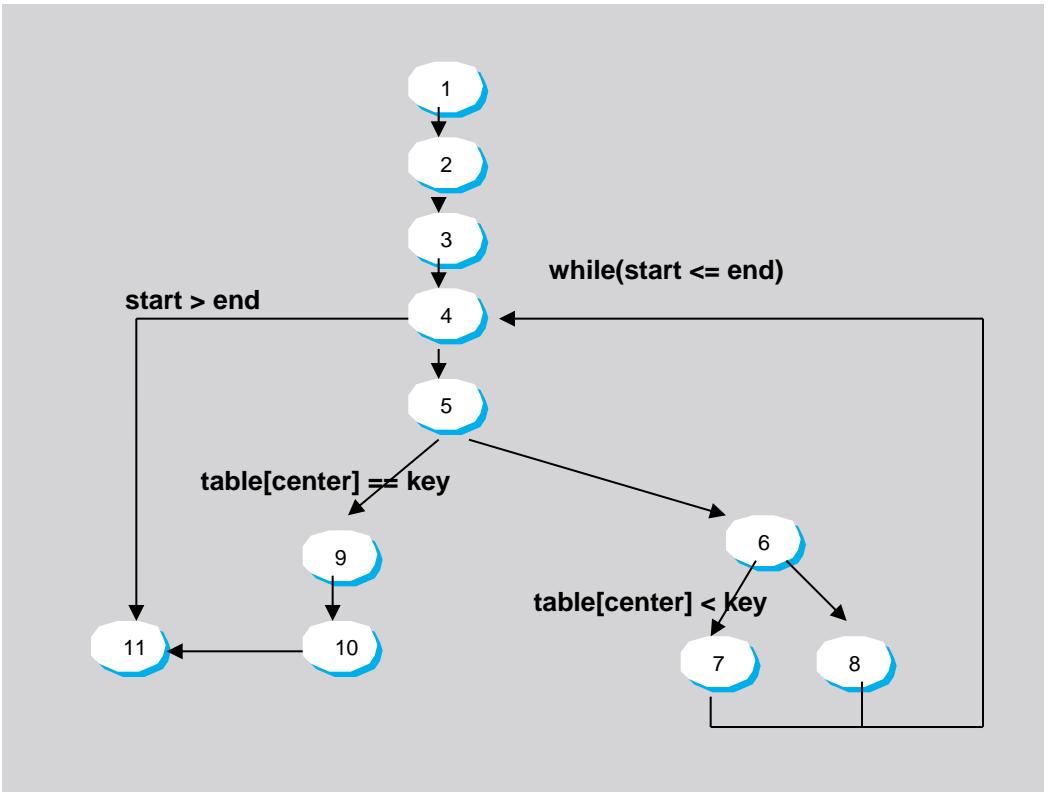


T	K	Result (found, position)
17	17	True, 1
17	1	False, ??
17, 21, 23, 29	17	True, 1
9, 16, 18, 30, 31, 41, 45	45	True, 7
17, 18, 21, 23, 29, 33, 38	23	True, 4
17, 18, 21, 23, 29, 33, 41	21	True, 3
17, 18, 21, 23, 32	23	True, 4
21, 23, 29, 33, 38, 55, 64, 65	22	False, ??

Testowanie ścieżek

- Odmiana testowania strukturalnego, której celem jest zbadanie każdej niezależnej ścieżki wykonania programu
- Punktem startu jest opracowanie grafu strumieni (grafu przepływu) programu
 - Węzły reprezentują decyzje (instrukcje decyzyjne)
 - Krawędzie reprezentują przepływ sterowania
- Metoda wykorzystywana przede wszystkim w ramach testów jednostkowych

Wyszukiwanie binarne – graf strumieni



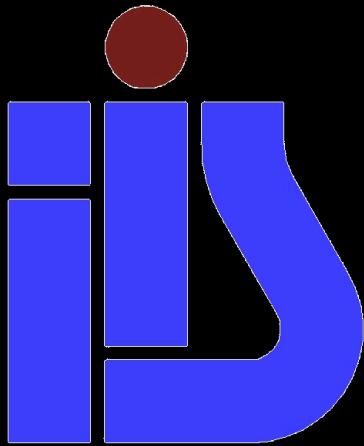
Niezależne ścieżki

- 1,2,3,4,5,9,10,11
- 1,2,3,4,11
- 1,2,3,4,5,6,7,4...
- 1,2,3,4,5,6,8,4...
- Przypadki testowe powinny uruchomić wszystkie ścieżki

Podsumowanie



- Testowanie może tylko wykazać istnienie błędów – nie może udowodnić, że nie błędów nie ma Edsger Dijkstra
- Za przeprowadzenie testów w trakcie wytwarzania odpowiedzialny jest zespół rozwijający system. Za wykonanie testów wersji systemu przygotowanej do wydania odpowiedzialny powinien być odrębny zespół.
- Na testy w trakcie wytwarzania składają się testy jednostkowe, w ramach których testowane są pojedyncze obiekty i metody, testy komponentów, w których testowane są powiązane grupy obiektów oraz testy systemowe, które testują częściowo zintegrowany lub kompletny system.



Część 2

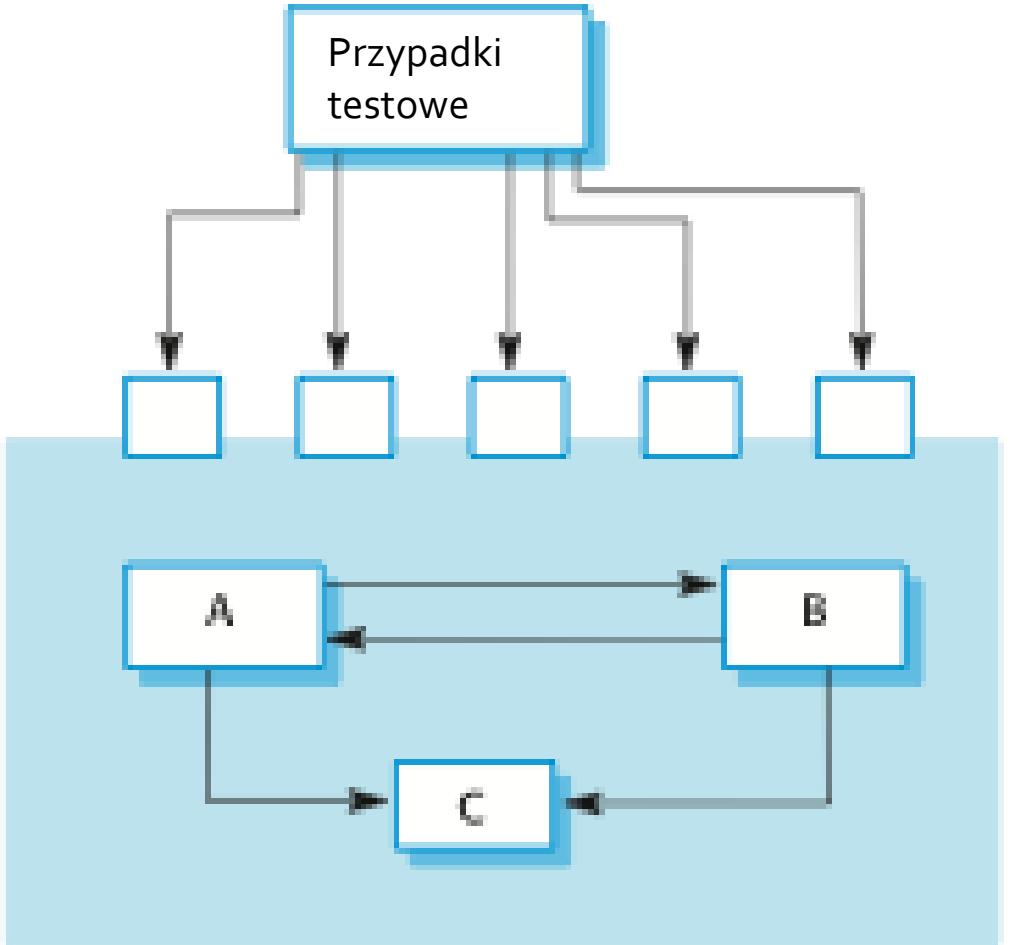
Testowanie oprogramowania

Testy komponentów



- Komponenty oprogramowania to najczęściej agregaty złożone z kilku współpracujących obiektów.
- Dostęp do funkcjonalności komponentu odbywa się za pośrednictwem zdefiniowanego interfejsu.
- Z tego powodu, testowanie komponentu skupia się przede wszystkim na zgodności zachowania interfejsu z jego specyfikacją.
 - Zakładamy, że testy jednostkowe obiektów stanowiących komponent zostały ukończone.

Testy komponentu – testy interfejsu



Testowanie interfejsów



- Wykorzystywane po zintegrowaniu modułów lub podsystemów
- Celem jest wykrycie usterek (testy defektów), które pojawiły się z powodu błędów w interfejsach (np. błędne założenia)
 - Niewłaściwe użycie interfejsu
 - Nierozumienie interfejsu
 - Błędy synchronizacji

Typy interfejsów



- Interfejsy parametryczne
 - Przekazują dane z jednego komponentu do drugiego
- Interfejsy w pamięci dzielonej
 - Dane w pamięci współdzielonej
- Interfejsy proceduralne
 - Jeden podsystem obudowuje procedury udostępniane innym podsystemom
- Interfejsy z przekazywaniem komunikatów
 - Jeden podsystem żąda usługi innego przez przesłanie komunikatu. Komunikat zwrotny zawiera wynik wykonania usługi

Testowanie interfejsów - porady

- Jawnie wypisz wszystkie wywołania zewnętrznych komponentów. Opracuj zbiór testów, w których wartości parametrów leżą na granicach zakresów
- W przypadku przekazywania wskaźników (referencji) przetestuj z zerowymi wartościami (null)
- Gdy komponent wywoływany jest przez interfejs proceduralny opracuj testy powodujące awarię
 - Najczęstsze nieporozumienia
- Dla interfejsów z przekazywaniem komunikatów wykonaj testy obciążeniowe.
- Jeżeli komponenty komunikują się za pomocą pamięci współdzielonej opracuj testy różniące się porządkiem inicjalizacji komponentów

Testy systemowe a testy komponentów



- Testy systemowe związane są z integracją komponentów w celu utworzenia wersji systemu, która następnie zostaje poddana testowaniu.
 1. Integracja komponentów wielokrotnego użycia z nowymi komponentami.
 2. Integracja komponentów rozwijanych niezależnie przez programistów (i/lub odrębne zespoły).
 3. Proces kolektywny a nie indywidualny.

Rozważania na temat testów systemowych



- Nacisk na interakcję pomiędzy komponentami.
- Sprawdzają kompatybilność komponentów, poprawność interakcji, przesyłanie, pomiędzy interfejsami, poprawnych danych w odpowiednim momencie.
- Testowanie zachowania systemu związanego ze zjawiskiem **emergencji** (ang. emergence, emergent properties/behavior).

Testowanie przypadków użycia Use-case testing

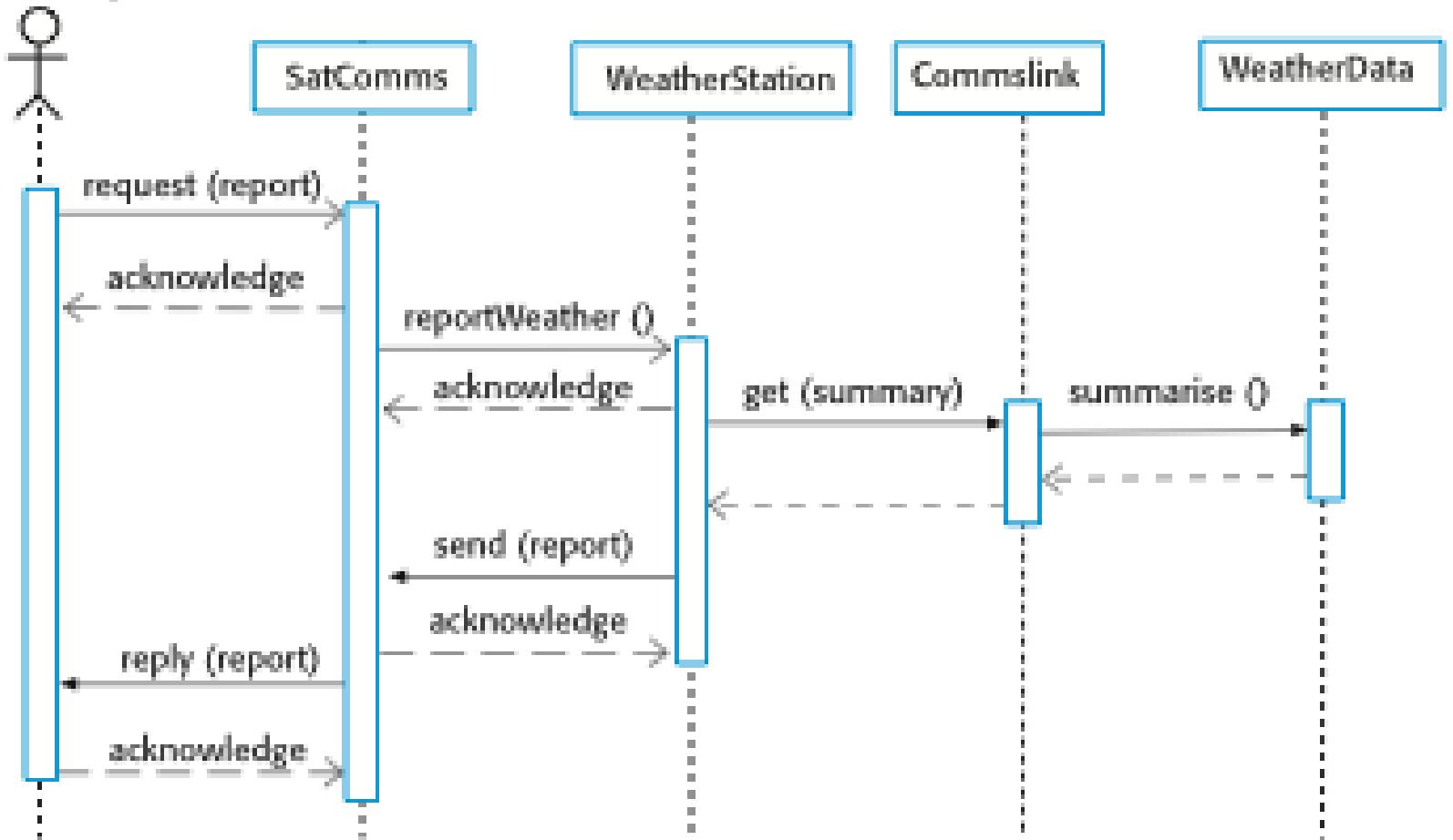


- Jako podstawę testów systemowych można wykorzystać przypadki użycia opracowane w celu identyfikacji sposobu interakcji systemu z otoczeniem.
- Każdy przypadek użycia angażuje zazwyczaj kilka komponentów systemu przez co jego wykonanie wymusza zaistnienie interakcji.

Diagram sekwencji dla wywołania usługi „stacji pogodowej”



Weather
information system



Polityka testowania



- Tylko wyczerpujące testy mogą wykazać, że program nie ma poważniejszych usterek.
 - Ale takie testy są w praktyce niemożliwe 😊
- Polityka testowania definiuje podejście wykorzystywane do wyboru testów systemowych:
 - Wszystkie funkcje dostępne z interfejsu powinny być przetestowane
 - Kombinacje funkcji dostępnych z tego samego menu powinny być przetestowane
 - Tam gdzie wymagane jest wprowadzanie danych funkcje muszą być testowane z poprawnymi i niepoprawnymi danymi

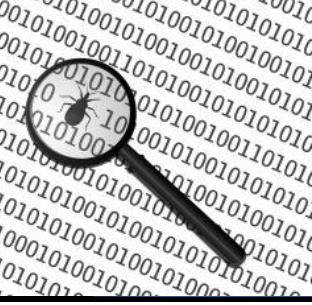
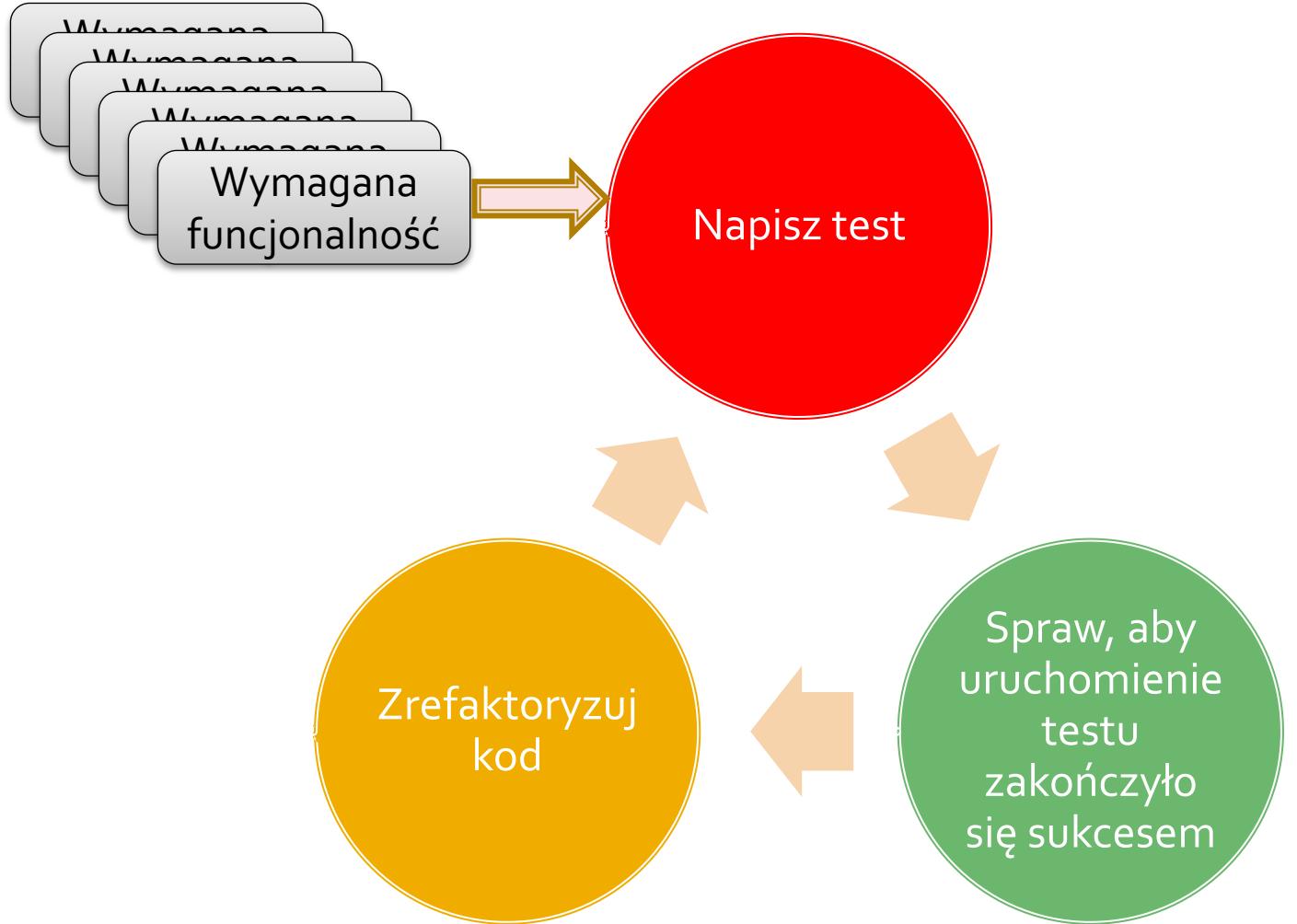
Wytwarzanie sterowane testami

Test-driven development (TDD)



- Metoda przyrostowego rozwoju oprogramowania w które testy stanowią podstawę wytwarzania.
- Testy pisane są przed kodem a poprawne wykonanie testu jest sterownikiem rozwoju.
 - Dla danego przyrostu, w pierwszej kolejności, pisany jest test . Kolejny przyrost nie może się rozpocząć dopóki kod nie „przejdzie” testu.
- Wprowadzona w ramach metodyki eXtreme Programming, wyewoluowała do niezależnej metodyki.

Cykl TDD



Zalety TDD

- 
1. Pokrycie kodu testem
 2. Testy regresyjne
 3. Uproszczenie debugowania
 4. Dokumentacja systemu (nazwy testów są ważne!)
 5. Koncentracja na API

Testy regresyjne Regression testing



- Kontrola czy zmiany nie spowodowały problemów w dotychczas istniejącym kodzie.
 - testy niezmienionych części oprogramowania po wykonaniu zmiany
- Bardzo kosztowne przy ręcznym wykonywaniu.
 - Ale automatyzacja jest prosta – wszystkie dotychczasowe testy muszą być wykonane każdorazowo po wprowadzeniu zmiany.
- Zatwierdzenie zmiany wymaga poprawnego wykonania testów.

Testy wydania

Release testing



1. Test wersji systemu, która ma być używana poza zespołem rozwijającym system
2. Celem jest zwiększenie przekonania klientów, że produkt spełnia wymagania
3. Zazwyczaj są to testy typu czarnej skrzynki (ang. black-box) nazywane również testami funkcjonalnymi
 - Bazują tylko na specyfikacji systemu
 - Zespół testujący nie ma wiedzy o implementacji systemu



Testy wydania a testy systemowe

- Testy wydania są formą testów systemowych.
- Istotne różnice:
 - W testy wydania powinien być zaangażowany zespół, który nie brał udziału w wytwarzaniu.
 - Testy systemowe skupiają się na wykrywaniu błędów (testy usterek). Celem testów wydania jest sprawdzenie czy system jest zgodny ze specyfikacją i nadaje się do użytku produkcyjnego (testy zatwierdzające).

Testowanie oparte na wymaganiach

Requirements based testing



- Testy dotyczące weryfikacji każdego wymagania poprzez opracowanie i uruchomienie dedykowanych testu.
- Przykładowe wymagania systemu obsługi pacjenta:
 - Jeżeli wiadomo, że pacjent jest uczulony na dany lek, przepisanie go powinno wygenerować ostrzeżenie dla użytkownika systemu.
 - Jeżeli przepisujący wybrał opcję ignorowania ostrzeżenia, powinien podać przyczynę takiego zachowania.

Testowanie wymagań



- Zainicjalizuj rekord pacjenta bez informacji o alergiach. Przepisz lekarstwo, dla którego wiadomo, że istnieją przypadki uczuleń. Sprawdź, czy informacja z ostrzeżeniem nie została wygenerowana przez system.
- Zainicjalizuj rekord pacjenta z informacją o alergii na lek. Przepisz lekarstwo, na które pacjent jest uczulony. Sprawdź, czy informacja z ostrzeżeniem została wygenerowana.
- Zainicjalizuj rekord pacjenta z informacją o uczuleniach na dwa lub więcej leków.
 - Przepisz odrębnie dwa leki uczulające pacjenta. Sprawdź czy poprawne ostrzeżenie dla każdego leku zostało wygenerowane.
 - Przepisz jednocześnie dwa leki uczulające pacjenta. Sprawdź czy dwa ostrzeżenia zostały poprawnie zgłoszone.
- Zainicjalizuj rekord pacjenta z informacją o alergii na lek. Przepisz lekarstwo generujące ostrzeżenie. Zignoruj ostrzeżenie. Sprawdź, czy system wymaga od użytkownika podania informacji wyjaśniających powód zignorowania.

Testy wydajnościowe Performance

testing



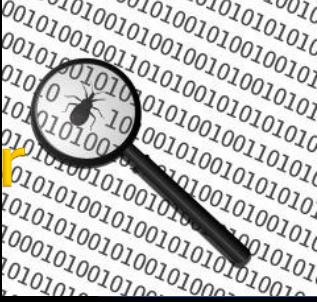
- Stanowią zazwyczaj część testów wydania i uwzględniają własności systemu takie jak wydajność czy niezawodność.
- Powinny odzwierciedlać profile wykorzystania systemu
- Wymagają serii testów o wzrastającym obciążeniu, aż do poziomu, który powoduje nieakceptowalny spadek wydajności.

Testy obciążenia Stress testing



- Testowanie powyżej maksymalnego obciążenia aż do pojawienia się awarii
 - Testowanie zachowania systemu w sytuacji błędu.
 - Presja wywierana na system może ujawnić usterki, które w trakcie normalnego pozostają ukryte
- Szczególnie użyteczne dla systemów rozproszonych, które mogą być narażone np. na przeciążenia sieci komputerowej.

Testy użytkowników User/customer testing



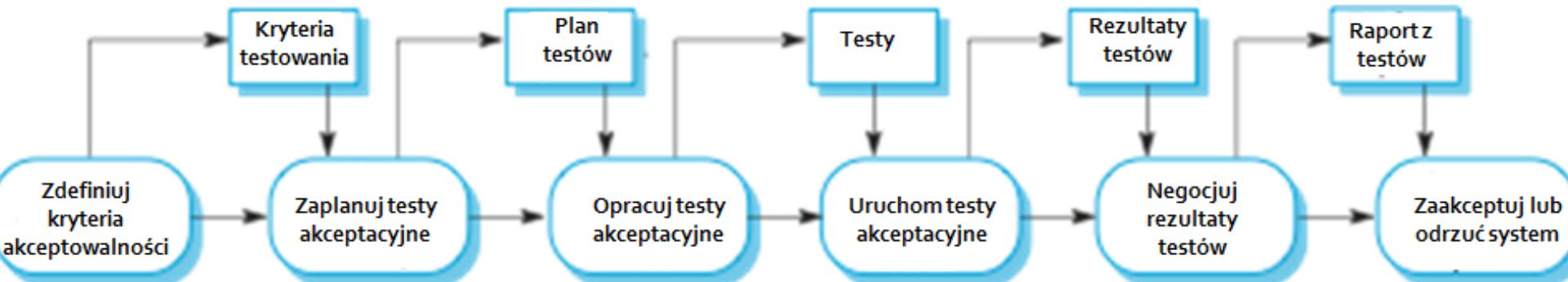
- Etap w procesie testowania, w którym to użytkownik (klient) dostarcza dane wejściowe oraz wskazówki co do sposobu wykonania testów.
- Istotny etap, nawet jeżeli przeprowadzone zostały kompleksowe testy systemowe oraz testy wydania.
 - Powodem tego jest to, że wpływy środowiska pracy użytkownika, ma istotny wpływ na niezawodność, wydajność i użyteczność systemu. Nie jest możliwe odtworzenie warunków tego środowiska w środowisku testowym.

Rodzaje testów użytkowników



- **Testy alpha**
 - Użytkownicy oprogramowania współpracują z zespołem rozwijającym oprogramowania w celu przetestowania oprogramowania po stronie wytwórcy.
- **Testy beta**
 - Wydanie systemu jest udostępniane użytkownikom. Mogą oni samodzielnie eksperymentować z systemem, informując producenta o wykrytych problemach.
- **Testy akceptacyjne**
 - Klient testuje system w celu podjęcia decyzji czy jest on gotowy do wdrożenia w środowisku produkcyjnym. Ty testów przeznaczony przede wszystkim dla oprogramowania dedykowanego.

Proces testowania akceptacyjnego



Metody zwinne a testy akceptacyjne



- Użytkownik/klient jest częścią zespołu rozwijającego oprogramowanie
 - Jest odpowiedzialny za podejmowanie decyzji o akceptowalności systemu.
- Testy definiowane przez klienta/użytkownika są integrowane z pozostałymi testami, które wykonuje się automatycznie po wprowadzeniu zmian.
- Nie ma oddzielnego procesu testowania akceptacyjnego.
- Podstawowym problemem jest „typowość” użytkownika
 - Na ile reprezentuje on interesy wszystkich zainteresowanych.

Rejestr wymagań a testy wymagań

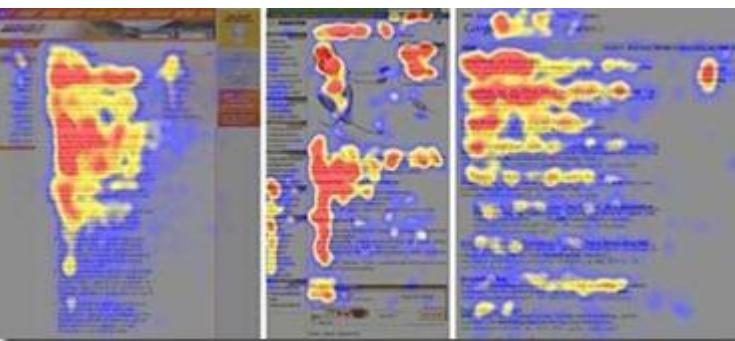


ID	Nazwa	Waga	Oszac.	Demo	Uwagi
1	Wpłata	30	5	Zaloguj się, otwórz stronę stan konta, sprawdź stan, otwórz stronę wpłat, wpłacić 100 PLN, przejdź do strony stan konta i sprawdź, że zwiększył się o 100 PLN.	Potrzebny diagram sekwencji (UML). Szyfrowanie jest na razie nieistotne.
2	Przeglądanie historii własnych transakcji	10	8	Zaloguj się, wybierz opcje „transakcje”. Dokonaj wpłaty. powróć do transakcji i sprawdź, że nowa transakcja wpłaty się pojawiła.	Wykorzystaj stronicowane w celu uniknięcia dużych zapytań, projekt podobny do strony przeglądania użytkowników.

Testowanie systemu w środowisku produkcyjnym



- A/B testing (Online Controlled Experimentation)
 - Obserwacja zachowania użytkowników



- Canary deployment
 - Przekierowanie podzbioru użytkowników do nowej funkcjonalności

Formalna weryfikacja oprogramowania



- Formalny proces dowodzenia zgodności programu z wymaganiami
 - Formułowanie wymagań w języku formalnym (logika, rachunek zdań)
 - Zamiana programu na model (uproszczony)
 - Konfrontacja modelu z wymaganiami
- Przykładowe Techniki weryfikacji
 - Model checking, logika Hoare'a, inferencja reguł

Podsumowanie



- W trakcie testowania oprogramowania powinno się próbować „złamać” oprogramowanie, wykorzystując własne doświadczenie, wskazówki (doświadczenie innych) do wyboru odpowiedniego zestawu efektywnych przypadków testowych.
- Kiedy to jest możliwe, testy powinny być automatyzowane. Daje to możliwość ich wielokrotnego uruchamiania (przy każdej zmianie).
- Wytwarzanie sterowane testami jest metodą wytwarzania oprogramowania, w ramach której test powstaje przed kodem, który ma być, za jego pomocą, testowany.
- Testowanie scenariuszy związane jest z definiowaniem typowych scenariuszy użycia systemu i wywodzeniem z nich przypadków testowych.
- Testy akceptacyjne to proces testowania przez użytkowników, którego celem jest pozyskanie informacji pozwalających na podjęcie decyzji czy produkt jest gotowy do wdrożenia i eksploatacji.