INŻYNIERIA OPROGRAMOWANIA

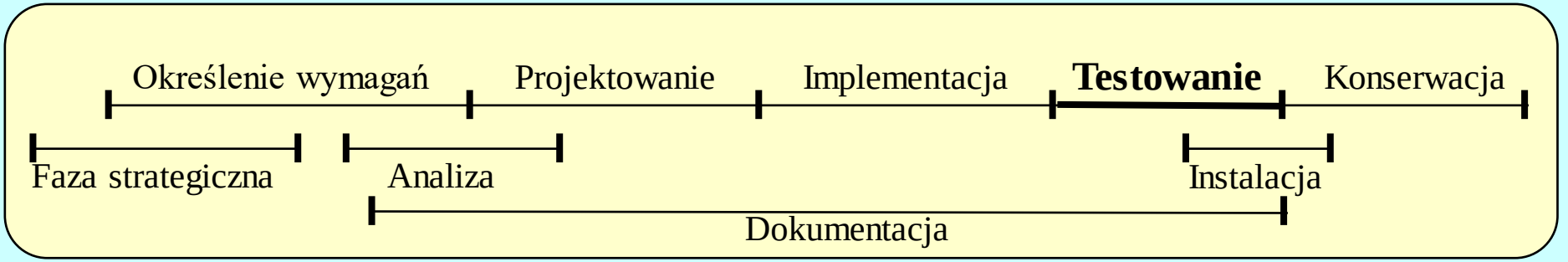
wykład 8:
TESTOWANIE

WERYFIKACJA
I WALIDACJA OPROGRAMOWANIA

dr inż. Leszek Grocholski

Zakład Inżynierii Oprogramowania
Instytut Informatyki
Uniwersytet Wrocławski

Etap - TESTOWANIE



Rozróżnia się następujące terminy:

WERYFIKACJA (*verification*) - sprawdzenie zgodności systemu i artefaktów systemu (dokumentacja, kod, itd….,) są zgodne z wymaganiami zdefiniowanymi w fazie określenia wymagań.

WALIDACJA (*validation*) inaczej ATESTOWANIE - ocena czy system spełnia oczekiwania klienta.

Weryfikacje i walidacje powinno się przeprowadzać we wszystkich fazach wytwarzania oprogramowania.

Dwa główne cele testowania (w wąskim zakresie tego terminu) **:**

- wykrycie i usunięcie błędów w systemie
- ocena niezawodności systemu

Weryfikacja oznacza...

- **Sprawdzanie czy artefakty są zgodne z wymaganiami**

Przeglądy, inspekcje, testowanie, sprawdzanie, audytowanie lub inna działalność ustalającą i dokumentującą czy system, komponenty, procesy, usługi lub dokumenty zgadzają się z wyspecyfikowanymi wymaganiami.

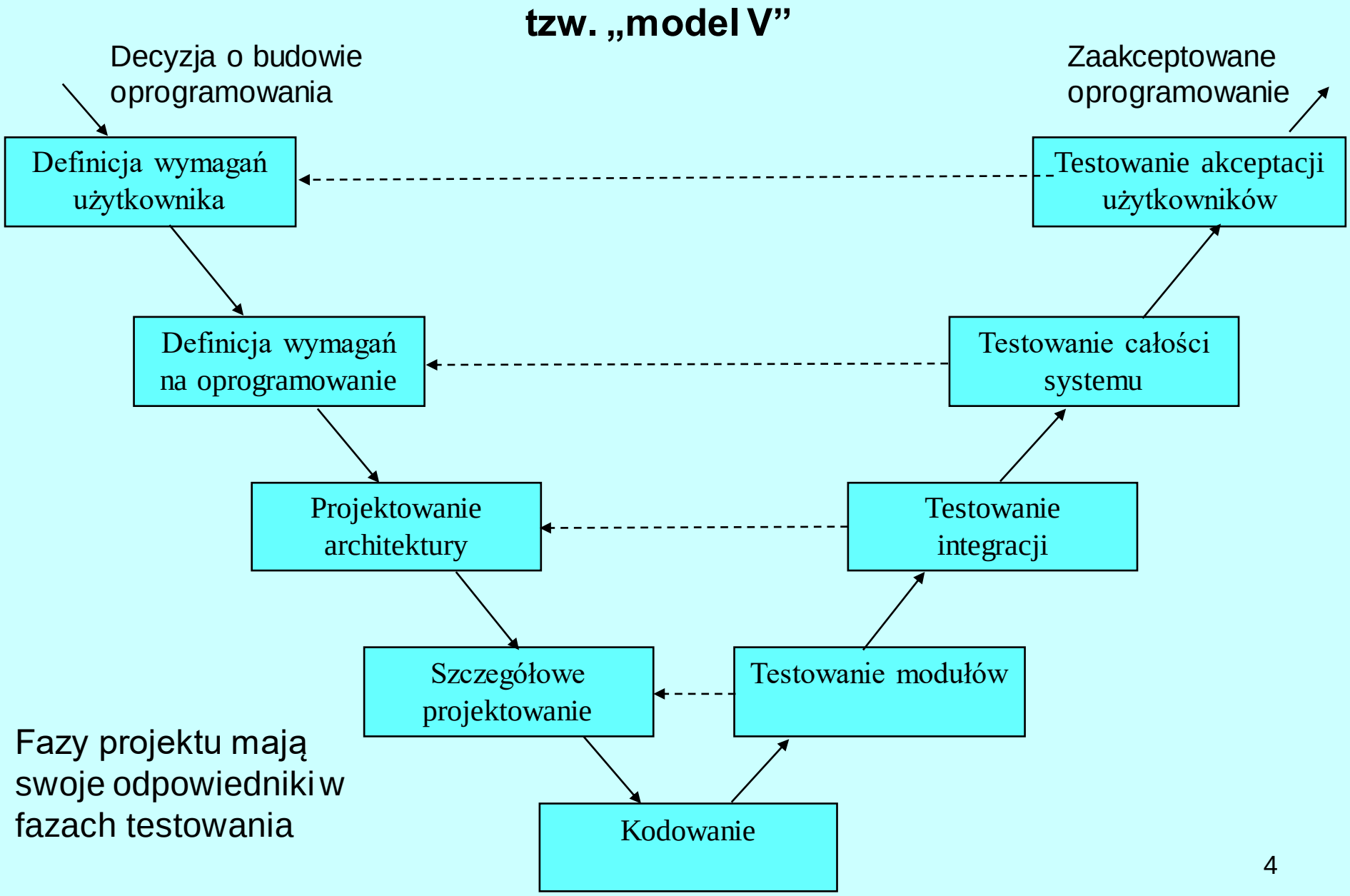
- **Sprawdzenie czy artefakty spełniają warunki początkowe fazy**

Oceny systemu lub komponentu mające na celu określenie czy produkt w danej fazie rozwoju oprogramowania spełnia warunki zakładane podczas startu tej fazy.

Weryfikacja praktycznie włącza następujące czynności:

- Przeglądy techniczne oraz inspekcje oprogramowania.
- Sprawdzanie czy wymagania na oprogramowanie są zgodne z wymaganiami użytkownika.
- Sprawdzanie czy komponenty oprogramowania są zgodne z wymaganiami na oprogramowanie.
- Testowanie jednostkowe jednostek oprogramowania (komponentów).
- Testowanie integracji oprogramowania, testowanie systemu.
- Testowanie akceptacji systemu przez użytkowników
- Audyt.

Związki procesu wytwarz. z fazami testowania



Fazy projektu mają swoje odpowiedniki w fazach testowania

Przeglądy oprogramowania (reviews)

Przegląd jest procesem lub spotkaniem, podczas którego produkt roboczy lub pewien zbiór produktów roboczych jest prezentowany dla personelu projektu, kierownictwa, użytkowników, klientów lub innych zainteresowanych stron celem uzyskania komentarzy, opinii i akceptacji.

Przeglądy mogą być formalne i nieformalne.
Formalne przeglądy mogą mieć następującą postać:

Przegląd techniczny. Oceniają elementy oprogramowania na zgodność postępu prac z przyjętym planem. (Szczegóły można znaleźć w **ANSI/IEEE Std 1028-1988 „IEEE Standard for Reviews and Audits").**

Przejście (*walkthrough*). Wczesna ocena dokumentów, modeli, projektów i kodu. Celem jest zidentyfikowanie defektów i rozważenie możliwych rozwiązań. Wtórnym celem jest szkolenie i rozwiązanie problemów stylistycznych (np. z formą kodu, dokumentacji, interfejsów użytkownika).

Audyt. Przeglądy potwierdzające zgodność oprogramowania, artefaktów, procesu wytwarzania z wymaganiami, specyfikacjami, zaleceniami, standardami, procedurami, instrukcjami, kontraktami i licencjami. Obiektywność audytu wymaga, aby był on przeprowadzony przez osoby niezależne od zespołu projektowego.

Skład zespołu oceniającego

Dla poważnych projektów ocena oprogramowania nie może być wykonana w sposób amatorski. Musi być powołany zespół, którego zadaniem będzie zarówno przygotowanie testów jak i ich przeprowadzenie.

Skład zespołu oceniającego:

- Kierownik
- Sekretarz
- Członkowie, w tym:
    - użytkownicy
    - kierownik projektu oprogramowania
    - inżynierowie oprogramowania
    - bibliotekarz oprogramowania
    - personel zespołu zapewnienia jakości
    - niezależny personel weryfikacji i atestowania
    - niezależni eksperci nie związani z rozwojem projektu

Zadania kierownika: nominacje na członków zespołu, organizacja przebiegu oceny i spotkań zespołu, rozpowszechnienie dokumentów oceny pomiędzy członków zespołu, organizacja pracy, przewodniczenie posiedzeniom, wydanie końcowego raportu, i być może inne zadania.

Co to jest audyt oprogramowania? (audit)

AUDYTEM nazywany jest niezależny przegląd i ocena JAKOŚCI OPROGRAMOWANIA, która zapewnia zgodność z wymaganiami na oprogramowanie, a także ze specyfikacją, generalnymi założeniami, standardami, procedurami, instrukcjami, kodami oraz kontraktowymi i licencyjnymi wymaganiami.

Aby zapewnić obiektywność, audyt powinien być przeprowadzony przez osoby niezależne od zespołu projektowego.

Audyt powinien być przeprowadzany przez odpowiednią organizację audytu (w Polsce, Polskie Stowarzyszenie Audytu Wewnetrznego) lub przez osoby posiadające uprawnienia/licencję audytorów.

Reguły i zasady audytu są określone w normie:
ANSI/IEEE Std 1028-1988 „IEEE Standard for Reviews and Audits".

Aspekty audytu

Przykłady

- Analiza stanu projektu
- Analiza celowości
- Analiza procesu wytwórczego
- Analiza dowolnego procesu np. zarządzania ryzykiem
- Analiza systemu jakości
- Analiza stosowania systemu jakości

Relacje odbiorca dostawca

- audyt wewnętrzny
- audyt zewnętrzny – firmy certyfikujące (ISO, CMM)
- audyt „trzeciej strony” – firmy doradcze np. Ernst & Young

Etapy

- planowanie i przygotowanie
- wykonywanie
- raportowanie
- zamknięcie

Audyt projektu IT - cel

- Celem audytu projektu informatycznego jest dostarczenie odbiorcy i dostawcy obiektywnych, aktualnych i syntetycznych informacji o stanie całego projektu

- Jest to osiągane przez zbieranie dowodów, że projekt:
  - posiada możliwości (zasoby, kompetencje, metody, standardy) by osiągnąć sukces,
  - optymalnie wykorzystuje te możliwości,
  - rzeczywiście osiąga założone cele (cząstkowe).

- Zebrane informacje służą jako podstawa do podejmowania strategicznych decyzji w/o projekcie.

Przedmioty i perspektywy audytu

Przedmioty

- procesy przedsięwzięcia informatycznego - ma to na celu sprawdzenie czy wykonywane prace oraz sposób ich wykonywania są prawidłowe
- produkty (cząstkowe) projektu informatycznego - ma to na celu sprawdzenie czy rezultaty poszczególnych prac odpowiadają zakładanym wymaganiom

Perspektywy

- zawansowanie przedsięwzięcia
- technologia - ma to na celu sprawdzenie czy użyte techniki oraz opracowane rozwiązania są prawidłowe i prawidłowo stosowane
- zarządzanie - ma to na celu sprawdzenie czy sposób zarządzania projektem umożliwia jego sukces

Inspekcje

Inspekcja to formalna technika oceny, w której wymagania na oprogramowanie, projekt lub kod są szczegółowo badane przez osobę lub grupę osób nie będących autorami, **w celu identyfikacji błędów, naruszenia standardów i innych problemów** [IEEE Std. 729-1983]

- Technika o najlepszej skuteczności (od 50% do 80%; średnia 60%; dla testowania średnia 30%, max 50%)

- Stosowane dla „elitarnych" systemów

- Dlaczego nie są powszechne?
  - trudne w sprzedaży: nie potrzeba narzędzi, potrzeba planowania i kompetentnych ludzi,
  - analiza kosztów i zysków nie jest prosta
  - psychologia: - ludzie nie lubią być sprawdzani,
    - dobry programista woli programować

Cechy inspekcji

- Sesje są zaplanowane i przygotowane
- Błędy i problemy są notowane
- Wykonywana przez techników dla techników (bez udziału kierownictwa)
- Dane nie są wykorzystywane do oceny pracowników
- Zasoby na inspekcje są gwarantowane
- Błędy są wykorzystywane w poprawie procesu programowania (prewencja)
- Proces inspekcji jest mierzony.

  mierzone są np. błędy – w celu poprawy

Proces wytwórczy inspekcji jest poprawiany - lessons learned

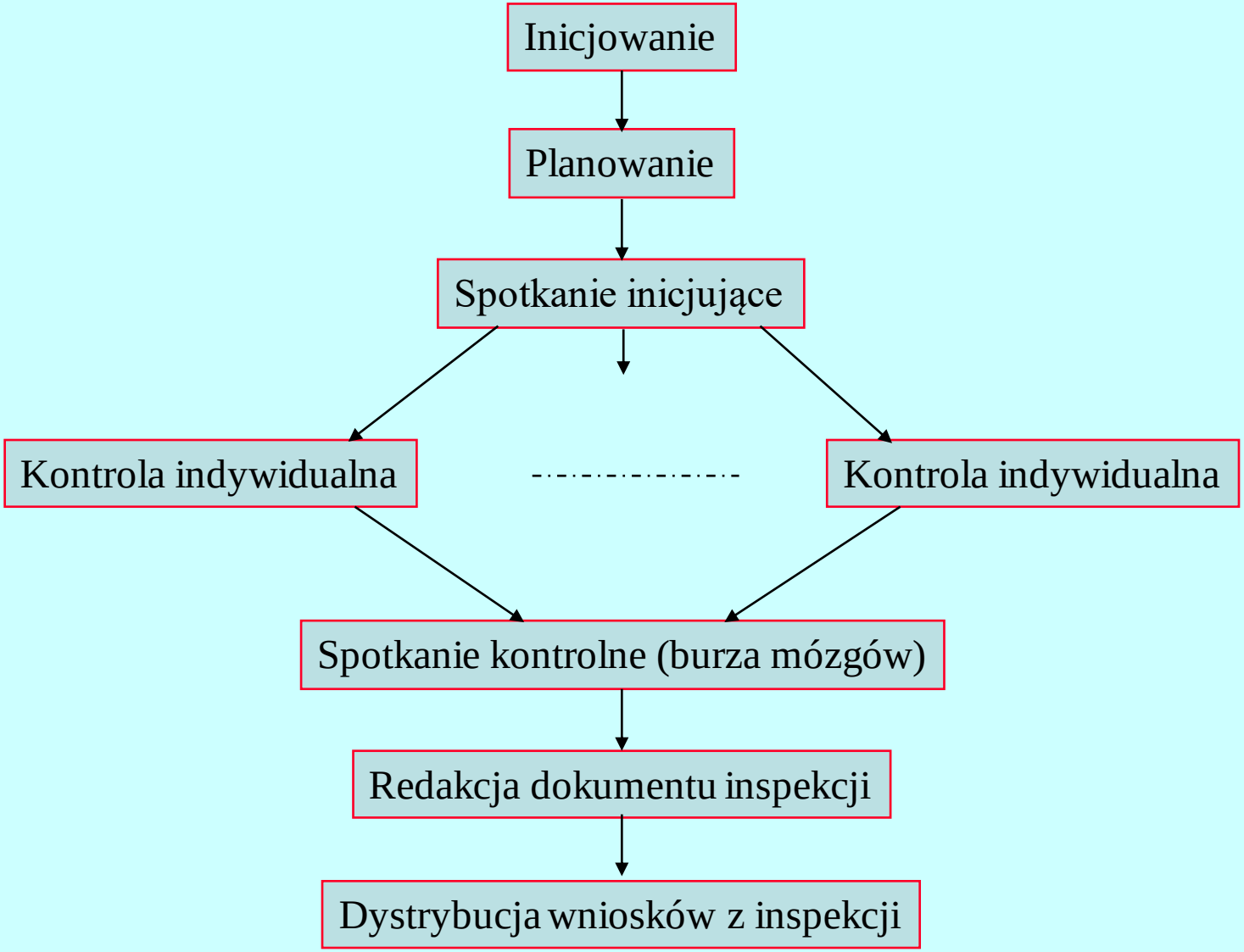
Korzyści z inspekcji

- Wzrost produktywności od 30% do 100%
- Skrócenie czasu projektu od 10% do 30%
- Skrócenie kosztu i czasu wykonywania testów od 5 do 10 razy (mniej błędów, mniej testów regresyjnych)

! 10 razy mniejsze koszty konserwacji (naprawczej) !

- Poprawa procesu i stylu programowania
- Dostarczanie na czas (bo wcześnie wiemy o problemach)
- Większy komfort pracy (brak presji czasu i nadgodzin)
- Zwiększenie motywacji
  - świadomość, że produkt będzie oceniany (wybór pomiędzy byciem zażenowanym a dumnym)
  - nauka przez znajdowanie błędów
- Mniejsze koszty marketingu
  - informacja o dobrej jakości rozpowszechnia się.

Przebieg inspekcji



Kroki inspekcji (1)

1. **Inicjowanie**
   Zgłoszenie potrzeby inspekcji; wyłonienie lidera inspekcji.
2. **Planowanie**
   Lider ustala uczestników, listy kontrolne, zbiory reguł, tempo kontroli, daty spotkań kontrolnych.
3. **Spotkanie inicjujące**
   Ustalenie ról, ustalenie celów i oczekiwań, dystrybucja dokumentu, szkolenie w inspekcjach.
4. **Kontrola indywidualna**
   Uczestnicy sprawdzają dokument względem zadanych kryteriów, reguł i list kontrolnych (znaleźć jak najwięcej unikalnych błędów)
5. **Spotkanie kontrolne (burza mózgów)**
   Notowanie uwag z kontroli indywidualnej.
   Każda uwaga jest kwalifikowana jako „zagadnienie” (potencjalny błąd), „pytanie o intencję”, „propozycja poprawy procesu”; Szukanie nowych zagadnień;.Poprawa procesu wytarzania oprogramowania, poprawa procesu inspekcji.

Kroki inspekcji (2)

6. **Poprawa produktu:** edytor (najczęściej autor) rozwiązuje zagadnienia; prawdziwy problem może być inny niż jest to zgłoszone; zagadania są kwalifikowane jako błędy lub dokument jest redagowany by uniknąć błędnych interpretacji
7. **Kontynuacja**: lider sprawdza, że obsłużono wszystkie zagadnienia: są poprawione lub są w systemie zarządzania konfiguracją; sprawdza kompletność a nie poprawność
8. **Decyzja o gotowości:** lider podejmuje decyzję czy produkt jest gotowy do przekazania dalej (np. liczba błędów w określonym limicie)
9. **Rozpowszechnienie dokumentu**
10. **Burza mózgów** ma na celu identyfikację przyczyn błędów (max 10 najpoważniejszych) i propozycji poprawy procesu programowania, by błędy te nie powtórzyły się w przyszłości; idee są notowane bez ich głębokiej oceny

Zagrożenia inspekcji

- **Niewłaściwie przeprowadzona inspekcja nie przynosi efektów.**
- Złe prowadzenie inspekcji - mała efektywność i skuteczność.
- Słabi (niekompetentni, nieodporni psychicznie) kontrolerzy.
- Kontrola indywidualna niewystarczająca (jakość i ilość).
- Skłonność autora do lekceważenia defektów na etapie opracowywania dokumentów (“inspekcja wskaże błędy...”).
- Dyskusje o rozwiązaniach podczas spotkania kontrolnego.
- Poczucie zagrożenia u autora - nieuzasadniona obrona własnych rozwiązań.
- Krytyczne nastawienie do autora.

Błąd i błędne wykonanie

Błąd (*failure, error*) - niepoprawna konstrukcja znajdująca się w programie, która może doprowadzić do niewłaściwego działania.

Błędne wykonanie (*failure*) - niepoprawne działanie systemu w trakcie jego pracy.

Błąd może prowadzić do różnych błędnych wykonań.
To samo błędne wykonanie może być spowodowane różnymi błędami.

Proces testowania oprogramowania można określić jako poszukiwanie i usuwanie błędów na podstawie obserwacji błędnych wykonań oraz innych testów.

Rodzaje testów

Testy można klasyfikować z różnych punktów widzenia:

1. Weryfikacji spełnienia wymagań niefunkcjonalnych

2. Dotyczące błędnych wykonań:

- Wykrywanie błędnego wykonań a następnie znalezienie błędu i jego poprawiene,

- Testy statystyczne, których celem jest spowodowanie błędnych wykonań, poznanie przyczyn najczęstszych błędnych wykonań oraz ocena niezawodności systemu.

Z punktu widzenia techniki wykonywania testów można je podzielić na:

- Testy dynamiczne, które polegają na wykonywaniu (fragmentów) programu i porównywaniu uzyskanych wyników z wynikami poprawnymi.

- Testy statyczne, oparte na analizie kodu

Typowe fazy testowania systemu

Testy modułów: Są one wykonywane już w fazie implementacji bezpośrednio po zakończeniu realizacji poszczególnych modułów.
Testy integracji …

Testy systemu: W tej fazie integrowane są poszczególne moduły i testowane są poszczególne podsystemy oraz system jako całość.
Testy wykonywane w organizacji, która wytworzyła oprogramowania, ale przez niezależny od zespołu, który wyprodukował oprogramowanie to testy alfa.

W przypadku oprogramowania sprzedawanego rynkowo testy takie polegają na nieodpłatnym przekazaniu pewnej liczby kopii systemu grupie użytkowników. Testy takie nazywa się testami **beta**.

Testy akceptacji (*acceptance testing*): W przypadku oprogramowania realizowanego na zamówienie system przekazywany jest podczas wdrażania do przetestowania przyszłemu użytkownikowi to testy wdrożeniowe.
Testy typowe testy akceptacyjne to testy, mające upewnić klienta, że dostarczono to co zamówiono i system można użytkować.

Co jeszcze podlega testowaniu (1)?
→ wymagania niefunkcjonalne
→ TO ZALEŻY OD KONTEKSTU !

Bardzo ważne:
Cechy na podstawie, których klient będzie ocenial jak system dziala
Własności operacyjne systemu, np. użyteczność/ stopień skomplikowania instrukcji kierowanych do systemu, czytelność ekranów, operacje wymagające zbyt wielu kroków, jakość komunikatów systemu, jakość informacji o błędach, jakość pomocy, wymagania logistyczne, organizacyjne, .

Wydajność systemu i poszczególnych jego funkcji (czy jest satysfakcjonująca).

Interfejsy systemu na zgodność z wymaganiami określonymi przez użytkowników

Testy zużycia zasobów: zużycie czasu jednostki centralnej, zużycie pamięci operacyjnej, przestrzeni dyskowej, itd.

Zabezpieczenie systemu: odporność systemu na naruszenia prywatności, tajności, integralności, spójności i dostępności. Testy powinny np. obejmować:

- zabezpieczenie haseł użytkowników
- testy zamykania zasobów przed niepowołanym dostępem

Co jeszcze podlega testowaniu (2)?

Przenaszalność oprogramowania: czy oprogramowanie będzie działać w zróżnicowanym środowisku (np. różnych wersjach Windows 95, NT, Unix), przy różnych wersjach instalacyjnych, rozmiarach zasobów, kartach graficznych, rozdzielczości ekranów, oprogramowaniu wspomagającym (bibliotekach), ...

Niezawodność oprogramowania, zwykle mierzoną średnim czasem pomiędzy błędami (MTF).

Odtwarzalność oprogramowania (*maintainability*), mierzoną zwykle średnim czasem reperowania oprogramowania po jego awarii. Pomiar powinien uwzględniać średni czas od zgłoszenia awarii do ponownego sprawnego działania.

Bezpieczeństwo oprogramowania: stopień minimalizacji katastrofalnych skutków wynikających z niesprawnego działania. (Przykładem jest wyłączenie prądu podczas działania w banku i obserwacja, co się w takim przypadku stanie.)

Kompletność i jakość założonych funkcji systemu.

Nie przekraczanie ograniczeń, np. na zajmowaną pamięć, obciążenia procesora,...

Co jeszcze podlega testowaniu (3)?

Modyfikowalność oprogramowania, czyli zdolność jego do zmiany przy zmieniających się założeniach lub wymaganiach

Obciążalność oprogramowania, tj. jego zdolność do poprawnej pracy przy ekstremalnie dużych obciążeniach. Np. maksymalnej liczbie użytkowników, bardzo dużych rozmiarach plików, dużej liczbie danych w bazie danych, ogromnych (maksymalnych) zapisach, bardzo długich liniach danych źródłowych. W tych testach czas nie odgrywa roli, chodzi wyłącznie o to, czy system poradzi sobie z ekstremalnymi rozmiarami danych lub ich komponentów oraz z maksymalnymi obciążeniami na jego wejściu.

Skalowalność systemu, tj. spełnienie warunków (m.in. czasowych) przy znacznym wzroście obciążenia.

Akceptowalność systemu, tj. stopień usatysfakcjonowania użytkowników.

Jakość dokumentacji, pomocy, materiałów szkoleniowych, zmniejszenia bariery dla nowicjuszy.

Schemat testów statystycznych

Losowa konstrukcja danych wejściowych zgodnie z rozkładem prawdopodobieństwa tych danych.

Określenie wyników poprawnego działania systemu na tych danych

Uruchomienie systemu oraz porównanie wyników jego działania z poprawnymi wynikami.

Powyższe czynności powtarzane są cyklicznie.

Stosowanie testów statystycznych wymaga określenia rozkładu prawdopodobieństwa danych wejściowych możliwie bliskiemu rozkładowi, który pojawi się w rzeczywistości. Dokładne przewidzenie takiego rozkładu jest trudne, w związku z czym wnioski wyciągnięte na podstawie takich testów mogą być nietrafne.

Założeniem jest przetestowanie systemu w typowych sytuacjach. Istotne jest także przetestowanie systemu w sytuacjach skrajnych, nietypowych, ale dostatecznie ważnych.

Istotną zaletą testów statystycznych jest możliwość ich automatyzacji, a co za tym idzie, możliwości wykonania dużej ich liczby. Technika ta jest jednak mało efektywna.

Testowanie czarnej skrzynki

black-box testing

Tak określa się sprawdzanie funkcji oprogramowania bez zaglądania do środka programu. **Testujący traktuje sprawdzany moduł jak „czarną skrzynkę", której wnętrze jest niewidoczne.**

Testowanie n/z czarnej skrzynki powinno obejmować cały zakres danych wejściowych.

Testujący powinni podzielić **dane wejściowe w „klasy równoważności**", co do których istnieje duże przypuszczenie, że będą produkować te same błędy. Np. jeżeli testujemy wartość „Malinowski", to prawdopodobnie w tej samej klasie równoważności jest wartość „Kowalski". Celem jest uniknięcie efektu „eksplozji danych testowych".

„**Klasy równoważności**" mogą być również zależne **od wyników zwracanych przez testowane funkcje**. Np. jeżeli wejściem jest wiek pracownika i istnieje funkcja zwracająca wartości „młodociany", „normalny" „wiek emerytalny", wówczas implikuje to odpowiednie klasy równoważności dla danych wejściowych.

Wiele wejść dla danych (wiele parametrów funkcji) może wymagać zastosowania pewnych systematycznych metod określania ich kombinacji, np. tablic decyzyjnych lub grafów przyczyna-skutek.

Testowanie białej skrzynki

white-box testing

Tak określa się sprawdzanie wewnętrznej logiki oprogramowania. (Lepszym terminem byłoby „*testowanie szklanej skrzynki*”.)

Testowanie n/z białej skrzynki pozwala sprawdzić wewnętrzną logikę programów poprzez **odpowiedni dobór danych wejściowych, dzięki czemu można prześledzić wszystkie ścieżki przebiegu sterowania programu.**

Tradycyjnie programiści wstawiają kod diagnostyczny do programu aby śledzić wewnętrzne przetwarzanie. Debuggery pozwalają programistom obserwować wykonanie programu krok po kroku.

Często niezbędne staje się wcześniejsze przygotowanie danych testowych lub specjalnych programów usprawniających testowanie (np. programu wywołującego testowaną procedurę z różnymi parametrami).

Dane testowe powinny być dobrane w taki sposób, aby każda ścieżka w programie była co najmniej raz przetestowana.

Ograniczeniem testowania na zasadzie białej skrzynki jest **niemożliwość pokazania brakujących funkcji w programie**.

Wadę tę usuwa **testowanie metodą czarnej skrzynki**.

Podstawowy proces testowy

Podstawowy proces testowy składa się z następujących czynności:

1. Planowanie i nadzór nad testami
2. Analiza i projektowanie testów
3. Implementacja i wykonanie testów
4. Ocena kryteriów zakończenia i raportowanie
5. Czynności zamykające test

Chociaż wyżej wymienione czynności układają się w logiczny ciąg, to w praktyce mogą się zazębiać lub występować jednocześnie.

Zwykle konieczne jest ich dostosowanie do kontekstu systemu i projektu.

Psychologia testowania

Sposób myślenia stosowany podczas testowania i przeglądania różni się od stosowanego podczas rozwoju oprogramowania.

Programiści posiadający odpowiednie nastawienie są w stanie testować swój kod, ale zwykle odpowiedzialność za testowanie przekazuje się testerom żeby wzmocnić koncentrację wysiłków oraz uzyskać dodatkowe korzyści, takie jak niezależne spojrzenie wyszkolonych, zawodowych testerów.

INŻYNIERIA OPROGRAMOWANIA

Dziękuję za uwagę