

The image features the GlobalLogic logo in white text, centered against a dark, blurred background. The background shows an office setting with a person sitting in a chair, and a pattern of faint, light blue circles and 'X' marks overlaid on the image.

GlobalLogic®










GlobalLogic®

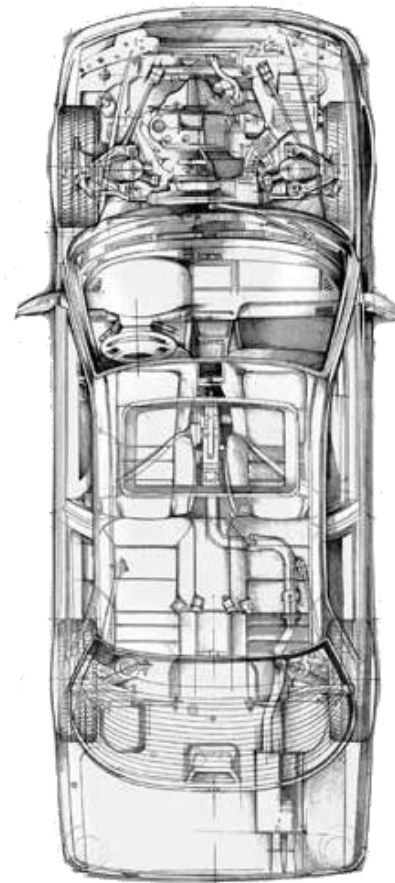
Testowanie Oprogramowania

Marcin Caryk



PLAN PREZENTACJI

-  1. Wstęp
-  2. Dlaczego testować?
-  3. Rodzaje testów
-  4. Planowanie i tworzenie testów
-  5. Continuous integration
-  6. Techniki tworzenia testów
-  7. Zadania





WSTEP



- QA
 - Testowanie oprogramowania
 - ISO 9001
 - ISO 9001 a testowanie
 - Automotive Spice
-



QA

- QA = Quality Assurance
- Jest to systematyczny proces sprawdzania produktu pod względem spełnienia określonych wymagań.
- Zapewnienie jakości oprogramowania (SQA) składa się ze środków monitorowania procesów inżynierii oprogramowania i metod stosowanych w celu zapewnienia jakości.





Testowanie oprogramowania

- Testowanie oprogramowania jest to proces związany z wytwarzaniem oprogramowania.
- Jest on jednym z procesów kontroli jakości oprogramowania (SQC), zapewniający jakość produktu
- Jest to rodzaj technologicznego badania
- Testy nie wpływają na jakość produktu
- Testowanie ma dwa główne cele: weryfikację oprogramowania i walidację oprogramowania





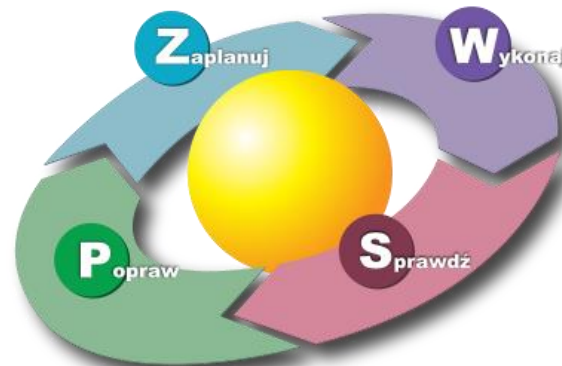
SQA vs SQC

Criteria	Software Quality Assurance (SQA)	Software Quality Control (SQC)
Definicja	SQA to zbiór aktywności dla zapewnienia jakości w procesie inżynierii oprogramowania	SQC to zbiór aktywności dla zapewnienia jakości oprogramowania
Ukierunkowanie	Proces	Produkt
Orientacja	Prewencja	Detekcja
Zasięg	Organizacja	Produkt lub projekt
Zakres	Wszystkie wytwarzane produktu	Konkretny produkt
Aktywności	Definicja oraz implementacja procesów Audyty Szkolenia	Review Testowanie



ISO 9001

- ISO 9001 jest to międzynarodowa norma określająca wymagania, które powinien spełniać system zarządzania jakością w organizacji, Systemu Zarządzania Jakością (QMS)
- Ukierunkowana jest ona na zrozumienie i spełnienie wymagań klienta,
- Standard ten zaleca objęcie procesów organizacji **cyklem Deminga**,
- Norma jest skonstruowana uniwersalnie. Nie jest normą techniczną, zawiera tylko wymagania dotyczące systemu zarządzania.





ISO 9001 a testowanie

ISO 9001 wskazuje, że organizacja powinna zaplanować realizację wyrobu i określić:

- Cele dotyczące jakości wyrobu
- Specyficzne działania badające zgodność wyrobu (weryfikacja, walidacja, monitorowanie, kryteria dotyczące przyjęcia wyrobu)
- Zapisy potrzebne do dostarczenia dowodów, że procesy realizacji spełniają wymagania





Automotive Spice

- SPICE (Software Process Improvement and Capability Determination), jest zbiorem standardów technicznych.
- Zapoczątkowany został z cyklu życiowego standardu ISO 12207 i z rozwiniętych modeli takich jak CMM.
- A-SPICE – Automotive Software Process Improvement & Capability dEtermination, Pochodzi od ISO/IEC 15504 (ISO/IEC 33001)
- Stworzony przez Specjalną Grupę Zainteresowania (SIG): AUDI, BMW, Daimler, FIAT, Ford, Jaguar, VW, Porsche, Volvo
- Określa model referencyjny model oceny procesów zarządzania projektem, tworzenia oprogramowania, integracji oraz testowania





DLACZEGO TESTOWAĆ?



- Co to jest błąd...
- Skąd biorą się błędy w kodzie
- Co to jest testowanie
- Dlaczego testowanie jest niezbędne
- Testowanie a jakość produktu
- Podstawowe zasady testowania



Co to jest błąd...



Pomyłka



Defekt



Awaria



Nieprawidłowe działanie testowanego produktu (TP)

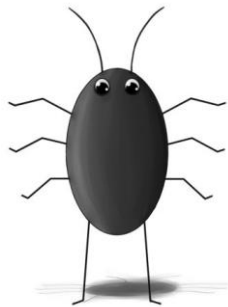


Co to jest błąd...

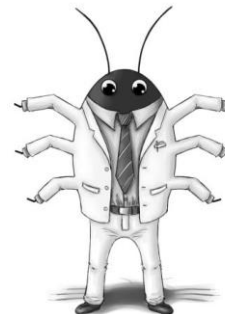
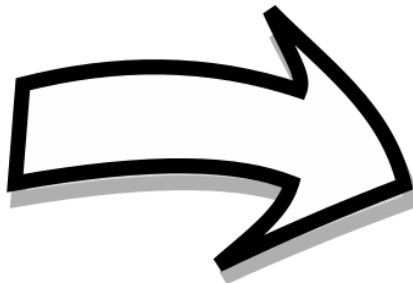
Pomyłka, błąd (mistake, error): działanie człowieka powodujące powstanie nieprawidłowego wyniku.

Defekt, usterka, pluskwa (defect, fault, bug): skutek błędu twórcy oprogramowania. Usterka może, ale nie musi spowodować awarii.

Awaria (failure): odchylenie od spodziewanego zachowania albo wyniku działania oprogramowania.



BUG



FEATURE



Skąd biorą się błędy w kodzie

- Zła komunikacja lub jej brak
- Złożoność oprogramowania
- Błędy w oprogramowaniu
- Zmiany w wymaganiach
- Ograniczenie projektowe
 - ograniczenia czasowe
 - ograniczenia zespołowe
 - ograniczenia narzędziowe



- Ograniczenia poznawcze
- Czynniki psychologiczne
 - Egoistyczne lub zbyt pewne podejście siebie osób
 - Zbyt ambitne podejście
 - Obniżona koncentracja
- Słabo udokumentowany kod
- Źle dobrane narzędzie deweloperskie

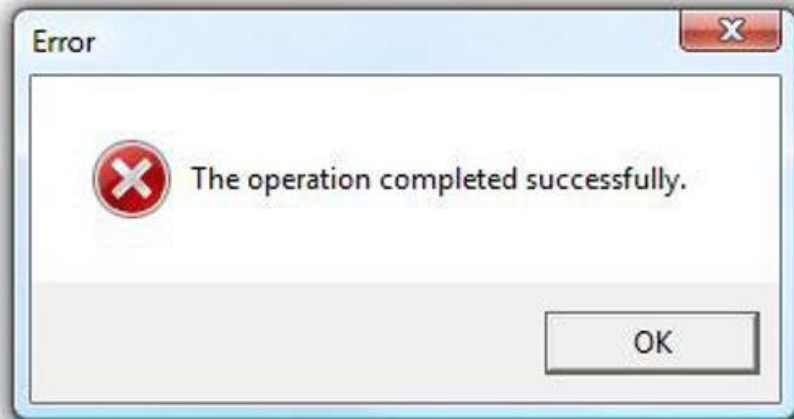


Skąd biorą się błędy w kodzie

Defekty (usterki) istnieją, ponieważ:

- ludzie są omylni
- wytwórcy działają
 - pod presją czasu,
 - ze skomplikowanym kodem,
 - w skomplikowanej infrastrukturze,
 - przy zmieniających się technologiach.
- jest wiele interakcji wewnątrz systemu i pomiędzy współpracującymi systemami

Awarie mogą być wywołane także przez czynniki środowiska co może również powodować niepoprawne działanie oprogramowania





Co to jest testowanie

- Rodzaj technologicznego badania pozwalające otrzymać informacje o jakości TP (testowanego produktu).
- Jeden z procesów pośrednio wpływających na zapewnienie jakości produktu
- Same testy nie wpływają na jakość produktu

Weryfikacja - proces oceny oprogramowania określający czy produkt na etapie fazy rozwojowej spełnia przedłożone w fazie startowej. Weryfikacja oprogramowania ma na celu sprawdzenie, czy wytwarzane oprogramowanie jest zgodne ze specyfikacją. Określenie czy produkt wytwarzany jest prawidłowo, zgodnie ze sztuką tworzenia oprogramowania

Walidacja - Proces oceny oprogramowania podczas albo na końcu procesu rozwojowego określający czy spełnione zostały określone wymagania. Walidacja sprawdza, czy oprogramowanie jest zgodne z oczekiwaniami użytkownika. Określenie czy spełnia wymagania postawione przez klienta.

Walidacja i **weryfikacja** to nie etapy procesu testowania, lecz jego cele



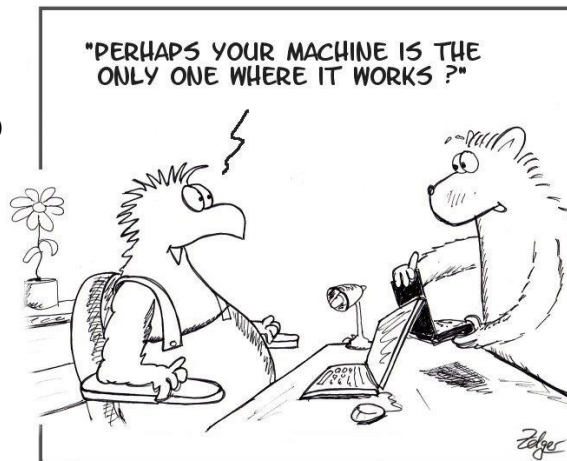
Co to jest testowanie

Ogólne cele testowania:

- Wykrywanie podstawowych błędów, by je usunąć,
- Sprawdzanie zgodności z innymi aplikacjami,
- Ułatwianie podjęcia decyzji interesariuszom o wypuszczaniu lub niewypuszczaniu produktu,
- Minimalizacja kosztów pomocy technicznej,
- Kontrola zgodności z wymaganiami i z prawnymi uregulowaniami.

Co **nie jest** celem testowania:

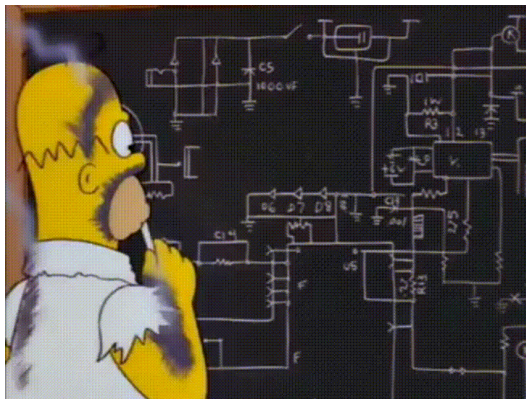
- debugging,
- udowodnienie, że oprogramowanie działa bezbłędnie,
- udowodnienie, że oprogramowanie nie działa.



It works on my machine



Co to jest testowanie



Czynności związane z testowaniem:

- planowanie,
- nadzorowanie,
- ustalanie warunków testowych,
- projektowanie zadań testowych,
- porównywanie wyników,
- ocena kryteriów zakończenia,
- raporty dotyczące procesu testowego i testowanej aplikacji,
- zakończenie i zamykanie testów (po zakończeniu fazy testów).

Dodatkowo

- przeglądy dokumentów i kodu
- analizę statyczną



Co to jest testowanie

Proces testowy składa się z minimum 5 faz (czynności)

1. planowanie i nadzór
2. analiza i projektowanie
3. implementacja i wykonanie
4. ocena spełnienia warunków zakończenia i raportowanie
5. czynności zamykające test.



Co to jest testowanie

1. Planowanie i nadzór:

- określa się:
 - cel i zakres testów (co podlega testowaniu),
 - metodologię testów (techniki testowania, metryki, pokrycie, itp.),
 - strategię testów,
 - zasoby (ludzie, narzędzia, sprzęt, środowisko);
- buduje się harmonogramy dla:
 - analizy testów i zadań projektowych,
 - implementacji i wykonania,
 - oceny testów;
- dokonuje się analizy ryzyka związanego z testami
- Nadzoruje się poprzez mierzenie i analiza rezultatów i monitorowanie i dokumentowanie postępu prac i pokrycia testowego



Co to jest testowanie

2. Analiza i projektowanie:

- zdefiniowanie i przeglądanie podstawy testów (wymagania, architektura, projekt, interfejsy)
- Identyfikacja warunków testowych, wymagań testowych, potrzebnych danych testowych na podstawie analizy przedmiotów testu, specyfikacji, zachowania i struktury
- projektowanie testów
- ocena testowalności wymagań i systemu
- projektowanie środowiska testowego
- identyfikacja całej potrzebnej infrastruktury oraz narzędzi



Co to jest testowanie

3. Implementacja i wykonanie:

Tworzone jest środowisko testowe

Wykonanie testów składa się z następujących czynności:

- wytwarzanie przypadków testowych i określanie priorytetów,
- tworzenie danych testowych,
- pisanie procedur testowych,
- przygotowywanie jarzma testowego i pisanie automatycznych skryptów testowych (opcjonalnie),
- tworzenie scenariuszy testowych na podstawie przypadków testowych celem efektywnego wykonania testu,
- weryfikacja poprawności utworzonego środowiska testowego,
- wykonanie przypadków testowych ręcznie lub przy pomocy narzędzi, w zaplanowanej kolejności logowanie wyniku wykonania testu w dzienniku testów oraz rejestrowanie jednostek i wersji testowanego oprogramowania, narzędzi testowych oraz testaliów



Co to jest testowanie

- porównywanie wyników rzeczywistych z oczekiwanymi,
- raportowanie rozbieżności jako incydentów i ich analiza mająca na celu
- znalezienie przyczyny wystąpienia błędu
- powtarzanie czynności testowych:
 - testowanie potwierdzające (retest) – ponowne wykonanie testu, który poprzednio wykrył awarię celem potwierdzenia usunięcia usterki,
 - testy regresywne (regresyjne) – wykonanie testów celem upewnienia się, że nowe defekty nie zostały wprowadzone do niezmiennych obszarów oprogramowania, lub że usuwanie usterek nie powoduje innych awarii,
 - wykonanie poprawionego testu



Co to jest testowanie

4. Ocena stopnia spełnienia warunków zakończenia i raportowanie:

- Wykonanie testu jest oceniane względem zdefiniowanych celów.
- Ocena powinna być wykonywana dla każdego poziomu testów.
- Oceniany jest stopień spełnienia warunków zakończenia

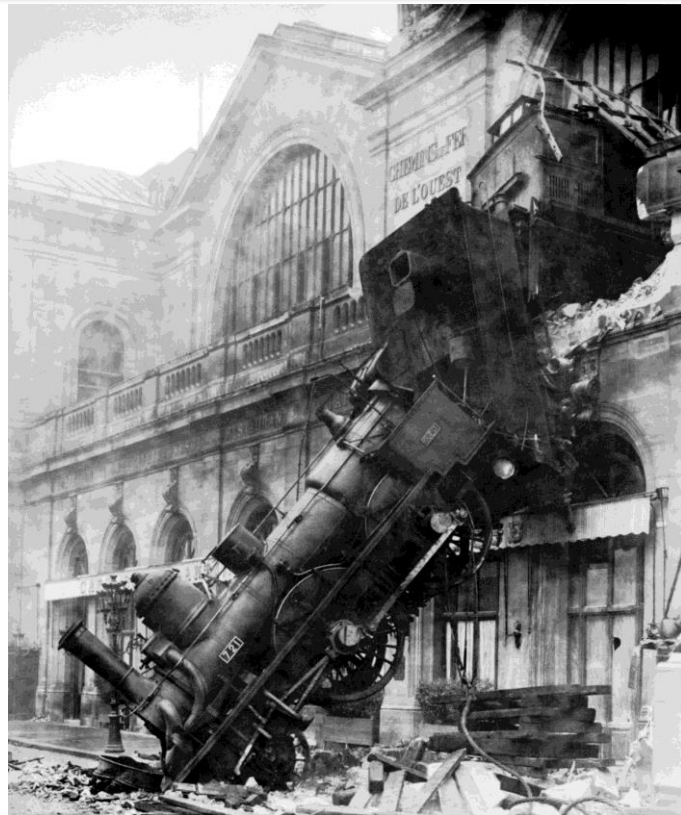
5. Czynności zamykające testowanie:

- sprawdzenie, które zaplanowane dostawy zostały dostarczone,
- zamknięcie raportów incydentów lub zgłoszenie zmian do tych, które pozostały otwarte,
- wyliczenie potrzebnych metryk,
- dokumentowanie akceptacji systemu,
- zakończenie i archiwizacja testaliów, środowiska testowego i infrastruktury testowej celem ponownego wykorzystania,
- przekazanie testaliów do zespołu serwisowego
- analiza wniosków na potrzeby przyszłych projektów oraz poprawy zdolności /dojrzałości testów



Dlaczego testowanie jest niezbędne

- Testowania umożliwia wykrycie błędów we wczesnych stadiach rozwoju oprogramowania, co pozwala zmniejszyć koszty usuwania tego błędu.
- Warto przeprowadzać testy na każdym etapie tworzenia oprogramowania.
- Testować należy jak najwcześniej, ponieważ podstawowymi źródłami błędów są specyfikacja i projekt.
- Im później wykryty zostanie błąd tym większy jest koszt jego usunięcia.
- Samo testowanie nie podnosi jakości oprogramowania i dokumentacji ale ułatwia zapobieganie awariom





Dlaczego testowanie jest niezbędne

- Przypadek Ariane 5 - 4 czerwca 1996 roku, rakieta Ariane 5, przygotowana przez Europejską Agencję Kosmiczną (European Space Agency) eksplodowała 40 sekund po starcie w Kourou,
Koszt: 7 miliardów USD (rakieta i ładunek 500 milionów USD)
Przyczyna: 64-bitowa liczba zmiennoprzecinkowa określająca poziomą prędkość rakiety była konwertowana na 16-bitową liczbę całkowitą. Ponieważ liczba była większa niż 32767, konwersja zawiodła
- 22 czerwca 1962 roku Rakieta Mariner 1 zostaje zniszczona przez oficer do spraw bezpieczeństwa naciska poprzez naciśnięcie czerwonego guzika
Sonda międzyplanetarna miała zbadać Wenus. Zamiast tego szczątki Marinera po 293 sekundach lotu spadły na Karaiby.
Przyczyna: Brak znaku „_” w kodzie nie pozwolił poprawnie zdetonować rakiety. Służył jako dodatkowy bajt do poprawnego przeliczenia uśrednionej prędkości rakiety.
- Rakiety Patriot, 25 lutego 1991 roku w Dharan w Arabii Saudyjskiej podczas pierwszej wojny w Zatoce Perskiej, amerykańska rakieta Patriot nie przechwyciła irackiego Scud'a. Scud trafił w barak armii amerykańskiej, zabijając 28 i raniąc ponad 100 osób
Przyczyna: Błąd arytmetyczny - System mierzył czas w dziesiątych częściach sekundy, używając 24-bitowego stałoprzecinkowego rejestru. Po około 100 godzinach pracy, błąd numeryczny skumulował się do 0,34 sekundy, co przy prędkości Scuda ponad 1676 m/s dawało odległość ponad 0,5 km. System śledzący Patriot uznał więc, że Scud jest poza jego zasięgiem.



Testowanie a jakość produktu

- Testowanie umożliwia określenie jakości produktu funkcjonalne i нефunkcjonalne (niezawodność, użyteczność, efektywność, utrzymywalność)
- Testowanie podnosi zaufanie co do jakości oprogramowania
- Testowanie umożliwia poprawę jakości, gdy błędy są znalezione i usunięte.
- Testowanie ułatwia zapobieganie awariom poprzez redukcję poziom ryzyka
- Testowanie jest tylko jednym z wielu środków służącym zapewnieniu jakości obok standardów kodowania, szkolenia, analiza błędów

To tell
somebody that
they are wrong
is called
criticism.

To do so
officially is
called testing.



Podstawowe zasady testowania

7 podstawowych zasad testowania

- I. Testowanie ujawnia usterki, a nie ich nieobecność
- II. Testowanie gruntowne jest niemożliwe
- III. Testowanie powinno zacząć się jak najwcześniej
- IV. Kumulowanie się defektów blisko siebie
- V. Paradoks pestycydów
- VI. Testowanie zależne jest od kontekstu
- VII. Mylne przekonanie o braku błędów



faza	analiza	projekt	kodowanie	testy jednostkowe	testy integracyjne	testy systemowe	po wdrożeniu
Koszt	1	5	10	15	22	50	100+
Czas	6 minut	0,5h	1h	1,5h	2,2h	5h	10h+



RODZAJE TESTÓW

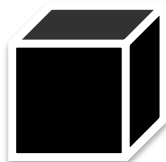


- Testy białoskrzynkowe i czarnoskrzynkowe
- Testy manualne i automatyczne
- Testowanie statyczne i dynamiczne
- Poziomy testów
- Testy ze względu na cele
- Statyczne techniki testowania

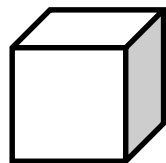


Testy białoskrzynkowe i czarnoskrzynkowe

Testowanie oprogramowania dzieli się na dwa główne nurty z punktu widzenia testera



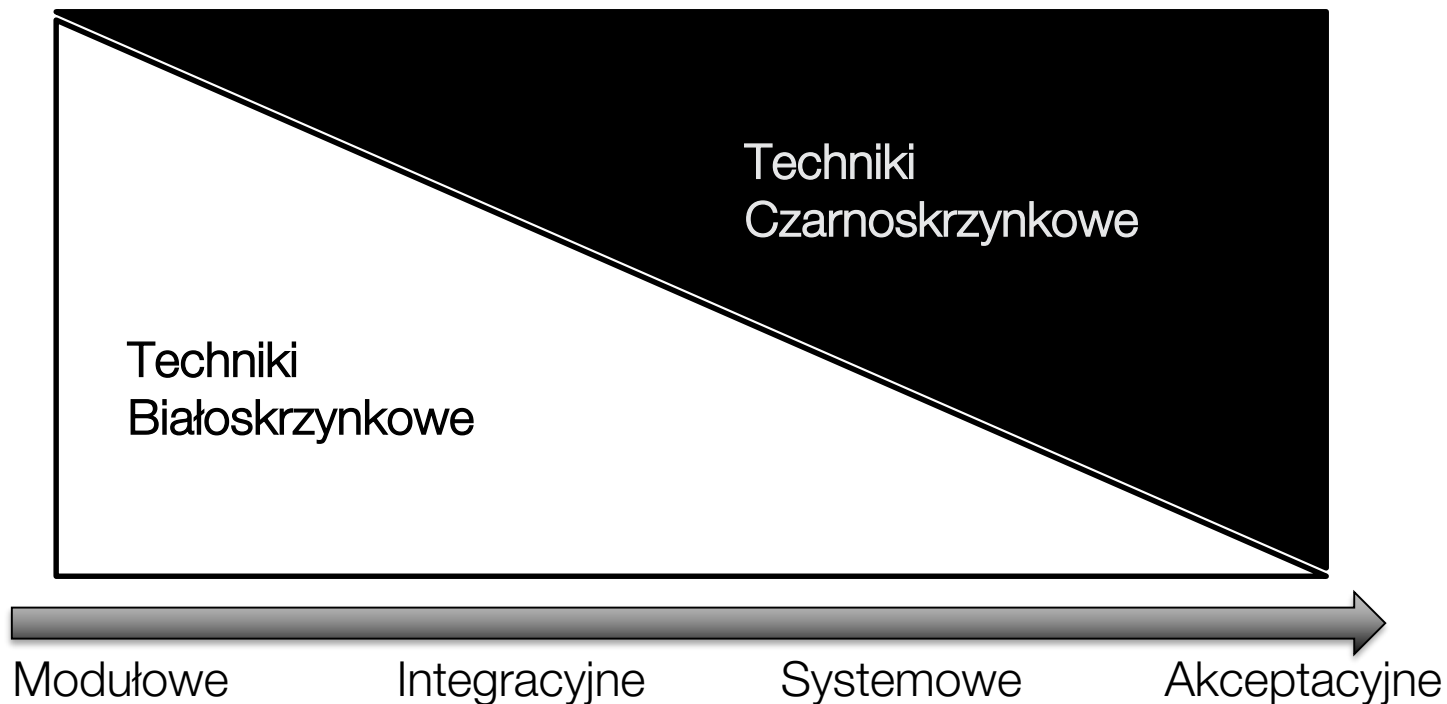
czarnej skrzynki (black box testing) – inaczej testy funkcjonalne, gdzie oprogramowanie traktowane jest jako „czarna skrzynka” a tester nie wnika w sam kod źródłowy. Brana jest pod uwagę funkcjonalność testowanego obiektu.



białej skrzynki (white box testing) – inaczej testy strukturalne, gdzie tester ma dostęp do kodu źródłowego. Testowane są poszczególne moduły testowanej aplikacji lub kodu.



Testy białoskrzynkowe i czarnoskrzynkowe





Testy manualne i automatyczne

Testowanie ze względu na sposób wykonywania można podzielić:



Testy manualne – polegające na manualny wykonaniu sekwencji lub szeregu testów



Testy automatyczne – wykorzystujące języki skryptowe w celu zautomatyzowania procesu testowego



Testowanie statyczne i dynamiczne

Testowanie statyczne - testowanie modułu lub systemu na poziomie specyfikacji lub implementacji bez wykonywania tego oprogramowania, np. przeglądy (review) lub analiza statyczna kodu.



Testowanie dynamiczne - testowanie, podczas którego wykonywany jest kod modułu lub systemu.





Poziomy testów

Testy można podzielić ze względu na ich poziom, który jest ściśle powiązany z etapem w procesie wytwarzania oprogramowania



testy modułowe / testy jednostkowe - testują oprogramowanie na najbardziej podstawowym poziomie – na poziomie działania pojedynczych funkcji (metod), klasy lub modułu



testy integracyjne - Testowanie integracyjne wykonywane jest w celu wykrycia błędów w interfejsach i interakcjach pomiędzy modułami, komponentami.



testy systemowe - Testy systemowe przeprowadzane są, aby stwierdzić czy zintegrowany już system spełnia jako całość wymagania zawarte w specyfikacji.



testy akceptacyjne - Walidacja systemu pod kątem zgodności z wymaganiami klienta, który na swoim środowisku wykonuje przypadki testowe przy udziale przedstawicieli projektu



Poziomy testów

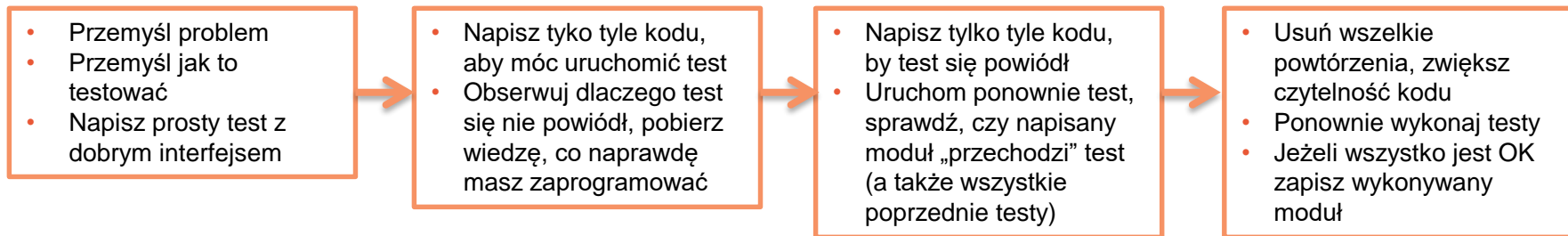
Testy modułowe / testy jednostkowe (unit tests)

- Testowanie na najniższym poziomie, podczas którego poszczególne metody (funkcje) testowane są pojedynczo, w oderwaniu od reszty aplikacji
- Testują pod kątem zgodności ze zdefiniowanym typem/zakresem danych wejściowych testy modułów powinny być przeprowadzane dla każdego produktu informatycznego
- Testowanie modułowe pozwala na wprowadzenie elementów zapewnienia jakości już we wczesnym etapie wytwarzania TP
- Do testów modułowych można zaliczyć
 - Analiza ścieżek (path analysis)
 - Klasy równoważności (equivalence partition)
 - Analiza wartości brzegowych
 - testowanie składniowe



Poziomy testów

- Alternatywą dla testów modułowych może być metodologia pochodząca z „eXtreme Programming” DD – Test Driven Development (*wytwarzanie sterowane testowaniem*) lub Test-first programming (*najpierw przygotuj testy*)
- Polega na idei – najpierw napisz testy, a następnie twórz kod.
- Dwa cele: sposób przemyślenia problemu przed stworzeniem kodu (bardziej specyfikacja niż walidacja), technika programistyczna zwiększająca poprawność tworzonego kodu,
- Wspiera testy jednostkowe.

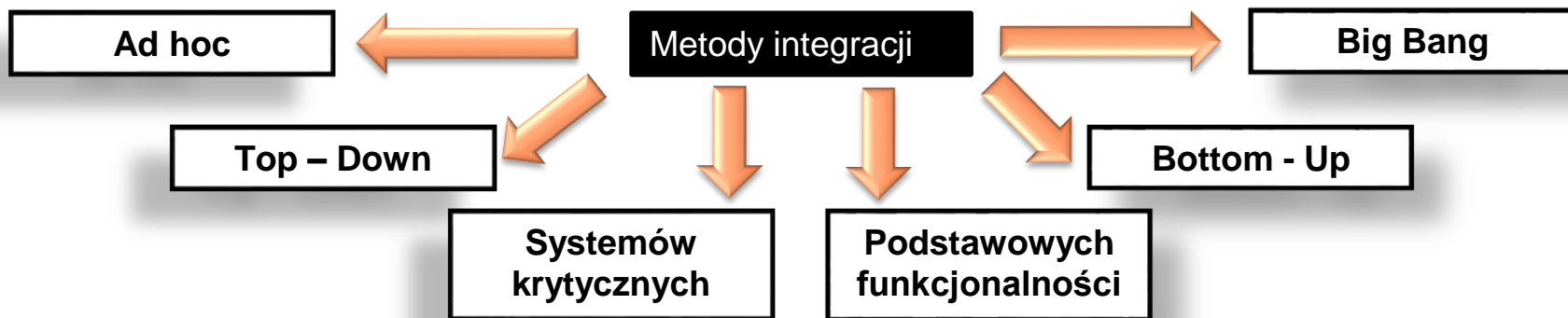




Poziomy testów

Testy integracyjne (integration tests)

- Testowanie wykonywane w celu wykrycia błędów w interfejsach i interakcjach pomiędzy integrowanymi elementami
- Testy integracyjne mogą być pomiędzy modułami (testy funkcjonalne i wydajnościowe) jak i pomiędzy systemami (testy funkcjonalne, wydajnościowe i regresywne)
- Metoda integracji i wykonywania testów zależy od czynników na podstawie których następuje integracja (architektura systemu, plan projektu, strategia testowania)





Poziomy testów

Typowe błędy znajdowane podczas testów integracyjnych:

- nie następuje przekazanie danych,
- nadawca przekazuje syntaktycznie nieprawidłowe dane, tak że odbiorca nie może ich odczytać,
- komunikacja ma miejsce, lecz jest nieprawidłowa semantycznie – przekazywane dane rozumiane są w różny sposób,
- dane przekazywane są prawidłowo, ale w złym czasie, z dużym opóźnieniem lub w zbyt krótkich odstępach czasu.

Zaśleпка (stub) – szkieletowa albo specjalna implementacja modułu używana podczas produkcji lub testów innego modułu, który tę zaślepkę wywołuje albo jest w inny sposób od niej zależny.

Sterownik testowy (test driver) – program lub narzędzie testowe używane do uruchamiania oprogramowania w celu wykonania zestawu przypadków testowych.



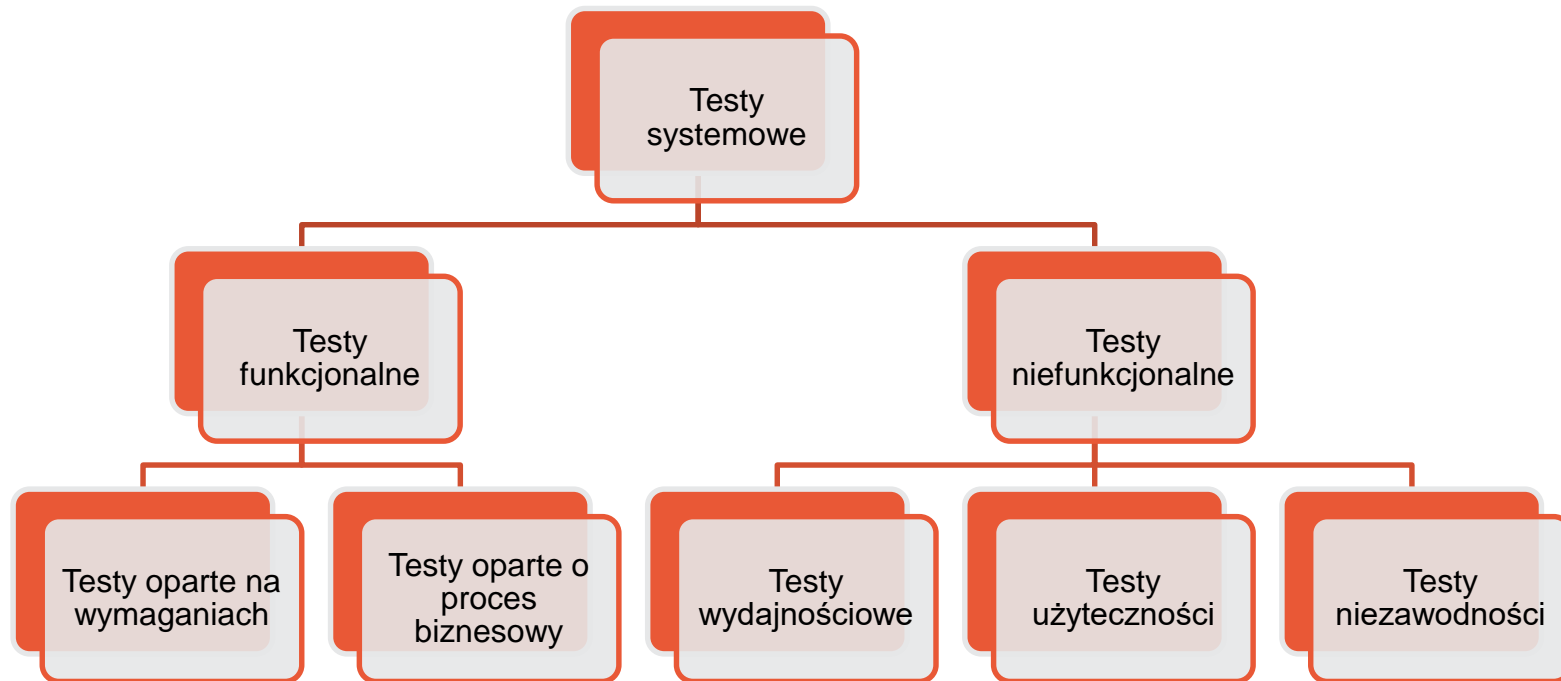
Poziomy testów

testy systemowe

- proces testowania sprawdzający, czy zintegrowany system spełnia wyspecyfikowane wymagania
- Podczas testów systemowych cały system jest weryfikowany pod kątem zgodności z: wymaganiami funkcjonalnymi i wymaganiami нефunkcjonalnymi (wydajność, użyteczność, niezawodność)
- Testy systemowe są pierwszym poziomem na którym system jest testowany jako całość
- Testy systemowe przeprowadzane są na środowisku zbliżonym do produkcyjnego , aby odwzorować rzeczywiste warunki w których będzie działał system
- Wymagania systemu muszą być dokładnie zdefiniowane i opisane,
- Testowany produkt powinien spełniać warunki wejściowe przed rozpoczęciem testów systemowych (np. przejść smoke test)



Poziomy testów





Poziomy testów

Testy akceptacyjne

- przeprowadza się w celu umożliwienia użytkownikowi, klientowi lub innemu uprawnionemu podmiotowi ustalenia, czy można zaakceptować system lub moduł.
- testy akceptacyjne powinny być przeprowadzane w środowisku produkcyjnym
- testy przeprowadzane się na podstawie dokumentu opisującego specyfikację testów akceptacyjnych
- Podział testów akceptacyjnych ze względu na miejsce, środowisko i profil działalności testerów: testy alfa i testy beta
- Podział testów akceptacyjnych ze względu na charakter akceptacji
 - Testowanie akceptacyjne przez użytkownika – Akceptacja użytkownika.
 - Testowanie zgodności oprogramowania z kontraktem - Akceptacja zgodności z kontraktem biznesowym.
 - Testowanie zgodności legislacyjnej – Akceptacja zgodności legislacyjnej.
 - Testowanie zgodności operacyjnej – Akceptacja zgodności operacyjnej.



Testy ze względu na cele

Testy funkcjonalne

- testy funkcjonalne to inaczej testy wymagań funkcjonalnych (funkcji),
- wymagania funkcjonalne zawierają opis zachowania systemu,
- to najczęściej wykonywane testy,
- dotyczą funkcji lub cech systemu (udokumentowanych lub znanych testerom)
- wykonuje się je na wszystkich poziomach testów,
- są mierzalne jeżeli chodzi o pokrycie („wszystkie funkcje zostały przetestowane”)
- są podatne na automatyzację

Software Projects

What Client expected.



What was delivered.

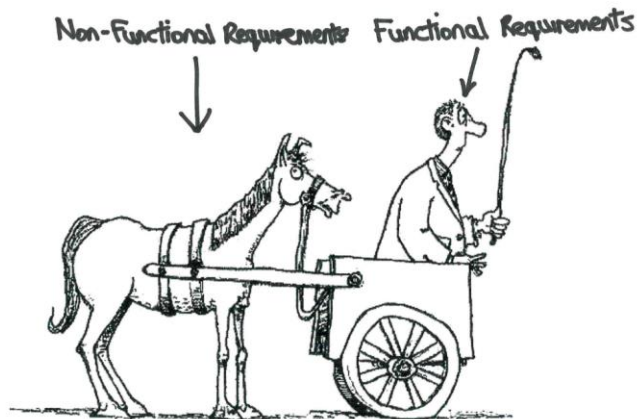




Testy ze względu na cele

Testy нефункционалне

- Testy нефункционалне то иначеж тесты влăściwości / wymagań нефункционалных. Wymagania te opisują atrybuty нефункционалне,
- влăściwości нефункционалне opisują następujące cechy testowanego produktu: wydajność (testy wydajności, obciążeniowe, przeciążające), użyteczności, niezawodność, efektywność, przenaszalność, możliwości procedury instalacyjnej,
- testowanie нефункционалне można przeprowadzać na wszystkich poziomach testów.





Testy ze względu na cele

Testy strukturalne

- Inaczej są to testy architektury systemu, białoskrzynkowe.
- Wykonywane są na wszystkich poziomach testów zarówno w testach dynamicznych jak i statycznych.
- Niezbędna jest znajomość kodu, architektury, zasad komunikacji modułów / systemów.
- Testy strukturalne można wykonywać zgodnie z architektury systemu, np. hierarchii wywołań.
- Testy strukturalne mogą być również stosowane na poziomie testów systemowych, testów integracji systemów lub testów akceptacyjnych lub w odniesieniu do modelu biznesowego



Testy ze względu na cele

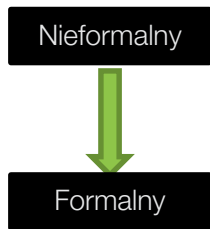
Testowanie związane ze zmianami

- Re-test (powtórzenie testu, test potwierdzający) – powtórne wykonanie przypadku testowego, którego pierwotne wykonanie wykazało awarię (nieprawidłowe działanie) testowanego produktu. Celem re-testu jest sprawdzenie, czy defekt będący przyczyną awarii został usunięty.
- Testy regresywne (regresyjne) – ponowne przetestowanie uprzednio testowanego programu po dokonaniu w nim modyfikacji, by upewnić się, że w wyniku zmian nie powstały nowe błędy lub nie ujawniły się wcześniej nieznanne.



Statyczne techniki testowania

- **Testowanie statyczne** – testowanie bez wykonywania go na komputerze. Są to zarówno techniki ręczne jak i zautomatyzowane. Przegląd jest to najczęściej używana forma testowania statycznego.
- Przeglądy są sposobem testowania produktów pracy programistycznej.
- Defekty wykryte podczas przeglądów na wczesnym etapie cyklu życia oprogramowania są dużo tańsze do usunięcia niż defekty wykryte podczas wykonywania późniejszych testów.
- Przejrzeć można każdy produkt pracy programistycznej np. specyfikację wymagań, specyfikację projektu, kod, plany testów, specyfikację testów, przypadki testowe, skrypty testowe, instrukcje użytkownika, strony internetowe.
- Podstawowe rodzaje przeglądów:
 - przejrzanie (przejście),
 - przegląd techniczny,
 - inspekcja





PLANOWANIE I TWORZENIE TESTÓW



- Identyfikacja warunków testowych
 - Organizacja i planowanie testów
 - Projektowanie i tworzenie przypadków testowych
 - Ustalenie formy raportowania wyników testów
-



Identyfikacja warunków testowych

Budowa warunków testowych

- identyfikacja elementu lub zdarzenia, które powinno być zweryfikowane przez jeden lub więcej przypadków testowych,
- śledzenie powiązań pomiędzy warunkami testowymi, a specyfikacją lub wymaganiami
- analiza pokrycia wymagań określonych zestawem testów

Specyfikacja przypadków testowych

- Budowa i opisywanie danych, przypadków testowych, wyników oczekiwanych
- Przypadek testowy (test case) składa się z zestawu wartości wejściowych, warunków początkowych, oczekiwanych wyników i warunków końcowych, zaprojektowanych w celu pokrycia pewnych warunków testowych

Procedura testowa

- przygotowane przypadki testowe ustawione są w kolejności wykonywania
- w przypadku testów automatycznych, kolejność określana jest w skrypcie testowym

Harmonogram wykonania testu

- według kolejności wykonania
- terminarz
- osoba odpowiedzialna
- priorytet



Organizacja i planowanie testów

Organizacja testów:

- Testowanie niezależne (niezależny zespół testowy) – autor często nie widzi swoich błędów, inne osoby mają inne podejście na temat testów.
- Poziom niezależności – inna osoba, inny zespół, outsourcing
- Testowanie na wielu poziomach
- Dobór narzędzi do raportowania, wykonywania testów i śledzenia i analizy wyników (tracability)

Główne zadania zespołu testowego:

- planu testów,
- przegląd i analiza wymagań,
- tworzenie środowiska testowego,
- przygotowanie i pozyskiwanie danych testowych,
- implementacja testów,
- używanie narzędzi do administrowania lub zarządzania testami,
- automatyzacja testów,
- mierzenie wydajności,
- praca zespołowa,



Organizacja i planowanie testów

Planowanie testów ma na celu określić

- przedmiotów testowania,
- funkcji do przetestowania,
- czynności, które trzeba wykonać,
- osób odpowiedzialnych za każdą z tych czynności,
- ryzyka związanego z planem
- dostępność zasobów

Czynniki brane pod uwagę przy planowaniu

- polityka testów,
- zakres testowania,
- cele,
- zagrożenia,
- ograniczenia,
- krytyczność,
- skomplikowanie testowanego produktu,
- dostępność zasobów



Projektowanie i tworzenie przypadków testowych

- Analiza wymagań lub/i usecasów.
- Tworzenie specyfikacji przypadków testowych (TC specification), opisu przypadków testowych (TC description) i innych potrzebnych dokumentów.
- Planowanie.
- Implementacja przypadków testowych z uwzględnieniem procesu raportowania błędów, automatyzacji i CI.
- Uruchamianie procesu testowania.
- Raportowanie wyników.
- Poprawianie przypadków testowych.
- Przy każdym etapie często następuje proces rewiew i akceptacji klienta.

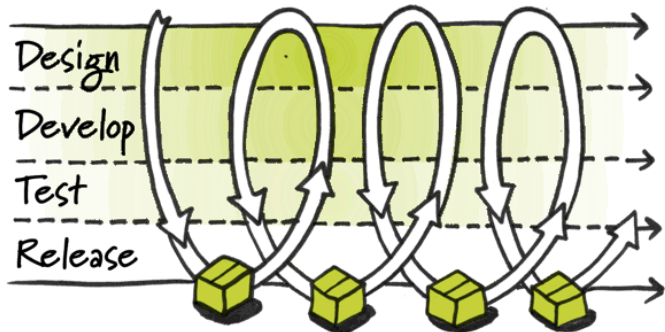


Ustalenie formy raportowania wyników testów

- Często zapewniane jest to narzędziowo lub poprzez skrypty.
- Ustalona musi być forma raportowania wyników.
- Analiza wyników odbywać się może automatycznie poprzez wyznaczenie współczynników KPI (mierzących pokrycie testów itp.).
- Wykryte błędy muszą być raportowane, logi muszą być przechowywane (narzędzie do raportowania błędów).
- Implementacja przypadków testowych musi być wersjonowana (svn, git) i powiązana z opisem przypadków testowych (dokumentacji).
- Wszystko musi zapewniać traceability (wersja testowanego oprogramowania, zaraportowany błąd, opis przypadku testowego, implementacja przypadku testowego).
- Dokumentacja powinna być też wersjonowana i przechowywana(z review).



CONTINUOUS INTEGRATION



- Co to jest CI w inżynierii oprogramowania?
- Dobre praktyki w CI
- Co powinno być wersjonowane
- Zalety i wady CI
- Narzędzia do CI



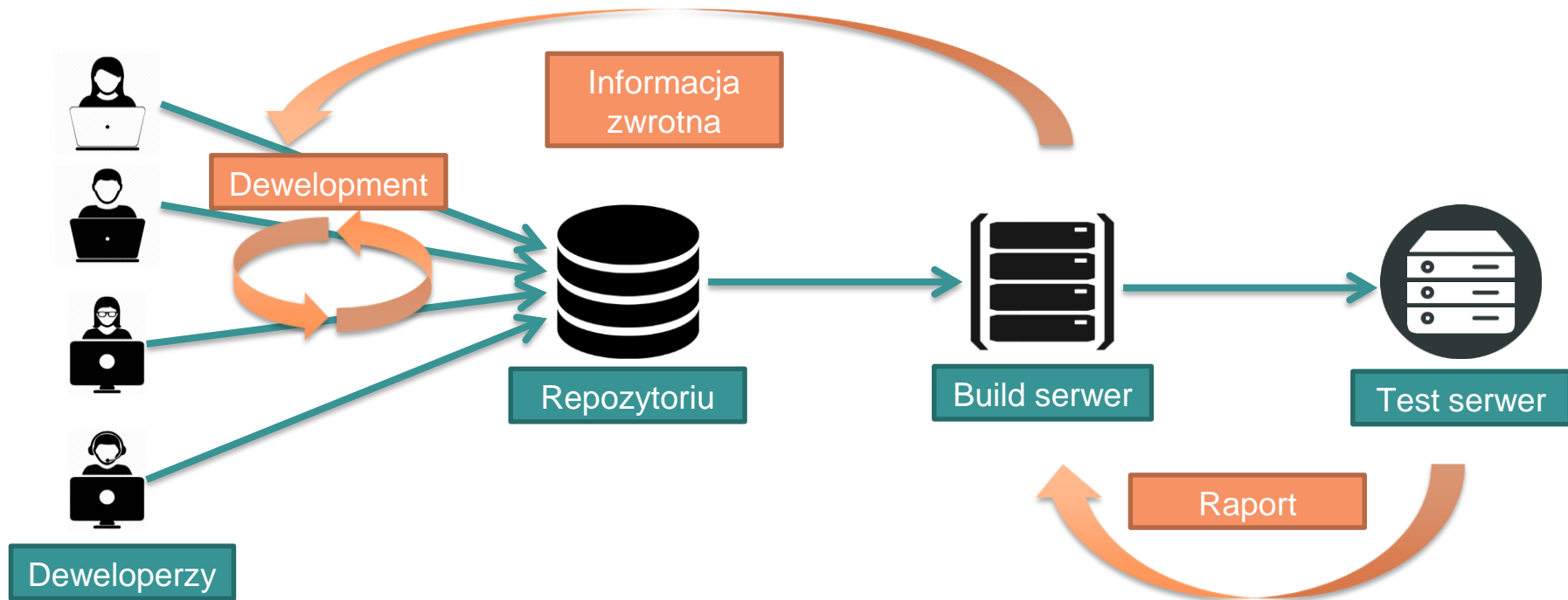
Co to jest CI w inżynierii oprogramowania?

- Jest to ciągły proces sprawdzania jakości dostarczanego kodu
- Jest to metoda rozwoju oprogramowania polegająca na codziennej integracji
- Ciągłej weryfikacji automatyczny buildów
- Automatycznego wykonywania testów
- Generowanie raportów i wyników





Co to jest CI w inżynierii oprogramowania?





Dobre praktyki w CI

- Pojedyncze repozytorium kodu i testów
- Automatyczne wykonywanie buildów i testów
- Wszyscy komitują codziennie
- Buildy powinny wykonywać się szybko
- Wszyscy widzą jaki jest aktualny stan
- Wyniki są generowane automatycznie





Co powinno być wersjonowane



- Kod źródłowy
- Kod testów
- Skrypty budujące (Build scripts)
- Narzędzia budowania (Build Tools)
- Skrypty Konfiguracyjne (Configuration scripts)
- Raporty i wyniki



Zalety i wady CI

PRO's

- **Lepsza** i łatwa kontrola kodu
- **Szybkie** testowanie kodu w czasie rzeczywistym
- Pozwala usunąć błędy szybko na samym początku ich powstania co **zmniejsz koszty** ich powstania

CON's

- Potrzeba dodatkowego **sprzętu**
- Potrzeba dodatkowego **personelu**
- Potrzeba dodatkowego **czasu**



Narzędzia do CI

- Jenkins / Hudson



- Bamboo



- Team City



- Cruise Control

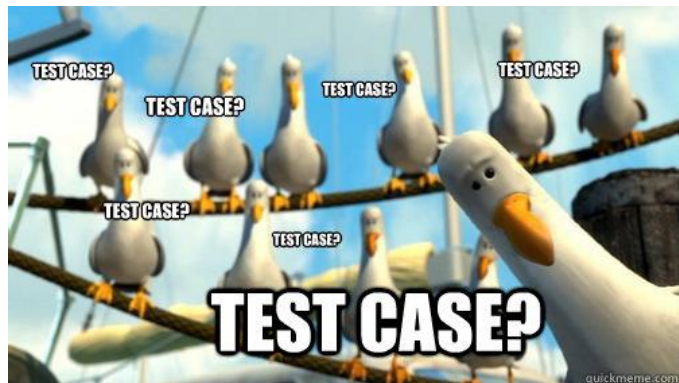


- Team Foundation Server





TECHNIKI TWORZENIA TESTÓW



- Techniki testowania
- Techniki czarnoskrzynkowe
- Techniki białoskrzynkowe
- Techniki oparte o incydent
- Techniki oparte o doświadczenie



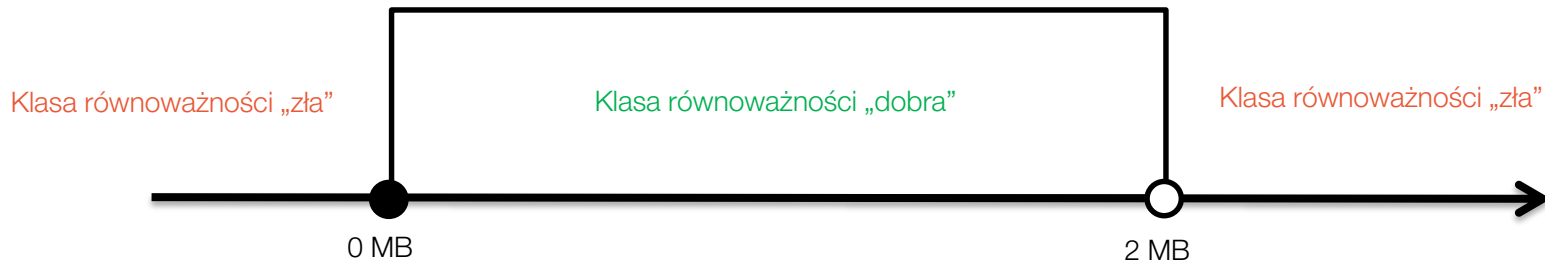
Techniki testowania

- oparte o specyfikacje (czarnoskrzynkowe, black-box),
 - klasy równoważności,
 - badanie wartości brzegowych,
 - tablice decyzyjne,
 - graf przyczynowo - skutkowy,
 - grafy stanów przejść,
 - tablice ortogonalne,
 - tablice testów wszystkich par,
 - use case testing – przypadki testowe.
- oparte o strukturę (białoskrzynkowe, white-box),
 - testowanie instrukcji,
 - testowanie decyzyjne (rozgałęzień),
 - LCSAJ (Liniowa Sekwencja Kodu i Skok)
- oparte o incydenty (defect-based techniques),
- oparte o doświadczenie (experienced-based techniques).



Techniki czarnoskrzynkowe

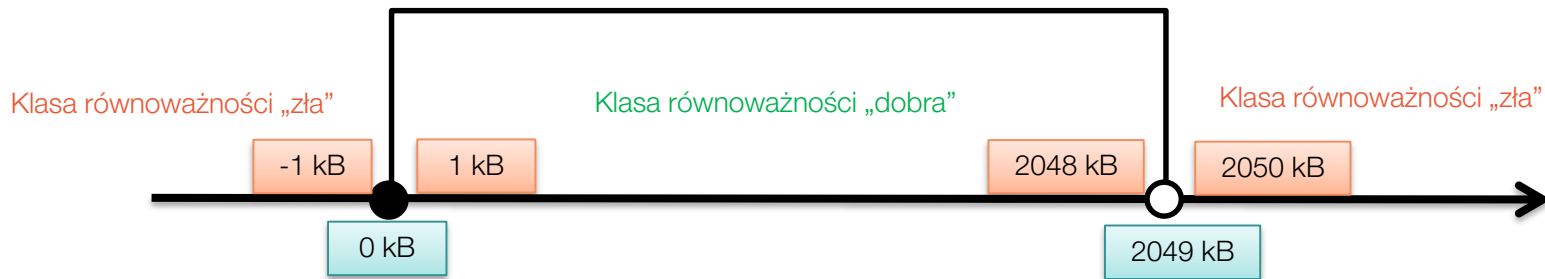
- **Klasy równoważności (KR) (*equivalence partition*)** - zbiór danych możliwych zmiennych dzielony jest na rozłączne podzbiory klasy pewnej relacji równoważności danych używanych do przeprowadzenia testu. Wykonanie testu z użyciem kilku elementów zbioru, powoduje uznanie całej klasy za poprawną i zwalnia nas od testowania wszystkich elementów w zbiorze.
- **Ścieżki równoważne** - Dwie wartości testowe są równoważne, jeżeli powodują wykonanie przez przypadek testowy tej samej ścieżki (np. tyle samo iteracji w pętli)
- **Testowanie w oparciu o klasy równoważności** – technika projektowania przypadków testowych, w której przypadki testowe projektowane są tak, aby użyć elementów z klas równoważności.





Techniki czarnoskrzynkowe

- **Wartość brzegowa** - to wartość znajdująca się wewnątrz, pomiędzy lub tuż przy granicy danej klasy równoważności.
- **Analiza wartości brzegowych (AWB)** (*boundary value analysis*) - wartość brzegowa to wartość znajdująca się wewnątrz, pomiędzy lub tuż przy granicy danej klasy równoważności.



Wartości dla przypadków testowych z klasy abstrakcji:

- Dla klasy „dobrej” = 100
- Dla klas „złych” = -10 , 2060

Wartości dla przypadków testowych z klasy wartości brzegowych:

- Dla klasy „dobrej” = 0 , 2048
- Dla klasy „złej” = -1 , 2049



Techniki czarnoskrzynkowe

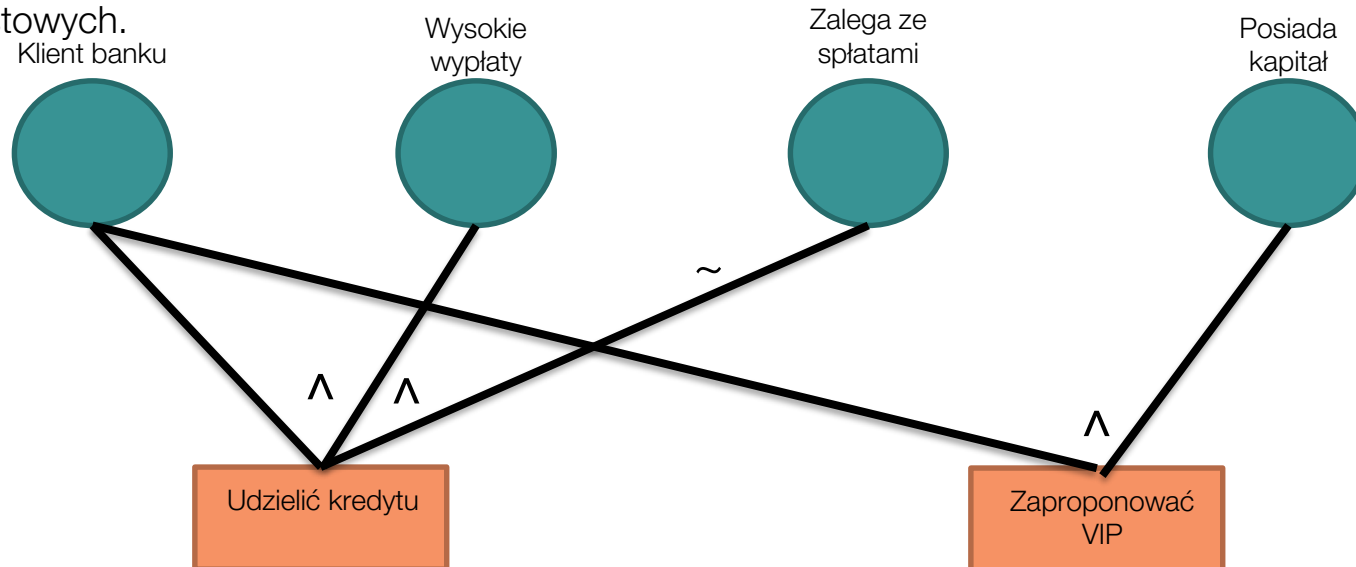
- **Testowanie w oparciu o tablicę decyzyjną** (*decision table testing*) - rodzaj projektowania przypadków testowych polegająca na sprawdzeniu działania modułu lub systemu w reakcji na kombinacje warunków wejściowych podanych w tablicy decyzyjnej.
- Testowanie te oparte o wymagania zawierające warunki logiczne i udokumentowanie wewnętrznego projektu systemu.
- warunki wejściowe i akcje ustawia się w taki sposób, aby odzwierciedlały prawdę lub fałsz

	K1	K2	K3	K4	K5	K6	K7	K8	K9
Warunki									
Ma konto w banku	False	True	True	True	True	True	True	True	True
Ma > 100 000 zł	False	False	False	False	False	True	True	True	True
Zalega ze spłatą kredytu	False	False	False	True	True	False	False	True	True
Miesięcznie > 5 000 zł	False	False	True	False	True	False	True	False	True
Akcje									
Nowy kredyt	False	False	True	False	False	True	True	False	False
Zaproponował VIP	False	False	False	False	False	True	False	True	True



Techniki czarnoskrzynkowe

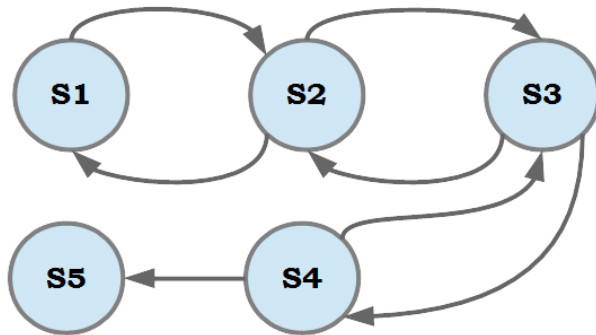
Graf przyczynowo-skutkowy (*cause-effect graphing*): Graficzna reprezentacja wejść i/lub bodźców (przyczyn) z odpowiadającymi im wyjściami (efektami), które mogą być wykorzystane do zaprojektowania przypadków testowych.





Techniki czarnoskrzynkowe

- **Testowanie przejść pomiędzy stanami** (*state transition testing*) – testowanie zachowania się przypadku testowego oparty na dobrze zdefiniowanych stanach systemu i przejściach pomiędzy nimi (**automaty skończone**). Inaczej technika projektowania przypadków testowych, w której przypadki te są tak dobierane, aby sprawdzały przejścia między stanami
- **Zmiana stanu** – przejście pomiędzy dwoma dozwolonymi stanami systemu lub modułu
- Przypadki testowe to sekwencje zdarzeń powodujących przejście między stanami





Techniki czarnoskrzynkowe

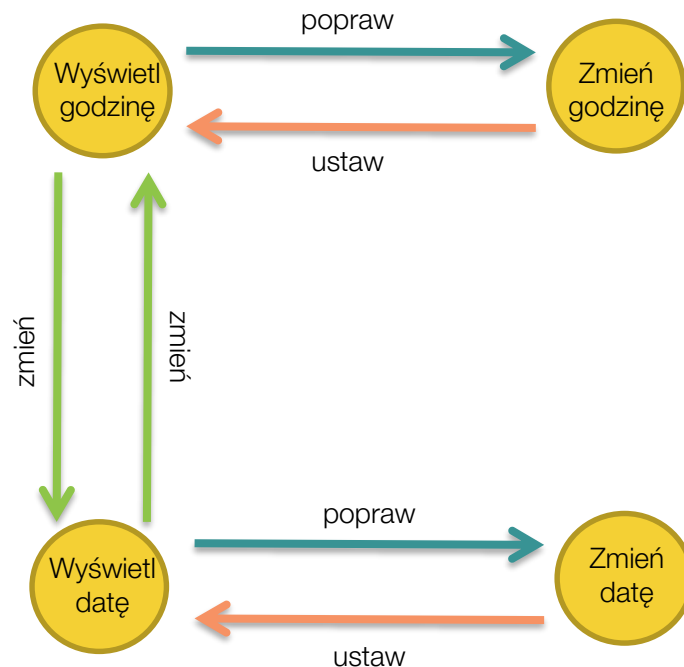
Przykład – zegarek elektroniczny

Ma 4 tryby pracy:

- *wyświetl godzinę*,
- *wyświetl datę*,
- *zmień godzinę*,
- *zmień datę*

i 3 przyciski:

- *zmień* – przełącznik między *wyświetlaniem godziny* i *wyświetlaniem daty*,
- *popraw* – przełącznik pomiędzy trybami *wyświetl/i* *zmień*;
- *ustaw* – przełącznik między trybami *zmień i* *wyświetl*.

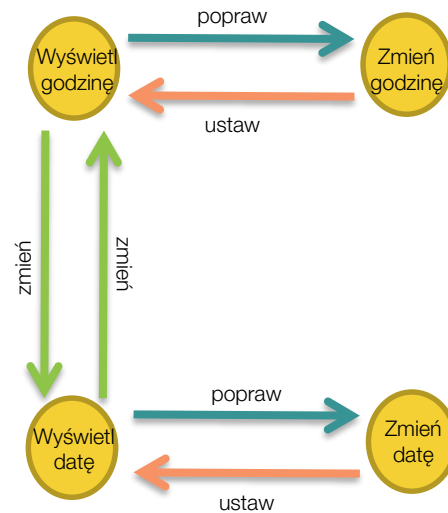




Techniki czarnoskrzynkowe

Tablica przejść


	Popraw	Ustaw	Zmień
Wyświetl Godzinę	Zmień Godzinę		Wyświetl datę
Zmień Godzinę		Wyświetl Godzinę	
Wyświetl datę	Zmień datę		Wyświetl Godzinę
Zmień datę		Wyświetl datę	

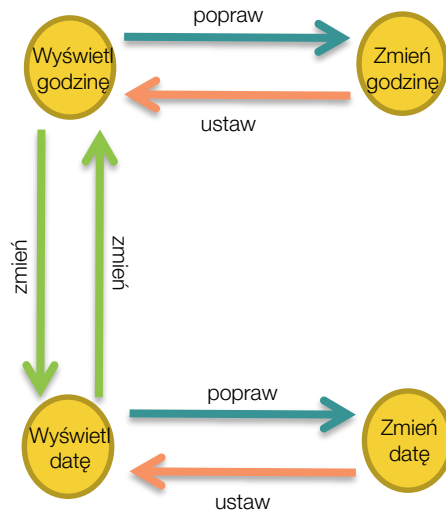




Techniki czarnoskrzynkowe

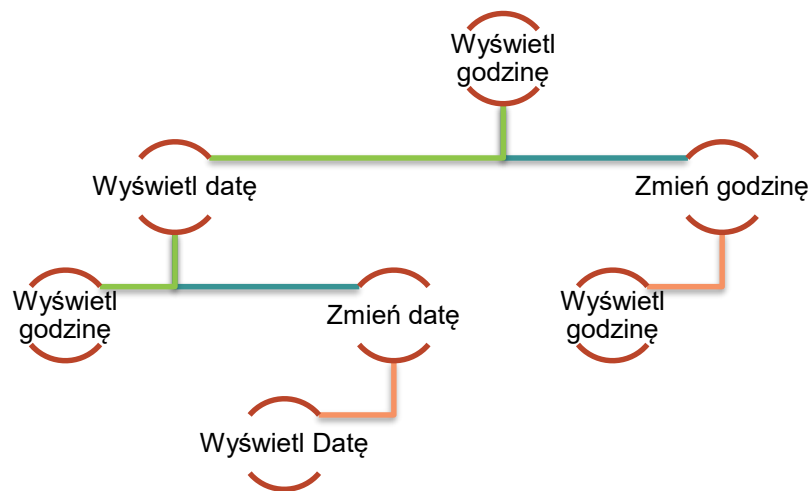
Tablica przejść

	Wyświetl Godzinę	Zmień Godzinę	Wyświetl datę	Zmień datę
Wyświetl Godzinę		Ustaw	Zmień	
Zmień Godzinę	Popraw			
Wyświetl datę	Zmień			Ustaw
Zmień datę			Popraw	

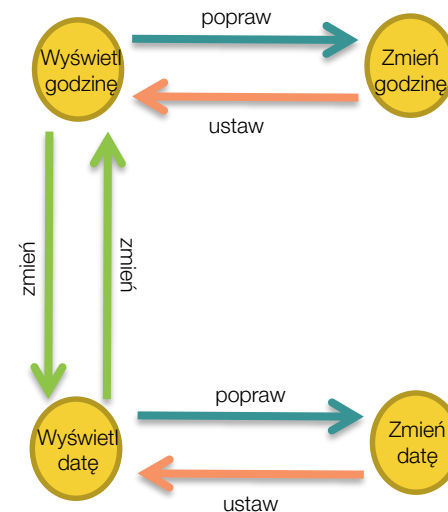




- Testowanie przejść pomiędzy stanami. Pokrycie 0-przejściowe – pokrycie przejść

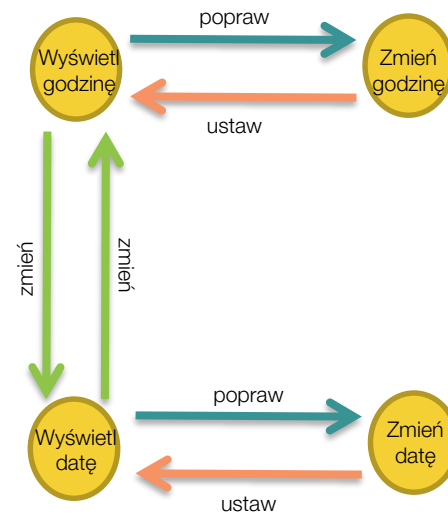
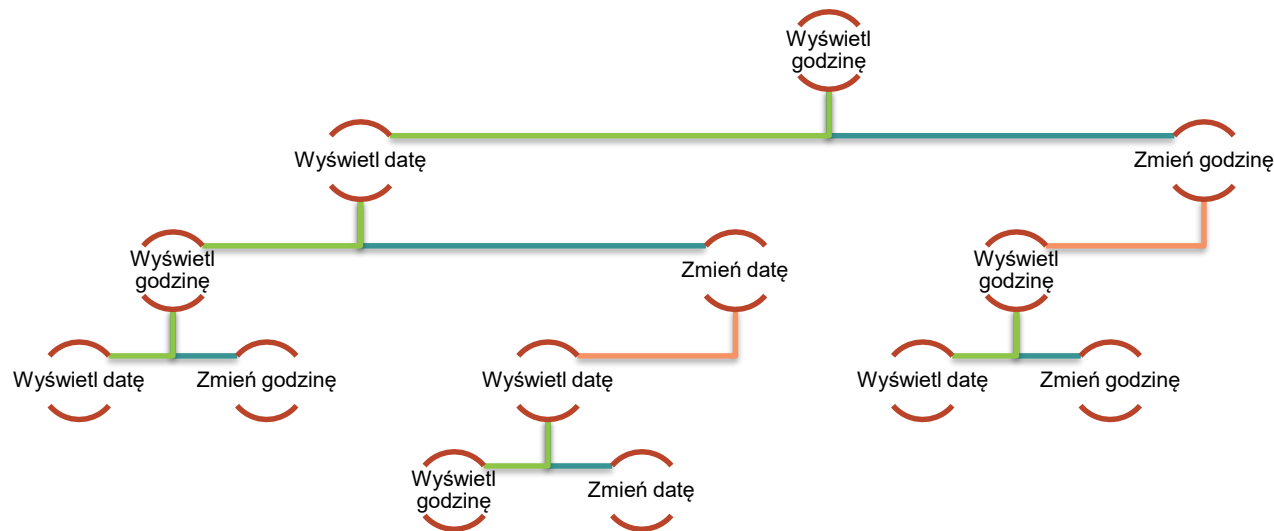


3 przypadki testowe





- Testowanie pętli pomiędzy stanami. Pokrycie 1-przejęciowe

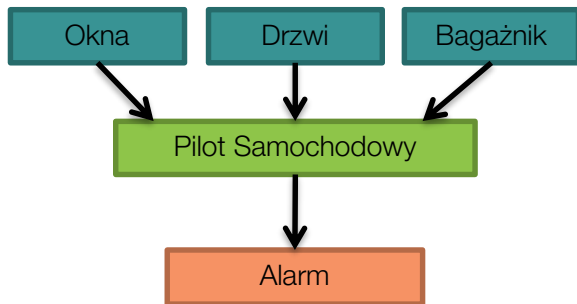


6 przypadki testowe



Techniki czarnoskrzynkowe

Tablica ortogonalna (*orthogonal arrays*)- 2-wymiarowa tablica wybrana ze zbioru predefiniowanych tablic, opartych o kombinacje pewnej liczby zmiennych i zakresu wartości tych zmiennych. Każda zmienna reprezentuje kolumnę, a każda wartość tej zmiennej pojawia się w tablicy wielokrotnie. Ilość wierszy odpowiada liczbie przypadków testowych potrzebnych do pokrycia każdej pary kombinacji wartości dwóch zmiennych.



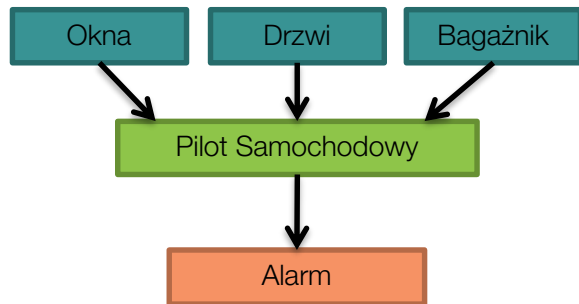
	Okna	Drzwi	Bagażnik
TC1	0	0	0
TC2	0	0	1
TC3	0	1	0
TC4	0	1	1
TC5	1	0	0
TC6	1	0	1
TC7	1	1	0
TC8	1	1	1



Techniki czarnoskrzynkowe

Testowanie par (*Pairwise testing*): projektowania przypadków testowych w której przypadki testowe są projektowane tak, aby wykonać wszystkie możliwe dyskretne kombinacje każdej pary parametrów wejściowych.

Założenia: Okna i drzwi pracują w jednym systemie (okna i drzwi pracują w parze)



	Okna	Drzwi	Bagażnik
TC1	0	0	0
TC2	0	0	1
TC3	1	1	0
TC4	1	1	1

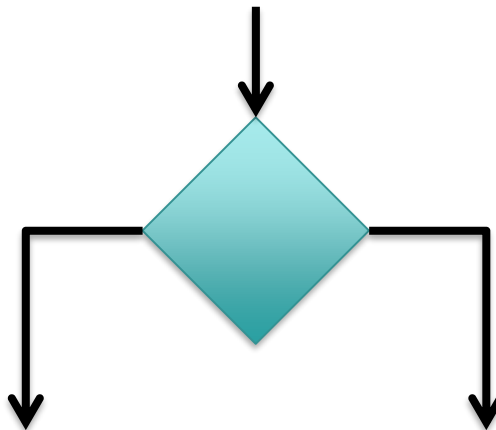


Techniki białoskrzynkowe

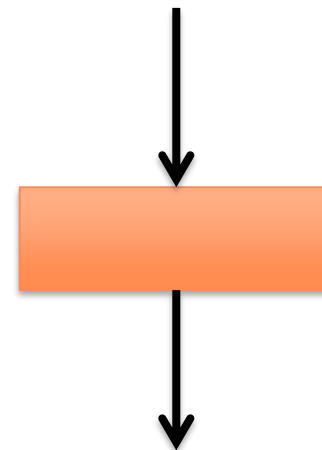
Graf przepływu sterowania:



Skok



Rozgałęzienie



Instrukcja



Techniki białoskrzynkowe

Miary złożoności:

- Ilość linii kodu (*LoC*) – prosta i użyteczna miary
- Indeks Mc Cabe (*Indeks Cyklomatycznej Złożoności CC*)

- Podstawowy wzór:

$$CC = E - N + p$$

E – ilość krawędzi w grafie

N – ilość wierzchołków w grafie

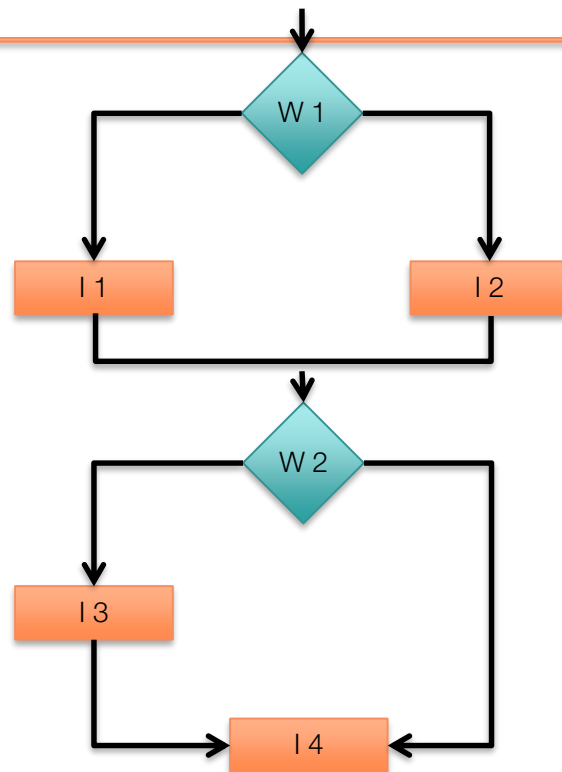
p – ilość wykonywanych modułów

- Uproszczony wzór:

$$CC = E - N + 1$$

- W rezultacie wyznaczana jest niestabilność

CC	Interpretacja
1 - 10	kod dość prosty stwarzający nieznaczne ryzyko
11 - 20	kod złożony powodujący ryzyko na średnim poziomie
21 - 50	kod bardzo złożony związany z wysokim ryzykiem
> 50	kod niestabilny grożący bardzo wysokim poziomem ryzyka



$$CC = E - N + p = 4 - 2 + 1 = 3$$

wierzchołki – romby

krawędzie – kreski wychodzące z rombów



Techniki białoskrzynkowe

Test pokrycia instrukcji - nazywany C0-Test, jest minimalnym testem spośród testów pokrycia. Podczas testu każda instrukcja z grafu przepływu sterowania jest wykonywana co najmniej raz.

Testowanie decyzyjne (rozgałęzień): zwany C1-Test, wykonuje co najmniej raz każdą krawędź (gałąź) w grafie przepływu sterowania. Przy tym każda decyzja w rozgałęzieniach w grafie osiąga przynajmniej raz wartość "true" i przynajmniej raz "false". Test pokrycia rozgałęzień zawiera całkowicie test pokrycia instrukcji programu.

- testowanie warunku rozgałęzienia – przypadki testowe zaprojektowane są tak, aby sprawdzić wyniki warunku decyzji,
- testowanie kombinacji warunków w decyzjach – przypadki testowe są tak zaprojektowane, aby wykonywać różne kombinacje warunków w decyzjach,
- testowanie zmodyfikowanych warunków w decyzji – przypadki testowe są tak zaprojektowane, aby sprawdzać wartości warunków, które niezależnie od siebie wpływają na wynik decyzji

Testowanie LSKiS – przypadki testowe są projektowane tak, aby wykonywały LSKiS. Liniowa Sekwencja Kodu i Skok składająca się z trzech następujących punktów: rozpoczęcie liniowej sekwencji wykonywanych instrukcji, koniec sekwencji liniowej i docelowa linia, do której wykonywanie programu jest przekazywane po zakończeniu liniowej sekwencji



Techniki białoskrzynkowe

1. if (a > b)
2. printf („a większe od b”);
3. else
4. printf („a jest niewiększe od b”);
5. if (a == 5)
6. printf („a równa się 5”);
7. printf („Koniec”);

TC1: a = 5; b = 2

TC2: a = 5; b = 7

Pokrycie instrukcji: TC1 + TC2 = 100%

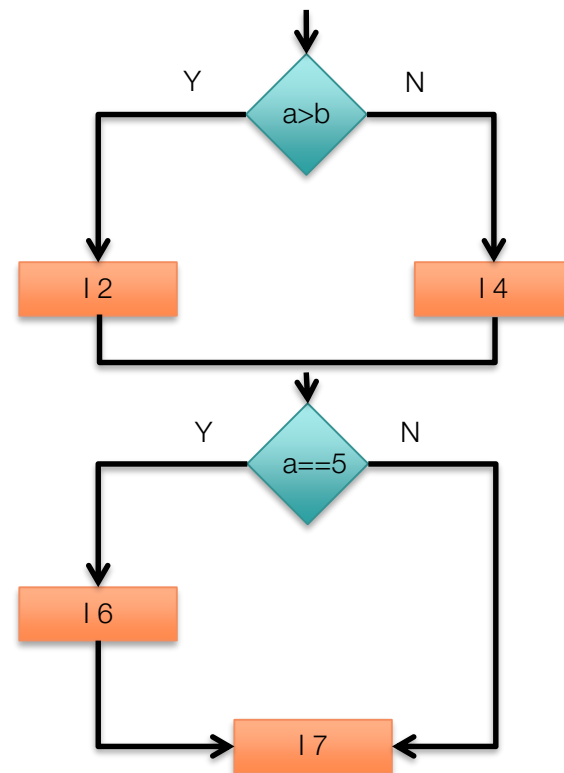
Pokrycie decyzji: TC1 + TC2 = 75%

$$P_i \leq P_d \leq CC$$

P_i – pokrycie instrukcji

P_d – pokrycie decyzji

CC – indeks Mc Cabe





Techniki białoskrzynkowe

```
1. z=2;  
2. if (x==100)  
3. {printf(„x=100”);}  
4. else  
5. {z=5;}  
6. if (z==2) {  
7. printf („z=2”);  
8. if (y==7)  
9. printf („y=7”);}  
10.else  
11. printf („z różne od 2”);
```

100% pokrycie instrukcji:

TC1: x = 100; y = 7

TC2: x = 1; y = 7

Minimalna ilość przypadków
testowych: **2**

100% pokrycie decyzji:

TC1: x = 100; y = 7

TC2: x = 100; y = 5

TC3: x = 1; y = 5

Minimalna ilość przypadków
testowych: **3**



Techniki białoskrzynkowe

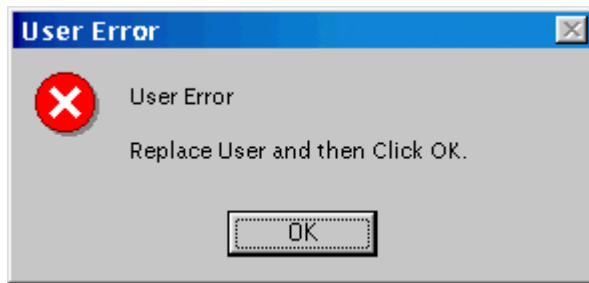
```
1 #include <stdlib.h>
2 #include <string.h>
3 #include <math.h>
4
5 #define MAXCOLUMNS 26
6 #define MAXROW 20
7 #define MAXCOUNT 90
8 #define ITERATIONS 750
9
10 int main (void)
11 {
12     int count = 0, totals[MAXCOLUMNS], val = 0;
13
14     memset (totals, 0, MAXCOLUMNS * sizeof(int));
15
16     count = 0;
17     while ( count < ITERATIONS )
18     {
19         val = abs(rand()) % MAXCOLUMNS;
20         totals[val] += 1;
21         if ( totals[val] > MAXCOUNT )
22         {
23             totals[val] = MAXCOUNT;
24         }
25         count++;
26     }
27
28     return (0);
29
30 }
```

LSKiS	Rozpoczęcie liniowej sekwencji	Wykonanie instrukcji	Skok
1	10	17	28
2	10	21	25
3	10	26	17
4	17	17	28
5	17	21	25
6	17	26	17
7	25	26	17



Techniki oparte o incydent

- Technika tworzenia testów, która wykorzystują wiedzę na temat zgłoszonych, znanych defektów testowanego systemu.
- Może być stosowana w projektach które posiadają historię zgłoszonych incydentów.
- Technika nazywana jest taksonomią (taxonomies), czyli listą potencjalnych defektów o odpowiednich kategoriach.
- Błędy w oprogramowaniu lubią tłum i zwykle są występować w podobnych sytuacjach:
 - oprogramowanie tworzone jest przez ludzi,
 - ludzie narażeni są na monotonię swoich zadań,
 - często popełniają te same pomyłki,
 - mają tendencje do zachowania swoich nawyków...





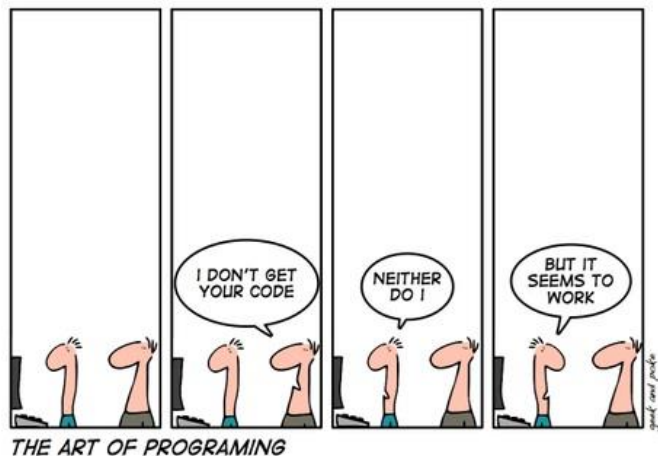
Techniki oparte o doświadczenie

- W tworzeniu przypadków testowych wykorzystywana jest/są:
 - umiejętności,
 - intuicji,
 - doświadczenia testerów z podobnymi aplikacjami i technologiami,
 - wiedza teoretyczna
 - znajomości zespołu wytwórczego
- priorytetyzowanie obszary do testów:
 - wiemy jak bardzo szczegółowe powinny być scenariusze testowe,
 - jakich danych testowych używać,
 - jaką dokumentacją się wspierać,
 - jak szukać przyczyny znalezionych defektów.
- Do technik testowania opartego o doświadczenie wyróżniamy m.in.
 - zgadywanie błędów (error guessing)
 - lista zagadnień testowych (checklist – based)
 - testy eksploracyjne (exploratory testin)
 - ataki (attacks)





ZADANIA



- Zadanie 1
- Zadanie 2
- Zadanie 3
- Zadanie 4
- Zadanie 5



Zadanie 1

- Pompa benzynowa pokazuje cenę paliwa w zakresie od 3,00 zł do 5,99 zł włącznie. Które z poniższych wartości wejściowych będą najlepsze do projektowania przypadków testowych w oparciu o analizę przypadków testowych z poprawnymi przypadkami granicznymi?
- Przewidywalna liczba użytkowników e-banku w ciągu 2 lat od wdrożenia systemu jest pomiędzy 100 000 włącznie a 500 000 wyłącznie. Które z poniższych wartości wejściowych będą najlepsze do projektowania przypadków testowych o oparciu podział na klasy równoważności dla przypadków poprawnych oraz analizę przypadków testowych z poprawnymi granicznymi?



Zadanie 2

	A	B	C	D	E	F
S0	S1	S2				
S1		S2				
S2	S0		S3		S1	
S3				SE		S3
SE						

1. Narysuj graf przejść między stanami

2. Który ze stanów jest nieprawidłowy.

- a) E ze stanu S2
- b) F ze stanu S3
- c) B ze stanu S1
- d) E ze stanu S3



Zadanie 3

Zasada 1		Zasada 2	Zasada 3	Zasada 4
Warunki				
Temp > 38.5	False	True	True	True
Czasowe utraty przytomności	False	True	False	True
Wymioty	True	True	False	False
Akcje				
Szpital	False	True	False	False
Leczenie ambulatoryjne	True	False	True	False

1. Sporządzić graf przyczynowo-skutkowy

2. Jakie będą rezultaty przypadków testowych:

TC1: Chory z wysoką gorączką i wymiotami

TC2: Chory z wymiotami i czasową utratą przytomności



Zadanie 4

```
1. x=100;
2. if (y>100)
3. {printf(„y większe od x”);}
4. else
5. {x=120;}
6. If (x==100) {
7. printf(„ x równa się 100”);
8. If (z==0)
9. {printf(„z równa się 0”);}
10. }
11. else
12. {printf(„x różne od 100”);}
13. printf(„Koniec”)
```

```
1. Narysować graf
2. Wyznaczyć minimalną liczbę
przypadków testowych dla pokrycia
instrukcji
3. Wyznaczyć minimalną liczbę
przypadków testowych dla pokrycia
decyzji
```



Zadanie 5

Specyfikacja:

Zapytaj: „Czy kupuje Pan bilet w jedną stronę czy powrotny?”

Jeżeli użytkownik odpowie „Powrotny”

To zapytaj: „normalny czy zniżkowy?”

Jeżeli odpowiedź użytkownika to „Zniżkowy”

To stwierdź: „Bilety kosztują 48,50”

W przeciwnym przypadku stwierdź: „Bilet kosztuje 85,40”

W przeciwnym przypadku stwierdź: „Bilet kosztuje 32,80”

Ile przypadków testowych należy przygotować, by w pełni sprawdzić powyższe wymagania?



Thank you