Tester Oprogramowania


E
L
C
O
M
E

O mnie

Marek Waszak

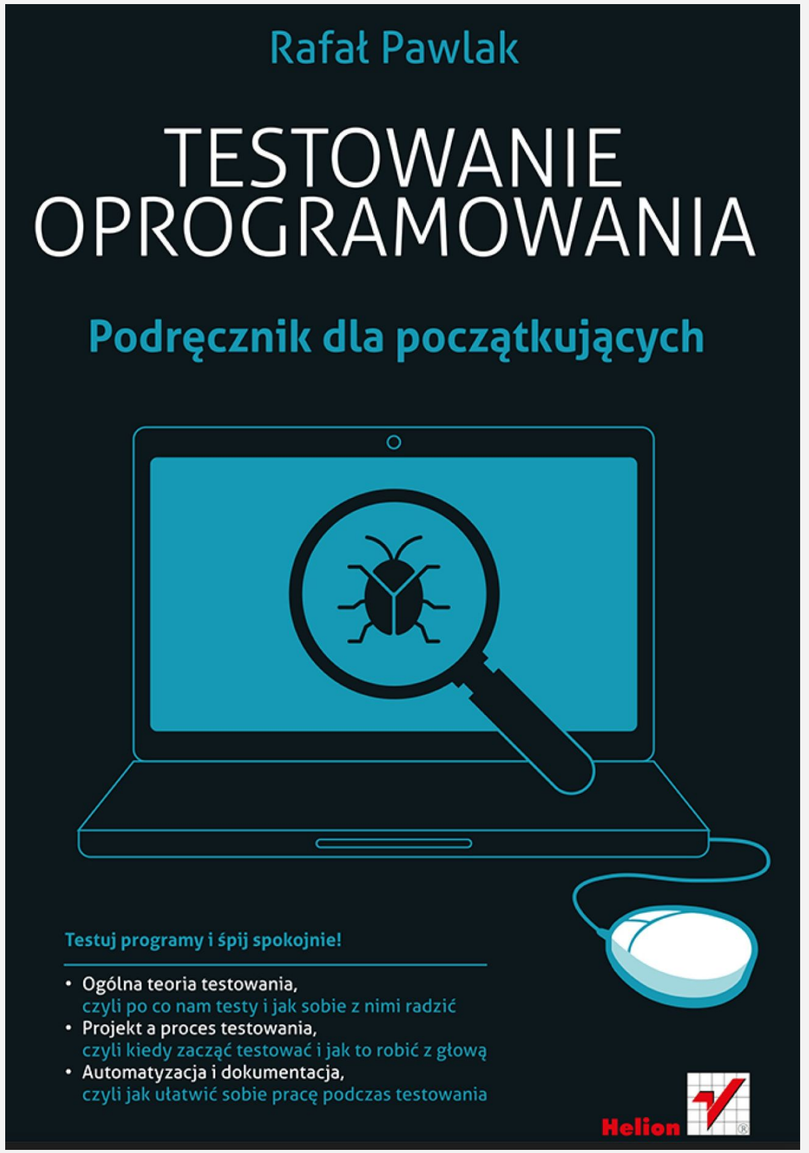
- **Test Automation Architect** w firmie **Huuuge Games**
- Od **2013 r** w procesie testowania oprogramowania
- **Mnóstwo** projektów i **tworzenia** procesów jakościowych
- Trener **od 2016 r**
- Zajęć szkoleniowych: **ponad 700 godzin**
- **Zainteresowania**: piłka nożna, siłownia, motoryzacja

Materiały uzupełniające

1. **Testowanie Oprogramowania - Podręcznik dla początkujących**
   Rafał Pawlak

2. **ISTQB FL - Poziom podstawowy**
   wersja 3.1.3

3. **Słownik Testerski - Poziom podstawowy**
   wersja 3.1



Program szkolenia

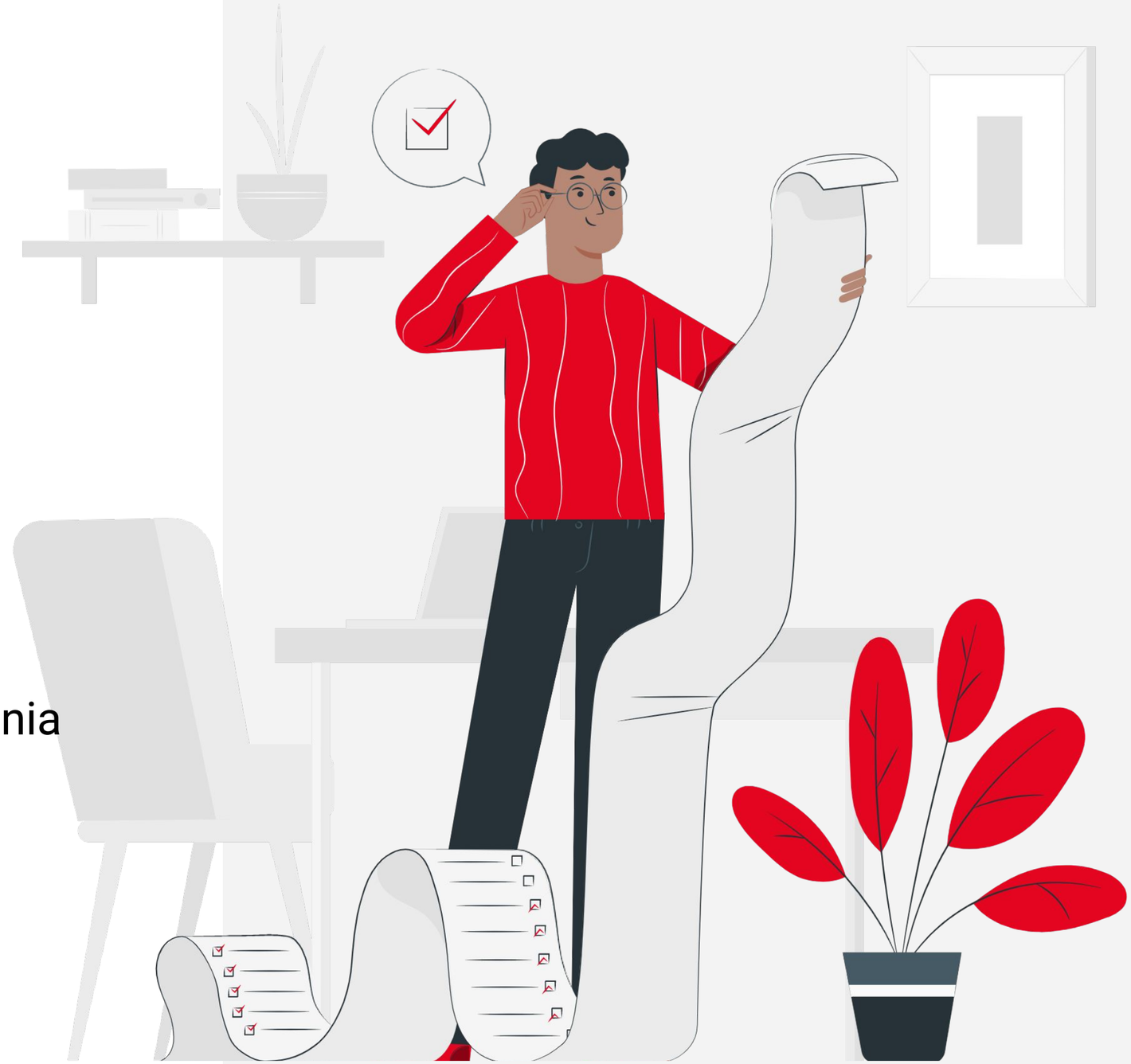
01 Definicje testowania

02 Zasady testowania

03 Podział testowania

04 Modele wytwarzania oprogramowania

05 Techniki projektowania testów



Program szkolenia

06 Statyczne techniki testowania

:

07 Testy potwierdzające

:

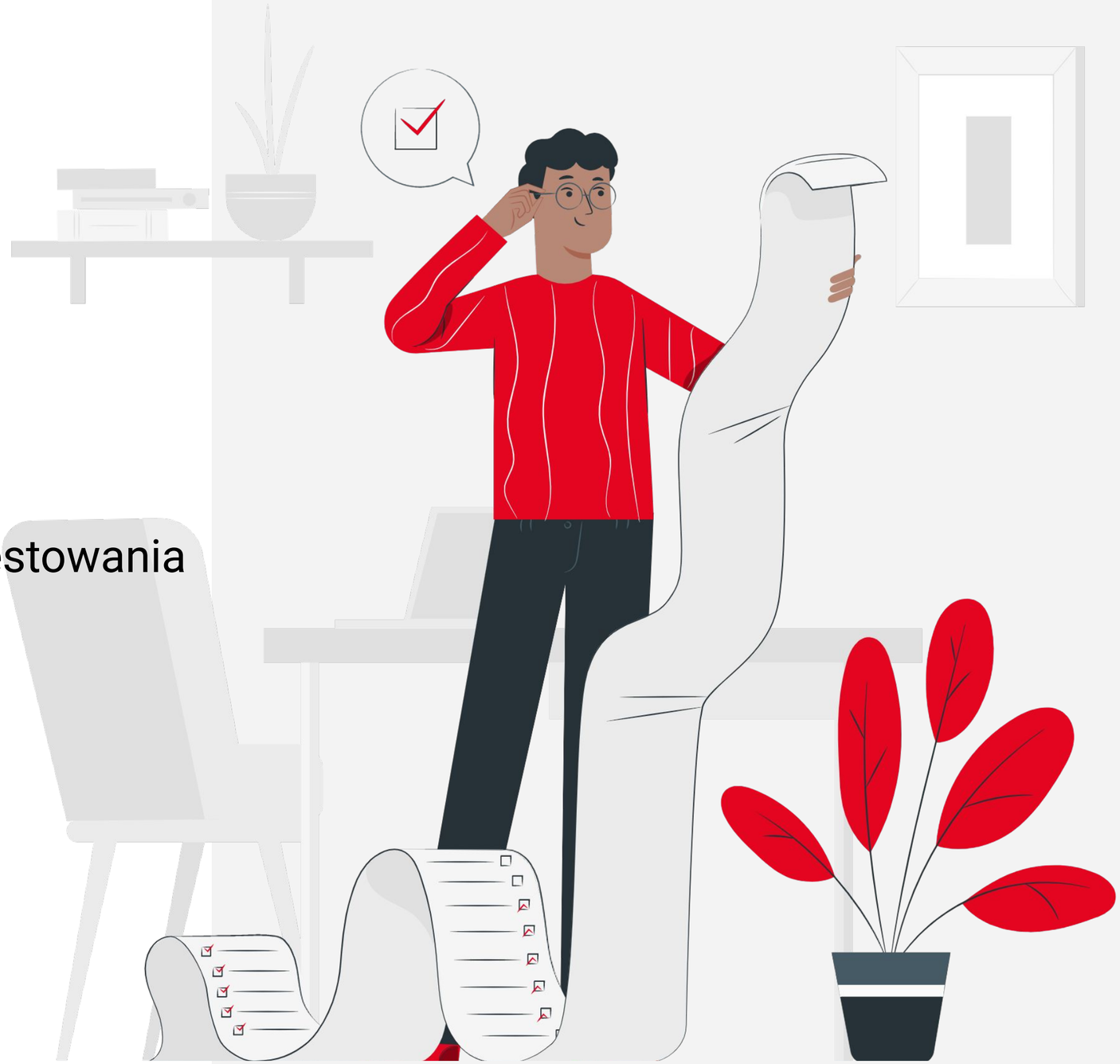
08 Narzędzia wspomagające proces testowania

:

09 Współpraca z programistami

:

10 Weryfikacja / Walidacja



Program szkolenia

11 Organizacja testowania

:

12 Planowanie testowania

:

13 Strategia testowania

:

14 Monitorowanie testowania

:

15 Kierowanie testami

01
Definicje testowania

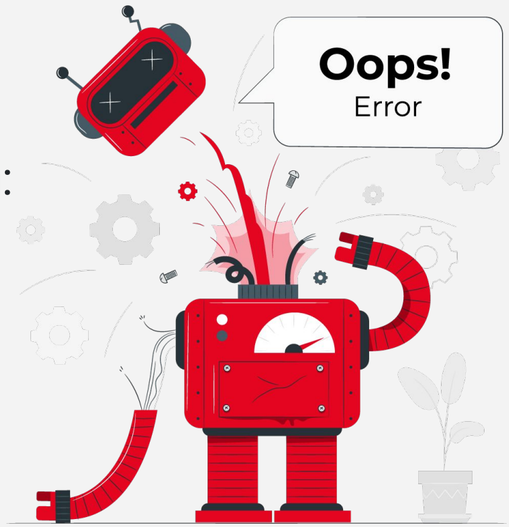
#01# Definicje testowania

Dlaczego testowanie jest niezbędne?

Systemy oprogramowanie są nieodłączną częścią naszego życia we wszystkich jego obszarach - od aplikacji biznesowych (np. bankowości) po produkty użytkowe (np. samochody).

Jednocześnie większość z nas miała zapewne do czynienia z oprogramowaniem, które...

nie zadziałało tak, jak powinno.



Oprogramowanie, które działa **niepoprawnie** może powodować wiele **problemów**, w tym:

- straty finansowe,
- utratę czasu,
- utratę reputacji firmy,
- a nawet utratę zdrowia lub życia.

#01# Definicje testowania

Dlaczego testowanie jest niezbędne?

Człowiek może popełnić **błąd** (pomyłkę), która powoduje powstanie **defektu** (usterki, pluskwy) w kodzie programu lub dokumencie.

Jeśli kod zawierający **defekt** zostanie wykonany, system nie zrobi tego co powinien (lub wykona to czego nie powinien) powodując **awarię**.

Usterki w oprogramowaniu, systemach lub dokumentach **mogą prowadzić do** wystąpienia **awarii**, **ale nie wszystkie usterki powodują awarię.**

#01# Definicje testowania

Dlaczego testowanie jest niezbędne?

Pomyłki (defekty) istnieją z wielu powodów, takich jak:

- presja czasu,
- omylność człowieka,
- brak doświadczenia lub niedostateczne umiejętności uczestników projektu,
- problemy z komunikacją w zespole developerskim (wymagania, dokumentacja - złe lub ich brak),
- złożoność kodu, architektury,
- zmieniające się technologie, stosowanie nowych/nieznanych technologii,
- duża ilość interakcji pomiędzy systemami.

Awarie (poza tymi występującymi w kodzie) mogą powstać na skutek, np. **warunków atmosferycznych**:

- promieniowanie lub pole elektromagnetyczne,
- zanieczyszczenia.

#01# Definicje testowania

Testowanie a jakość

Za pomocą **testów** można **zmierzyć jakość** oprogramowania **wyrażoną** przez **ilość znalezionych usterek** zarówno dla funkcjonalnych jak i niefunkcjonalnych wymagań i atrybutów oprogramowania (np. niezawodność, użyteczność).

Testowanie **może budować zaufanie** do jakości oprogramowania jeżeli:

- osoby testujące znajdują **mało usterek,**
- osoby testujące **nie znajdują usterek**.

Testowanie wykrywa usterki, ale jakość systemu podnosi się wraz z ich naprawieniem.

SAMO TESTOWANIE NIE PODNOSI JAKOŚCI OPROGRAMOWANIA I DOKUMENTACJI!

#01# Definicje testowania

Jak dużo testowania jest potrzebne?

Podczas podejmowania decyzji jak dużo testów należy wykonać, powinno się wziąć pod uwagę **poziom ryzyka**:

- technicznego,
- związanego z bezpieczeństwem,
- biznesowego.

Ważne są także **ograniczenia projektowe** takie jak:

- czas,
- budżet.

#01# Definicje testowania

Jak dużo testowania jest potrzebne?

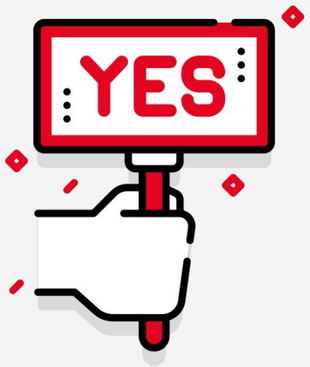
Docelowo **testowanie powinno dostarczać** interesariuszom **informacji** wystarczających do podjęcia świadomych **decyzji o dopuszczeniu** testowanego **oprogramowania** lub **systemu** do **następnej fazy rozwoju** lub **przekazania go klientowi**.

#01# Definicje testowania

Co to jest testowanie?

Za testowanie najczęściej uważa się **tylko** wykonywanie testów...

Wykonywanie testów stanowi tylko **część** testowania!
Istnieją jeszcze inne czynności związane z testowaniem.
Czynności te występują zarówno przed jak i po wykonywaniu testów.



#01# Definicje testowania

Co to jest testowanie?

Testowanie to:

- Planowanie testów
- Nadzór testów
- Wybór warunków testowych
- Projektowanie przypadków testowych
- Wykonywanie warunków testowych
- Sprawdzanie wyników testów
- Ocena spełnienia kryteriów zakończenia
- Raportowanie procesu testowania i testowanego systemu
- Zakończenie i zamykanie testów (po zakończeniu fazy testów)

#01# Definicje testowania

Co to jest testowanie?

Do testowania zalicza się także:

- przeglądy dokumentacji,
- przeglądy kodu źródłowego,
- analizę statyczną.

Przypominając powyższe czynności (testowanie) pozwala na:

- **ocenę jakość** danego oprogramowania,
- **zmniejszenie ryzyka** wystąpienia **awarii** podczas eksploatacji.

#01# Definicje testowania

Cele testowania

Do **celów** testowania zaliczają się:

- znajdowanie usterek,
- nabieranie zaufania do poziomu jakości,
- dostarczanie informacji potrzebnych do podejmowania decyzji,
- zapobieganie defektom.

#01# Definicje testowania

Cele testowania - w poszczególnych typach testów

Testowanie wytwórcze (testy jednostkowe, integracyjne, systemowe)

Głównym celem jest **wywołanie** tylu **awarii** ile się da, żeby zidentyfikować i naprawić usterki występujące w oprogramowaniu.

Testowanie pielęgnacyjne - zawiera testy sprawdzające - czy nie wprowadzono nowych usterek podczas wykonywania zmian (**testy regresji**).

Testy akceptacji (testy akceptacyjne)

Głównym celem jest **potwierdzenie**, że **system działa** jak powinien oraz nabieranie pewności, że spełnia wymagania.

#01# Definicje testowania

Cele testowania

W niektórych przypadkach **celem** testowania może być
ocena jakości oprogramowania - **bez intencji naprawiania defektów**.

↓

Takie testowanie **dostarcza** interesariuszom **informacji**
o **ryzyku** związanym z **wydaniem** systemu w danej chwili.

#01# Definicje testowania

Debugowanie vs Testowanie

Testowanie - może pokazać awarie, których źródłem są usterki.

Debugowanie - czynność programistyczna, która znajduje, analizuje i umożliwia usunięcie przyczyny awarii.

Odpowiedzialność, za każdą z tych czynności jest zwykle inna, tj.

Testerzy -> Testują

Programiści -> Debugują

Debugowanie różni się od testowania!

#01# Definicje testowania

Podstawowy proces testowy

Najbardziej **widocznym** elementem testowania jest **wykonywanie** testów.

Jednak, żeby testy były skuteczne i efektywne, **plany testów** powinny uwzględniać:

- czas potrzebny na zaplanowanie testów,
- zaprojektowanie przypadków testowych,
- przygotowanie do ich wykonania,
- ocenę wyników,

#01# Definicje testowania

Podstawowy proces testowy

Podstawowy proces testowy składa się z:

- **planowanie i nadzór nad testami**

Planowanie testów polega na zdefiniowaniu celów testowania i określeniu czynności testowych potrzebnych do wypełnienia misji i celów testowania.

#01# Definicje testowania

Podstawowy proces testowy

Podstawowy proces testowy składa się z:

- **analiza i projektowanie testów**

Podczas analizy i projektowania testów **ogólne cele** testowania **są przekształcane** w:

- konkretne warunki testowe,
- przypadki testowe.

#01# Definicje testowania

Podstawowy proces testowy

Podstawowy proces testowy składa się z:

- **implementacja i wykonywanie testów**

Czynności, podczas których:

- specyfikowane są procedury,
- tworzone są skrypty testowe,
- konfiguracja środowiska,
- dołączenie innych informacji potrzebnych do wykonania testów,
- wykonywanie testów.

#01# Definicje testowania

Podstawowy proces testowy

Podstawowy proces testowy składa się z:

- **ocena spełnienia kryteriów zakończenia i raportowanie**

Ocena spełnienia kryteriów zakończenia polega na **ocenie wykonania testów** zgodnie z przyjętymi **celami** testowania.

Ocena powinna być wykonywana **dla każdego poziomu testowania**.

#01# Definicje testowania

Podstawowy proces testowy

Podstawowy proces testowy składa się z:

- **czynności zamykające testy**

W skład tych czynności wchodzą:

- zbieranie danych z zakończonych czynności,
- testalia oraz metryki.

Czynności te wykonywane są przy kamieniach milowych projektu:

- wydanie systemu,
- osiągnięcie kamienia milowego,
- zakończenie lub anulowanie projektu testowego.

02
Zasady testowania

#02# Zasady testowania

7 zasad testowania oprogramowania

Zasada 1 - Testowanie ujawnia usterki

Zasada 2 - Testowanie gruntowne jest niewykonalne

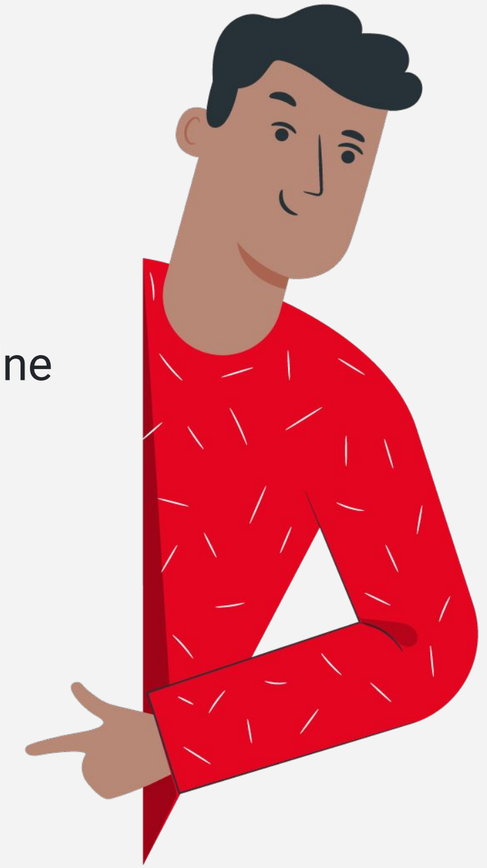
Zasada 3 - Wczesne testowanie

Zasada 4 - Kumulowanie się błędów

Zasada 5 - Paradoks pestycydów

Zasada 6 - Testowanie zależy od kontekstu

Zasada 7 - Mylne przekonanie o braku błędów



#02# Zasady testowania

Zasada 1 - Testowanie ujawnia usterki

Testowanie **może pokazać**, że istnieją **usterki** - ale **nie może dowieść**, że oprogramowanie **nie posiada** defektów.

Testowanie **zmniejsza prawdopodobieństwo** występowania w oprogramowaniu **niewykrytych defektów**, ale nawet jeżeli **nie zostały znalezione** żadne usterki, to **nie stanowi** to dowodu o **poprawności** oprogramowania.

#02# Zasady testowania

Zasada 2 - Testowanie gruntowne jest niewykonalne

Przetestowanie **wszystkiego** (wszystkich kombinacji wejść i warunków początkowych) jest **wykonalne** tylko **w trywialnych przypadkach**.

Zamiast testowania gruntownego, do kierowania testami należy wykorzystać:

- analizę ryzyka,
- priorytetyzację

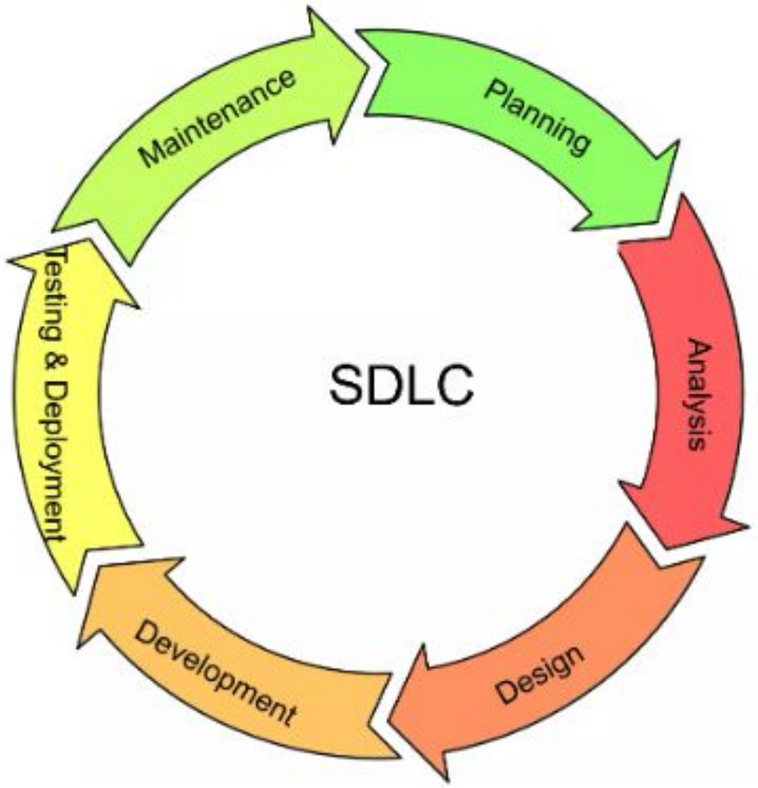
#02# Zasady testowania

Zasada 3 - Wczesne testowanie

Żeby wykryć usterki **jak najwcześniej**, testowanie:

- rozpoczyna się tak wcześnie jak to tylko możliwe (w cyklu rozwoju oprogramowania - **SDLC**),
- jest nakierowane na spełnienie zdefiniowanych celów.

#02# Zasady testowania



SDLC (Software Development Life Cycle)

#02# Zasady testowania

Zasada 4 - Kumulowanie się błędów

Większość **defektów** lub **awarii** wykrytych podczas testowania przed przekazaniem oprogramowania do użycia **występuje** lub ma swoje źródło w **niewielkiej liczbie modułów**.

Skupiska defektów zaobserwowane na etapie testowania lub używania oprogramowania są **ważnym elementem analizy ryzyka**.

#02# Zasady testowania

Zasada 5 - Paradoks pestycydów

Jeżeli **te same testy** są **powtarzane bez zmian**, to w końcu **przestaną znajdować nowe usterki**.

Aby zapobiec i przezwyciężyć “paradoks pestycydów”, testy muszą być regularnie:

- przeglądane,
- uaktualniane.

#02# Zasady testowania

Zasada 6 - Testowanie zależy od kontekstu

Testowanie wykonuje się:

- w różny sposób
- w różnym kontekście.

Oprogramowanie krytyczne ze względu na bezpieczeństwo testuje się inaczej niż sklep internetowy.

#02# Zasady testowania

Zasada 7 - Mylne przekonanie o braku błędów

Znajdowanie i naprawa usterek **nie pomoże**, jeżeli produkowany system:

- nie nadaje się do użytkowania,
- nie spełnia wymagań i oczekiwań użytkownika.

03
Podział testowania

#03# Podział testowania

Poziomy testów

Dla każdego poziomu testowania można zdefiniować:

- ogólne cele testowania,
- produkty, na podstawie których tworzy się przypadki testowe,
- przedmiot testów (to co jest testowane),
- typowe defekty i awarie do wykrycia,
- wsparcie narzędziowe,
- środowisko testowe,
- specyficzne podejście,
- odpowiedzialność.

#03# Podział testowania

Testy modułowe (jednostkowe)

Polegają na wyszukiwaniu błędów i weryfikacji funkcjonalności które można testować oddzielnie:

- moduły,
- programy,
- obiekty,
- klasy.

Mogą być wykonywane w izolacji od reszty systemu.

Można podczas nich użyć: **zaślepek**, **sterowników** testowych, **symulatorów**.

#03# Podział testowania

Testy integracyjne

Polegają na sprawdzeniu interfejsów pomiędzy modułami (lub systemami) z innymi częściami systemu:

- system operacyjny,
- system plików,
- sprzęt.

#03# Podział testowania

Testy systemowe

Polegają na sprawdzeniu jak zachowuje się system/produkt.

Zakres takich testów powinien być jasno określony w głównym planie testów.

Podczas wykonywania takich testów, środowisko testowe powinno być w jak największym stopniu **odwzorowywać środowisko docelowe (produkcyjne)**.

#03# Podział testowania

Testy akceptacyjne

Polegają na **nabraniu zaufania** do systemu, jego części lub pewnych atrybutów niefunkcjonalnych.

Wyszukiwanie usterek nie jest tym, na czym skupiają się testy akceptacyjne.

Mogą one **oceniać gotowość systemu** do wdrożenia i użycia.

Odpowiedzialność za testy akceptacyjne często leży po stronie **klienta** lub **użytkowników** systemu.

#03# Podział testowania

Typy testów

Typ testów to **grupa dynamicznych czynności** testowych **wykonywanych z myślą** o przetestowaniu określonych **charakterystyk** systemu/oprogramowania zgodnie z **określonymi celami** testów.

Celami mogą być:

- ocena **funkcjonalnych** charakterystyk jakościowych (kompletność, prawidłowość)
- ocena **niefunkcjonalnych** charakterystyk jakościowych (niezawodność, wydajność, kompatybilność)
- dokonanie oceny **skutków zmian** (np. po naprawie defektów)

#03# Podział testowania

Testowanie funkcji (testowanie funkcjonalne)

Funkcje (jakie ma pełnić system, podsystem, moduł) są opisane w:

- specyfikacji wymagań,
- przypadkach użycia,
- specyfikacji funkcjonalnej,
- nieudukomentowane.

Funkcje są tym **“co”** system robi.

Testy funkcjonalne dotyczą funkcji oraz ich współdziałania z innymi systemami.

Można je wykonywać na **wszystkich poziomach** (np. testy modułowe mogą bazować na specyfikacji modułów)

#03# Podział testowania

Testowanie funkcji (testowanie funkcjonalne)

Testy funkcjonalne zajmują się zewnętrznym zachowaniem oprogramowania **(testy czarnoskrzynkowe).**

Przykłady typów testów funkcjonalnych:

Testowanie zabezpieczeń - sprawdzanie funkcji (np. zapory) pozwalające na wykrycie zagrożeń, takich jak wirusy.

Testowanie współdziałania - ocenia zdolność oprogramowania do współpracy z jednym lub większą liczbą wskazanych modułów lub systemów.

#03# Podział testowania

Testowanie atrybutów niefunkcjonalnych (testowanie niefunkcjonalne)

Testowanie niefunkcjonalne m.in. obejmuje:

- testowanie wydajnościowe,
- testowanie obciążeniowe,
- testowanie użyteczności,
- testowanie niezawodności.

Testowanie niefunkcjonalne polega na sprawdzeniu **“jak”** system działa.

04
Modele wytwarzania oprogramowania

#04# Modele wytwarzania oprogramowania

Model V (model sekwencyjny)

Najczęściej spotykany model V posiada:

4 poziomy testowania,

którym odpowiadają

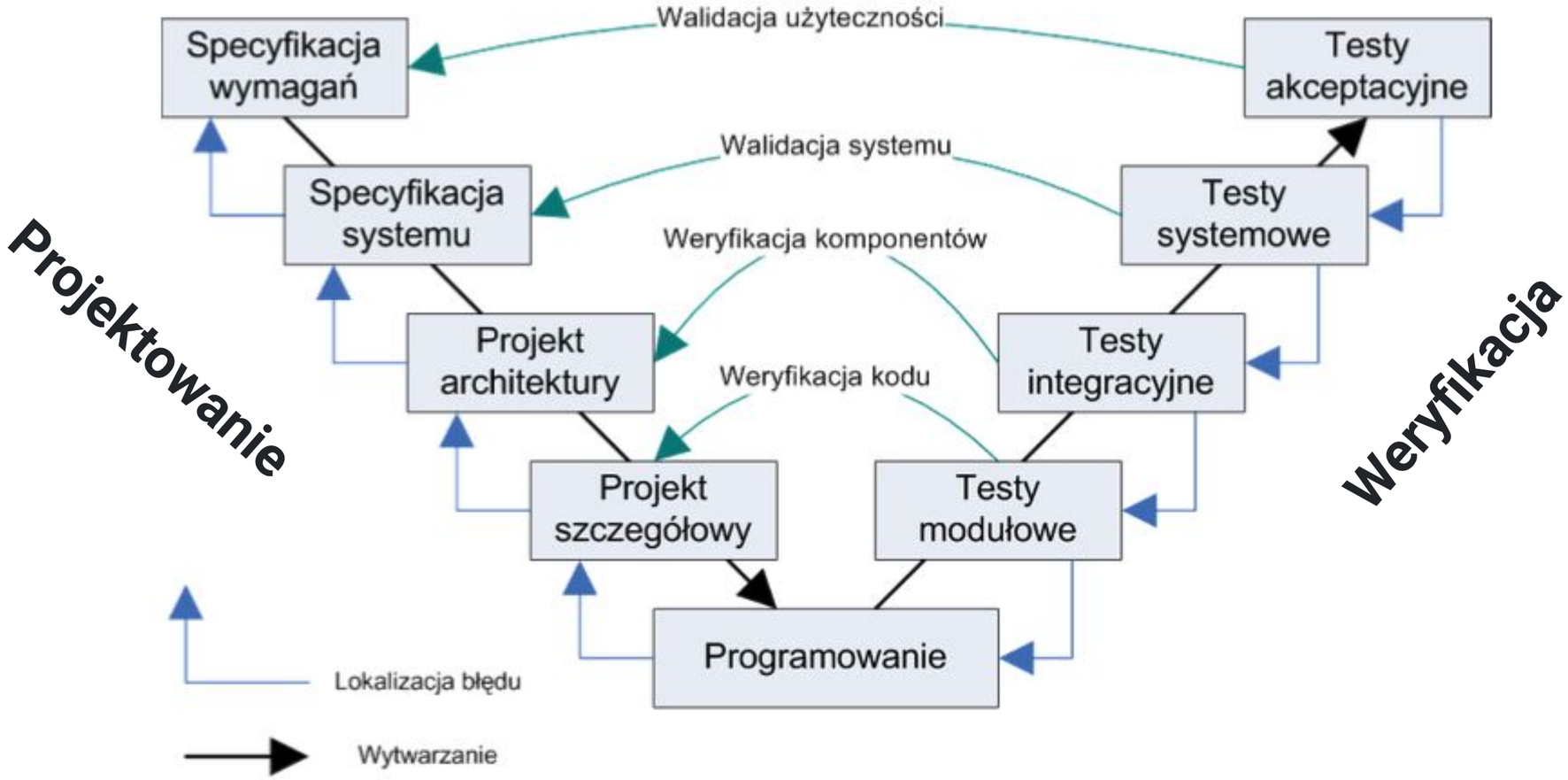
4 poziomy rozwoju oprogramowania.

4 poziomy testowania:

- testy modułowe (jednostkowe),
- testy integracyjne,
- testy systemowe,
- testy akceptacyjne.

#04# Modele wytwarzania oprogramowania

Model V:



Projektowanie i weryfikacja w modelu V, nie następują po sobie, ale wykonywane są równolegle.

#04# Modele wytwarzania oprogramowania

Model V (model sekwencyjny)

Zalety modelu V:

- precyzyjnie pokazuje **zależności**, jakie powinny łączyć **kolejne etapy projektu** Każdy etap projektowania kończy się stworzeniem danych wejściowych dla następnej fazy oraz do korespondującej z nim fazy weryfikacji.

- zachęca do jak **najwcześniejszego** rozpoczęcia procesu **tworzenia planów testów**, **specyfikacji testowej** i samego **testowania**.

- dzięki analizie grafu możemy łatwo zidentyfikować obszary, gdzie stworzona **dokumentacja** powinna być **sprawdzona**.

#04# Modele wytwarzania oprogramowania

Model iteracyjno - przyrostowy (Agile)

Procesy, takie jak:

- zbierania wymagań,
- projektowanie,
- budowanie,
- testowanie systemu

zorganizowane są w krótsze cykle rozwojowe.

System wyprodukowany według takiego modelu może być przetestowany na kilku poziomach w każdej **iteracji (Sprint)**.

#04# Modele wytwarzania oprogramowania

Agile

#04# Modele wytwarzania oprogramowania

Model iteracyjno - przyrostowy (Agile)

W 2001 roku powstał zbiór wartości i zasad. Zbiór ten znany jest jako:

- **Manifest zwinnego wytwarzania oprogramowania** lub
- **Manifest Agile**.

Manifest Agile zawiera **4 główne zasady:**

1. Ludzie i współpraca ponad procesy i narzędzia
2. Działające oprogramowanie ponad szeroką dokumentację
3. Współpraca z klientem ponad formalne ustalenia
4. Reagowanie na zmiany ponad podążanie za planem

#04# Modele wytwarzania oprogramowania

Model iteracyjno - przyrostowy (Agile)

Podejście **“cały zespół”** w Agile:

- poprawia współpracę i komunikację w zespole,

  (Planning, Refinement, Daily Scrum, Review, Retro)
- czyni wszystkich odpowiedzialnymi za jakość.

W projekcie zwinnym za jakość odpowiada cały zespół!

#04# Modele wytwarzania oprogramowania

Model iteracyjno - przyrostowy (Agile)

Przyrost, dodany do innych wytworzonych wcześniej, staje się **rosnącym systemem częściowym**, który również **powinien być przetestowany**.

Testowanie regresji - bardzo ważne w każdej iteracji oprócz pierwszej.

Każdy przyrost może podlegać zarówno **weryfikacji** jak i **walidacji**.

Przy testach regresji najlepiej sprawdzają się **testy automatyczne**.

#04# Modele wytwarzania oprogramowania

Model iteracyjno - przyrostowy (Agile)

Zalety:

- częsty kontakt z klientem
- brak konieczności zdefiniowania z góry całości dokumentacji
- wczesne wykorzystanie przez klienta fragmentów systemu (funkcjonalności)
- możliwość elastycznego reagowania na opóźnienia realizacji fragmentu (lub przyspieszenie prac nad inną częścią)

#04# Modele wytwarzania oprogramowania

Testowanie w cyklu oprogramowania

W każdym modelu rozwoju oprogramowania **dobre testowanie** posiada **kilka niezmiennych cech**:

- dla **każdej czynności** związanej z wytwarzaniem oprogramowania istnieją **odpowiadające jej czynności** związane z testowaniem,
- **każdy poziom** testowania ma zdefiniowane **cele**,
- **analiza i projektowanie** testów dla danego poziomu powinny **rozpoczynać** się już podczas **odpowiadającej im fazy** wytwarzania,
- **testerzy** powinni **uczestniczyć** w przeglądach już **od wczesnych wersji** dokumentacji tworzonej podczas wytwarzania.

05
Techniki projektowania testów

#05# Techniki projektowania testów

Techniki projektowania testów

Klasyczny **podział** projektowania testów wyróżnia techniki:

- czarnoskrzynkowe,
- białoskrzynkowe,
- oparte na doświadczeniu.

Celem technik projektowania testów jest:

- zdefiniowanie **warunków** testowych,
- zdefiniowanie **przypadków** testowych,
- zdefiniowanie **danych** testowych.

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

Techniki **czarnoskrzynkowe** nazywane są również technikami **opartymi na specyfikacji.**

Są sposobem na wybranie **warunków**, **przypadków** i **danych** testowych bazując na analizie podstawy testów.

Można ich używać **zarówno** w testowaniu **funkcjonalnym** jak i **niefunkcjonalnym**.

Z definicji **nie wykorzystują** żadnych **informacji o strukturze** testowanego modułu lub systemu.

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

1. Podział na klasy równoważności

Testy można tak zaprojektować, żeby pokrywały **akceptowalne** i **nieakceptowalne** klasy równoważności.

W technice podziału na klasy równoważności **wejścia** programu lub systemu są **dzielone na grupy (klasy równoważności)**.

Klasy równoważności można wyznaczyć dla:

- danych **poprawnych** (wartości, które powinny zostać zaakceptowane)
- danych **niepoprawnych** (wartości, które powinny zostać odrzucone)

Podział na klasy równoważności można **zastosować** na **każdym poziomie** testowania.

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

2. Analiza wartości brzegowych

Minimum i **maksimum klasy równoważności** to jej **wartości brzegowe**.

Wartość brzegowa poprawnego przedziału jest nazywana **poprawną wartością brzegową**.

Wartość brzegowa niepoprawnego przedziału jest nazywana **niepoprawną wartością brzegową**.

Testy można zaprojektować tak, aby **pokrywały** zarówno **poprawne** jak i **niepoprawne** wartości brzegowe.

#05# Techniki projektowania testów

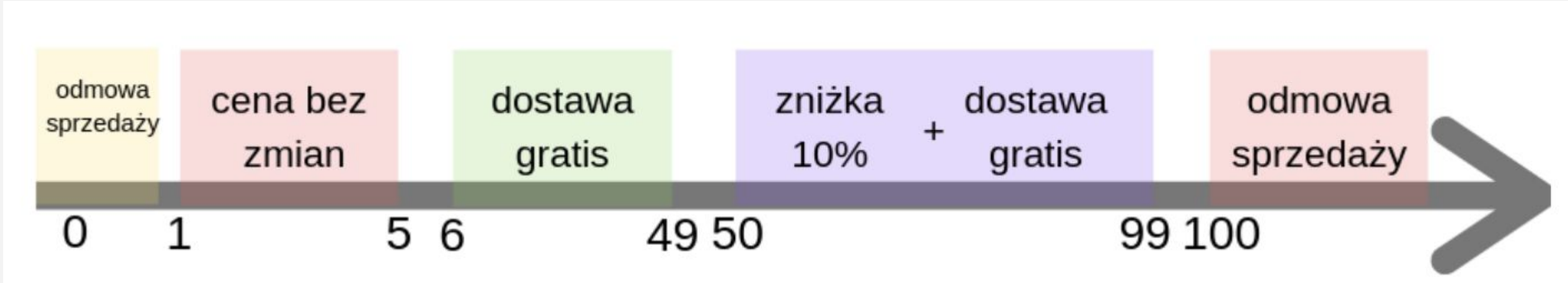
Przykład (zadanie):

Wyznacz **klasy równoważności** i **wartości brzegowe** dla **koszyka sklepu online.**

Treść dokumentacji:

Przy zakupie do 5 sztuk towaru cena nie ulega zmianie. Natomiast przy zakupie powyżej 5 sztuk przysługuje darmowa dostawa, ale jeśli zamówienie jest na co najmniej 50 sztuk, to klient otrzymuje dodatkowo 10% zniżki. Przy tym wszystkich istnieje warunek, że nie można zamówić więcej niż 99 sztuk towaru.

#05# Techniki projektowania testów



Klasy równoważności:

- 0 sztuk - brak zakupu
- <1,6) sztuk - normalna cena
- <6,50) sztuk - darmowa dostawa
- <50,99) sztuk - darmowa dostawa + 10% zniżki
- >99 sztuk - brak możliwości zamówienia towaru.

Otrzymaliśmy **5 klas równoważności.**

Wartości brzegowe:

0, 1, 5, 6, 49, 50, 99, 100

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

3. Testowanie w oparciu o tablicę decyzyjną

Podczas **tworzenia** tablic decyzyjnych **analizuje** się specyfikację systemu i **wyszukuje** się warunki oraz wynikające z nich zachowanie systemu.

Warunki wejściowe często muszą być zapisywane jako **prawda** lub **fałsz**.

Tabele decyzyjne są dobrym sposobem dla systemów, które zawierają **zależności logiczne**.

#05# Techniki projektowania testów

Przykład (zadanie):

Przekształć tablicę decyzyjną w przypadki testowe.

Tytuł: promocja - rabat 40zł za zapis do newslettera.					
Warunki wejścia:	W1	W2	W3	W4	W5
Klient zapisał się na Newsletter	P	P	P	P	F
Klient dostał kupon za zapis do newslettera	P	P	P	F	*
Kupon jest aktywny (nie został wykorzystany wcześniej)	P	P	F	*	*
Produkty w koszyku spełniają warunki promocji	P	F	*	*	*
Akcje					
Zostaje naliczony rabat 40zł	P	F	F	F	F

Tablica decyzyjna przedstawiająca udzielenie Klientowi rabatu za zapisanie do newslettera.

#05# Techniki projektowania testów

Tablica przypadków testowych:

Warunki wejścia:	T1	T2	T3	T4	T5
Klient zapisał się na Newsletter	TAK	TAK	TAK	TAK	NIE
Klient dostał kupon za zapis do Newslettera	TAK	TAK	TAK	NIE	NIE
Kupon jest aktywny (nie został wykorzystany wcześniej)	TAK	TAK	NIE	NIE	NIE
Produkty w koszyku spełniają warunki promocji	TAK	NIE	NIE	NIE	NIE
Akcje					
Zostaje naliczony rabat 40zł	TAK	NIE	NIE	NIE	NIE

#05# Techniki projektowania testów

Przypadek testowy (T1)

ID: TC-1

Tytuł: Rabat 40 zł za zapis do newslettera - przypadek T1

Warunki wejściowe:

Klient zapisał się do newslettera [PRAWDA]

Klient dostał kupon za zapis do newslettera [PRAWDA]

Kupon jest aktywny (nie został wykorzystany wcześniej) [PRAWDA]

Produkty w koszyku spełniają warunki promocji [PRAWDA]

Warunki wyjściowe:

Zostaje naliczony rabat 40 zł [PRAWDA]

#05# Techniki projektowania testów

Przypadek testowy (T1)

Oczekiwany rezultat: Rabat za zapis do newslettera w wysokości 40 zł zostaje prawidłowo dodany, wartość koszyka zmniejsza się.

Kroki do wykonania:

1. Zapisz się na newsletter.
2. Odbierz kod rabatowy przesłany na skrzynkę e-mail podaną przy zapisie na newsletter.
3. Dodaj produkty do koszyka spełniające warunki promocji (link do regulaminu promocji).
4. Wejdź w koszyk.
5. W polu [kod rabatowy] wpisz numer otrzymanego kuponu.

Warunek końcowy:

Rabat w wysokości 40 zł zostaje prawidłowo odjęty od wartości koszyka.

#05# Techniki projektowania testów

Ćwiczenia: Na podstawie tablicy przypadków testowych rozpisz **pozostałe** przypadki testowe.

Warunki wejścia:	T1	T2	T3	T4	T5
Klient zapisał się na Newsletter	TAK	TAK	TAK	TAK	NIE
Klient dostał kupon za zapis do Newslettera	TAK	TAK	TAK	NIE	NIE
Kupon jest aktywny (nie został wykorzystany wcześniej)	TAK	TAK	NIE	NIE	NIE
Produkty w koszyku spełniają warunki promocji	TAK	NIE	NIE	NIE	NIE
Akcje					
Zostaje naliczony rabat 40zł	TAK	NIE	NIE	NIE	NIE

#05# Techniki projektowania testów

Przypadek testowy (T2)

ID: TC-2

Tytuł: Rabat 40 zł za zapis do newslettera - przypadek T2

Warunki wejściowe:

Klient zapisał się do newslettera [PRAWDA]

Klient dostał kupon za zapis do newslettera [PRAWDA]

Kupon jest aktywny (nie został wykorzystany wcześniej) [PRAWDA]

Produkty w koszyku spełniają warunki promocji [FAŁSZ]

Warunki wyjściowe:

Zostaje naliczony rabat 40 zł [FAŁSZ]

#05# Techniki projektowania testów

Przypadek testowy (T2)

Oczekiwany rezultat: Rabat za zapis do newslettera w wysokości 40 zł **NIE** zostaje dodany, wartość koszyka **NIE** zmienia się.

Kroki do wykonania:

1. Zapisz się na newsletter.
2. Odbierz kod rabatowy przesłany na skrzynkę e-mail podaną przy zapisie na newsletter.
3. Dodaj produkty do koszyka, które **NIE** spełniają warunków promocji (link do regulaminu promocji).
4. Wejdź w koszyk.
5. W polu [kod rabatowy] wpisz numer otrzymanego kuponu.

Warunek końcowy:

Rabat w wysokości 40 zł **NIE** zostaje naliczony.

#05# Techniki projektowania testów

Przypadek testowy (T3)

ID: TC-3

Tytuł: Rabat 40 zł za zapis do newslettera - przypadek T3

Warunki wejściowe:

Klient zapisał się do newslettera [PRAWDA]

Klient dostał kupon za zapis do newslettera [PRAWDA]

Kupon jest aktywny (nie został wykorzystany wcześniej) [FAŁSZ]

Produkty w koszyku spełniają warunki promocji [NIEISTOTNE]

Warunki wyjściowe:

Zostaje naliczony rabat 40 zł [FAŁSZ]

#05# Techniki projektowania testów

Przypadek testowy (T3)

Oczekiwany rezultat: Rabat za zapis do newslettera w wysokości 40 zł **NIE** zostaje dodany, wartość koszyka **NIE** zmienia się.

Kroki do wykonania:

1. Zapisz się na newsletter.
2. Odbierz kod rabatowy przesłany na skrzynkę e-mail podaną przy zapisie na newsletter.
3. Dodaj produkty do koszyka, które spełniają warunki promocji (link do regulaminu promocji).
4. Wejdź w koszyk i w polu [kod rabatowy] wpisz numer otrzymanego kuponu.
5. Dokonaj pełnego procesu zakupu.
6. Ponownie dodaj produkty do koszyka, które spełniają warunki promocji.
7. Wejdź w koszyk i w polu [kod rabatowy] wpisz numer otrzymanego kuponu.

Warunek końcowy:

Rabat w wysokości 40 zł **NIE** zostaje naliczony. Pojawiła się informacja, że **kupon jest nieprawidłowy**.

#05# Techniki projektowania testów

Przypadek testowy (T4)

ID: TC-4

Tytuł: Rabat 40 zł za zapis do newslettera - przypadek T4

Warunki wejściowe:

Klient zapisał się do newslettera [PRAWDA]

Klient dostał kupon za zapis do newslettera [FAŁSZ]

Kupon jest aktywny (nie został wykorzystany wcześniej) [NIEISTOTNE]

Produkty w koszyku spełniają warunki promocji [NIEISTOTNE]

Warunki wyjściowe:

Zostaje naliczony rabat 40 zł [FAŁSZ]

#05# Techniki projektowania testów

Przypadek testowy (T4)

Oczekiwany rezultat: Rabat za zapis do newslettera w wysokości 40 zł **NIE** zostaje dodany, wartość koszyka **NIE** zmienia się.

Kroki do wykonania:

1. Zapisz się na newsletter.
2. Przejdź na skrzynkę e-mail podaną przy zapisie na newsletter.
3. Sprawdź czy kod rabatowy został przesłany.

Warunek końcowy:

Brak kodu rabatowego.

#05# Techniki projektowania testów

Przypadek testowy (T5)

ID: TC-5

Tytuł: Rabat 40 zł za zapis do newslettera - przypadek T5

Warunki wejściowe:

Klient zapisał się do newslettera [FAŁSZ]

Klient dostał kupon za zapis do newslettera [NIEISTOTNE]

Kupon jest aktywny (nie został wykorzystany wcześniej) [NIEISTOTNE]

Produkty w koszyku spełniają warunki promocji [NIEISTOTNE]

Warunki wyjściowe:

Zostaje naliczony rabat 40 zł [FAŁSZ]

#05# Techniki projektowania testów

Przypadek testowy (T5)

Oczekiwany rezultat: Rabat za zapis do newslettera w wysokości 40 zł **NIE** zostaje dodany, wartość koszyka **NIE** zmienia się.

Kroki do wykonania:

Przypadek jest odpowiedzią na założenie, że klient **nie zapisał się na newsletter.**

Dalsze rozpatrywanie danych wejściowych i ich rezultatu jest **nieistotne**.

Warunek końcowy:

Brak.

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

4. Testowanie przejść pomiędzy stanami

System może różnie odpowiadać w zależności od aktualnych warunków oraz od historii **(od stanu)**.

Tabela stanów pokazuje **zależności pomiędzy stanami** oraz **wejściami** i może uwypuklić przejścia nieprawidłowe.

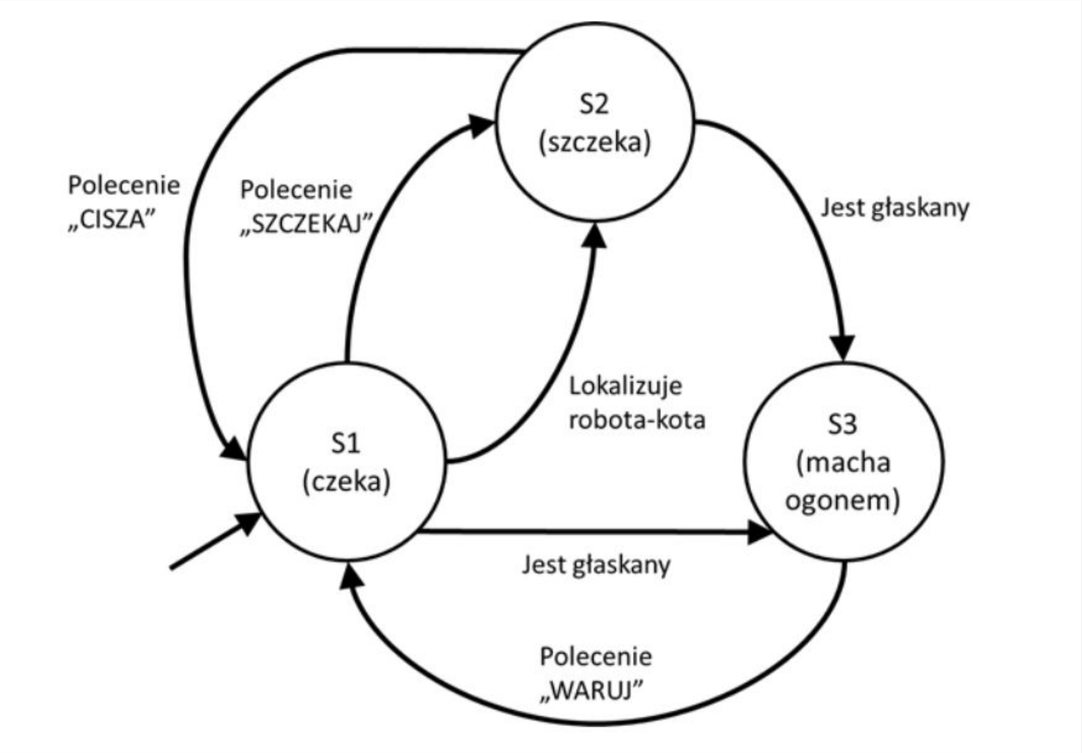
Testy można **zaprojektować** tak aby pokryły:

- typowe sekwencje stanów,
- każde przejście,
- konkretny ciąg przejść,
- aby testować przejścia nieprawidłowe.

#05# Techniki projektowania testów

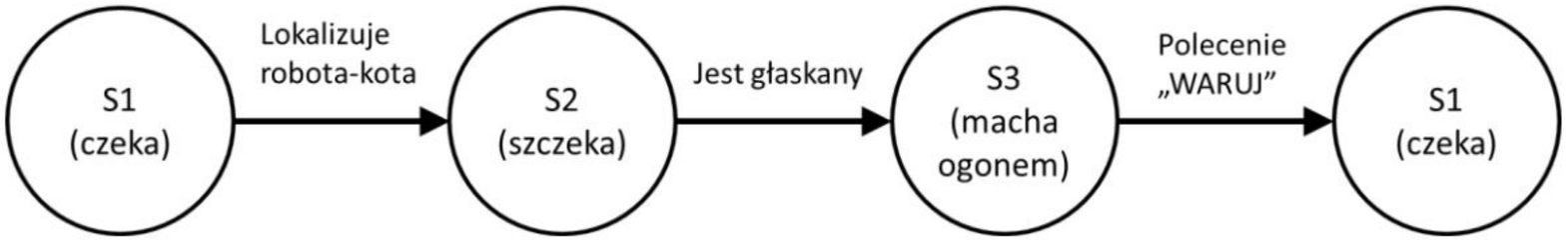
Przykład (zadanie):

Wyznacz minimalną liczbę przypadków testowych pokrywających wszystkie przejścia (strzałki). Ponowne osiągnięcie stanu S1 wymusza zakończenie przypadku testowego.

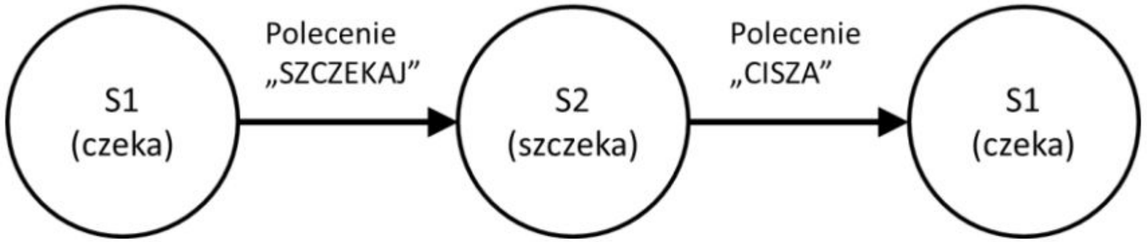


#05# Techniki projektowania testów

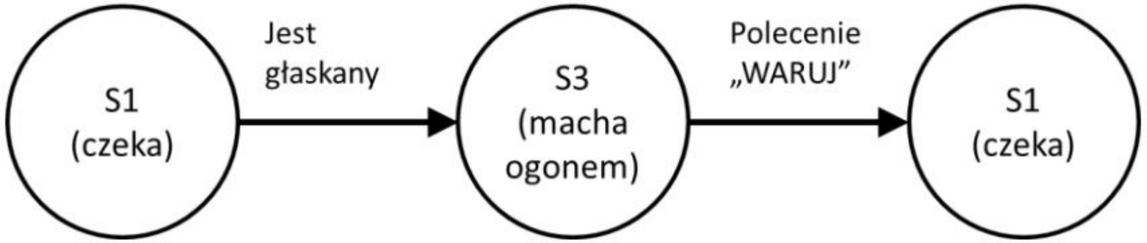
TC1:



TC2:



TC3:



Odpowiedź:

Aby pokryć **wszystkie przejścia**, potrzeba **3 przypadków testowych**.

Przypadki **TC1, TC2 i TC3** razem pokrywają wszystkie **6 przejść** w diagramie stanów.

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

5. Testowanie w oparciu o przypadki użycia

Przypadek użycia opisuje interakcje pomiędzy aktorami (użytkownikami lub systemami).

Każdy przypadek użycia posiada **warunki wstępne**, które muszą zostać **spełnione**, żeby przypadek użycia został wykonany.

Każdy przypadek użycia kończy się **warunkami końcowymi**.

Są nimi **widoczne rezultaty** jego wykonania oraz **stan systemu** po zakończeniu przypadku użycia.

Przypadki użycia zwykle posiadają **scenariusz główny** oraz czasami **scenariusze poboczne**.

#05# Techniki projektowania testów

Przykład: Przypadek użycia

Nazwa przypadku użycia	Zaloguj się do systemu szkolnego
Opis przypadku użycia	Zalogowanie użytkownika do systemu w celu uzyskania dostępu do funkcjonalności
Aktorzy	Rodzice, uczniowie, nauczyciel, administrator
Warunek wstępny	Użytkownicy posiadają dostęp do systemu i są w nim zalogowani
Warunek końcowy	Po pomyślnym zalogowaniu na identyfikator poczty użytkownika zostanie wysłana wiadomość e-mail z powiadomieniem o logowaniu.

#05# Techniki projektowania testów

Przykład:

Przykładowy scenariusz

Główne scenariusze	ID kroków	Kroki wykonania
Aktorzy (użytkownicy)	1	Wpisz nazwę użytkownika, wprowadź hasło
	2	Sprawdź poprawność nazwy użytkownika oraz hasła
	3	Zezwól na dostęp do systemu
Rozszerzenia	a	Zła nazwa użytkownika, system wyświetla komunikat o błędzie
	b	Nieprawidłowe hasło, system wyświetla komunikat o błędzie
	c	4x niepoprawne hasło, aplikacja zostaje zablokowana

#05# Techniki projektowania testów

Techniki projektowania testów - czarnoskrzynkowe

Przypadek użycia - służy do określenia sposobu korzystania z systemu zaprojektowanego do wykonywania określonego zadania.

Przypadek testowy - to grupa danych wejściowych do testów, warunków wykonania i oczekiwanych wyników opracowanych dla określonego celu testu.

Przypadek użycia NIE jest uruchamiany ani wykonywany!

Ponieważ jest to najczęściej tekstowa lub schematyczna prezentacja dokumentu.

#05# Techniki projektowania testów

Techniki projektowania testów - białoskrzynkowe

Techniki **białoskrzynkowe** nazywane są również technikami **opartymi na strukturze.**

Testy te można wykonywać **na każdym poziomie testowania**.

Technik strukturalnych **najlepiej użyć po technikach opartych na specyfikacji** (by zmierzyć dokładność testowania poprzez ocenę stopnia **pokrycia**).

Pokrycie - stopień w jakim struktura została przetestowana przez zestaw testów.

Jeżeli **pokrycie** jest **niższe niż 100%** wtedy można zaprojektować **więcej testów**.

#05# Techniki projektowania testów

Techniki projektowania testów - białoskrzynkowe

1.Testowanie i pokrycie instrukcji

W testowaniu instrukcji stosuje się **pokrycie instrukcji**, które polega na **zmierzeniu**, jaki odsetek **instrukcji wykonywalnych** został **przetestowany** przez zestaw testów.

W technice tej **projektuje** się **przypadki testowe**, tak by **wykonać określone instrukcje**, zwykle po to żeby zwiększyć **pokrycie**.

Pokrycie instrukcji **oblicza** się poprzez **podzielenie**:

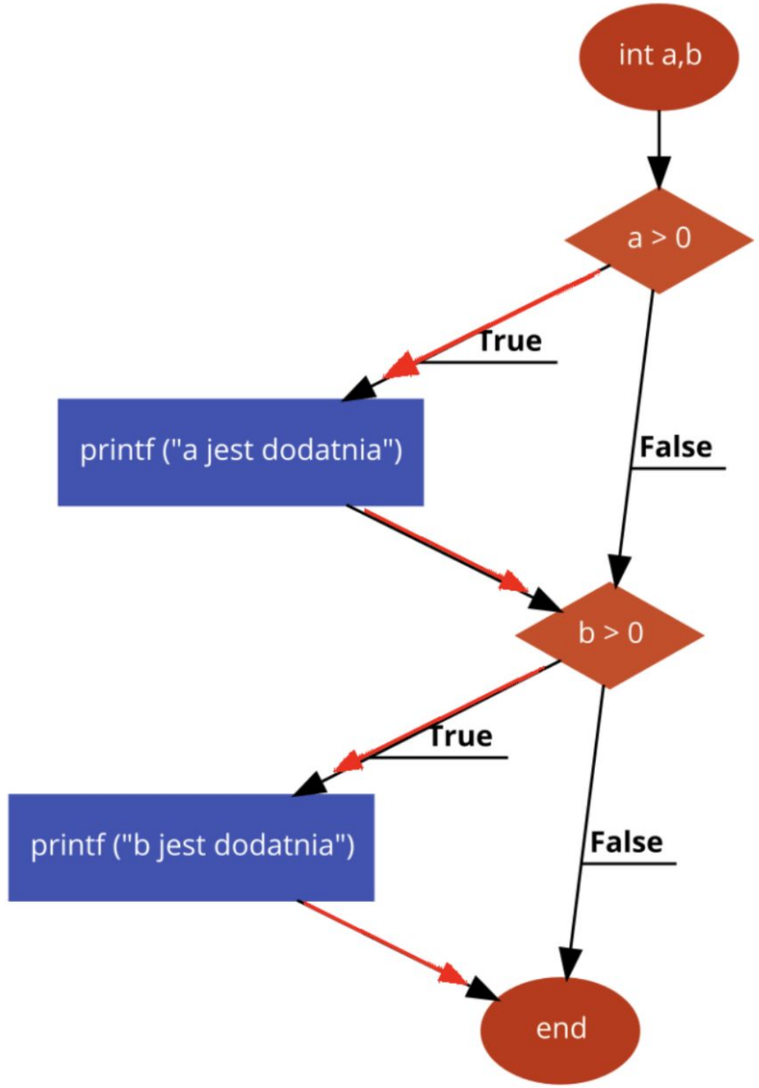
liczby wykonywanych instrukcji pokrytych przez **przypadki** testowe,

przez **liczbę wszystkich wykonywanych instrukcji** w testowanym **kodzie**.

#05# Techniki projektowania testów

Przykład:

```
int a,b
if (a > 0) {
printf (“a jest dodatnia”);
}
if (b > 0) {
printf (“b jest dodatnia”);
}
end
```



Aby **pokryć wszystkie instrukcje** tego pseudokodu wymagany będzie **1 przypadek testowy** (czerwone strzałki).

ponieważ

Pokryliśmy wszystkie bloki instrukcji zawarte we fragmencie pseudokodu **(niebieskie prostokąty).**

#05# Techniki projektowania testów

Techniki projektowania testów - białoskrzynkowe

2.Testowanie i pokrycie decyzji

Pokrycie decyzji - zmierzenie jaki odsetek **wyników decyzji** (np. wyniku prawda fałsz w instrukcji if) **został przetestowany** przez zestaw testów.

W technice testowania decyzji **projektuje** się **przypadki testowe** tak aby pokryć określone **wyniki decyzji**.

Pokrycie decyzji **oblicza** się poprzez **podzielenie**:

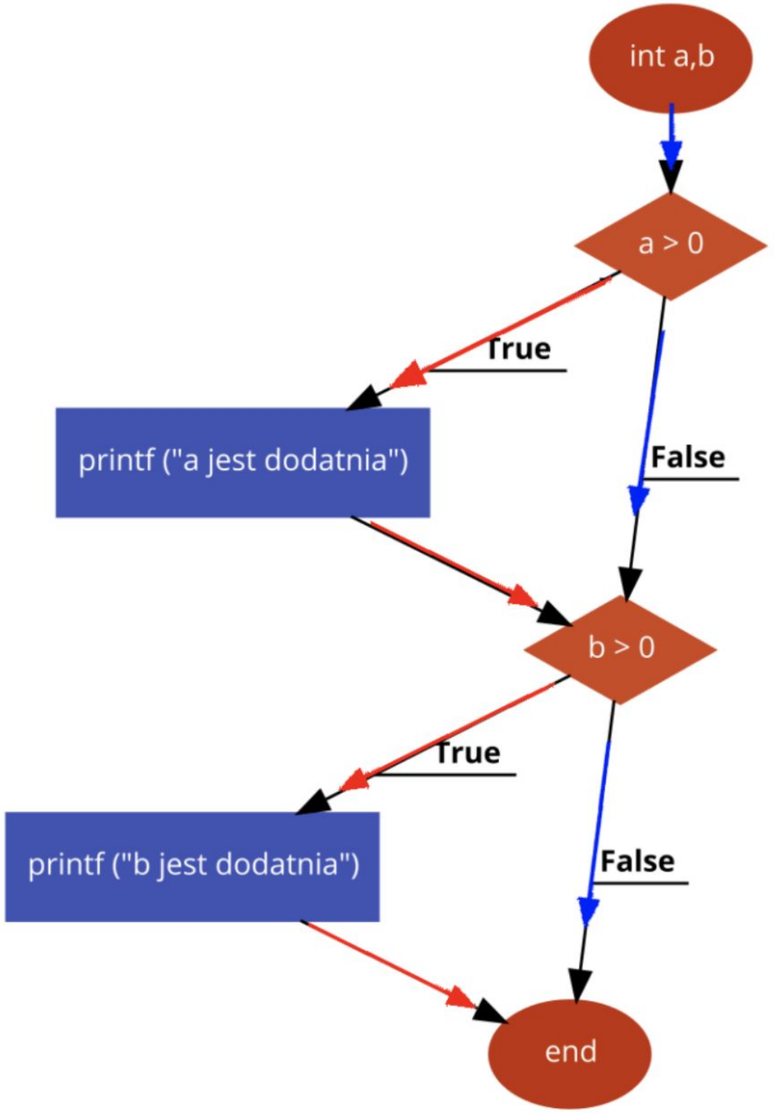
liczby wyników decyzji pokrytych przez **przypadki** testowe,

przez **liczbę wszystkich wyników decyzji** w testowanym **kodzie**.

#05# Techniki projektowania testów

Przykład:

```
int a,b
if (a > 0) {
printf (“a jest dodatnia”);
}
if (b > 0) {
printf (“b jest dodatnia”);
}
end
```



Do **pełnego pokrycia decyzji** będą wymagane 2 przypadki testowe.

Pierwszy przypadek testowy **pokrywa** wszystkie bloki instrukcji (czerwone strzałki) dla warunków **TRUE**, a **drugi pokrywa** warunki **FALSE** dla decyzji (niebieskie strzałki).

#05# Techniki projektowania testów

Techniki projektowania testów - białoskrzynkowe

Pokrycie decyzji jest mocniejsze niż pokrycie instrukcji.

100% pokrycia decyzji gwarantuje 100% pokrycia instrukcji, ale nie odwrotnie!

#05# Techniki projektowania testów

Techniki projektowania testów - oparte na doświadczeniu

Testowanie oparte na doświadczeniu zwane jako **testy eksploracyjne.**

Testy projektuje się na podstawie **wiedzy** i **intuicji** testerów oraz ich **doświadczeniu** z podobnymi aplikacjami i technologiami.

Równoległe projektowanie testów, ich wykonaniu i zapisywaniu wyników bazując na celach i trzymając się ram czasowych.

Szeroko wykorzystywaną techniką opartą na doświadczeniu jest zgadywanie błędów.

#05# Techniki projektowania testów

Techniki projektowania testów - wybór odpowiedniej techniki

Wybór, która technika testowania powinna zostać użyta, zależy od wielu czynników:

- typu systemu,
- regulacji prawnych,
- wymagań klienta,
- poziomu ryzyka,
- typu ryzyka,
- celu testów,
- dostępnej dokumentacji,
- wiedzy testerów,
- czasu i budżetu,
- cyklu życia oprogramowania.

#05# Techniki projektowania testów

Techniki projektowania testów - wybór odpowiedniej techniki

Podczas **projektowania** przypadków testowych
testerzy zwykle stosują **różne techniki projektowania**.

Ma to na celu zapewnienie odpowiedniego pokrycia testowanego obiektu.

06
Statyczne techniki testowania

#06# Statyczne techniki testowania

Techniki statyczne a proces testowania

Techniki dynamiczne -> wymagają uruchomienia programu.

Techniki statyczne -> polegają na sprawdzeniu bez uruchamiania kodu.

Techniki statyczne **polegają** na **sprawdzeniu ręcznym** (przeglądy) lub **analizie automatycznej** (analiza statyczna) kodu lub innych dokumentów testowych.

Typowe usterki wykrywane przez testy statyczne:

- odchylenia od standardów,
- usterki w wymaganiach lub projekcie,
- niedostateczna pielęgnowalność,
- nieprawidłowe specyfikacje interfejsów.

#06# Statyczne techniki testowania

Proces przeglądu - Typy

Istnieją **2 typy** przeglądów:

- **nieformalne** - brak spisanych instrukcji dla przeglądających
- **systematyczne** - uwzględniają przygotowanie zespołu, zapisanie wyników przeglądu, udokumentowania procedury wykonania przeglądu.

Stopień sformalizowanie przeglądu zależy od:

- dojrzałości procesu produkcyjnego,
- wymagań,
- konieczności dokumentowania przebiegu przeglądu.

#06# Statyczne techniki testowania

Proces przeglądu - Cele

Istnieją różne **cele** podczas przeglądów:

- znajdowanie usterek,
- zrozumienia,
- przekazywania wiedzy testerom i nowym członkom zespołu,
- przedyskutowania i podjęcia uzgodnionej decyzji.

#06# Statyczne techniki testowania

Proces przeglądu - Kroki przeglądu formalnego

1. **Planowanie:**
   - definiowanie **kryteriów** przeglądu,
   - wybór **uczestników** przeglądu,
   - przydział **ról**,
   - ustalenie **kryteriów wejścia i zakończenia**,
   - wybór **fragmentów dokumentu** do przejrzenia.

#06# Statyczne techniki testowania

Proces przeglądu - Kroki przeglądu formalnego

2. Rozpoczęcie:

- rozesłanie dokumentów,
- opisanie uczestnikom **celów** przeglądu, **procesu** i **dokumentów**,
- sprawdzenie **kryteriów wejścia**.

#06# Statyczne techniki testowania

Proces przeglądu - Kroki przeglądu formalnego

3. Przygotowanie indywidualne:

- **przygotowanie** przed spotkaniem (**przejrzenie** dokumentów),
- **zapisywanie** potencjalnych:
  -> defektów,
  -> pytań,
  -> komentarzy.

#06# Statyczne techniki testowania

Proces przeglądu - Kroki przeglądu formalnego

4. Kontrola / Ocena / Zapisywanie wyników (spotkanie przeglądowe)**:**

- dyskusja,
- spisywanie z **udokumentowaniem wyników**,
- spisanie **protokołu**,
- zapisywanie **defektów**,
- zapisywanie **rekomendacji**,
- podejmowanie **decyzji** co do defektów.

#06# Statyczne techniki testowania

Proces przeglądu - Kroki przeglądu formalnego

5. Poprawki:

- **naprawianie** znalezionych defektów,
- **uaktualnienie** statusów defektów.

6. Zakończenie:

- sprawdzenie, czy **usterki** zostały **obsłużone,**
- zbieranie **metryk,**
- sprawdzenie **kryteriów zakończenia**.

#06# Statyczne techniki testowania

Proces przeglądu - Role i odpowiedzialność przeglądu formalnego

1. **Kierownik -** decyduje o wykonaniu przeglądu.
2. **Moderator -** prowadzi przegląd dokumentu.
3. **Autor -** osoba która napisała lub nadzorowała powstanie dokumentu do przeglądu.
4. **Przeglądający -** osoby biznesowe lub techniczne znajdujący i spisujący uwagi.
5. **Protokólant -** dokumentuje wszystkie zagadnienia, problemy.

#06# Statyczne techniki testowania

Proces przeglądu - Przegląd nieformalny

Przegląd nieformalny:

- **brak** formalnego procesu,
- może przybrać formę **programowania w parach**,
- **może** ale nie musi **być udokumentowany**,
- **główny cel:** tani sposób na osiągnięcie niewielkich korzyści.

#06# Statyczne techniki testowania

Proces przeglądu - Przegląd nieformalny

Przejrzenie:

- spotkanie jest **prowadzone przez autora**,
- sesje **nieograniczone czasowo**,
- w praktyce może przekształcić się **od nieformalnego w formalny**,
- **główny cel:** uczenie się, zrozumienie, znajdowanie usterek.

#06# Statyczne techniki testowania

Proces przeglądu - Przegląd <u>nieformalny</u>

Przegląd techniczny:

- posiada **zdefiniowany proces** wykrywania defektów,,
- może być organizowany jako przegląd koleżeński **bez udziału kierownictwa**,
- prowadzony przez **moderatora**,
- **przygotowanie przeglądających** przed spotkaniem,
- opcjonalnie z wykorzystaniem **list kontrolnych**,
- **główny cel:** przedyskutowanie, podjęcie decyzji, ocena alternatyw, wyszukiwanie usterek, rozwiązywanie problemów technicznych, sprawdzenie zgodności ze specyfikacją.

#06# Statyczne techniki testowania

Proces przeglądu - Przegląd nieformalny

Inspekcja:

- prowadzona przez **moderatora**,
- zwykłe **sprawdzenie przez kolegów**,
- posiada **metryki**,
- posiada **listy kontrolne**,
- **przygotowanie przeglądających** przed spotkaniem,
- **raport** z inspekcji zawierający listę uwag,
- **główny cel:** wyszukiwanie usterek.

#06# Statyczne techniki testowania

Analiza statyczna - Cel

Celem analizy statycznej jest wyszukiwanie **usterek w kodzie** programu lub **modelach bez uruchamiania oprogramowania**.

Analiza statyczna **może znaleźć usterki które są trudne do znalezienia podczas testowania dynamicznego**.

Analiza statyczna **znajduje usterki - a nie awarie!**

#06# Statyczne techniki testowania

Analiza statyczna - Korzyści

Korzyści:

- wczesne wykrywanie usterek,
- wczesne wykrywanie wysokiego stopnia złożoności,
- identyfikacja defektów trudnych do wykrycia podczas testów dynamicznych,
- wykrywanie niespójności w modelach oprogramowania,
- zwiększenie pielęgnowalności kodu i projektu,
- zapobieganie defektom.

#06# Statyczne techniki testowania

Analiza statyczna - Korzyści

Analiza statyczna zwykle wykrywa następujące typy usterek:

- odwołanie do niezainicjalizowanej zmiennej,
- niespójne interfejsy pomiędzy modułami,
- niewykorzystywane lub niepoprawne zadeklarowane zmienne,
- martwy kod,
- brakująca lub błędna logika (nieskończone pętle),
- zbyt skomplikowane konstrukcje,
- naruszenie zasad kodowania,
- słabe punkty zabezpieczeń,
- naruszenie reguł składni kodu.

#06# Statyczne techniki testowania

Wsparcie narzędziowe dla testów statycznych

Narzędzia wspierające testy statyczne stanowią opłacalny sposób na znajdowanie większej ilości defektów we wcześniejszych fazach życia oprogramowania.

Mogą to być:

- **narzędzia wspomagające przeglądy**

  Narzędzia te wspomagają **proces przeglądu, listy kontrolne, wytyczne dla przeglądów**. Są wykorzystywane do przechowywania i komunikowania komentarzy z przeglądów, raportów defektów oraz pracochłonności.

- **narzędzia do analizy statycznej**

  Narzędzia te pomagają programistom i testerom jeszcze przed testami dynamicznymi poprzez **wymuszanie standardów kodowania, analizę struktur i zależności.**

#06# Statyczne techniki testowania

Wsparcie narzędziowe dla testów statycznych

- **narzędzia do modelowania**

  Narzędzia te używane są w **walidacji** modeli oprogramowania (np. fizycznego modelu danych w bazach relacyjnych) do wymieniania **niezgodności** i wyszukiwania **defektów**.

  Narzędzia te pomagają dosyć często w generowaniu przypadków testowych z modeli.

07
Testy potwierdzające

#07# Testy potwierdzające

Testowanie regresji / testy potwierdzające (testowanie związane ze zmianami)

Po **wykryciu** i **naprawieniu defektu**, powinien zostać wykonany **RETEST**.

Celem Retestu jest **potwierdzenie** usunięcia usterki.

Takie testy nazywane są **testami potwierdzającymi.**

Testy regresji - można wykonywać na **wszystkich poziomach** testów i dla **wszystkich typów**.

Zestawy testów regresywnych są wykonywane wiele razy i są stosunkowo niezmienne, wobec czego są dobrymi kandydatami do **AUTOMATYZACJI.**

#07# Testy potwierdzające

Testowanie pielęgnacyjne

Testowanie pielęgnacyjne - testowanie **zmian** wdrożonego systemu lub **wpływ zmienionego środowiska** na wdrożony system.

Testy pielęgnacyjne, oprócz przetestowania tego co zostało zmienione, **zawierają testy regresywne** tych części systemy, które się nie zmieniły.

Zakres testów pielęgnacyjnych zależy od:

- ryzyka zmian,
- wielkości systemu,
- zakresu zmian.

08
Narzędzia wspomagające proces testowania

#08# Narzędzia wspomagające proces testowania

Znaczenie i cel wsparcia narzędziowego dla testów

Narzędzia testowe mogą być wykorzystywane w jednej lub wielu czynnościach wspierających testowanie.

Mogą to być:

- **narzędzia wspomagające zarządzanie procesem testowym**

  Narzędzia do zarządzania testami, wynikami testów, danymi, wymaganiami, incydentami oraz narzędzia raportujące i monitorujące.

- **narzędzia używane w testach**

  Narzędzia wspierające wykonywanie testów, generujące dane testowe, porównujące wyniki

#08# Narzędzia wspomagające proces testowania

Znaczenie i cel wsparcia narzędziowego dla testów

- **narzędzia używane w rozpoznaniu (eksploracji)**

  Narzędzia monitorujące dostęp do plików przez aplikację.

- **dowolne narzędzie wspierające testy**

  Narzędzia wspierające proces testowania, np. arkusz kalkulacyjny.

#08# Narzędzia wspomagające proces testowania

Znaczenie i cel wsparcia narzędziowego dla testów

W zależności od kontekstu **wsparcie narzędziowe** dla testów może mieć jeden lub kilka z poniższych **celów**:

- **zwiększenie efektywności czynności testowych**

  Automatyzacja powtarzających się zadań raportowania i monitorowania testów.

- **automatyzacja czynności testowych**

  Automatyzacja czynności, które wymagałyby wielkich nakładów, gdyby wykonywane ręcznie.

- **zwiększenie niezawodności testów**

  Automatyzacja porównywania dużej ilości danych lub symulacje.

#08# Narzędzia wspomagające proces testowania

Klasyfikacja narzędzi testowych

Narzędzia testowe można podzielić na wiele sposobów:

- według celów,
- komercyjne,
- darmowe,
- o otwartym kodzie (open-source),
- według wykorzystywanej technologii,
- itd..

#08# Narzędzia wspomagające proces testowania

Narzędzia do zarządzania testami

Dostarczają one interfejsów do:

- wykonywania zadań,
- śledzenia defektów,
- zarządzania wymaganiami,
- wspierania analizy ilościowej,
- raportowania.

#08# Narzędzia wspomagające proces testowania

Narzędzia do zarządzania wymaganiami

Narzędzie te:

- przechowywują tekst wymagań, ich atrybuty (np. priorytet),
- generują unikalne identyfikatory,
- wspierają śledzenie powiązań pomiędzy wymaganiami, a poszczególnymi testami,
- mogą pomóc w znalezieniu niespójnych lub brakujących wymagań.

#08# Narzędzia wspomagające proces testowania

Narzędzia do zarządzania incydentami

Narzędzie te:

- przechowywują raporty incydentów,
- zarządzają raportami incydentów,
- pomagają zarządzać cyklem życia incydentów,
- opcjonalnie mogą wspomagać analizy statyczne.

#08# Narzędzia wspomagające proces testowania

Narzędzia do zarządzania konfiguracją

Narzędzie te przechowują i zarządzają wersjami testaliów i innego oprogramowania, np:,

- wersje systemów operacyjnych,
- kompilatory,
- przeglądarki.

Ma to znaczenie, w szczególności, gdy konfigurowane jest **więcej niż jedno środowisko** sprzętowo-programowe.

#08# Narzędzia wspomagające proces testowania

Narzędzia do projektowania testów

Narzędzie te wykorzystywane są do:

- generowania **wejść** do testów,
- generowania graficznych **interfejsów** użytkownika,
- generowania **modeli** projektowych (stanów, danych obiektów),
- generowania **kodu.**

#08# Narzędzia wspomagające proces testowania

Narzędzia do <u>przygotowywania danych testowych</u>

Narzędzie te wykorzystywane są do:

- przygotowywania danych testowych,
- przetwarzają bazy danych
- przetwarzają pliki,
- przetwarzają wszystko, co można przekształcić w późniejsze dane do wykorzystania podczas wykonywania testów.

#08# Narzędzia wspomagające proces testowania

Narzędzia do <u>wykonywania testów</u>

Narzędzie te wykorzystywane są do:

- tworzenia logów z testów,
- umożliwiają automatyczne wykonywanie testów,
- wykonywania testów,
- nagrywania testów,
- mierzenia pokrycia,
- testowanie zabezpieczeń.

#08# Narzędzia wspomagające proces testowania

Narzędzia do <u>mierzenia wydajności i monitorowania testów</u>

Narzędzie te wykorzystywane są do:

- analizy dynamicznej (wycieki pamięci),
- testów wydajnościowych (zachowanie systemu),
- testów obciążeniowych (wirtualni użytkownicy),
- ogólnego monitorowania (analiza, weryfikacja, raportowanie zasobów systemowych)

#08# Narzędzia wspomagające proces testowania

Potencjalne korzyści i ryzyko wsparcia narzędziowego dla testów

Z użytkowaniem każdego narzędzia wiążą się potencjalne **korzyści** i **możliwości**, ale również może wiązać się pewne **ryzyko**.

Korzyści:

- **ocena jest obiektywna**

  Np. miary statyczne, pokrycie.

- **łatwiejszy dostęp do danych o testach i testowaniu**

  Statystyki i wykresy obrazujące postęp testów, współczynniki występowania incydentów oraz wydajność.

#08# Narzędzia wspomagające proces testowania

Potencjalne korzyści i ryzyko wsparcia narzędziowego dla testów

- **zredukowana zostaje powtarzające się praca**

  Np. uruchamianie testów regresywnych, sprawdzanie zgodności ze standardami kodowania.
- **zwiększa się spójność i powtarzalność**

  Testy wykonywane przez narzędzie są w tej samej kolejności i z tą samą częstotliwością

#08# Narzędzia wspomagające proces testowania

Potencjalne korzyści i ryzyko wsparcia narzędziowego dla testów

Ryzyko:

- **nierealistyczne oczekiwania od narzędzia**

  Nierealistyczne wymagania co do funkcjonalności lub łatwości użycia.

- **niedoszacowanie czasu, kosztów oraz pracochłonności wdrożenia narzędzia**

  Włączając w to szkolenia oraz ekspertów zewnętrznych.

- **niedoszacowanie czasu i pracochłonności do osiągnięcia znaczących i trwałych korzyści**

  Potrzeba zmian w procesie testowym

- **zbytnie poleganie na narzędziu**
- **słabe wsparcie od dostawcy oprogramowania (lub jego brak)**
- **brak kompatybilności**

#08# Narzędzia wspomagające proces testowania

Wdrażanie narzędzi w organizacji

Główne aspekty do wzięcia pod uwagę podczas **wyboru narzędzia** dla organizacji to:

- **ocena dojrzałości organizacji,**

  Mocne i słabe strony oraz identyfikacja możliwości doskonalenia procesu testowania.

- **ocena według jasnych wymagań oraz obiektywnych kryteriów**
- **PoC - proof-of-concept**

  Wstępne zbadanie czy narzędzie jest skuteczne i jakie zmiany w istniejącej infrastrukturze są potrzebne do skutecznego użycia narzędzia.

- **ocena dostawcy**

  Czy posiadamy odpowiednie wsparcie, szkolenia.

- **oszacowanie stosunku korzyści do kosztów**

  Na podstawie konkretnego przypadku biznesowego

#08# Narzędzia wspomagające proces testowania

Wdrażanie narzędzi w organizacji - Projekt pilotażowy

Wdrażanie wybranego narzędzia w organizacji zaczyna się od **projektu pilotażowego**.

Cele projektu pilotażowego:

- szczegółowe zapoznanie się z narzędziem,
- ocena czy i jak narzędzie pasuje do obowiązujących procesów,
- ustalenie standardów użycia narzędzia,
- ocena korzyści płynących z narzędzia - osiągnięta przy rozsądnych kosztach.

#08# Narzędzia wspomagające proces testowania

Wdrażanie narzędzi w organizacji - Projekt pilotażowy

Czynniki wpływające na sukces wdrożenia narzędzia w organizacji:

- **stopniowe wdrażanie** narzędzia,
- adaptacja i **udoskonalenie procesu** tak, aby pasował do sposobu używania narzędzia,
- zapewnienie **szkoleń** oraz doradztwa nowym użytkownikom,
- wdrożenie sposobu **zbierania informacji** z **wykorzystania** narzędzia,
- zapewnienie **wsparcia** dla zespołu testowego,
- **monitorowanie** wykorzystania narzędzia oraz osiąganych korzyści.

09
Współpraca z programistami

#09# Współpraca z programistami

Psychologia testowania

Znajdowanie defektów podczas procesu testowania może być odbierane jako **krytyka produktu** lub jego **autora**.

Na skutek owej krytyki, **testowanie** może być postrzegane jako **czynność destrukcyjna**.

Aby **ograniczyć** takie postrzeganie, należy przekazywać **informację** o defektach i awariach w sposób jak najbardziej **konstruktywny**.

10
Weryfikacja / Walidacja

#10# Weryfikacja / Walidacja

Zarówno **weryfikacja** jak i **walidacja** produktu są **czynnościami**, które służą **sprawdzeniu** czy wytworzony produkt **spełnia wymagania** produktu.

Obie czynności zachodzą **wielu momentach** i mogą pojawić się w **wielu fazach** tworzenia systemu.

Różnice między weryfikacją i walidacją dotyczą przede wszystkim tego, z jakiej **perspektywy** dokonujemy sprawdzenia:

- perspektywa zespołu budującego system,
- perspektywa użytkownika końcowego.

#10# Weryfikacja / Walidacja

Weryfikacja - weryfikujemy to co da się **wyliczyć**, **wykazać logicznie** i **nie ma jak zanegować** tego co jest już zweryfikowane.
Przy dobrze spisanych wymaganiach weryfikacji dokonać może każdy kto wie jak czytać specyfikację i wiedzący jak wykonać test.

Walidacja - walidacja jest po stronie odbiorcy i to zaspokojenie jego (odbiorcy) potrzeb jest ostatecznym kryterium sukcesu.

11
Organizacja testowania

#11# Organizacja testowania

Organizacja testów a ich niezależność

Skuteczność wykrywania usterek w testach może zostać **podniesiona** przez zaangażowanie **niezależnych testerów**.

Niezależność może występować w różnych wariantach, włączając w to następujące:

- brak niezależnych testerów -> **programiści sami testują swój kod**
- niezależni testerzy **wewnątrz zespołu** projektowego
- niezależny zespół testowy lub grupa testerów **wewnątrz organizacji** podlegająca kierownikowi projektu
- **niezależni specjaliści** od określonych **typów testów** takich jak: użyteczność, zabezpieczenia
- niezależni testerzy, którzy zostali **wynajęci** lub są na **zewnątrz organizacji**

#11# Organizacja testowania

Organizacja testów a ich niezależność

Korzyści:

- niezależni testerzy widzą inne i odmienne usterki niż twórcy,
- niezależni testerzy nie mają uprzedzeń do oprogramowania,
- niezależny tester może zweryfikować założenia poczynione podczas specyfikacji i implementacji systemu.

Wady:

- izolacja od zespołu developerskiego,
- programiści mogą utracić poczucie odpowiedzialności za jakość,
- niezależni testerzy mogą być postrzegani jako wąskie gardło lub obwiniani za opóźnienia w wydaniach.

#11# Organizacja testowania

Organizacja testów - zadania lidera testów oraz testera (kompetencje)

Typowe zadania Lidera Testów:

- **koordynowanie strategii** oraz **planu testów** z kierownikami projektu,
- **tworzenie** lub **przeglądanie strategii testów** w projekcie oraz polityki testowania w organizacji,
- przedstawianie **perspektywy testowania** w innych zadaniach projektowych takich jak planowanie integracji,
- **planowanie testów**, uwzględniając:

    -> ich kontekst,
    -> cel,
    -> ryzyko,
    -> pracochłonność,
    -> definiowanie poziomu testów, cykli testowych oraz planowanie zarządzania incydentami.

#11# Organizacja testowania

Organizacja testów - zadania lidera testów oraz testera (kompetencje)

Typowe zadania Lidera Testów:

- zorganizowanie odpowiedniego **zarządzania konfiguracją testaliów** w celu zwiększenia możliwości śledzenie powiązań,
- wprowadzenie **metryk** do mierzenia postępu prac,
- decydowanie co powinno zostać **zautomatyzowane**,
- **wybór narzędzi** wspierających testy,
- organizacja **szkoleń** dla testerów,
- sporządzanie raportów podsumowujących testy.

#11# Organizacja testowania

Organizacja testów - zadania lidera testów oraz testera (kompetencje)

Typowe zadania Testera:

- przeglądanie i wnoszenie wkładu do planów testów,
- analiza, przegląd oraz ocena wymagań użytkownika,
- współtworzenie środowiska testowego,
- przygotowywanie i pozyskiwanie danych testowych,
- implementacja testów na wszystkich poziomach,
- automatyzacja testów,
- pomiar wydajności modułów i systemów.

12
Planowanie testowania

#12# Planowanie testowania

Planowanie testowania

Wpływ na planowanie testów ma:

- polityka testowania w organizacji,
- zakres testów,
- cele,
- ryzyko,
- ograniczenia,
- krytyczność,
- testowalność,
- dostępność zasobów.

#12# Planowanie testowania

Planowanie testowania

Im **dłużej** trwa **planowanie** projektu i testów, tym **więcej informacji** jest dostępnych i tym więcej **szczegółów** może być **włączonych** do **planowania**.

Planowanie testów jest **czynnością stałą** i wykonuje się je **dla wszystkich procesów** i zadań **cyklu życia oprogramowania.**

Informacje zwrotne z czynności testowych są **używane do wykrywania zmieniających się ryzyk** tak, że **planowanie** może zostać **dopasowane** do bieżącej **sytuacji w projekcie.**

#12# Planowanie testowania

Czynności związane z planowaniem testowania

W planowanie testów dla całego systemu (lub jego części) mogą wchodzić następujące czynności:

- ustalenie **zakresu** testowania,
- ustalenie **ryzyka** testowania,
- zdefiniowanie **celów** testowania,
- zdefiniowanie **ogólnego podejścia** (kryteria wejścia/zakończenia),
- **koordynowanie zadań testowych** z innymi zadaniami w cyklu życia oprogramowania,
- **przydzielanie zasobów** do zadań testowych,
- **zdefiniowanie** poziomu **szczegółowości** oraz struktury **dokumentacji testowej**,
- **wybór metryk** do monitorowania i kontroli nad testami.

#12# Planowanie testowania

Kryteria wejścia

Definiują **warunki** pozwalające na **rozpoczęcie** testów.

Kryteria wejścia zwykle **mogą zawierać** następujące zagadnienia:

- dostępność i gotowość **środowiska testowego**,
- gotowość **narzędzi testowych** w środowisku testowym,
- dostępność **testowalnego kodu**,
- dostępność **danych wejściowych**.

#12# Planowanie testowania

Kryteria zakończenia

Celem jest **zdefiniowanie momentu zakończenia** testów na danym poziomie lub gdy cel testów został spełniony.

Kryteria zakończenia mogą składać się z:

- miar staranności (pokrycie kodu, funkcjonalności lub ryzyka),
- estymat gęstości błędów lub miar niezawodności,
- kosztu,
- istniejącego ryzyka, takiego jak: niepoprawione usterki lub brak pokrycia pewnych obszarów,
- harmonogramów, np. zdefiniowanych na podstawie czasu do wypuszczenia na rynek.

#12# Planowanie testowania

Szacowanie testów

Istnieją **2 podejścia** do **szacowania pracochłonności** testów:

- **podejście oparte na metrykach**

  Szacowanie pracochłonności testów bazując na pomiarach minionych lub podobnych projektów.

- **podejście oparte na ekspertach**

  Szacowanie zadań przez ich przyszłych wykonawców lub przez ekspertów.

Gdy pracochłonność zostanie już oszacowana
można przyporządkować zasoby do zadań
oraz szkicować harmonogram.

13
Strategia testowania

#13# Strategia testowania

Podejście do testowania - Strategia testowania

Podejście do testów jest implementacją strategii testów w konkretnym projekcie.

Typowe podejścia do testów:

- **podejście analityczne**

  Testy oparte na ryzyku, w którym testowanie kierowane jest na **obszary o największym ryzyku**.

- **podejście oparte na modelach**

  Wykorzystuje informacje statyczne na temat **współczynników awarii** (takich jak modele wzrostu niezawodności oprogramowania).

#13# Strategia testowania

Podejście do testowania - Strategia testowania

- **podejście metodyczne**

  Oparte na **awariach**, **doświadczeniu**, a także na **listach kontrolnych** lub **atrybutach jakościowych**.

- **podejście zgodne ze standardem lub procesem**

  Określone przez **standardy przemysłowe** lub **metodyki zwinne**.

#13# Strategia testowania

Podejście do testowania - Strategia testowania

- **podejście dynamiczne**

  Testowanie eksploracyjne, w którym testowanie bardziej **reaguje na zdarzenia podczas testów** niż jest wykonywane według planu i w którym **wykonywanie testów i ocena wyników dzieją się równolegle**.

- **podejście konsultatywne**

  Pokrycie testowe jest **sterowane** głównie **przez wskazówki i porady ekspertów** technologicznych lub **biznesowych z zewnątrz zespołu testowego**.

#13# Strategia testowania

Podejście do testowania - Strategia testowania

- **podejście regresywne**

  Używa się powtórnie istniejących **materiałów testowych**, rozbudowanej **automatyzacji** regresywnych testów funkcjonalnych oraz **standardowych zestawów testów**.

Różne podejścia mogą zostać połączone!

np. w dynamiczne testy oparte na ryzyku.

14
Monitorowanie testowania

#14# Monitorowanie testowania

Monitorowanie postępu prac

Celem monitorowania jest:

- uzyskanie informacji zwrotnych,
- uzyskanie wglądu w przebieg zadań testowych.

Informacje, które mają być **monitorowane**, mogą zostać **zebranie ręcznie** lub **automatycznie**.

Mogą zostać **użyte** do:

- pomiarów spełnienia kryteriów zakończenia (pokrycia),
- oceny postępów prac według zaplanowanego harmonogramu i budżetu.

#14# Monitorowanie testowania

Monitorowanie postępu prac

Najczęściej monitorowanymi metrykami są:

- procent pracy wykonanej przy przygotowywaniu przypadków testowych,
- procent prac wykonanych przy przygotowaniu środowiska testowego,
- liczba (nie)wykonanych przypadków testowych,
- liczba (nie)zaliczonych przypadków testowych,
- informacje o usterkach (gęstość błędów, współczynnik awarii wyniki testów),
- pokrycie testami wymagań, ryzyka, kodu,
- daty kamieni milowych,
- koszt testowania.

#14# Monitorowanie testowania

Monitorowanie postępu prac - Raportowanie testów

Raportowanie testów daje podsumowanie projektu testowego a w tym:

- co się zdarzyło w czasie testowania, np. daty spełnienia kryteriów zakończenia,
- analizę informacji oraz metryk np. opłacalność dalszego testowania.

Pomiary powinny być wykonywane **w trakcie** oraz **na koniec danego poziomu testów**, żeby ocenić:

- dopasowanie celów testów do danego poziomu testów,
- adekwatność wybranego podejścia do testów,
- skuteczność testów w odniesieniu do celów testowania.

15
Kierowanie testami

#15# Kierowanie testami

Kierowanie testami

Kierowanie testami to wszystkie **działania zarządcze** lub **korekcyjne** podjęte **na skutek zebranych** i zaraportowanych **informacji** i **pomiarów**.

Przykłady działań związanych z kierowaniem testami:

- podejmowanie decyzji na podstawie informacji uzyskanych z monitorowania testów,,
- zmiana priorytetów testów,
- zmiana harmonogramu testów (może być spowodowana dostępnością środowiska testowego).

#15# Kierowanie testami

Zarządzanie konfiguracją

Celem zarządzania konfiguracją jest **ustanowienie i utrzymanie integralności modułów/danych/dokumentacji** związanych z oprogramowaniem lub systemem przez cały projekt lub jego cykl.

Z punktu widzenia testowania, **zarządzanie konfiguracją** może wymagać zapewnienia, że:

- wszystkie elementy oprogramowania zostały zidentyfikowane, poddane kontroli wersji i że są śledzone,
- odwołania w dokumentacji testowej do wszystkich zidentyfikowanych dokumentów oraz elementów są jednoznaczne.

Zarządzanie konfiguracją **pomaga** testerom jednoznacznie **wskazać** (i odtworzyć) **testowany element, dokumenty testowe i testy.**

#15# Kierowanie testami

Ryzyko a testowanie

Ryzyko w testowaniu to możliwość wystąpienia zdarzenia lub sytuacji powodującej **niepożądane konsekwencje**.

Poziom ryzyka jest określony przez prawdopodobieństwo wystąpienia niekorzystnego zdarzenia oraz jego wpływ, tj. szkodę jaką może wyrządzić to zdarzenie.

#15# Kierowanie testami

Obszary ryzyka projektowego

Ryzyko projektowe - obszary ryzyka otaczające zdolność projektu do osiągnięcia postawionych przed nim celów, takie jak:

- **czynniki organizacyjne:**

  -> braki w umiejętnościach (szkolenia),
  -> problemy kadrowe,
  -> nieprawidłowe nastawienie i oczekiwania od testowania.

#15# Kierowanie testami

Obszary ryzyka projektowego

- **czynniki techniczne:**

  -> problemy ze zdefiniowaniem poprawnych wymagań,
  -> stopień, w jakim wymagania mogą zostać spełnione przy istniejących ograniczeniach,
  -> dostępność środowiska testowego,
  -> niska jakość projektu, kodu, danych testowych.

- **problemy z dostawcami:**

  -> niewywiązywanie się dostawców ze zobowiązań,
  -> problemy z kontraktami.

#15# Kierowanie testami

Obszary ryzyka produktowego

Ryzyko projektowe - potencjalne obszary wystąpienia awarii w oprogramowaniu lub systemie, takie jak:

- dostarczanie awaryjnego oprogramowania,
- możliwość wyrządzenia szkody człowiekowi lub firmie przez oprogramowanie lub sprzęt,
- niedostateczne atrybuty oprogramowania (np. funkcjonalność, niezawodność, wydajność),
- niska jakość lub brak spójnych danych (problemy z konwersją danych, naruszenie standardów danych),
- oprogramowanie, które nie spełnia założonych funkcji.

Ryzyka używa się do określania, kiedy rozpocząć testowanie

oraz co powinno zostać dokładniej przetestowane!

#15# Kierowanie testami

Zarządzanie incydentami

Incydenty (bugi) można zgłaszać podczas:

- programowania,
- przeglądów,
- testowania,
- użytkowania produktu.

Incydenty mogą być zgłoszone dla problemów w:

- kodzie,
- działającym systemie,
- dokumentach dowolnego typu.

#15# Kierowanie testami

Zarządzanie incydentami

Raporty incydentów istnieją w celu:

- dostarczania programistom i innym stronom informacji na temat problemów, w celu identyfikacji i naprawy, jeżeli okaże się to konieczne,

- dostarczania liderom testów środków do śledzenia jakości testowanego systemy oraz postępu prac,

- dostarczenia pomysłów na doskonalenie procesu testowania.

#15# Kierowanie testami

Zarządzanie incydentami

Raport Incydentów może zawierać:

- **datę** zgłoszenia,
- zgłaszającą **organizację** lub **osobę**,
- wyniki **oczekiwane** oraz **rzeczywiste**,,
- wskazanie na **element konfiguracji** oraz na **środowisko**,
- **proces cyklu życia oprogramowania** lub systemy w którym incydent został znaleziony (**wersja**),
- **opis** incydentu oraz **kroki** w celu umożliwienia odtworzenia incydentu,
- **logi, zrzuty ekranu, bazę danych,**
- **stopień** - stopień wpływu na system
- **priorytet** - pilność naprawy,
- **status** incydentu (open, in progress, blocked, to test, fixed, done).

Details

Type:	Bug	Status:	DONE (View Workflow)
Priority:	Major		
Component/s:	GUI Automation	Resolution:	Done
Labels:	None		
Epic Link:	GUI for CRM OPS		
Sprint:	Sprint 41		

Description

Build	ver 0.1
Device	n/a
OS version	n/a
Resolution	n/a
Aspect ratio	n/a
Repro rate	100%

Description

Headless mode not working as expected.

Steps to reproduce

1. Mark checkbox - headless mode
2. Run Machine

Suggested expected result

Tool will be working in the background.

Smart Checklist

Attachments

People

Assignee: Marek Waszak

Reporter: Marek Waszak

Votes: 0

Watchers: 1 Stop watching this issue

Dates

Created: 24/Jan/23 09:41

Updated: 24/Jan/23 09:43

Resolved: 24/Jan/23 09:43

Time Tracking

Collaborators

Development

Agile

Active Sprint:

Sprint 41 ends 02/Feb/23

View on Board

Slack

Dziękuję za uwagę!