Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

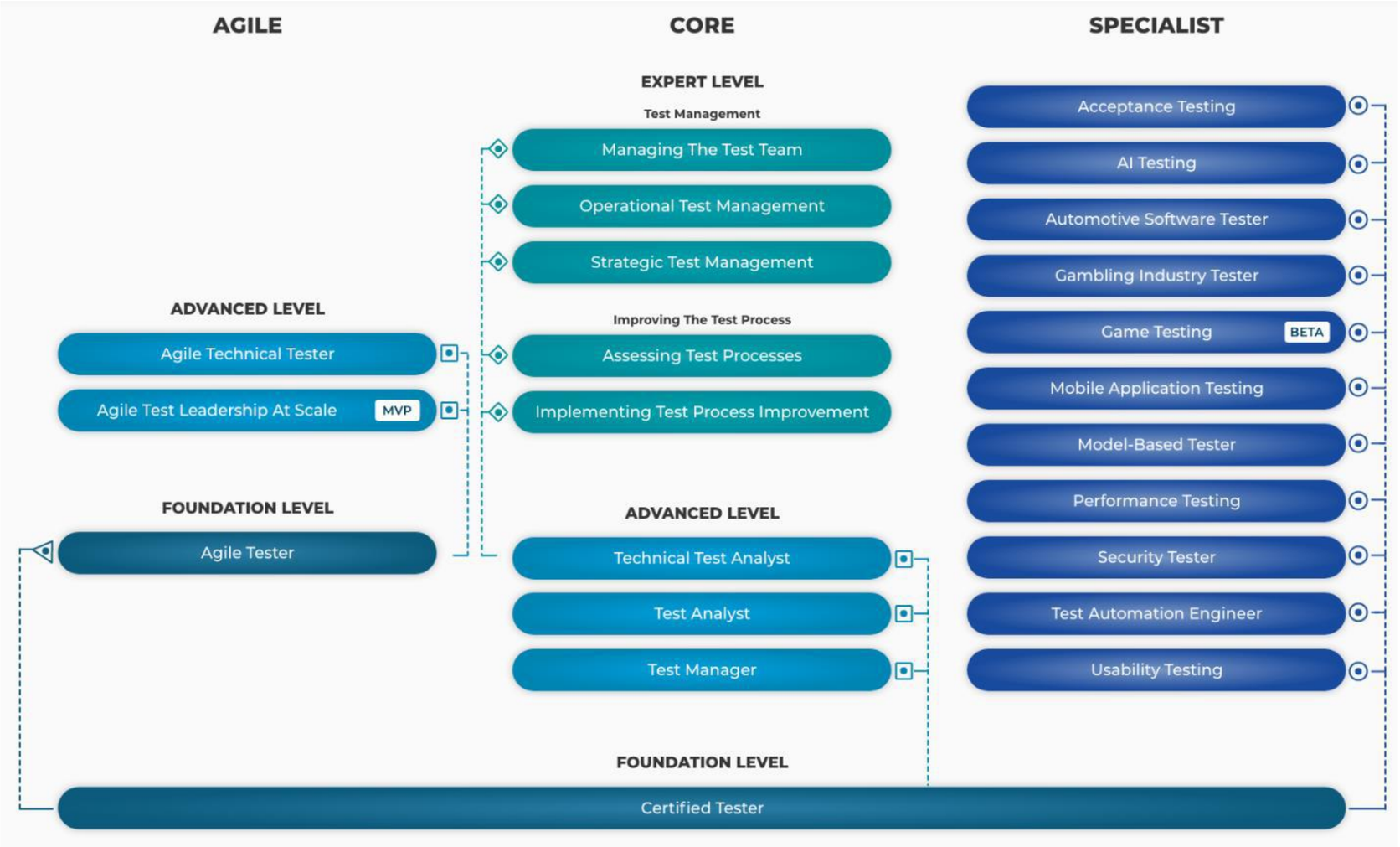
ISTQB® Poziom Podstawowy

Szkolenie akredytowane przez SJSI
zgodnie z wersją sylabusa ISTQB 2018 v.3.1

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

Informacje ogólne.

Wstęp
Ścieżka certyfikacji ISTQB®



Wstęp
Informacje ogólne

- 40 pytań jednokrotnego lub wielokrotnego wyboru (czyli jedno z czterech lub dwa z pięciu) opartych na celach nauczania podanych na początku każdego rozdziału
- żeby zdać egzamin, należy odpowiedzieć na co najmniej 65% pytań (tj. 26 pytań)
- każda poprawna odpowiedź to 1 punkt
- nie ma punktów ujemnych
- egzamin trwa 60 minut
- jeżeli język ojczysty kandydata nie jest językiem egzaminacyjnym, kandydatowi przysługuje dodatkowe 25% czasu

Wstęp
Cele nauczania objęte egzaminem

Poziomy wiedzy na temat poszczególnych celów nauczania są przedstawione na początku każdego rozdziału i są klasyfikowane w następujący sposób :

- **K1:** zapamiętać
- **K2:** zrozumieć
- **K3:** zastosować

Wstęp
Podział pytań wg poziomów K

Poziom	Liczba pytań	Czas przeznaczony na każde pytanie	Łączny czas przeznaczony na dany poziom K (dane uśrednione)
K1	8	1	8
K2	24	1	24
K3	8	3	24
Suma	**40**		**56**

Wstęp
Rozkład czasu

Rozdział	Czas
I. Podstawy testowania	175
II. Testowanie w cyklu życia oprogramowania	100
III. Testowanie statyczne	135
IV. Techniki testowania	330
V. Zarządzanie testami	225
VI. Narzędzia wspomagające testowanie	40

Wstęp
Podział Pytań

Rozdział	Pytania	Poziom
I. Podstawy testowania	8	K1 = 2, K2 = 6, K3 = 0
II. Testowanie w cyklu życia oprogramowania	5	K1 = 1, K2 = 4, K3 = 0
III. Testowanie statyczne	5	K1 = 1, K2 = 3, K3 = 1
IV. Techniki testowania	11	K1 = 1, K2 = 5, K3 = 5
V. Zarządzanie testami	9	K1 = 2, K2 = 5, K3 = 2
VI. Narzędzia wspomagające testowanie	2	K1 = 1, K2 = 1, K3 = 0

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

01

Podstawy testowania.

1. Podstawy testowania
Cele Nauczania

1.1 Co to jest testowanie?

- FL-1.1.1 (K1) Kandydat potrafi wskazać typowe cele testowania
- FL-1.1.2 (K2) Kandydat potrafi odróżnić testowanie od debugowania

1.2 Dlaczego testowanie jest niezbędne?

- FL-1.2.1 (K2) Kandydat potrafi podać przykłady wskazujące, dlaczego testowanie jest niezbędne
- FL-1.2.2 (K2) Kandydat potrafi opisać relację między testowaniem a zapewnieniem jakości
- FL-1.2.3 (K2) Kandydat potrafi rozróżnić pomyłkę, defekt i awarię
- FL-1.2.4 (K2) Kandydat potrafi odróżnić podstawową przyczynę od skutków defektu

1. Podstawy testowania
Cele Nauczania

1.3 Siedem zasad testowania

FL-1.3.1 (K2) Kandydat potrafi objaśnić siedem zasad testowania

1.4 Proces testowy

FL-1.4.1 (K2) Kandydat potrafi wyjaśnić wpływ kontekstu na proces testowy

FL-1.4.2 (K2) Kandydat potrafi opisać czynności testowe i odpowiadające im zadania

FL-1.4.3 (K2) Kandydat potrafi rozróżnić produkty pracy wspomagające proces testowy

FL-1.4.4 (K2) Kandydat potrafi wyjaśnić korzyści wynikające ze śledzenia powiązań między podstawą testów a produktami pracy związanymi z testowaniem

1. Podstawy testowania
Cele Nauczania

1.5 Psychologia testowania

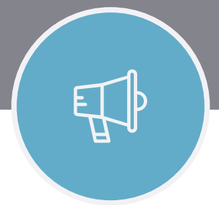
FL-1.5.1 (K1) Kandydat potrafi wskazać czynniki psychologiczne wpływające na powodzenie testowania

FL-1.5.2 (K2) Kandydat potrafi wyjaśnić różnice w sposobie myślenia testerów i programistów

Co to jest **testowanie**?

1. Podstawy testowania
1.1 Wprowadzenie

W dzisiejszych czasach systemy oprogramowania są prawie wszędzie.
Są integralną częścią naszego życia i nikt dziś w to nie wątpi.



Aplikacje biznesowe

…bankowość internetowa

Komunikacja

…komunikatory, e-maile, media społecznościowe

Produkty użytkowe

…w aucie, w domu, w pracy

Ochrona

…oprogramowanie antywirusowe, kopie zapasowe

1. Podstawy testowania
1.1 Wprowadzenie

Większość z nas pamięta oprogramowanie, które nie działało zgodnie z oczekiwaniami. Testowanie może pomóc w ocenie jakości produktu i może zmniejszyć ryzyko działającego oprogramowania.

Poprawa:		Utrata:	
jakości	✓	✗	klientów
niezawodności	✓	✗	czasu
wydajności	✓	✗	życia
komfortu pracy	✓	✗	reputacji biznesowej

1. Podstawy testowania
1.1 Wprowadzenie

TERMINY

testowanie: proces składający się z wszystkich czynności cyklu życia, zarówno statycznych jak i dynamicznych; skoncentrowany na planowaniu, przygotowaniu i ewaluacji oprogramowania oraz powiązanych produktów w celu określenia czy spełniają one wyspecyfikowane wymagania oraz wykazania, że są one dopasowane do swoich celów oraz do wykrywania usterek.

Testerzy są odpowiedzialni za testowanie

debugowanie: proces wyszukiwania, analizowania i usuwania przyczyn awarii oprogramowania.

Programiści są odpowiedzialni za debugowanie

1. Podstawy testowania
1.1 Wprowadzenie

Co to jest testowanie ?

Testowanie to nie tylko wykonanie testów. Może się wydawać, że jedyną aktywnością testerów jest wykonywanie testów, ale proces testowania jest czymś więcej. Chodzi także o projektowanie i analizowanie, planowanie, wdrażanie testów, a także przeglądanie produktów pracy, takich jak wymagania, projekty, historyjki użytkowników, kod źródłowy, plany testów, raporty itp.

Więcej szczegółów na temat procesu testowego opisano w części 1.4.

1. Podstawy testowania
1.1 Wprowadzenie

Weryfikacja kontra Walidacja

weryfikacja: egzaminowanie poprawności i dostarczenie obiektywnego dowodu, że produkt procesu wytwarzania oprogramowania spełnienia zdefiniowane wymagania.

→ **Czy tworzymy produkt poprawnie?**

walidacja: sprawdzanie poprawności i dostarczenie obiektywnego dowodu, że produkt procesu wytwarzania oprogramowania spełnienia potrzeby i wymagania użytkownika.

→ **Czy tworzymy właściwy produkt?**

1. Podstawy testowania
1.1 Wprowadzenie

Typowe nieporozumienia

Wykonywanie testów

Nie chodzi tylko o wykonanie testu. Jest to najbardziej widoczna aktywność, ale nie jedyna.

Nie testujemy bez oprogramowania

Testy bez oprogramowania są możliwe. Koncentrujemy się na weryfikacji wymagań, historyjek użytkowników i innych specyfikacji.

Sprawdzanie wymagań

Skupiamy się na jakości oprogramowania, nie tylko na spełnieniu wymagań.

Poprawność produktu

Testy obejmują weryfikację systemu, a także walidację - czy aplikacja spełnia potrzeby użytkownika.

1. Podstawy testowania
1.1.1 Typowe cele testowania

Cele testowania mogą obejmować : [1/2]

Sprawdzanie i walidowanie

czy obiekt testowy jest kompletny i działa tak, jak oczekuje tego użytkownik i inni interesariusze.

Budowanie zaufania

odnośnie poziomu jakości testowanego produktu.

Weryfikacja

czy wszystkie wyspecyfikowane wymagania zostały spełnione

Zapobieganie defektom

poprzez dokonywanie oceny produktów pracy, takich jak: wymagania, historyjki użytkownika, projekt i kod

1. Podstawy testowania
1.1.1 Typowe cele testowania

Cele testowania mogą obejmować : [2/2]

Wykrywanie defektów

oraz obserwowanie awarii, a tym samym zmniejszenie poziomu ryzyka związanego z niedostateczną jakością oprogramowania

Dostarczanie informacji

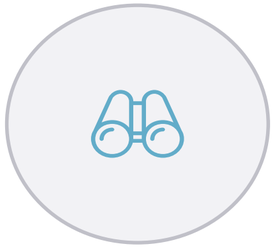
zainteresowanym stronom, aby mogły podejmować świadome decyzje, w szczególności w odniesieniu do poziomu jakości obiektu testowego.

Przestrzeganie standardów

przepisów prawa lub regulacyjnych i weryfikacji zgodności testowanego produktu z takimi wymaganiami lub standardami.

1. Podstawy testowania
1.1.1 Typowe cele testowania

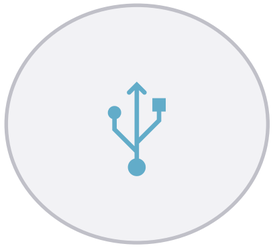
Wszystkie cele testowania wymienione na poprzednim slajdzie są poprawne, ale nie wszystkie są obecne na każdym poziomie testowania. Wszystko zależy od modelu cyklu rozwoju oprogramowania i od kontekstu testowanego systemu (SUT).



Powodować tyle defektów, ile to możliwe

Nabieranie zaufania

Ocena charakterystyki produktu



Sprawdzenie jak system będzie działał po zmianie

1. Podstawy testowania
1.1.1 Typowe cele testowania

Testowanie modułowe

Spowodować tyle awarii, ile jest to możliwe, i w rezultacie naprawić je wcześniej.

Testowanie akceptacyjne

Aby uzyskać pewność co do jakości testowanego oprogramowania i dostarczyć informacji o ryzyku związanym z wydaniem produktu.

1. Podstawy testowania
1.1.1 Typowe cele testowania

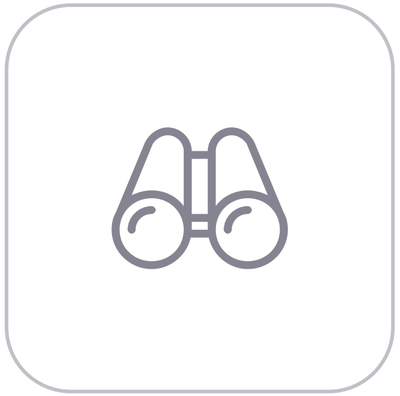
Testowanie pielęgnacyjne

Testowanie zmian we wdrożonym systemie lub testowanie wpływu zmienionego środowiska na wdrożony system.

Testowanie operacyjne

W celu oceny cech produktu, takich jak wydajność, dostępność, przenaszalność.

1. Podstawy testowania
1.1.2 Testowanie i debugowanie



testowanie

Pozwala na znajdowanie awarii spowodowanych defektami

debugowanie

Wyszukanie, analiza i naprawa istniejącej usterki

testy potwierdzające

Aby sprawdzić, czy poprawka naprawiła defekt

1. Podstawy testowania
1.1.2 Testowanie i debugowanie

PRZYKŁAD

Rozważmy rozwiązanie e-commerce. Klient kupując produkty za ponad 1000 USD otrzymuje 7% zniżki. Produkty zostały zakupione na łączną kwotę 2000$.

1 → 2 → 3

wykonywanie testów

Podczas testów zauważyliśmy, że rabat nie został naliczony, utworzyliśmy raport o defekcie

debugowanie

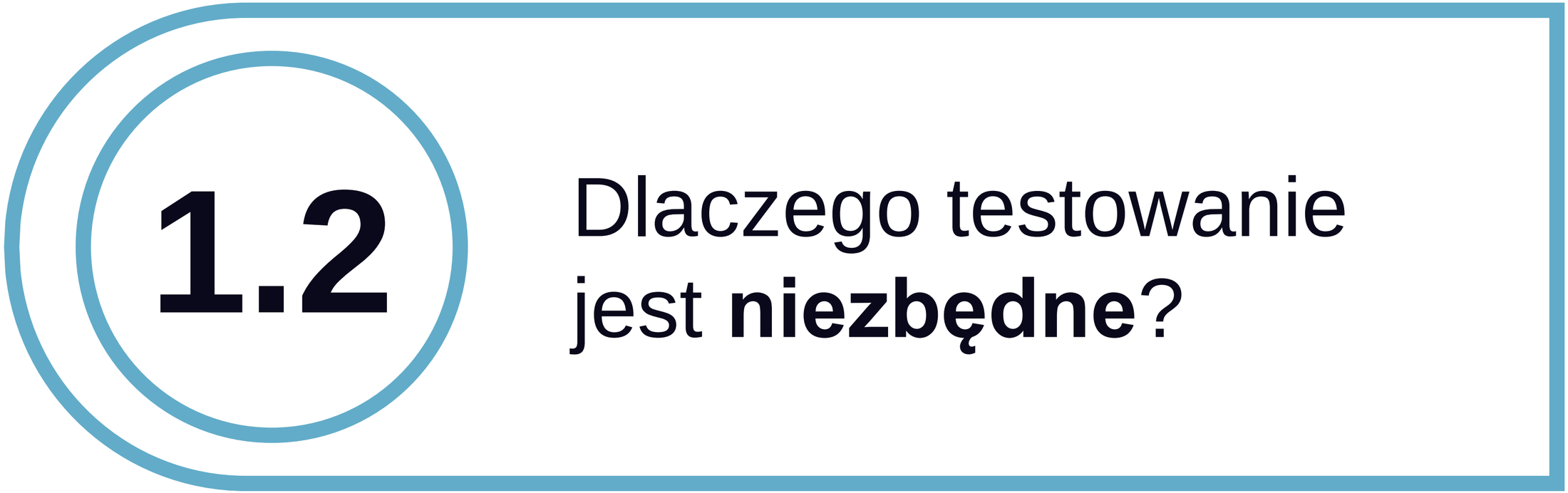
Deweloper przypisany do defektu, próbuje znaleźć, przeanalizować i naprawić defekt

retestowanie

Defekt jest naprawiony. Sprawdzamy, czy rabat jest obliczony poprawnie, wykonując ten sam test

1. Podstawy testowania
1.1.2 Testowanie i debugowanie

- W niektórych przypadkach test początkowy i końcowy test potwierdzający są wykonywane przez testerów, a programistom powierzane jest debugowanie odpowiedniego modułu i zadania związane z ciągłą integracją
- W środowisku Agile mogą to być testerzy biorący udział w debugowaniu i testowaniu modułów.


1.2
Dlaczego testowanie jest **niezbędne**?

1. Podstawy testowania
1.2 Wprowadzenie

Dlaczego testowanie jest niezbędne?

Rygorystyczne testowanie

Zmniejszamy ryzyko wystąpienia awarii występujących podczas eksploatacji

Podnoszenie jakości

Kiedy defekt zostanie wykryty, a następnie naprawiony, przyczynia się to do podniesienia jakości systemu

Normy i standardy branżowe

Testowanie może być wymagane w celu spełnienia wymagań wynikających z norm lub standardów branżowych

1. Podstawy testowania
1.2 Wprowadzenie

PRZYKŁADY

Rygorystyczne testowanie

Rygorystyczne testowanie modułów i systemów a także związanej z nimi dokumentacji może pomóc w zmniejszeniu ryzyka wystąpienia awarii podczas eksploatacji oprogramowania. Np. dzięki sprawdzeniu czy generowanie raportów działa poprawnie, możemy zmniejszyć ryzyko związane z tym obszarem już po wdrożeniu.

Podnoszenie jakości

Samo testowanie nie poprawia jakości, jakość wzrasta, gdy defekty są naprawiane, np. stwierdzenie, że "testowanie wykazało obecność 15 defektów i zostały one wyeliminowane" jest miarą jakości, i możemy powiedzieć, że jakość poprawiła się poprzez naprawienie tych wad defektów.

Normy i standardy branżowe

Wyobraź sobie, że nasz produkt ma być sprzedawany na rynku amerykańskim i musimy przestrzegać standardów na rynku amerykańskim. Np. w przypadku oprogramowania stosowanego w medycynie musimy być zgodni ze standardem FDA (Regulacja systemu jakości (QS) 21 CFR część 820) co spowoduje konieczność odpowiedniego dokumentowania naszego procesu testowego i przetwarzanie dokumentacji w sposób zgodny ze standardem FDA.

1. Podstawy testowania

1.2.1 Znaczenie testowania dla powodzenia projektu

Co możemy z tym zrobić?

Istnieje wiele znanych przykładów oprogramowania i systemów, które zostały przekazane do eksploatacji, ale na skutek defektów uległy awarii lub z innych powodów nie zaspokoiły potrzeb interesariuszy.

Aby zmniejszyć ryzyko związane z dostarczaniem oprogramowania o niskiej jakości, możemy przetestować je poprawnie dzięki zastosowaniu odpowiednich technik testowania, na odpowiednim poziomie testowania i we właściwych fazach cyklu życia oprogramowania.

1. Podstawy testowania
1.2.1 Znaczenie testowania dla powodzenia projektu

Jak testowanie przyczynia się do sukcesu? [1/2]

Wczesne testowanie

Testerzy mogą brać udział w przeglądach dokumentacji i udoskonalaniu historyjek użytkowników. Usterki wykryte na początku cyklu wytwarzania oprogramowania zmniejszają ryzyko zaimplementowania nieprawidłowej funkcjonalności, której być może nie da się przetestować.

Współpraca z architektami systemu

Lepsze zrozumienie produktu. W ten sposób możemy zmniejszyć ryzyko elementarnych wad projektowych.

Współpraca z programistami

Lepsze zrozumienie kodu i efektywne jego testowanie. Mając to na uwadze możemy zmniejszyć ryzyko wystąpienia defektów w kodzie i spowodowanych przez nie awarii w testach.

1.2.1 Znaczenie testowania dla powodzenia projektu

Jak testowanie przyczynia się do sukcesu? [2/2]

Weryfikacja i walidacja

Testerzy są zaangażowani w weryfikację i walidację oprogramowania i dzięki temu mogą pomóc wyeliminować awarie przed przekazaniem produktu do eksploatacji. W przeciwnym razie takie awarie mogłyby zostać przeoczone. Testowanie może również wspomagać czynności związane z debugowaniem. Co więcej, testowanie może zwiększyć prawdopodobieństwo, że oprogramowanie spełnia potrzeby biznesowe.

Ogólne podejście

Testowanie przyczynia się do ogólnego powodzenia procesu wytwarzania i pielęgnacji oprogramowania.

1. Podstawy testowania
1.2.2 Zapewnienie jakości a testowanie

Czy QA = testowanie ?

To nie dokładnie to samo, ale
jest ze sobą powiązane

Zarządzanie jakością to szersza koncepcja, która łączy je wszystkie i obejmuje **wszystkie działania** w danej organizacji pod względem jakości.

Testowanie

Kontroluje czy został osiągnięty odpowiedni poziom jakości. Jest to **tylko część całego procesu** wytwarzania i pielęgnacji oprogramowania.

Zapewnienie jakości

Skupia się na śledzeniu procesów i standardowych procedur operacyjnych. Celem jest uzyskanie pewności, że osiągnęliśmy odpowiedni poziom jakości. Im lepiej śledzimy procesy, tym wyższa jakość produktu (zapobieganie defektom). Dodatkowo analiza przyczyn źródłowych (wykrywanie i usuwanie przyczyn defektów) oraz retrospekcyjne spotkania skoncentrowane na zastosowaniu wyników są kluczem do sukcesu w zakresie zapewniania jakości.

Zapewnienie jakości dotyczy całego procesu. Wspiera testowanie,
bo testowanie przyczynia się do osiągnięcia wymaganej jakości produktów pracy.

1. Podstawy testowania
1.2.3 Pomyłki, defekty i awarie

1. Pomyłka

Ludzki błąd, który może doprowadzić do nieoczekiwanego wyniku.

3. Awaria

System nie robi tego, co powinien i jest to widoczne dla użytkownika końcowego.

2. Defekt (usterka, pluskwa)

Efekt nieprawidłowej implementacji programisty. Kod został wdrożony na podstawie dokumentacji z błędem. Usterka może (ale nie musi) spowodować awarię.

1. Podstawy testowania
1.2.3 Pomyłki, defekty i awarie

PRZYKŁADY

Wymaganie do systemu e-commerce	*„Jeśli klient podczas płatności online ma w swoim koszyku produkty kategorii F, to jest proszony o zakup dodatkowego produktu z kategorii H”.*
Pomyłka	Niezrozumienie przez analityka biznesowego jak należy określać kategorię uzupełniającą produkty w koszyku klienta.
Defekt	Określone wymaganie jest nieprawidłowo zaimplementowane, a niewłaściwa kategoria produktów H jest pokazywana jako produkty powiązane z kategorią F.
Awaria	Zgłoszono awarię z rynku. Klienci skarżą się, że po wybraniu produktów kategorii F nie ma możliwości ukończenia zakupu – aplikacja wyświetla komunikat o nieznanym kodzie błędu i otwiera się główna strona sklepu online.

1. Podstawy testowania
1.2.3 Pomyłki, defekty i awarie

Pamiętaj o tym, że

Jeden błąd, wiele produktów

Pomyłka, która doprowadzi do defektu w jednym produkcie roboczym, może skutkować wprowadzeniem defektu w innym produkcie roboczym.

Błąd … awaria

Defekt w wymaganiu może przełożyć się na usterkę w kodzie i ostatecznie doprowadzić do awarii.

Defekt to nie awaria

Defekt może spowodować awarię, ale nie zawsze. System może wymagać określonego środowiska, konfiguracji lub wartości wejściowych, aby zakończyć się niepowodzeniem.

Warunki środowiskowe

Awarie mogą być spowodowane warunkami środowiskowymi (np. niską temperaturą, promieniowaniem, wilgotnością, polem elektromagnetycznym).

1. Podstawy testowania
1.2.3 Pomyłki, defekty i awarie

Przyczyny powstawania pomyłek

Niedostateczna wymiana informacji

Nieporozumienia dotyczące dokumentacji.

Złożoność oprogramowania

Złożony kod, projekt, architektura, kwestia "niskiego wysiłku" staje się trudna.

Interfejsy pomiędzy systemami

Nieporozumienia dotyczące interfejsów w systemie i między innymi systemami, zwłaszcza gdy tych systemów jest wiele.

Presja czasu i nowe technologie

Pracujemy pod presją czasu (terminy) i musimy podążać za zmieniającymi się technologiami.

Ludzie są omylni

Popełniamy błędy.

Braki w umiejętnościach

Rotacja personelu między firmami lub projektami w firmie.

1. Podstawy testowania
1.2.3 Pomyłki, defekty i awarie

Błędna interpretacja wyniku testu

Rezultat fałszywie pozytywny
(false-positive result)

Test, w którym defekt został zaraportowany, chociaż defekt ten nie występuje.

→ Np. awaria wynikająca z niepoprawnego środowiska testowego

Rezultat fałszywie negatywny
(false-negative result)

Test, w którym nie zidentyfikowano obecności usterki występującej w testowanym obiekcie.

→ Np. niejednoznaczna interpretacja wymagań przez dewelopera i testera

1. Podstawy testowania
1.2.4 Defekty, podstawowe przyczyny oraz skutki

TERMINY

analiza przyczyny podstawowej: Technika analizy zorientowana na identyfikację podstawowych przyczyn defektów. Przez wprowadzenie miar ukierunkowanych na podstawowe przyczyny defektów, liczy się, że prawdopodobieństwo ponownego wystąpienia defektu będzie zminimalizowane.

podstawowa przyczyna: przyczyna defektu, która – gdy zostanie wyeliminowana – wystąpienie tego typu defektu redukuje lub usuwa.

1.2.4 Defekty, podstawowe przyczyny oraz skutki

Podstawowe przyczyny defektów

- ✓ Skupiają się na pierwotnych przyczynach defektu.
- ✓ Mogą być przeanalizowane w celu zmniejszenia prawdopodobieństwa wystąpienia podobnych defektów w przyszłości.
- ✓ Mogą prowadzić do usprawnienia procesu poprzez wyeliminowanie najważniejszych przyczyn źródłowych.

1.2.4 Defekty, podstawowe przyczyny oraz skutki

Przykład analizy przyczyny podstawowej : [1/2]

Przykład użycia

Organizacja, w której pracujesz zbiera dane o wykorzystaniu swojego produktu i przechowuje je w chmurze. Z powodu drobnego błędu w konfiguracji danych, które są gromadzone w chmurze organizacja przestała otrzymywać wiarygodne informacje na temat wykorzystania oprogramowania. Skutkuje to brakiem możliwości monitorowania bieżącego wykorzystania oprogramowania. Niepoprawna konfiguracja została przygotowana na podstawie informacji od programisty, który nie pracował wcześniej w takiej architekturze. W przypadku dalszego rozwijania rozwiązania, które ma na celu wykorzystanie zasobów z chmury konieczne będzie zapewnienie szkolenia dla osób wdrażających taki produkt. Zmniejszy to możliwość występowania podobnych defektów w przyszłości.

Zidentyfikuj defekt, podstawową przyczynę i skutek

1.2.4 Defekty, podstawowe przyczyny oraz skutki

Przykład analizy przyczyny podstawowej : [2/2]

Defekt

Nieprawidłowa konfiguracja danych przechowywanych w chmurze.

Przyczyna podstawowa

Brak wiedzy; jedynie przypuszczenia co do poprawnej konfiguracji rozwiązania.

Skutek

Brak wiarygodnych informacji na temat wykorzystania oprogramowania.


1.3
Siedem zasad testowania

1. Podstawy testowania
1.3 Siedem zasad testowania

Zasady testowania są ogólnymi wytycznymi,
które są wspólne dla wszystkich testów.

1. Testowanie ujawnia usterki, ale nie może dowieść ich braku
2. Testowanie gruntowne jest niemożliwe
3. Wczesne testowanie oszczędza czas i pieniądze
4. Kumulowanie się defektów
5. Paradoks pestycydów
6. Testowanie zależy od kontekstu
7. Przekonanie o braku błędów jest błędem

1. Podstawy testowania
1.3 Siedem zasad testowania

Testowanie ujawnia usterki, ale nie może dowieść ich braku

Możemy pokazać, że defekty są w oprogramowaniu, ale nie możemy udowodnić, że oprogramowanie nie ma defektów. Testowanie zmniejsza prawdopodobieństwo, że w oprogramowaniu pozostaną niewykryte defekty, a także koncentruje się na wykrywaniu awarii i ich zapobieganiu po wdrożeniu oprogramowania.

1. Podstawy testowania
1.3 Siedem zasad testowania

PRZYKŁAD

Wykonaliśmy wszystkie testy i nie ma żadnych defektów – czy to poprawne stwierdzenie ?

Kierownik testów mówi, że zespół wykonał 1000 testów i nie znalazł żadnego defektu. Na tej podstawie stwierdza, że w produkcie nie ma żadnego defektu. To jest oczywiście nieprawidłowe stwierdzenie.

1.3 Siedem zasad testowania

Testowanie gruntowne jest niemożliwe

Przetestowanie wszystkiego jest możliwe, ale tylko w trywialnych przypadkach. W przeważającej większości systemów, które testujemy, nie można przetestować wszystkich kombinacji. Musielibyśmy sprawdzić wszystkie wyniki decyzji, wszystkie konfiguracje, sekwencje testów itd. Nawet, gdy przetestujemy każdą znaną kombinację to następnego dnia, może pojawić się nowa nieznana dotąd konfiguracja, zmiana środowiska lub warunki testu, których jeszcze nie sprawdziliśmy.

1.3 Siedem zasad testowania

PRZYKŁAD

Wszystko już przetestowane – czy to poprawne stwierdzenie ?

Rozważmy przetestowanie formularza gdzie podajemy numer telefonu oraz kod doładowania prepaid, który jest przesyłany w postaci SMSa. Numer telefonu to 9 cyfr a kod doładowania jest w postaci 8 znaków (litery lub cyfry bez polskich liter oraz znaków specjalnych). Można więc wpisać 10^9 różnych numerów telefonicznych oraz 8^{36} różnych kodów. W sumie mamy 10^9 x 8^{36}. Chcąc przetestować to gruntowanie musielibyśmy sprawdzić wszystkie możliwe dane wejściowe i warunki, które mają wpływ na system. Czy będziemy to robić ręcznie? Ile czasu nam to zajmie? Przetestowanie wszystkich kombinacji wejść i warunków początkowych jest wykonalne ale tylko w trywialnych przypadkach.

1.3 Siedem zasad testowania

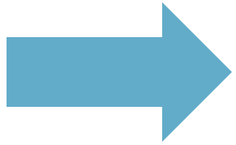
Wczesne testowanie oszczędza czas i pieniądze

Testowanie powinno się rozpocząć znacznie wcześniej, niż dopiero w chwili, gdy kod i interfejs użytkownika są ukończone. Im szybciej odkryjemy defekty, tym bardziej jest to opłacalne. Wyobraźmy sobie, że znajdujemy defekt podczas testowania na poziomie systemu. Mogłoby to wymagać modyfikacji zarówno testów na poziomie integracji, modułu, kodu, zmiany projektu i analizy.

Dlatego testerzy powinni brać udział w statycznych testach, takich jak: przeglądy dokumentacji, projektu, dowolnej podstawy testów itp.

1.3 Siedem zasad testowania

PRZYKŁAD



Czy możemy sobie pozwolić na znalezienie defektu na samym końcu procesu testowego?

Nasza aplikacja podlega odrębnym regulacjom wynikającym ze standardu FDA. Aby móc sprzedawać nasze produkty, musimy udowodnić, że insulina jest prawidłowo dawkowana przez nasze oprogramowanie, nie powodując szkód dla chorych (użytkowników końcowych). Podczas testów akceptacyjnych klient zauważa, że algorytm dawkowania insuliny działa nieprawidłowo, bo nieprawidłowo wylicza dawkowanie w pewnych sytuacjach.

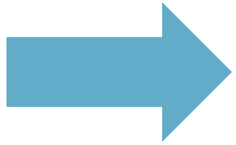
Defekt znaleziony dopiero podczas testów akceptacyjnych wymusza zmianę w podstawowym module, co pociąga za sobą testy regresji dużej części systemu.

1.3 Siedem zasad testowania

Kumulowanie się defektów

Uznaje się, że niewielka liczba komponentów zwykle zawiera większość defektów wykrytych przed wydaniem. Jeśli nasz produkt jest wdrożony z defektami, to mówimy, że wynikiem tego mogą być awarie występujące w fazie eksploatacji. W tym miejscu należy wziąć pod uwagę zasadę Pareto [80/20]. Większość defektów jest skupiona wkoło jednej funkcjonalności, ponieważ wykorzystywane algorytmy mogą być trudne do wdrożenia, stale zmieniają się technologie, mamy rotację personelu, krótkie terminy itp.

PRZYKŁAD



Gdzie są defekty?

Testujemy rozwiązanie e-commerce.

Podczas testowania systemowego zgłoszono sporo defektów w obszarze wydajności. Zgłoszone defekty zostały naprawione. Następnie przeprowadziliśmy analizę ryzyka i w wyniku tego, podczas testowania integracji systemów (nasze rozwiązanie z zewnętrzną obsługą płatności online) zdecydowano aby położyć duży nacisk na testy wydajności.

Testy zostały ukierunkowane na podstawie faktycznie zaobserwowanych skupisk defektów znalezionych podczas testowania systemowego.

Paradoks pestycydów

Pestycydy używane do ochrony roślin po pewnym czasie przestają być skuteczne i można powiedzieć, że to samo dzieje się podczas testowania. Testy nie powinny powtarzać w koło tego samego schematu. Znacznie mniej defektów można znaleźć, jeśli powtarzamy te same testy, z tymi samymi danymi wejściowymi i w tym samym środowisku. Właśnie dlatego, aby skutecznie wykrywać defekty, powinniśmy stale modyfikować testy. Jest to kluczowe przy rozważaniu automatycznych testów regresji. Paradoks pestycydów może być także korzystny – automatyczne testowanie regresji może potwierdzić niewielką liczbę defektów związanych z regresją.

1. Podstawy testowania
1.3 Siedem zasad testowania

PRZYKŁAD

Rozwijamy nasz produkt w długoterminowym projekcie z 3 wdrożeniami rocznie

Wykorzystujemy zestaw testów, który zaprojektowano dwa lata temu. Opisywane testy są w użyciu od tej pory i nie znajdują już żadnych defektów. W międzyczasie opracowano nowe funkcjonalności. Na podstawie danych z rynku widać, że defekty mogłyby zostać znalezione, gdyby istniejące "stare" scenariusze (zaprojektowane dwa lata temu) zostały zmodyfikowane.

Nie stosuj tego samego schematu - modyfikuj dane testowe, kolejność itp.

1.3 Siedem zasad testowania

Testowanie zależy od kontekstu

Sposób, w jaki testujemy i techniki, których używamy, zależą od kontekstu. Rozwiązanie e-commerce będzie testowane inaczej niż system krytyczny pod względem bezpieczeństwa lub taki, który musi spełniać odrębne przepisy branżowe. Z drugiej strony testowanie może się różnić pomiędzy projektami prowadzonymi w sposób zwinny od tych prowadzonych w sekwencyjnym modelu wytwarzania.

1. Podstawy testowania
1.3 Siedem zasad testowania

PRZYKŁAD

Testowanie zależy od kontekstu

Jeśli weźmiemy pod uwagę rozwiązanie e-commerce, możemy skupić się przykładowo na testach wydajności, użyteczności lub dostępności.

Podczas rozważania systemu krytycznego ze względu na poziom bezpieczeństwa, który musi spełniać surowe przepisy, raczej skupimy się na bezpieczeństwie, niezawodności i zgodności z wymaganymi przepisami.

1.3 Siedem zasad testowania

Przekonanie o braku błędów jest błędem

Dobra weryfikacja nie oznacza dobrej walidacji. Innymi słowy, możemy znaleźć wszystkie możliwe usterki i wykonać wszystkie potrzebne testy i udowodnić, że złagodziliśmy wszystkie powiązane ryzyka, ale niekoniecznie oznacza to, że dostarczamy w pełni funkcjonalny system, który spełnia oczekiwania użytkowników.

Ta zasada powinna się nazywać „Błędne przekonanie o braku defektów”, tym niemniej zdecydowano się zostawić „Przekonanie o braku błędów jest błędem” jako najbliższe angielskiemu „Absence-of-errors fallacy”.

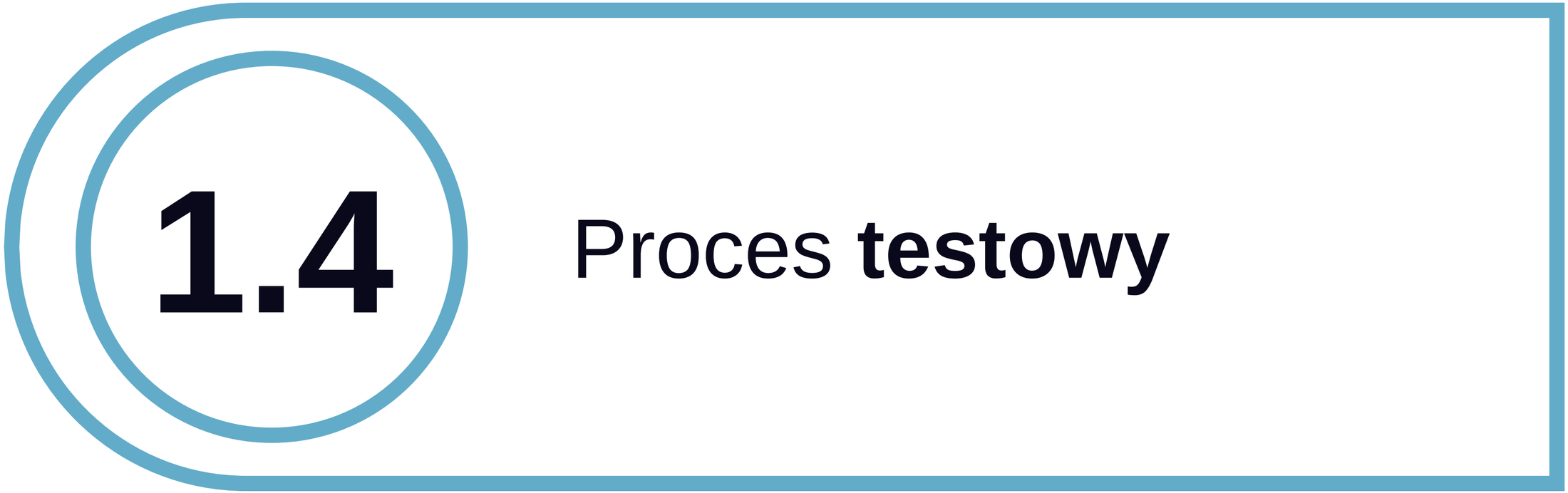
1. Podstawy testowania
1.3 Siedem zasad testowania

PRZYKŁAD

Znalezione defekty kontra wartość biznesowa

Rozważmy internetową platformę wymiany walut. Przeprowadziliśmy różnorodne testy i z radością informujemy, że naprawiliśmy większość znalezionych usterek.

Wydany produkt spełnia wysokie standardy jakości, ale okazuje się, że wydajność platformy jest poniżej oczekiwań klientów i nikt nie chce z niej korzystać.


1.4
Proces testowy

1. Podstawy testowania
1.4 Wprowadzenie

Proces testowania zależy od wielu czynników. Nie ma jednego, uniwersalnego procesu testowania oprogramowania. Zestawy czynności testowych opisane w tym rozdziale są rozumiane jako proces testowy. Jest to strategia testowa, którą każda organizacja może rozważyć w celu opisania, w które działania testowe są zaangażowane oraz w jaki sposób są wdrażane i kiedy występują.

1. Podstawy testowania
1.4.1 Proces testowy w kontekście

Co wpływa na proces testowy w organizacji ? [1/2]

01	02	03	04
Model wytwarzania oprogramowania i stosowane metodologie projektowe	Uwzględniane poziomy i typy testów	Ryzyko związane z produktem i projektem	Dziedzina biznesowa

1. Podstawy testowania
1.4.1 Proces testowy w kontekście

Co wpływa na proces testowy w organizacji ? [2/2]

05	06	07
Ograniczenia operacyjne, w tym: budżety i zasoby, skala czasowa, złożoność, wymagania prawne i kontraktowe	Polityka testów i praktyki w danej organizacji	Wymagane normy wewnętrzne i zewnętrzne

1. Podstawy testowania
1.4.1 Proces testowy w kontekście

PRZYKŁADY

model wytwarzania oprogramowania	W iteracyjnym modelu Agile/Scrum czynności testowe mogą odbywać się inaczej niż np. w modelu sekwencyjnym.
małe aplikacje	Mała aplikacja mobilna, która konwertuje temperaturę, może nie wymagać złożonego zestawu testów i wielu poziomów testowych. Wykonany może być jedynie zestaw testów modułowych. Nie zawsze jest konieczność wykonania testów na wszystkich poziomach.
zewnętrzni konsultanci	Testowanie konkretnego produktu może zmusić nas do skonsultowania naszego rozwiązania z zewnętrznymi ekspertami domenowymi, aby móc zaprojektować i wdrożyć dane rozwiązanie.
umowy/standardy	Nasze rozwiązanie musi uwzględniać specyficzne przepisy (np. bezpieczeństwa), które należy spełnić, żeby móc wejść na dany rynek.

1.4.1 Proces testowy w kontekście

Podstawa testów, a kluczowe wskaźniki wydajności (KPI)

Pokrycie KPI

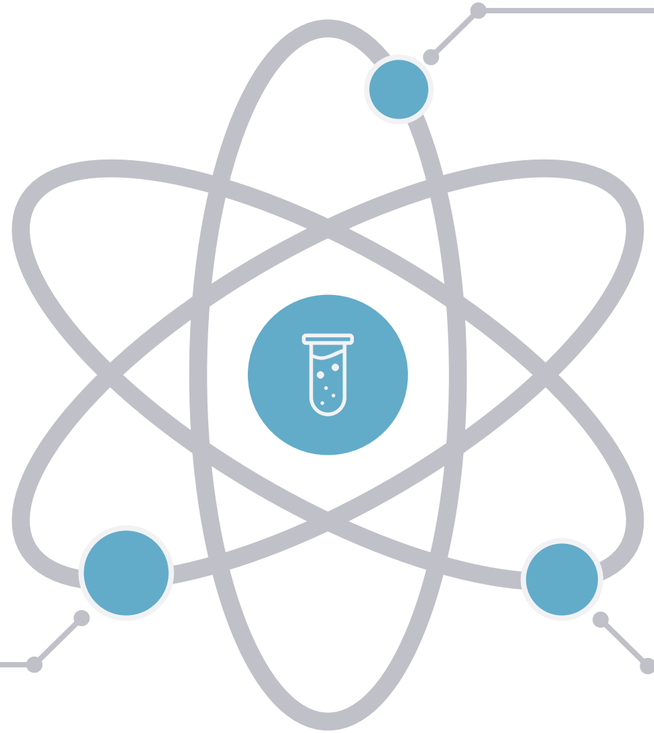
Podczas definiowania podstawy testów bardzo przydatne jest odniesienie się do tego jako mierzalnego kryterium pokrycia. Takie kryteria pokrycia można uznać za kluczowe wskaźniki wydajności, które wykazują osiągnięcie celów testowych.

Przykład

Aplikacja mobilna, która jest obsługiwana na wielu platformach (iOS, Android itd.). Każda z nich jest elementem podstawy testów (osobnym wymaganiem). Ponieważ wszystkie wymagania są objęte testami, po ich wykonaniu wyniki mogą jasno potwierdzić, że wszystkie urządzenia są obecnie obsługiwane.

1. Podstawy testowania
1.4.1 Proces testowy w kontekście

Procesy testowe można opisać w następujący sposób:



Produkty pracy związane z testowaniem

Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

Czynności i zadania testowe

1. Podstawy testowania

1.4.2 Czynności i zadania testowe

W obrębie procesu testowego można wyróżnić następujące grupy czynności:

- Planowanie testów
- Monitorowanie testów i nadzór nad testami
- Analiza testów
- Projektowanie testów
- Implementacja testów
- Wykonywanie testów
- Ukończenie testów

1. Podstawy testowania
1.4.2 Czynności i zadania testowe

- ✓ Każda grupa aktywności jest opisana osobno.
- ✓ Wszystkie składają się z oddzielnych zadań, które mogą nie być takie same we wszystkich rodzajach projektów lub wdrożeń.
- ✓ Należy zauważyć, że niezależnie od tego ile z tych działań wydaje się logicznie sekwencyjnych, mogą one być wdrażane równolegle.
- ✓ Nawet w czystym sekwencyjnym modelu rozwoju oprogramowania zadania te mogą nakładać się lub pojawiać się jednocześnie.
- ✓ Dostosowanie głównych działań w ramach procesu testowego ma kluczowe znaczenie dla powodzenia projektu.

1.4.2 Czynności i zadania testowe

Planowanie testów

- Określenie podejścia, aby osiągnąć zdefiniowane cele testowania
- Określenie technik i zadań testowych
- Formułowanie harmonogramu testów

- Plan testów może zostać skorygowany na podstawie informacji z monitorowania i działań kontrolnych

Zostało to wyjaśnione dokładniej w sekcji 5.2

1. Podstawy testowania
1.4.2 Czynności i zadania testowe

TERMINY

monitorowanie testów: zadanie zarządzania testami, które zajmuje się działaniami związanymi z okresowym monitorowaniem statusu aktywności testowych, identyfikowaniem odchyleń od planu lub oczekiwanego statusu oraz raportowaniem statusu do interesariuszy.

nadzór nad testami: zadanie z zakresu zarządzania testami, którego celem jest opracowanie i zastosowanie działań korygujących projekt testowy, kiedy monitorowanie pokazuje odchylenie od planu.

1.4.2 Czynności i zadania testowe

Monitorowanie testów i nadzór nad testami

- ✓ Oba działania są związane z kryteriami wyjścia (DoD – definicja ukończenia)
- ✓ Czy spełniliśmy określone kryteria pokrycia?
- ✓ Czy poziom jakości jest zapewniony?
- ✓ Czy potrzebujemy więcej testów?
- ✓ Rezultatem monitorowania i kontroli są istotne informacje przekazane zainteresowanym stronom w celu podjęcia decyzji o zakończeniu testowania.

1. Podstawy testowania
1.4.2 Czynności i zadania testowe

TERMINY

podstawa testów: zasób wiedzy używany jako podstawa dla analizy i projektowania testów.

warunek testowy: aspekt podstawy testów, który jest istotny dla osiągnięcia określonych celów testowych.

przypadek testowy: zestaw warunków wstępnych, danych wejściowych, akcji (w stosownych przypadkach), oczekiwanych rezultatów i warunków końcowych opracowany w oparciu o warunki testowe.

testalia: produkty prac stworzone w ramach procesu testowego używane do planowania, projektowania, wykonywania, oceny i raportowania testów.

Analiza testów [1/3]

- ✓ Ta aktywność polega na analizie testowanych funkcjonalności i zdefiniowaniu określonych warunków testowych. Analizujemy "CO" przetestować.
- ✓ Przydaje się tu stosowanie technik białoskrzynkowych, czarnoskrzynkowych lub opartych na doświadczeniu. Zmniejszamy wtedy ryzyko niedostatecznie określonych warunków testowych i szanse na ich pominięcie.
- ✓ W testowaniu eksploracyjnym karta opisu testów odzwierciedla zdefiniowany warunek testowy (cel testu)

Analiza testów [2/3]

- ✓ Wykrycie defektów podczas analizy może występować np. w technikach takich jak wytwarzanie sterowane zachowaniem (ang. Behavior Driven Development — BDD) i wytwarzanie sterowane testami akceptacyjnymi (ang. Acceptance Test Driven Development — ATDD). Obie techniki obejmują generowanie warunków testowych i przypadków testowych na podstawie historyjek użytkownika i kryteriów akceptacji przed rozpoczęciem tworzenia kodu.

Analiza testów [3/3]

Główne aktywności analizy testów:

- ✓ Analizowanie podstawy testów stosownie do rozważanego poziomu testu (sprawdzenie wymagania, projektu, kodu, identyfikowane ryzyka).
- ✓ Ocena podstawy testów i elementów testowych w celu sprawdzenia, czy potrafimy zidentyfikować testowalne warunki testowe (czy wymagania są jednoznaczne, spójne i dokładne).
- ✓ Identyfikowanie funkcjonalności i zestawów funkcjonalności do przetestowania.
- ✓ Definiowanie warunków testowych dla każdej funkcji i ustalenie ich kolejności.
- ✓ Inicjowanie śledzenia (wymaganie <—> warunek testowy).

Projektowanie testów [1/2]

- ✓ Warunki testowe są przekształcane w przypadki testowe wysokiego poziomu oraz inne testalia.
- ✓ Próbujemy odpowiedzieć na pytanie ”JAK” należy testować.
- ✓ W fazie projektowania testów wykorzystujemy znane nam techniki testowania.
- ✓ Wykrywanie możliwych defektów w podstawie testów tak samo jak w przypadku analizy testów.

Projektowanie testów [2/2]

Główne aktywności podczas projektowania testów :

- ✓ Projektowanie i nadanie priorytetów przypadkom testowym i zbiorom przypadków testowych
- ✓ Identyfikowanie niezbędnych danych testowych do wykonania warunków i przypadków testowych.
- ✓ Projektowanie środowiska testowego i identyfikowanie wymaganej infrastruktury i narzędzi.
- ✓ Kontynuowanie tworzenia dwukierunkowego śledzenia między podstawą testową, warunkami i przypadkami testowymi.

1. Podstawy testowania
1.4.2 Czynności i zadania testowe

TERMINY

procedura testowa: sekwencja przypadków testowych w kolejności wykonywania oraz wszelkie powiązane działania, które mogą być wymagane do ustawienia warunków wstępnych i wszelkich czynności podsumowujących po wykonaniu.

zestaw testowy: zestaw przypadków testowych lub procedur testowych zorganizowanych we wspólnym celu lub dla celu testowego.

Implementacja testów [1/3]

- ✓ Tworzymy / dokańczamy niezbędne testalia do wykonania testu.
- ✓ Uszeregowujemy przypadki testowe i procedury testowe.
- ✓ Próbujemy odpowiedzieć na pytanie: "czy mamy wszystko, co jest potrzebne do uruchomienia testów?"

Implementacja testów [2/3]

Główne czynności podczas implementacji testów: [1/2]

- ✓ Opracowanie i priorytetyzacja procedur testowych (także tworzenie zautomatyzowanych skryptów testowych).
- ✓ Tworzenie zestawów testów z procedur testowych i (jeśli to możliwe) zautomatyzowanych skryptów testowych.
- ✓ Uporządkowanie zestawów testowych w harmonogram w taki sposób, aby uzyskać efektywne wykonanie testów.

Implementacja testów [3/3]

Główne czynności podczas implementacji testów : [2/2]

- ✓ Tworzenie środowiska testowego (np. zaślepki i sterowniki, wirtualizację usług, symulatory) i upewnienie się czy jest poprawnie skonfigurowane.
- ✓ Przygotowanie danych testowych i upewnienie się, że są poprawnie załadowane w środowisku testowym.
- ✓ Kontynuacja tworzenia, weryfikacji i aktualizacji dwukierunkowego śledzenia pomiędzy podstawą testu, warunkami, przypadkami i procedurami testowymi i zestawami testów.

Projektowanie i implementacja testów

Pamiętaj, że:

- ✓ Zadania związane z projektowaniem testów i implementacją testów są często łączone i odbywają się jednocześnie.
- ✓ Podczas wykonywania testów eksploracyjnych oraz opartych na doświadczeniu wykonujemy jednocześnie projektowanie i implementację.
- ✓ Jest możliwość wykonania testów eksploracyjnych natychmiast po zakończeniu projektowania i implementacji testów.

Wykonywanie testów [1/2]

Główne czynności podczas wykonywania testów: [1/2]

- ✓ Zapisywanie wszelkich danych identyfikacyjnych i wersji testaliów, a w szczególności oprogramowania testowego.
- ✓ Ręczne wykonanie testów, lub przy użyciu narzędzi do wykonywania testów
- ✓ Porównanie rzeczywistych wyników z oczekiwanymi
- ✓ Analiza ewentualnych anomalii przez deweloperów, i ustalenie ich prawdopodobnych przyczyn (np. defekt w kodzie lub rezultat fałszywie pozytywny).

Wykonywanie testów [2/2]

Główne czynności podczas wykonywania testów : [2/2]

- ✓ Raportowanie defektów w oparciu o zaobserwowane awarie.
- ✓ Logowanie wyników wykonania testu.
- ✓ Powtarzanie czynności testowych w wyniku działań podjętych w przypadku anomalii lub w ramach planowanych testów (np. testy potwierdzające lub testowanie regresji).
- ✓ Kontynuacja tworzenia, weryfikacji i aktualizacji dwukierunkowego śledzenia pomiędzy podstawą testu, warunkami, przypadkami i procedurami testowymi i wynikami testów.

Ukończenie testów [1/3]

- ✓ Zbieramy dane z zakończonych czynności testowych.
- ✓ Ukończenie testów na ogół odbywa się przy okazji kamieni milowych np. kiedy zakończymy sprint lub ukończymy dany poziom testów. Przy ukończeniu sprintu początkowego nie wykonujemy tych czynności.

Ukończenie testów [2/3]

Główne czynności podczas ukończenia testów [1/2]:

- ✓ Sprawdzanie, czy wszystkie raporty o defektach są zamknięte.
- ✓ Tworzenie raportu podsumowującego testy, który zostanie przekazany zainteresowanym stronom.
- ✓ Finalizowanie i archiwizacja środowiska testowego, danych testowych, infrastruktury testowej do późniejszego wykorzystania.
- ✓ Przekazanie testaliów do zespołu wsparcia / konserwacji oraz innym zespołom projektowym i / lub innym interesariuszom, którzy mogliby skorzystać z jego użycia.

Ukończenie testów [3/3]

Główne czynności podczas ukończenia testów: [2/2]

- ✓ Analizowanie wniosków wyciągniętych z zakończonych czynności testowych, aby określić ewentualne zmiany potrzebne w przyszłej pracy (np. retrospekcje).
- ✓ Poprawa procesu testowania na podstawie zebranych informacji.

1. Podstawy testowania
1.4.2 Czynności i zadania testowe

PRZYKŁAD [1/2]

Pójdźmy za przykładem …

Jesteś testerem w projekcie motoryzacyjnym. Na początku procesu testowego pomożesz tworzyć plan testów, i aktywnie zaplanujesz harmonogram testów, kryteria wejścia i definicję zakończenia.

Kiedy to zrobisz, zostaniesz poproszony o rozpoczęcie analizy podstawy testów w celu określenia warunków testowych, które zainicjują macierz śledzenia.

W wyniku analizy testowej rozpoczynasz projektować testy. Opracowujesz warunki testowe, przypadki testowe wysokiego poziomu, identyfikujesz, jakie dane testowe są potrzebne, projektujesz środowisko testowe, identyfikujesz potrzebne narzędzia i rozszerzasz już istniejącą macierz śledzenia.

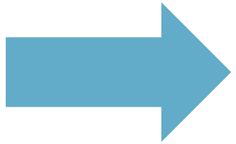
1.4.2 Czynności i zadania testowe

PRZYKŁAD [2/2]

Pójdźmy za przykładem…

Po zaprojektowaniu testów możesz rozpocząć opracowywanie procedur testowych dla każdego zidentyfikowanego przypadku testowego i pogrupować je w zestawy testów.

W końcu jesteś gotowy do rozpoczęcia wykonywania testów, gdzie możesz porównać wyniki rzeczywiste z oczekiwanym i o ile to konieczne zgłosić defekty. Pamiętaj, że po zidentyfikowaniu nowego ryzyka należy je dodać do planu testów i odpowiednio złagodzić.



Po osiągnięciu definicji zakończenia pomagasz zebrać dane z dotychczasowych działań testowych, tego co zostało zrobione i upewniasz się, że wszystkie raporty o usterkach są zamknięte. Dobrą praktyką jest także organizowanie spotkań retrospekcyjnych na zakończenie projektu.

1. Podstawy testowania

1.4.3 Produkty pracy związane z testowaniem

Czym jest produkt pracy związany z testowaniem ?

- ✓ Produkty pracy związane z testowaniem są integralną częścią procesu testowego.
- ✓ Różnorodność produktów prac związanych z testowaniem jest nieograniczona i zazwyczaj uzależniona jest od procesu testowego.
- ✓ Większością produktów prac związanych z testowaniem może zarządzać za pomocą narzędzi do zarządzania testami i zarządzania defektami.

1. Podstawy testowania
1.4.3 Produkty pracy związane z testowaniem

Różnorodne produkty pracy związane z testowaniem:

- Produkty pracy planowania testów
- Produkty pracy monitorowania testów i nadzoru nad testami
- Produkty pracy analizy testów
- Produkty pracy projektowania testów
- Produkty pracy implementacji testów
- Produkty pracy wykonywania testów
- Produkty pracy ukończenia testów

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy planowania testów

- ✓ Zwykle wyróżniamy jeden lub kilka planów testów.
- ✓ Informacje na temat podstawy testów, z którą zostaną powiązane inne produkty pracy
- ✓ Kryteria wyjścia (definicja ukończenia – Definition of Done) do wykorzystania podczas monitorowania i kontroli.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy monitorowania testów i nadzoru nad testami

- ✓ Różnego rodzaju raporty z testów (raporty postępu testów, raport sumaryczny z testów).
- ✓ Raporty powinny zawierać szczegółowe podsumowanie wyników wykonania testów.
- ✓ Należy zająć się kwestiami dotyczącymi zarządzania projektem (ukończenie zadań, wykorzystanie zasobów i pracochłonność).

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy analizy testów

- ✓ Zdefiniowane i uszeregowane według priorytetów warunki testowe.
- ✓ Ustanowienie możliwości śledzenia powiązań pomiędzy warunkami testowymi, a wymaganiami, które pokrywa.
- ✓ W przypadku testów opartych na doświadczeniach analizę testów można określić jako tworzenie kart opisu testów.
- ✓ Wykryte i zgłoszone defekty w podstawie testów.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy projektowania testów

- ✓ Tworzymy przypadki testowe i zbiory przypadków testowych na podstawie warunków testowych zdefiniowanych w analizie testowej.
- ✓ Tworzymy przypadki testowe wysokiego poziomu bez żadnych konkretnych wartości danych wejściowych i oczekiwanych wyników.
- ✓ Takie przypadki testowe można ponownie wykorzystać w kolejnych cyklach testowych już z użyciem konkretnych danych testowych.
- ✓ Tworzymy dwukierunkowe śledzenie powiązań przypadków testowych z pokrywanym warunkiem testowym (lub kilkoma warunkami testowymi).

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy projektowania testów

- ✓ Zidentyfikowano niezbędne dane testowe
- ✓ Zaprojektowano środowisko testowe
- ✓ Identyfikowanie narzędzia i wymaganej infrastruktury

Warunki testowe, które zostały wcześniej zdefiniowane podczas analizy testów, można doprecyzować podczas projektowania testów.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy implementacji testów

- ✓ Procedury testowe i kolejność ich wykonania
- ✓ Zestawy testowe
- ✓ Harmonogram testów

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy implementacji testów

- ✓ Dwukierunkowe śledzenie pomiędzy podstawą testową a procedurami testowymi. Na tym etapie powinniśmy być w stanie uzyskać pełne pokrycie podstawy testów.
- ✓ Czasami może być konieczne utworzenie takich produktów pracy, które korzystają z innych narzędzi np. zautomatyzowany skrypt testowy.
- ✓ Określane są dane testowe czyli przypisywane są wartości do danych wejściowych i oczekiwanych wyników dla każdego przypadku testowego – w tym momencie przypadki testowe niskiego poziomu wraz z danymi testowymi zastępują istniejące już przypadki testowe wysokiego poziomu.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy implementacji testów

- ✓ Utworzone jest środowisko testowe
- ✓ W przypadku testów opartych na doświadczeniu, projektowanie i implementacja są tworzone podczas wykonywania testów

Warunki testowe, które zostały wcześniej zdefiniowane podczas analizy testów, mogą być dalej doprecyzowane podczas implementacji testów.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy wykonywania testów

- ✓ Dokumentacja statusu poszczególnych przypadków lub procedur testowych (np. gotowych do uruchomienia, zaliczony, niezaliczony, zablokowany, celowo pominiętych itp.).
- ✓ Raporty o defektach.
- ✓ Dokumentacja pokazująca, które testalia zostały wykorzystane podczas testowania.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy wykonywania testów

- ✓ Dwukierunkowe śledzenie jest zaktualizowane i pozwala na uzyskanie rzeczywistego statusu danej podstawy testów.
- ✓ Wszelkie kryteria wyjściowe powinny być zmierzone, zaraportowane a w idealnym przypadku spełnione.

1.4.3 Produkty pracy związane z testowaniem

Produkty pracy ukończenia testów

- ✓ Raporty ukończenia testów
- ✓ Czynności, które mają na celu udoskonalenie procesu testowego w kolejnych projektach lub iteracjach
- ✓ Żądania zmian
- ✓ Ewentualne zaległości produktowe i sfinalizowane testalia

1. Podstawy testowania

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

TERMINY

śledzenie: stopień, w jakim można ustalić relację pomiędzy dwoma lub większą liczbą produktów prac.

macierz śledzenia: dwuwymiarowa tabela, która koreluje dwa pojęcia (np. wymagania i przypadki testowe). Macierz ta umożliwia śledzenie w tył i z powrotem od jednego elementu do drugiego, tym samym umożliwiając określenie pokrycia oraz szacowanie wpływu proponowanych zmian.

1. Podstawy testowania

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

TERMINY

śledzenie poziome (horyzontalne): śledzenie odwzorowania wymagań testowych w dokumentacji testowej na kolejnych poziomach (np. plan testów, specyfikacja projektu testów, specyfikacja przypadku testowego i specyfikacja procedury testowej lub skryptu testowego).

śledzenie pionowe (wertykalne): śledzenie wymagań poprzez kolejne poziomy dokumentacji projektowej aż do modułów.

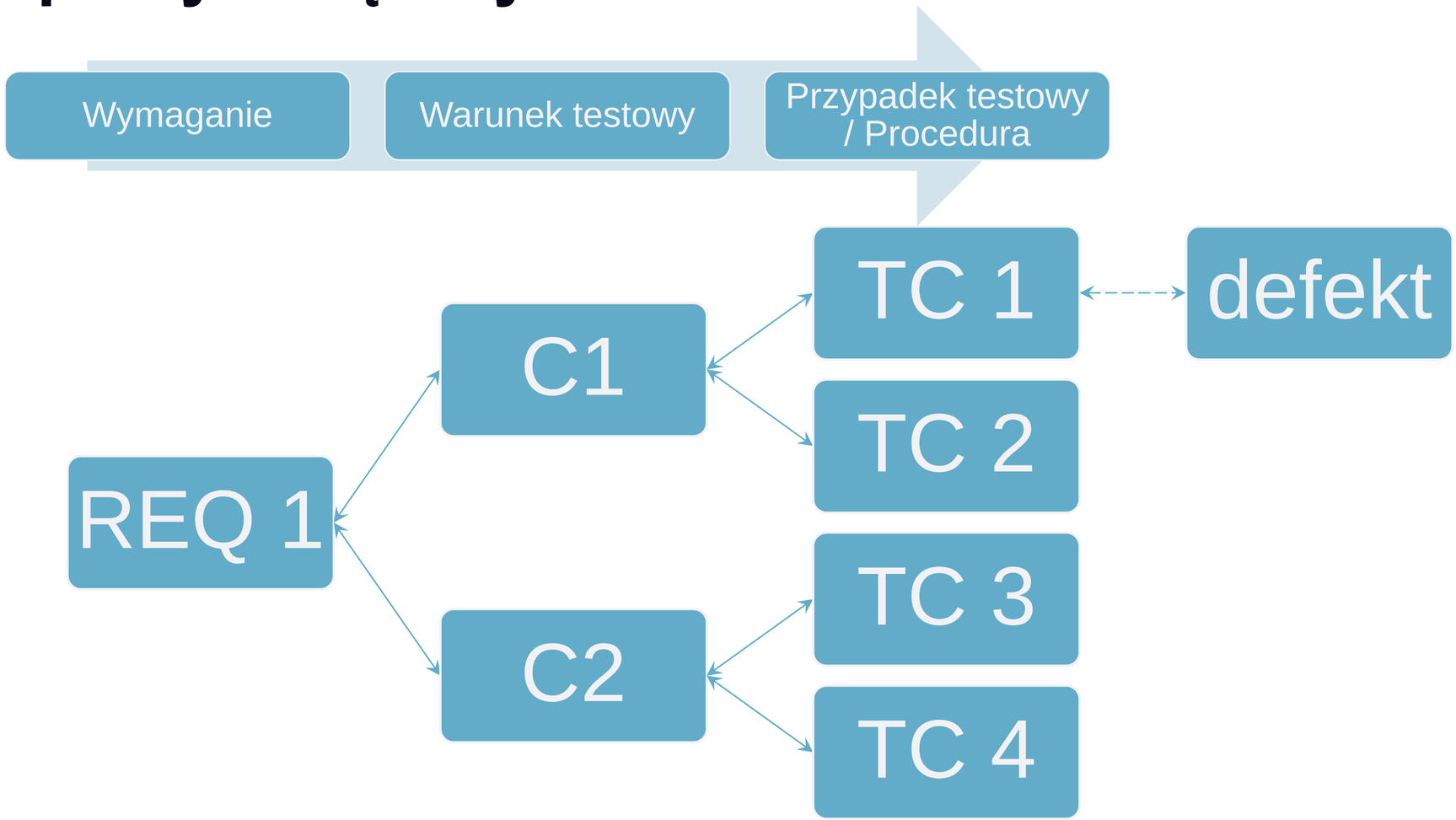
1. Podstawy testowania

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

- ✓ W celu ustanowienia skutecznego monitorowania i kontroli procesu testowego konieczne jest utrzymanie mechanizmu dwukierunkowego śledzenia w całym procesie testowym.
- ✓ Każdy element podstawy testu jest powiązany z innymi produktami związanymi z testowaniem, które są związane z tym elementem przez cały czas trwania procesu testowego.
- ✓ Narzędzia do zarządzania testami mogą dostarczać informacji, które są pomocne, gdy istnieje potrzeba generowania macierzy śledzenia.

1. Podstawy testowania

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem



1. Podstawy testowania

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

Sprawne śledzenie umożliwia:

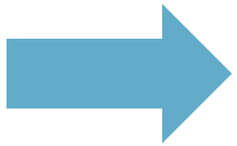
- ✓ Analizowanie wpływu zmian.
- ✓ Audyt testów i spełnianie kryteriów związanych z zarządzaniem w IT.
- ✓ Lepsza zrozumiałość raportów postępu testów i sumarycznego raportu z testów przy uwzględnieniu statusu poszczególnych elementów podstawy testów.
- ✓ Przekazywanie interesariuszom technicznych aspektów testowania w bardziej zrozumiałej formie.
- ✓ Dostarczanie informacji umożliwiających ocenę jakości produktu, możliwości procesu i postępu projektu w odniesieniu do celów biznesowych.

1.4.4 Śledzenie powiązań między podstawą testów a produktami pracy związanymi z testowaniem

PRZYKŁAD

Znaleźliśmy defekt - które wymaganie nie jest w pełni pokryte?

Wyobraź sobie, że testujesz produkt, który ma wkroczyć w fazę testów akceptacyjnych. Znalazłeś defekt – wiesz, że określona funkcjonalność nie działa poprawnie.



W oparciu o macierz śledzenia jesteś w stanie przypisać defekt związany z tą procedurą testową, warunkiem testowym i danym wymaganiem. Bez dwukierunkowej macierzy śledzenia nie byłoby możliwości sprawdzenia które wymaganie nie jest spełnione ze względu na znaleziony defekt.


1.5
Psychologia testowania

1. Podstawy testowania

1.5.1 Psychologia człowieka a testowanie

Efekt potwierdzenia

Zidentyfikowane defekty mogą być postrzegane jako krytyka autora. To, że kod jest niepoprawny może być trudne do zaakceptowania przez autora.

krytyka

Z drugiej strony wszelkie uprzedzenia poznawcze sprawiają, że jeszcze trudniej jest zaakceptować wszelkie informacje uzyskane podczas testów. Błędny wynik testu jest zwykle uznawany za złą wiadomość.

akceptacja

1.5.1 Psychologia człowieka a testowanie

Czy testowanie to czynność destrukcyjna ?

Niektóre osoby mogą uznać to za niekorzystne działanie. Dlatego też informacje o defektach i awariach należy przekazywać w poprawny sposób - konstruktywnie.

Potrzebne są dobre umiejętności interpersonalne, w szczególności te miękkie, aby zarządzać dobrymi relacjami z innymi członkami zespołu.

1. Podstawy testowania

1.5.1 Psychologia człowieka a testowanie

Jak powinniśmy pracować?

Komunikuj się w neutralny sposób

Nie krytykuj, pisz obiektywne raporty o faktycznych defektach.

Współpracuj - nie walcz

Przypomnij wszystkim o wspólnym celu, jakim są systemy lepszej jakości.

Podkreślaj zalety testowania

Popraw produkty pracy i umiejętności, oszczędzając czas i pieniądze oraz redukując ogólne ryzyko związane z jakością produktu.

Pytaj o potwierdzenie

Aby dowiedzieć się, czy druga osoba zrozumiała, co zostało powiedziane i na odwrót.

Wzajemny szacunek

Spróbuj się wczuć w sytuację innych osób i zrozumieć, dlaczego mogą reagować negatywnie na podane informacje.

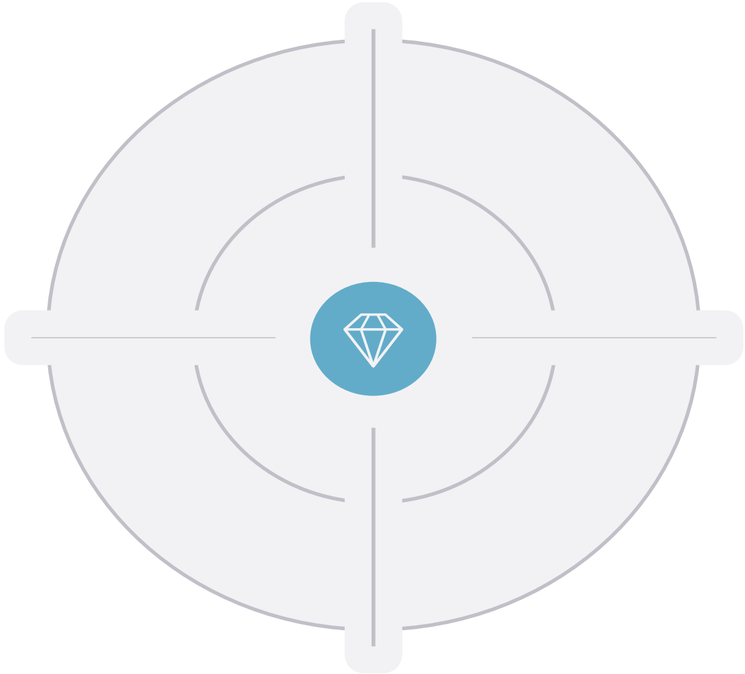
1. Podstawy testowania
1.5.2 Różnice w sposobie myślenia testerów i programistów

Programowanie

Celem jest zaprojektowanie
i zbudowanie produktu.

Testowanie

Celem jest zweryfikowanie i walidacja produktu, znalezienie usterek przed wydaniem itp.



Do osiągnięcia wspólnego celu potrzebne są różne sposoby myślenia, który jest wyższym poziomem jakości produktu.

1. Podstawy testowania
1.5.2 Różnice w sposobie myślenia testerów i programistów

Nastawienie testera

Ciekawość, profesjonalny pesymizm, krytyczne podejście, dbałość o szczegóły, motywacja do dobrej i pozytywnej komunikacji oraz dobrych relacji.

Nastawienie programisty

Są bardziej nastawieni na projektowanie i budowanie rozwiązań niż kwestionowanie, co może być niepoprawne w tych rozwiązaniach.

1. Podstawy testowania
1.5.2 Różnice w sposobie myślenia testerów i programistów

PRZYKŁAD

Zaprojektuj i zbuduj kontra *weryfikuj i waliduj*

Wyobraź sobie prosty program do konwersji temperatury pomiędzy stopniami Celsjusza i Farenheita. Ma dwa pola i przycisk do konwertowania.

Pomimo, że gotowe rozwiązanie działa poprawnie, nie ma żadnych informacji o tym co gdzie należy wpisać ani przy poszczególnym polu, ani przy przycisku do konwersji. Spełnia on określone wymagania, ale w ogóle nie jest przyjazny użytkownikowi.

1. Podstawy testowania

1.5.2 Różnice w sposobie myślenia testerów i programistów

Kto powinien testować oprogramowanie ?

Programiści mogą testować swój własny kod, ale tylko przy odpowiednim nastawieniu. Ponieważ pracujemy w różnych cyklach życia oprogramowania, działania testowe mogą być organizowane na różne sposoby. Posiadanie niezależnych testerów jest ważne, ponieważ mogą oni być bardzo skuteczni w znajdowaniu defektów.

1. Podstawy testowania

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

02

Testowanie w cyklu życia oprogramowania.

2. Testowanie w cyklu życia oprogramowania
Cele Nauczania

2.1 Modele cyklu życia oprogramowania

FL-2.1.1 (K2) Kandydat potrafi wyjaśnić relacje między czynnościami związanymi z wytwarzaniem oprogramowania a czynnościami testowymi w cyklu życia oprogramowania

FL-2.1.2 (K1) Kandydat potrafi wskazać powody, dla których konieczne jest dostosowanie modeli cyklu życia oprogramowania do kontekstu wynikającego z charakterystyki

2.2 Poziomy testów

FL-2.2.1 (K2) Kandydat potrafi porównać poszczególne poziomy testów z punktu widzenia celów, podstawy testów, przedmiotów testów, typowych defektów i awarii, podejść i odpowiedzialności

2. Testowanie w cyklu życia oprogramowania
Cele Nauczania

2.3 Typy Testów

FL-2.3.1 (K2) Kandydat potrafi porównać testowanie funkcjonalne, niefunkcjonalne i białoskrzynkowe

FL-2.3.2 (K1) Kandydat zdaje sobie sprawę z tego, że testy funkcjonalne, niefunkcjonalne i białoskrzynkowe mogą występować na dowolnym poziom

FL-2.3.3 (K2) Kandydat potrafi porównać przeznaczenie testowania potwierdzającego i testowania regresji

2.4 Testowanie pielęgnacyjne

FL-2.4.1 (K2) Kandydat potrafi podsumować zdarzenia wyzwalające testowanie pielęgnacyjne

FL-2.4.2 (K2) Kandydat potrafi opisać rolę analizy wpływu w testowaniu pielęgnacyjnym

Modele cyklu życia oprogramowania

2. Testowanie w cyklu życia oprogramowania

2.1 Wprowadzenie

Co to jest cykl życia oprogramowania ?

- ✓ Model cyklu życia oprogramowania to ogólne wytyczne dotyczące tego, jak prowadzimy projekt, w jaki sposób realizowany jest każdy etap projektu i jaki jest ich związek.
- ✓ Logiczna i chronologiczna kolejność etapów projektu.
- ✓ Możemy wyróżnić różne modele i ich cechy, szczególnie jeśli chodzi o podejście do testowania.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Czym jest dobre testowanie ?

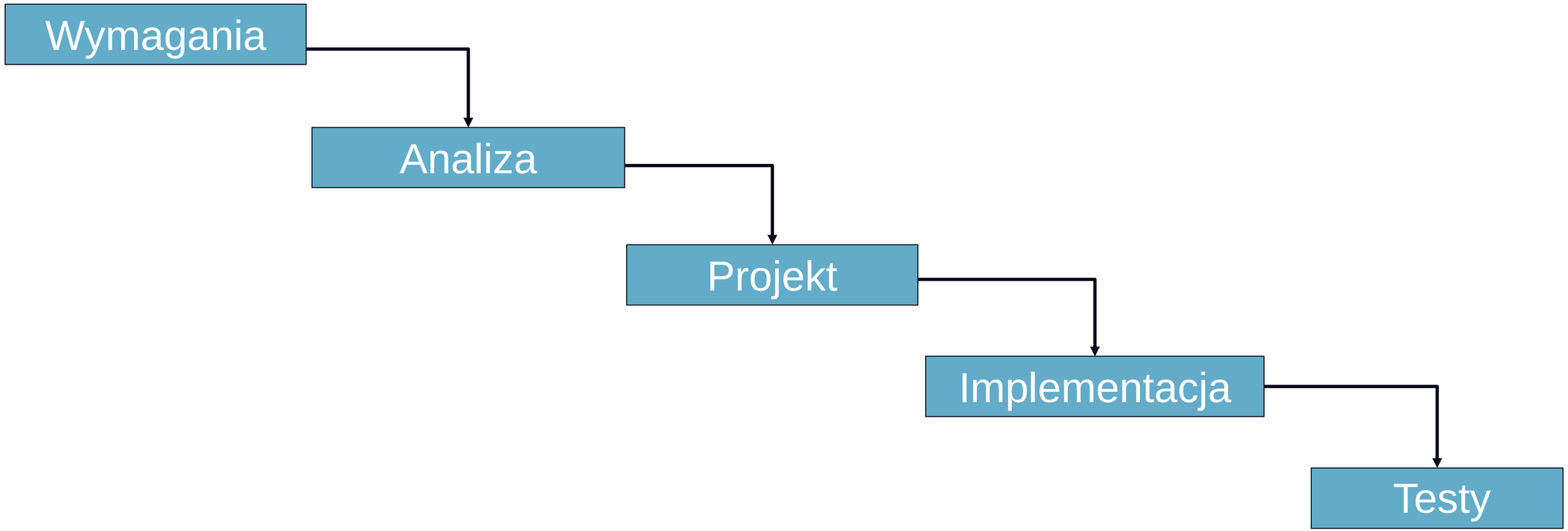
- ✓ Dla każdej czynności związanej z wytwarzaniem oprogramowania istnieje odpowiednia czynność testowa.
- ✓ Każdy poziom testowania ma cele testowe właściwe dla tego poziomu.
- ✓ Analizę i projektowanie testów rozpoczynają się podczas odpowiadającego im działania programistycznego (wczesne testowanie).
- ✓ Testerzy uczestniczą w dyskusjach mających na celu zdefiniowanie i udoskonalenie wymagań i projektów oraz w przeglądaniu dostępnych produktów pracy.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Model kaskadowy

Klasycznym przykładem modelu sekwencyjnego jest model waterfall:



2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

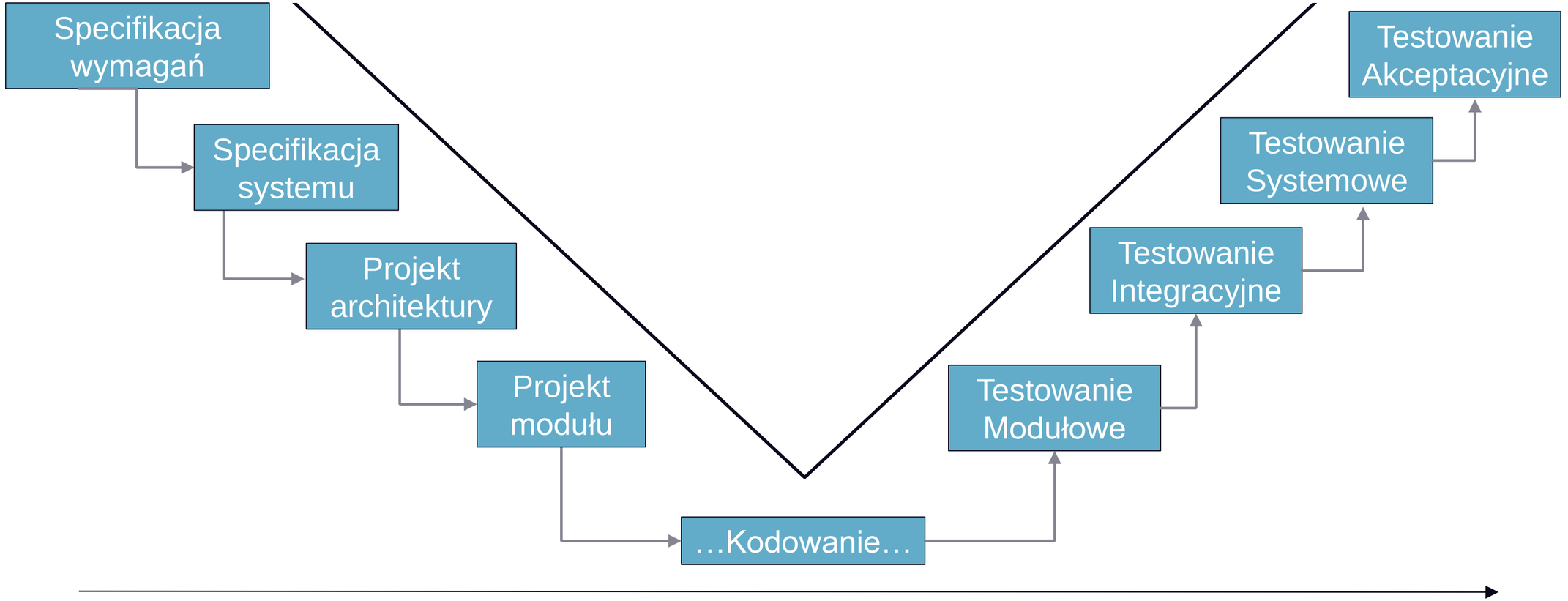
Cechy modelu kaskadowego

- ✓ Proces rozwoju oprogramowania podzielony jest na fazy, w których każda z nich ma miejsce po zakończeniu poprzedniej.
- ✓ Wymagania pozostają "zamrożone" i nie podlegają dalszym modyfikacjom w trakcie trwania projektu.
- ✓ Testowanie pojawia się na końcu, gdy wszystkie działania programistyczne są zakończone.
- ✓ Potrzeby klienta są rozpoznawane już na początku projektu.
- ✓ Długoterminowe projekty.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Model V



2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Cechy modelu V

- ✓ Jasne określenie poziomów testów związanych z każdą czynnością tworzenia oprogramowania.
- ✓ Skupiamy się na testowaniu statycznym - testerzy powinni uczestniczyć w przeglądach dokumentów podczas kolejnych faz wytwarzania oprogramowania.
- ✓ Model V rozgranicza poszczególne fazy wytwarzania oprogramowania od faz testowania.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

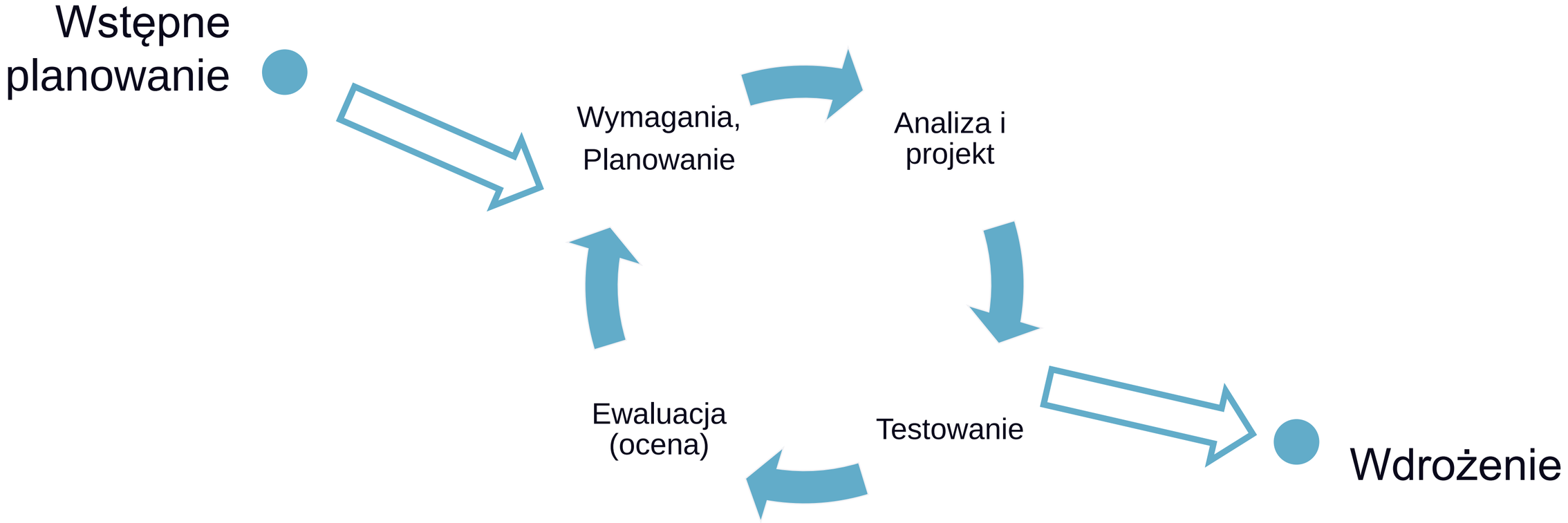
Cechy modeli sekwencyjnych

W większości przypadków są to projekty długoterminowe, które potrzebują wielu miesięcy (a nawet lat) żeby dostarczyć kompletną funkcjonalność

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Model iteracyjno-przyrostowy



2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Cechy modelu iteracyjno-przyrostowego

- ✓ System jest dostarczany przyrostowo w iteracjach, co oznacza, że jest podzielony na małe przyrosty, gdzie każdy z nich obejmuje inny podzbiór wymagań użytkownika (np. opowieści, zbiór historyjek użytkownika itp.).
- ✓ W modelach iteracyjno-przyrostowych, zwłaszcza Agile, nie wszystkie typy testów muszą być robione. Nie koniecznie musi tam być specyfikacja systemu albo zamodelowana architektura, to zależy od projektu.
- ✓ Czynności testowe mogą zachodzić na siebie i mają miejsce w każdej iteracji.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

SCRUM jako model iteracyjno-przyrostowy

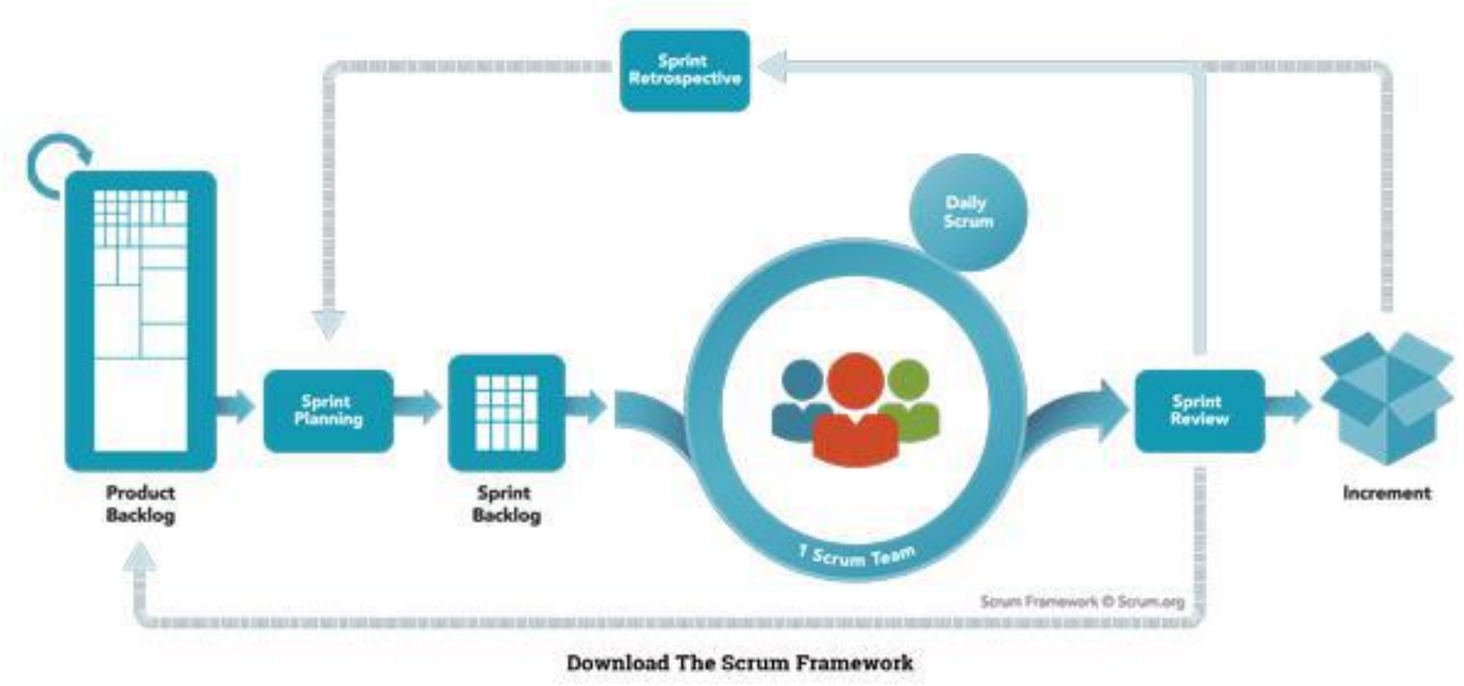
- ✓ Skupiamy się na automatycznych testach regresji i zautomatyzowanym środowiskiem testowym (CI).
- ✓ Oparte na samoorganizujących się zespołach i atrybutach Scrum.
- ✓ Klient jest zaangażowany w rozwój systemu (sprint demo, stała informacja zwrotna).
- ✓ Każda iteracja kończy dostarczanie działającego produktu z zestawem funkcji.
- ✓ Użyteczne oprogramowanie dostarczane jest w ciągu kilku tygodni, ale kompletny produkt jest dostarczany przez kilka miesięcy / lat.

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Model iteracyjno-przyrostowy PRZYKŁADY

- ✓ Scrum:



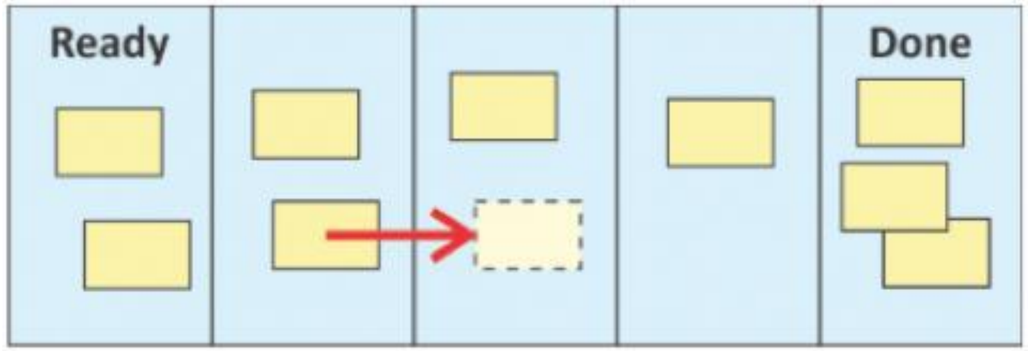
Źródło: https://www.scrum.org/resources/what-is-scrum

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Tablica Kanban, a model iteracyjno-przyrostowy PRZYKŁADY

- ✓ Tablica Kanban to jedno z narzędzi wykorzystywanych w celu implementacji metody kanban do zarządzania procesami i projektami w organizacjach. Nie jest modelem wytwarzania - to podejście zarządcze ułatwiające pracę w SCRUMie.



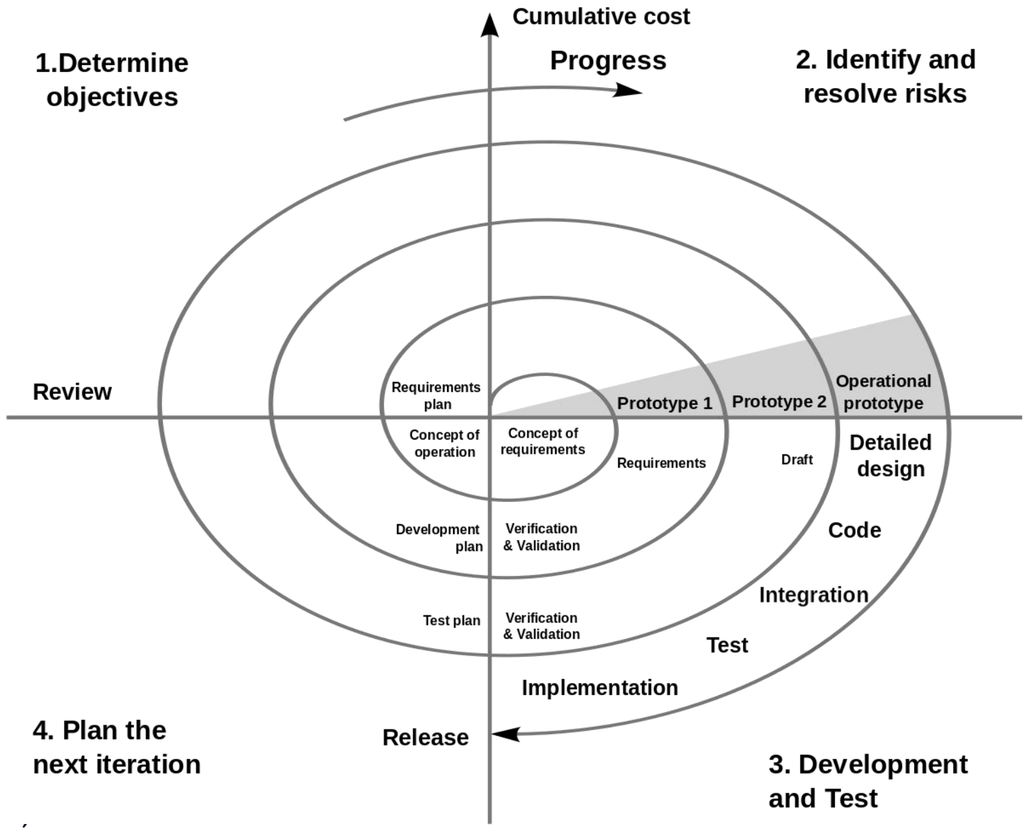
Źródło: https://help.rallydev.com/what-is-kanban

2. Testowanie w cyklu życia oprogramowania

2.1.1 Wytwarzanie oprogramowania a testowanie oprogramowania

Model iteracyjno-przyrostowy PRZYKŁADY

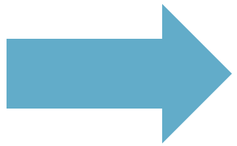
- ✓ Model spiralny:



Źródło: https://en.wikipedia.org/wiki/Spiral_model

2. Testowanie w cyklu życia oprogramowania
2.1.2 Modele cyklu życia oprogramowania w kontekście

PRZYKŁAD



Który model jest najlepszy?

…to zależy. Każdy model musi być dostosowany do naszego projektu. Wybór modelu zależy od celu, jaki chcemy osiągnąć, rodzaju produktu, priorytetów biznesowych i zidentyfikowanego ryzyka.

- ✓ Niewielki system administracyjny będzie opracowany inaczej niż system krytyczny ze względu na poziom bezpieczeństwa.
- ✓ Problemy z komunikacją mogą powodować trudności przy wdrażaniu iteracyjnych modeli przyrostowych.

2.1.2 Modele cyklu życia oprogramowania w kontekście

PRZYKŁAD

Który model jest najlepszy?

Poziomy testów mogą wymagać reorganizacji (połączenia lub rozdzielenia), aby pasowały do danego modelu.

- ✓ Np. w przypadku integracji oprogramowania do powszechnej sprzedaży z większym systemem nabywca może wykonywać **testy współdziałania na poziomie testowania integracji systemów** (np. w zakresie integracji z infrastrukturą i innymi systemami) **oraz na poziomie testowania akceptacyjnego** (testowanie funkcjonalne i niefunkcjonalne wraz z testowaniem akceptacyjnym przez użytkownika i produkcyjnymi testami akceptacyjnymi).

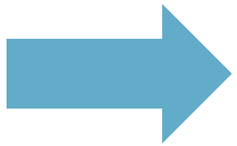
2. Testowanie w cyklu życia oprogramowania
2.1.2 Modele cyklu życia oprogramowania w kontekście

PRZYKŁAD

Który model jest najlepszy?

Model może być łączony

- ✓ Modelu V użyjemy do tworzenia i integracji systemu back-end`u. Modelu Agile użyjemy do rozwoju GUI, a na początku projektu możemy skupić się na prototypowaniu za pomocą modelu spiralnego.



2.1.2 Modele cyklu życia oprogramowania w kontekście

PRZYKŁAD

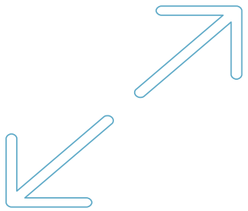
Który model jest najlepszy?

W przypadku systemu Internet of Things (IoT) (pol. Internet rzeczy):

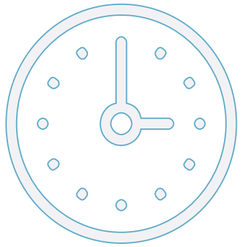
- ✓ mogą korzystać z różnych, oddzielnie obsługiwanych modeli, ponieważ składają się z wielu urządzeń, produktów i usług opracowywanych osobno.
- ✓ system IoT ma możliwość przesyłania danych przez sieć bez konieczności interakcji człowiek-człowiek lub człowiek-komputer.
- ✓ wyzwanie polega na tym, że można integrować różne obiekty jako jeden system IoT.
- ✓ nacisk na późniejsze fazy modelu, gdy wydanie zostanie wprowadzone do powszechnego użycia (system operacyjny, aktualizacja i likwidacja w końcu).

2. Testowanie w cyklu życia oprogramowania
2.1.2 Modele cyklu życia oprogramowania w kontekście

Pamiętaj o tym, że



- ✓ Dla każdej czynności programowania istnieje odpowiednia czynność testowa.



- ✓ Testerzy powinni brać udział w przeglądach tak szybko, jak to tylko możliwe.

2. Testowanie w cyklu życia oprogramowania
2.1.2 Modele cyklu życia oprogramowania w kontekście

Dlaczego musimy dostosować model rozwoju oprogramowania do kontekstu projektu i charakterystyki produktu ?

Złożoność projektu

- ✓ Różnica w ryzykach produktowych systemów

Połączenie rozwoju sekwencyjnego i zwinnego

- ✓ Wiele jednostek biznesowych może być częścią projektu lub programu

Krótki czas na dostarczenie produktu na rynek

- ✓ łączenie poziomów testów i/lub integracja typów testów na poziomach testowych

Poziomy **testów**

2. Testowanie w cyklu życia oprogramowania
2.2 Wprowadzenie

Poziomy testów

Poziom testów jest uznawany za grupę działań zorganizowanych i zarządzanych razem. Każdy poziom testowy odzwierciedla działania opisane w procesie testowym, które są związane z danym poziomem rozwoju oprogramowania. Poziomy testów opisane w sylabusie są następujące:

2.2 Wprowadzenie

Poziomy testów

Poziomy testów cechują:

- ✓ Szczegółowe cele
- ✓ Podstawa testów (można się do niej odwoływać żeby wyprowadzić przypadki testowe)
- ✓ Przedmiot testów (to, co jest testowane)
- ✓ Typowe defekty i awarie
- ✓ Konkretne podejścia i obowiązki

Każdy poziom testu wymaga odpowiedniego środowiska testowego. Programiści używają własnego środowiska programistycznego, podczas testów akceptacyjnych jest to kopia środowiska produkcyjnego.

2. Testowanie w cyklu życia oprogramowania
2.2.1 Testowanie modułowe

skoncentruj się na komponencie, który można testować oddzielnie

Cele testowania modułowego

- ✓ Zmniejszanie ryzyka.
- ✓ Sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych modułu z projektem i specyfikacjami.
- ✓ Budowanie zaufania do jakości modułu.
- ✓ Wykrywanie defektów w module.
- ✓ Zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

Przedmioty testów

- ✓ Moduły, jednostki lub komponenty
- ✓ Kod i struktury danych
- ✓ Klasy
- ✓ Moduły baz danych

Podstawa testów

- ✓ Szczegółowy projekt.
- ✓ Kod.
- ✓ Model danych.
- ✓ Specyfikacja komponentów.

Typowe defekty i awarie

- ✓ Niepoprawna funkcjonalność (np. niezgodna ze specyfikacją projektu).
- ✓ Problemy z przepływem danych.
- ✓ Niepoprawny kod i logika.

2. Testowanie w cyklu życia oprogramowania
2.2.1 Testowanie modułowe

skoncentruj się na komponencie, który można testować oddzielnie

Konkretne podejścia i odpowiedzialności

- ✓ Zwykle wykonuje je programista, który napisał kod.
- ✓ W przypadku zastosowania TDD, testy komponentów są utworzone przed pisaniem kodu aplikacji.

Główna charakterystyka

- ✓ Testy komponentów są często wykonywane w izolacji od pozostałych części systemu.
- ✓ Defekty mogą być zwykle przypisywane bez formalnego procesu.
- ✓ W niektórych przypadkach, gdzie zmiany kodu mają charakter ciągły, kluczową rolę pełni automatyczne testowanie regresji.

Pojęcie TDD (wytwarzanie sterowane testami) w testowaniu modułowym

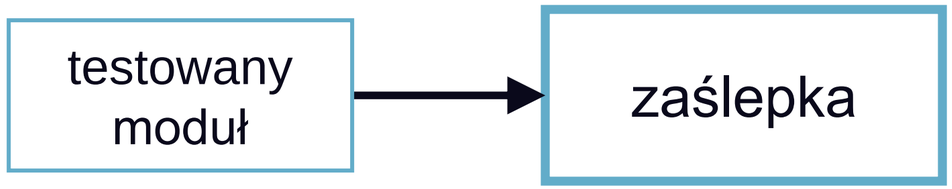
- ✓ Dodaj test, który sprawdza daną funkcjonalność.
- ✓ Uruchom test, który powinien zakończyć się niepowodzeniem, ponieważ kod nie istnieje.
- ✓ Napisz kod i ponownie przeprowadź test, aż przejdzie poprawnie.
- ✓ Ulepsz kod przed ponownym uruchomieniu testu - upewnij się, że nadal przechodzi.
- ✓ Powtórz to ponownie za każdym razem, gdy następny kawałek kodu jest ukończony.

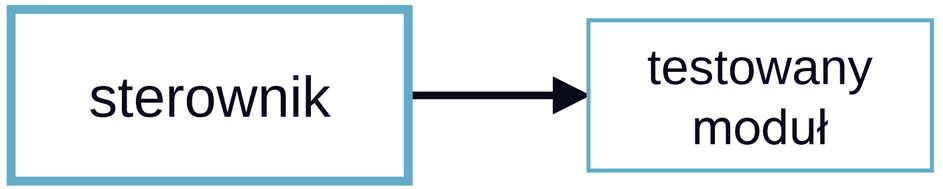
2.2.1 Testowanie modułowe

Zaślepka i Sterownik

zaślepka: szkieletowa albo specjalna implementacja modułu używana podczas produkcji lub testów innego modułu, który tę zaślepkę wywołuje albo jest w inny sposób od niej zależny. Zaślepka zastępuje wywoływany moduł.

sterownik: moduł oprogramowania lub narzędzie testowe, które zastępuje moduł kontrolujący lub wywołujący funkcje testowanego modułu lub systemu.





2. Testowanie w cyklu życia oprogramowania
2.2.2 Testowanie integracyjne

DEFINICJA

Ciągła integracja (CI od ang. Continuous Integration): praktyka stosowana w trakcie rozwoju oprogramowania, polegająca na częstym, regularnym włączaniu (integracji) bieżących zmian w kodzie do głównego repozytorium i każdorazowej weryfikacji zmian, poprzez zbudowanie projektu (jeśli jest taka potrzeba) i wykonanie testów jednostkowych.

2. Testowanie w cyklu życia oprogramowania
2.2.2 Testowanie integracyjne

skupiamy się na interakcjach pomiędzy komponentami lub systemami

Cele testowania integracyjnego

- ✓ Zmniejszanie ryzyka.
- ✓ Sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych interfejsów z projektem i specyfikacjami.
- ✓ Budowanie zaufania do jakości interfejsów.
- ✓ Wykrywanie defektów (które mogą występować w samych interfejsach lub w modułach/systemach).
- ✓ Zapobieganie przedostawaniu się defektów na wyższe poziomy testowania.

Podstawa testów

- ✓ Projekt oprogramowania i systemu
- ✓ Diagramy sekwencji
- ✓ Specyfikacje interfejsów
- ✓ Przypadki użycia
- ✓ Architektura na poziomie modułów i systemu
- ✓ Przepływy pracy
- ✓ Definicje interfejsów zewnętrznych

Przedmioty testów

- ✓ Podsystemy
- ✓ Bazy danych
- ✓ Infrastruktura
- ✓ Interfejsy
- ✓ Interfejsy programowania aplikacji (API)
- ✓ Mikrousługi

Dwa różne poziomy dla testowania integracyjnego

- ✓ Testy integracji modułów ze szczególnym uwzględnieniem interfejsów między komponentami. Testy te są zwykle częścią procesu ciągłej integracji.
- ✓ Testy integracji systemów ze szczególnym uwzględnieniem interfejsów między systemami. Mogą być wykonane po zakończeniu testowania systemowego.

2. Testowanie w cyklu życia oprogramowania
2.2.2 Testowanie integracyjne

skupiamy się na interakcjach pomiędzy komponentami lub systemami

Typowe defekty i awarie testów integracji modułów

- ✓ Niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych.
- ✓ Niepoprawne uszeregowanie lub niepoprawna synchronizacja wywołań interfejsów.
- ✓ Niezgodności interfejsów.
- ✓ Błędy komunikacji między modułami.
- ✓ Brak obsługi lub nieprawidłowa obsługa błędów komunikacji między modułami.
- ✓ Niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między modułami.

Typowe defekty i awarie testów integracji systemów

- ✓ Niespójne struktury komunikatów przesyłanych między systemami.
- ✓ Niepoprawne lub brakujące dane bądź niepoprawne kodowanie danych.
- ✓ Niezgodność interfejsów.
- ✓ Błędy komunikacji między systemami.
- ✓ Brak obsługi lub nieprawidłowa obsługa błędów komunikacji między systemami.
- ✓ Niepoprawne założenia dotyczące znaczenia, jednostek lub granic danych przesyłanych między systemami.
- ✓ Nieprzestrzeganie bezwzględnie obowiązujących przepisów dotyczących zabezpieczeń.

2. Testowanie w cyklu życia oprogramowania
2.2.2 Testowanie integracyjne

skupiamy się na interakcjach pomiędzy komponentami lub systemami

Konkretne podejścia i odpowiedzialności

- ✓ Testy integracji modułów lub systemów powinny koncentrować się na integracji, czyli komunikacji pomiędzy poszczególnymi elementami, a nie samej funkcjonalności, która jest objęta testowaniem systemu.
- ✓ Zwykle to deweloperzy są odpowiedzialni za te testy integracyjne na poziomie modułów.
- ✓ Stosujemy tu funkcjonalne, niefunkcjonalne i strukturalne typy testów.

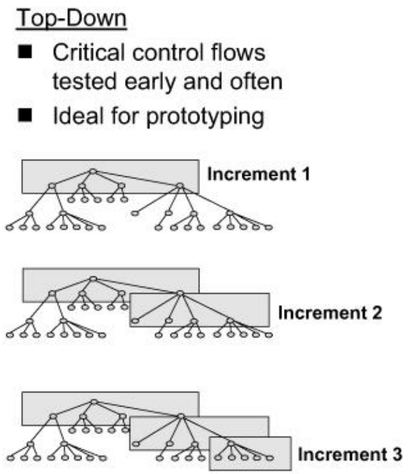
2. Testowanie w cyklu życia oprogramowania
2.2.2 Testowanie integracyjne

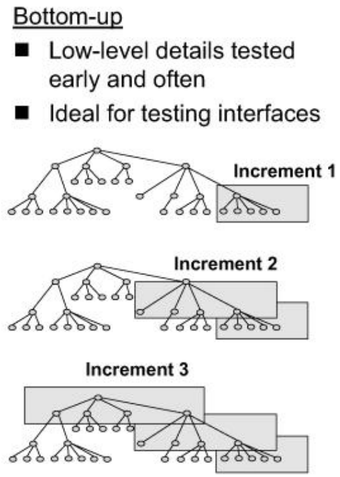
skupiamy się na interakcjach pomiędzy komponentami lub systemami

Możliwe strategie:

Metody przyrostowe:
- ✓ Zstępująca (Góra – Dół)
- ✓ Wstępująca (Dół – Góra)

Metoda nieprzyrostowa:
- ✓ „wielki wybuch”





Source: The Software Test Engineer's Handbook, 2nd Edition by Judy McKay; Graham Bath

Im większy zakres integracji, tym trudniej jest wyizolować defekty do danego modułu lub systemu

2. Testowanie w cyklu życia oprogramowania

2.2.3 Testowanie systemowe

skupiamy się na funkcjonalności całego systemu lub produktu

Cele testowania systemowego

- ✓ Zmniejszanie ryzyka.
- ✓ Sprawdzanie zgodności zachowań funkcjonalnych i niefunkcjonalnych systemu z projektem i specyfikacjami.
- ✓ Sprawdzanie kompletności systemu i prawidłowości jego działania.
- ✓ Budowanie zaufania do jakości systemu jako całości
- ✓ Wykrywanie defektów.
- ✓ Zapobieganie przedostawaniu się defektów na poziom testowania akceptacyjnego lub na produkcję.

Podstawa testów

- ✓ Specyfikacje wymagań dotyczących systemu i oprogramowania (funkcjonalnych i niefunkcjonalnych).
- ✓ Raporty z analizy ryzyka.
- ✓ Przypadki użycia.
- ✓ Historyjki użytkownika.
- ✓ Modele zachowania systemu i diagramy stanów.
- ✓ Instrukcje obsługi systemu i podręczniki użytkownika.

Przedmioty testów

- ✓ Aplikacje.
- ✓ Systemy łączące sprzęt i oprogramowanie.
- ✓ Systemy operacyjne.
- ✓ System podlegający testowaniu.
- ✓ Konfiguracja i dane konfiguracyjne systemu.

Typowe defekty i awarie

- ✓ Niepoprawne obliczenia.
- ✓ Niepoprawne lub nieoczekiwane zachowania funkcjonalne lub niefunkcjonalne systemu.
- ✓ Niepoprawne przepływy sterowania i/lub przepływy danych w systemie.
- ✓ Problemy z prawidłowym i kompletnym wykonywaniem całościowych zadań funkcjonalnych.
- ✓ Problemy z prawidłowym działaniem systemu w środowisku produkcyjnym.
- ✓ Niezgodność działania.
- ✓ Defekty w dokumentacji lub niezgodności pomiędzy działaniem systemu a opisem w dokumentacji.

2. Testowanie w cyklu życia oprogramowania

2.2.3 Testowanie systemowe

skupiamy się na funkcjonalności całego systemu lub produktu

Główna charakterystyka

- ✓ Wynik testów jest używany do podejmowania decyzji o wdrożeniu.
- ✓ W niektórych przypadkach można użyć automatycznego testowania regresji.
- ✓ Poziom testowy w celu spełnienia norm prawnych i regulacyjnych.
- ✓ Środowisko testowe możliwie zbliżone do produkcji.
- ✓ W testowanie systemowe zaangażowani są niezależni testerzy.

Konkretne podejścia i odpowiedzialności

- ✓ Koncentrujemy się na zachowaniu systemu jako produktu gotowego.
- ✓ Można zastosować zarówno funkcjonalne, jak i niefunkcjonalne techniki testowania.
- ✓ Skupiamy się na zaspokajaniu potrzeb biznesowych, np. użycie tablic decyzyjnych.
- ✓ Wyniki fałszywie pozytywne i fałszywie negatywne mogą być spowodowane wadami specyfikacji.
- ✓ W celu zmniejszenia nieprawidłowego wyniku testu testerzy powinni wcześnie zaangażować się w testowanie statyczne.

2. Testowanie w cyklu życia oprogramowania
2.2.4 Testowanie akceptacyjne

ocena gotowości systemu do wdrożenia

Cele testowania akceptacyjnego

- ✓ Budowanie zaufania do systemu.
- ✓ Sprawdzanie kompletności systemu i jego prawidłowego działania.
- ✓ Sprawdzanie zgodności zachowania funkcjonalnego i niefunkcjonalnego systemu ze specyfikacją.

Podstawa testów

- ✓ Procesy biznesowe.
- ✓ Wymagania użytkowników lub wymagania biznesowe.
- ✓ Przepisy, umowy, normy i standardy.
- ✓ Przypadki użycia.
- ✓ Wymagania systemowe.
- ✓ Dokumentacja systemu lub podręczniki dla użytkowników.
- ✓ Procedury instalacji.
- ✓ Raporty z analizy ryzyka.

Przedmioty testów

- ✓ System podlegający testowaniu.
- ✓ Konfiguracja i dane konfiguracyjne systemu.
- ✓ Procesy biznesowe wykonywane na ukończonym systemie.
- ✓ Systemy rezerwowe i ośrodki zastępcze (ang. hot site) (do testowania mechanizmów zapewnienia ciągłości biznesowej i usuwania skutków awarii).
- ✓ Procesy związane z użyciem produkcyjnym i utrzymaniem.
- ✓ Formularze i raporty.

Ponadto podstawą testów mogą być

- ✓ Procedury tworzenia kopii zapasowych i odtwarzania danych.
- ✓ Procedury usuwania skutków awarii.
- ✓ Wymagania niefunkcjonalne.
- ✓ Dokumentacja operacyjna.
- ✓ Instrukcje wdrażania i instalacji.
- ✓ Założenia wydajnościowe.
- ✓ Normy, standardy lub przepisy w dziedzinie zabezpieczeń.

2. Testowanie w cyklu życia oprogramowania
2.2.4 Testowanie akceptacyjne

ocena gotowości systemu do wdrożenia

Typowe defekty i awarie

- ✓ Systemowe przepływy pracy niezgodne z wymaganiami biznesowymi lub wymaganiami użytkowników.
- ✓ Niepoprawnie zaimplementowane reguły biznesowe.
- ✓ Niespełnienie przez system wymagań umownych lub prawnych.
- ✓ Awarie niefunkcjonalne, takie jak podatności zabezpieczeń, niedostateczna wydajność pod dużym obciążeniem bądź nieprawidłowe działanie na obsługiwanej platformie.

Konkretne podejścia i obowiązki

- ✓ Testy akceptacyjne są często obowiązkiem klienta, użytkowników biznesowych i osób odpowiedzialnych za dostarczanie potrzeb biznesowych, takich jak właściciele produktów i inni interesariusze.
- ✓ Zwykle to ostatni poziom testów, ale można to zrobić w przypadku produktów „do powszechnej sprzedaży” podczas ich instalacji lub integracji.
- ✓ Mogą być wykonane w przypadku nowego rozszerzenia funkcjonalnego przed testowaniem systemowym.
- ✓ Mogą być częścią iteracyjnego rozwoju oprogramowania, wykonane na końcu każdego sprintu.

2. Testowanie w cyklu życia oprogramowania
2.2.4 Testowanie akceptacyjne

„ocena gotowości systemu do wdrożenia"

Formy testowania akceptacyjnego:

Testowanie przez użytkownika

Wykonywane przez użytkowników z naciskiem na weryfikację systemu w rzeczywistym lub symulowanym środowisku w celu budowania zaufania w zaspokajaniu potrzeb użytkowników, spełniania wymagań lub procesów biznesowych.

Testowanie pod kątem zgodności z umową i prawem

Wykonywane przez użytkowników lub niezależnych testerów, wykonane zgodnie z kryteriami akceptacji określonymi w umowie lub dowolnymi przepisami, które należy spełnić, aby uzyskać pewność, że zgodność została osiągnięta.

Operacyjne testy akceptacyjne

Wykonywane przez administratorów systemu, z naciskiem na operacyjne działania administracyjne (np. odzyskiwanie po awarii, konserwacja, zarządzanie użytkownikami) w celu zbudowania pewności, że może ono działać poprawnie w środowisku operacyjnym.

Testy Alpha and beta

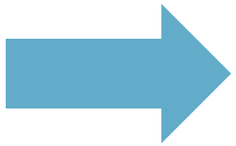
Testy alfa są przeprowadzane po stronie wytwórcy oprogramowania, jego operatorów lub niezależny zespół testowy. Testy beta przeprowadzane są przez klienta w jego lokalizacji. Mogą być wykonane po testach alfa lub bez testów alfa. Celem jest budowanie zaufania wśród osób, które będą korzystać z systemu, który działa dobrze i zgodnie z oczekiwaniami w codziennych warunkach.

2. Testowanie w cyklu życia oprogramowania

PRZYKŁAD [1/2]

Wyobraź sobie, że testujesz oprogramowanie…

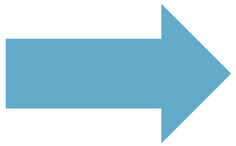
Podczas testowania modułowego programiści sprawdzają komponent odpowiedzialny za komunikację i wykrywają dwa defekty odnośnie niejednoznacznego wymagania do modułu. Programiści próbują przetestować funkcjonalność połączenia, a ty w miarę możliwości możesz wspierać programistów w tworzeniu automatycznych testów komponentów.



Po rozpoczęciu testów integracji otrzymasz zestaw przypadków użycia, spróbujesz uruchomić oprogramowanie, koncentrując się na interfejsach między komponentami. Niestety znajdujesz jedną dziwną sytuację i po dalszym zbadaniu z programistą zgłaszasz błędny interfejs.

2. Testowanie w cyklu życia oprogramowania

PRZYKŁAD [2/2]



Wyobraź sobie, że testujesz oprogramowanie…

Po rozpoczęciu fazy testowania systemowego skupiasz się na podręcznikach użytkownika i upewniasz się, że przepływ pracy systemu jest zgodny z opisem. Nie znajdujesz nowych defektów.

Na koniec następuje inicjacja testów akceptacyjnych. Bierzesz udział w testach beta i wspierasz klienta w sprawdzeniu głównych procesów biznesowych. Niestety zdajesz sobie sprawę, że jedna z reguł biznesowych nie jest prawidłowo zaimplementowana i należy ją traktować jako krytyczny defekt.


2.3
Typy testów

2. Testowanie w cyklu życia oprogramowania
2.3 Wprowadzenie

Typy testów

Typ testu to grupa czynności w testowaniu ukierunkowana na określone charakterystyki modułu lub systemu, oparta na specyficznych celach testowania np.

Ocena funkcjonalnych charakterystyk jakościowych

Takich jak kompletność, prawidłowość i adekwatność.

Ocena niefunkcjonalnych charakterystyk jakościowych

Takich jak niezawodność, wydajność, bezpieczeństwo, kompatybilność czy użyteczność.

Ocena struktury

Czy struktura lub architektura komponentu lub systemu jest poprawna, kompletna i zgodna ze specyfikacjami.

Oceny skutków zmian

Takie jak potwierdzenie usunięcia defektów (testowanie potwierdzające) lub wyszukanie niezamierzonych zmian działania wynikających ze zmian w oprogramowaniu lub środowisku (testowanie regresji).

2. Testowanie w cyklu życia oprogramowania
2.3.1 Testowanie funkcjonalne (black-box)

Charakterystyka testów funkcjonalnych

- ✓ oparte na specyfikacji (np. historyjki użytkowników, wymagania lub w postaci nieudokumentowanej).
- ✓ opisują "co" powinien robić system.
- ✓ można wykonać je na dowolnym poziomie testów.
- ✓ do określania warunków testowych i przypadków testowych można użyć technik czarnoskrzynkowych.
- ✓ staranność testowania funkcjonalnego można zmierzyć na podstawie pokrycia funkcjonalnego (procent elementów danego typu pokrytych przez testy).
- ✓ rozwiązanie danego problemu biznesowego może wymagać specjalnych umiejętności i wiedzy.

2. Testowanie w cyklu życia oprogramowania
2.3.1 Testowanie funkcjonalne (black-box)

PRZYKŁAD

Testy funkcjonalne aplikacji bankowej

- Testy modułowe są projektowane na podstawie tego, jak komponent ma obliczać odsetki na lokacie.
- Testy integracyjne są projektowane na podstawie tego, w jaki sposób informacje o koncie uzyskane z interfejsu użytkownika są przekazywane do logiki biznesowej.
- Testy systemowe są projektowane na podstawie tego, w jaki sposób posiadacze kont mogą ubiegać się o kartę płatniczą do swoich kont.
- Systemowe testy integracyjne są projektowane na podstawie tego, w jaki sposób system korzysta z zewnętrznego serwisu, aby sprawdzić wynik (scoring) właściciela konta.
- W przypadku testowania akceptacyjnego testy są projektowane w oparciu o to, jak pracownik banku obsługuje zatwierdzanie lub odrzucanie wniosku o kredyt.

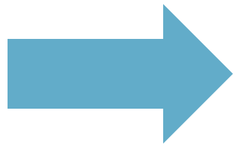
2. Testowanie w cyklu życia oprogramowania
2.3.2 Testowanie niefunkcjonalne

Charakterystyka testów niefunkcjonalnych

- ✓ celem tych testów jest dokonanie oceny charakterystyk systemu.
- ✓ opisują "jak dobrze" zachowuje się system.
- ✓ wykonuje się na dowolnych poziomach testów i tak szybko, jak to możliwe.
- ✓ techniki czarnoskrzynkowe są najprawdopodobniej wykorzystywane do określania warunków testowych i przypadków testowych.
- ✓ staranność testowania niefunkcjonalnego można zmierzyć na podstawie pokrycia niefunkcjonalnego (procent elementów danego typu pokrytych przez testy).
- ✓ mogą być potrzebne specjalne umiejętności i wiedza, aby rozpoznać słabe punkty zastosowanego projektu lub technologii.

2. Testowanie w cyklu życia oprogramowania
2.3.2 Testowanie niefunkcjonalne

PRZYKŁAD



Testy niefunkcjonalne aplikacji bankowej

- Podczas testów modułowych, testy wydajności są zaprojektowane do oceny liczby cykli procesora wymaganych do wykonania złożonej kalkulacji odsetek ogółem.
- Podczas testów integracji modułów projektowane są testy odzwierciedlające sposób, w jaki informacje na temat konta pozyskane w interfejsie użytkownika są przekazywane do warstwy logiki biznesowej.
- Testy systemowe sprawdzają, czy interfejs użytkownika działa we wszystkich obsługiwanych przeglądarkach i urządzeniach mobilnych.
- Podczas systemowych testów integracyjnych, testy niezawodnościowe mają na celu ocenę niezawodności systemu, jeśli mikrourządzenie do oceny zdolności kredytowej odpowiada tak jak powinno.
- Podczas testów akceptacyjnych testy użyteczności mają na celu ocenę dostępności interfejsu do obsługi kredytu bankowego dla osób niepełnosprawnych.

2. Testowanie w cyklu życia oprogramowania
2.3.3 Testowanie białoskrzynkowe

TERMINY

przepływ danych: abstrakcyjna reprezentacja sekwencji i możliwych zmian stanu obiektu danych, gdzie dostępne stany obiektu to utworzenie, użycie lub usunięcie.

testowanie przepływu danych: białoskrzynkowa technika projektowania przypadków testowych, w której testy projektowane są w oparciu o analizę par definicja - użycie zmiennych.

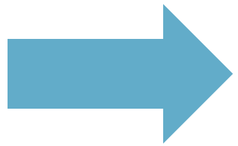
2. Testowanie w cyklu życia oprogramowania
2.3.3 Testowanie białoskrzynkowe

Charakterystyka testów białoskrzynkowych

- ✓ oparte na wewnętrznej strukturze systemu
- ✓ mogą obejmować kod, architekturę, przepływy pracy / danych
- ✓ testy strukturalne mogą być mierzone poprzez pokrycia strukturalne, do jakiego stopnia struktura jest wykonywana, zwykle wyrażone w procentach typu elementu objętego
- ✓ pokrycie jest mierzone w kategoriach wykonywanych instrukcji lub decyzji
- ✓ testowanie białoskrzynkowe jest często stosowane na poziomie integracji komponentów i samych komponentów, a także na poziomie testów systemowych i akceptacyjnych
- ✓ w celu poprawnej interpretacji pokrycia może być konieczne zaangażowanie specjalnych umiejętności i wiedzy

2.3.3 Testowanie białoskrzynkowe

PRZYKŁAD



Testy strukturalne aplikacji bankowej

- Podczas testów modułowych chcemy uzyskać pełne pokrycie decyzji i instrukcji dla wszystkich składników wykonujących obliczenia finansowe.
- Podczas testów integracji komponentów chcemy sprawdzić, w jaki sposób każdy ekran w interfejsie przeglądarki przekazuje dane do następnego ekranu i do logiki biznesowej.
- Testy systemowe mogą obejmować testowanie sekwencji stron internetowych, które mogą występować w aplikacji do badania zdolności kredytowej.
- W przypadku testów integracji systemu mamy na celu sprawdzenie wszystkich możliwych typów zapytań wysyłanych do mikroserwisu punktów kredytowych.
- Podczas testów akceptacyjnych chcemy sprawdzić wszystkie obsługiwane struktury plików danych finansowych i zakresów wartości dla przelewów między bankami.

2. Testowanie w cyklu życia oprogramowania
2.3.4 Testowanie związane ze zmianami

TERMINY

testowanie regresji: testowanie wcześniej przetestowanego modułu lub systemu po modyfikacji w celu upewnienia się, że w wyniku wprowadzonych zmian w niezmienionych obszarach oprogramowania nie zostały wprowadzone lub odkryte defekty. Jest wykonywane w przypadku zmiany oprogramowania lub jego środowiska.

testowanie potwierdzające: testy dynamiczne przeprowadzane po naprawieniu defektów w celu potwierdzenia, że awarie powodowane przez te defekty już nie występują.

2. Testowanie w cyklu życia oprogramowania
2.3.4 Testowanie związane ze zmianami

Cechy testowania związanego ze zmianami

- ✓ testowanie potwierdzające ma miejsce w przypadku naprawy defektów, jest wykonywane w celu potwierdzenia, że defekt jest poprawiony.
- ✓ testowanie regresji jest najczęściej zautomatyzowane, ma miejsce w przypadku zmiany systemu (**silny kandydat do automatyzacji**).
- ✓ zarówno testowanie potwierdzające jak i regresji może być wykonywane na dowolnym poziomie testów.

2. Testowanie w cyklu życia oprogramowania
2.3.4 Testowanie związane ze zmianami

Testowanie potwierdzające

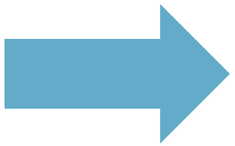
Ma miejsce, gdy w przypadku wykonania danego testu i znalezienia defektu. Po naprawieniu usterki musimy ponownie przetestować nową wersję oprogramowania, aby upewnić się, że defekt już nie istnieje.

Testowanie regresji

Odbywa się po zmianie kodu (defekt jest naprawiony, dodano nowy komponent, zmieniono środowisko, nowa wersja bazy danych lub systemu operacyjnego). Przeprowadzamy testowanie regresji, aby wykryć niezamierzone efekty uboczne.

2. Testowanie w cyklu życia oprogramowania

PRZYKŁAD



Testy związane ze zmianą

- Pracujesz przy tworzeniu oprogramowania, które służy wymiany danych za pośrednictwem komunikacji bezprzewodowej. Okazało się, że znaleziono defekt, który polegał na braku stabilnego połączenia pomiędzy urządzeniem peryferyjnym, a stacją bazową. Po naprawie defektu poinformowano Cię o wersji oprogramowania, która zawiera poprawkę. Wykonałeś retest na nowej wersji oprogramowania, a następnie testowanie regresji, które sprawdziło, że poprawka związana z usunięciem defektu nie wprowadziła nowych defektów.


2.4
Testowanie pielęgnacyjne

2.4 Wprowadzenie

Testowanie pielęgnacyjne

Cechy testowania pielęgnacyjnego

Ma miejsce już po wydaniu produktu kiedy istnieje potrzeba jego utrzymania - zmiana systemu w środowisku produkcyjnym. Istnieje wiele różnych celów testów pielęgnacyjnych, np. naprawianie usterek wykrytych podczas użytkowania produkcyjnego, nowych funkcji lub usuniętych funkcjonalności. Konieczne jest również utrzymywanie lub poprawa wybranych charakterystyk jakości oprogramowania, np. wydajność, kompatybilność, niezawodność, bezpieczeństwo i przenaszalność.

2. Testowanie w cyklu życia oprogramowania
2.4 Testowanie pielęgnacyjne

Testy konserwacyjne są wykonywane w celu określenia poprawności wprowadzanych zmian a także w celu sprawdzenia wszelkich skutków ubocznych

Zakres testowania pielęgnacyjnego zależy od:

Poziomu ryzyka związanego ze zmianą

Wielkości dotychczasowego systemu

Wielkości wprowadzonej zmiany

2. Testowanie w cyklu życia oprogramowania

2.4.1 Zdarzenia wywołujące pielęgnację

Testowanie pielęgnacyjne

Modyfikacja np.

- zaplanowane udoskonalenia (nowych wersji oprogramowania)
- zmiany korekcyjne i awaryjne
- zmiany środowiska operacyjnego (np. planowe uaktualnienia systemu operacyjnego lub bazy danych)
- uaktualnienia oprogramowania do powszechnej sprzedaży oraz poprawki usuwające defekty i podatności zabezpieczeń

Migracja np.

- przejście z jednej platformy na inną, co może wiązać się z koniecznością przeprowadzenia testów produkcyjnych nowego środowiska i zmienionego oprogramowania bądź testów konwersji danych (w przypadku migracji danych z innej aplikacji do pielęgnowanego systemu)

Wycofanie np.

- aplikacja jest wycofana z rynku

PRZYKŁADY

2. Testowanie w cyklu życia oprogramowania
2.4.2 Analiza wpływu związana z pielęgnacją

Co to jest analiza wpływu ?

analiza wpływu: identyfikacja wszystkich produktów pracy, na które zmiana ma wpływ, w tym oszacowanie zasobów potrzebnych do przeprowadzenia zmiany.

Zakres testów opiera się na analizie wpływu, w której identyfikujemy planowane konsekwencje i możliwe skutki uboczne zmiany. Na podstawie obszarów dotkniętych zmianą możemy zidentyfikować wpływ zmiany na istniejące testy. W wyniku analizy można wykonać zestaw testów regresji.

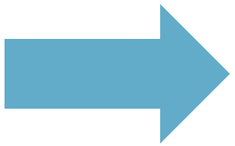
Taką analizę wpływu wykonuje się przed wprowadzeniem zmiany.

2. Testowanie w cyklu życia oprogramowania
2.4.2 Analiza wpływu związana z pielęgnacją

PRZYKŁAD

Jaki wpływ będzie miała nasza zmiana ?

Twój zespół ze sporym powodzeniem wdrożył na rynek nowe oprogramowanie. Po dwóch miesiącach od wdrożenia znajdujecie poważny defekt. Zespół programistów chce się upewnić, że ich naprawa rozwiąże problem.



Analiza wpływu jest przeprowadzana przed wprowadzeniem wymaganych zmian. W oparciu o analizę wpływu możliwe jest wybranie aktywności testowych i wybór zestawu testów regresji.

2. Testowanie w cyklu życia oprogramowania
2.4.2 Analiza wpływu związana z pielęgnacją

Co może być trudnego w wykonaniu analizy wpływu?

- niezadowalająca dbałość o łatwość konserwacji podczas projektowania
- wsparcie narzędziowe nie istnieje lub jest niewystarczające
- brak wiedzy dziedzinowej osób zaangażowanych w testowanie
- przypadki testowe są nieudokumentowane lub są nieaktualne
- specyfikacje (np. wymagania biznesowe, historyjki użytkowników, architektura) są nieaktualne lub ich brakuje
- brak dwukierunkowego śledzenia pomiędzy testami a podstawą testów

2. Testowanie w cyklu życia oprogramowania

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

03

Testowanie statyczne.

3. Testowanie statyczne
Cele Nauczania

3.1 Podstawy testowania statycznego

FL-3.1.1 (K1) Kandydat potrafi rozpoznać typy produktów pracy związanych z oprogramowaniem, które mogą być badane przy użyciu poszczególnych

FL-3.1.2 (K2) Kandydat potrafi opisać na przykładach wartość testowania statycznego

FL-3.1.3 (K2) Kandydat potrafi wyjaśnić różnicę między technikami testowania statycznego i dynamicznego z uwzględnieniem celów, typów identyfikowanych defektów oraz roli tych technik w cyklu życia oprogramowania

3. Testowanie statyczne
Cele Nauczania

3.2 Proces przeglądu

FL-3.2.1 (K2) Kandydat potrafi podsumować czynności wykonywane w ramach procesu przeglądu produktów pracy

FL-3.2.2 (K1) Kandydat potrafi rozpoznać poszczególne role i obowiązki w przeglądzie formalnym

FL-3.2.3 (K2) Kandydat potrafi wyjaśnić różnice między poszczególnymi typami przeglądów: przeglądem nieformalnym, przejrzeniem, przeglądem technicznym i inspekcją

FL-3.2.4 (K3) Kandydat potrafi zastosować technikę przeglądu do produktu pracy w celu wykrycia defektów

FL-3.2.5 (K2) Kandydat potrafi wyjaśnić, jakie czynniki decydują o powodzeniu przeglądu

Podstawy **testowania** statycznego

3. Testowanie statyczne
3.1 Wprowadzenie

Podstawy testowania statycznego

testowanie statyczne: Testowanie produktu prac bez uruchamiania kodu.

Testowanie dynamiczne wymaga wykonania kodu, zaś testowanie statyczne opiera się na wykonywaniu **przeglądów** (tj. ręcznym badaniu produktów roboczych) i **analizie statycznej** (tj. dokonanie oceny kodu przy pomocy narzędzi). Zarówno przeglądy i analiza statyczna nie wymagają wykonania kodu.

3. Testowanie statyczne
3.1 Wprowadzenie

Podstawy testowania statycznego

Analiza statyczna:

- istotna dla systemów komputerowych krytycznych ze względów bezpieczeństwa
- jest ważną częścią testów bezpieczeństwa
- w modelach zwinnego rozwoju oprogramowania jest częścią ciągłej integracji, oraz częścią automatycznego systemu budowania i dostarczania oprogramowania

3. Testowanie statyczne

3.1.1 Produkty pracy badane metodą testowania statycznego

Co możemy przetestować bez uruchamiania kodu?

- specyfikacje (w tym: wymagania biznesowe, wymagania funkcjonalne i wymagania w zakresie bezpieczeństwa)
- opowieści, historyjki użytkowników i kryteria akceptacji
- architekturę i specyfikacje projektowe
- kod
- testalia (w tym: plany testów, procedury testowe, przypadki testowe i skrypty testów automatycznych)
- podręczniki użytkownika
- strony internetowe
- umowy, plany projektów, harmonogramy i budżety
- modele takie jak diagramy aktywności, które mogą być używane do testowania opartego na modelu

3. Testowanie statyczne

3.1.1 Produkty pracy badane metodą testowania statycznego

Rodzaje testów statycznych

Przeglądy

Stosowanie do każdego produktu roboczego, który wiemy, w sposób czytać i rozumieć.

Analiza statyczna

Stosowanie do dowolnego produktu roboczego o formalnej strukturze (zazwyczaj kodu lub modeli) za pomocą dedykowanego narzędzia. Może się np. odbywać z wykorzystaniem narzędzia do sprawdzania błędów ortograficznych lub gramatycznych.

3. Testowanie statyczne
3.1.2 Zalety testowania statycznego

Główne zalety testowania statycznego

Jest bardzo opłacalne, aby usunąć defekty na wczesnym etapie cyklu życia

Wczesne wykrywanie defektów przed rozpoczęciem testowania dynamicznego

Defekty znalezione podczas testowania statycznego są tańsze do usunięcia niż te z testów dynamicznych (dodatkowe koszty testów regresji i potwierdzających)

3. Testowanie statyczne
3.1.2 Zalety testowania statycznego

Dodatkowe korzyści:

- identyfikowanie defektów, które trudno jest wykryć metodą testowania dynamicznego
- zapobieganie wystąpieniu defektów w projekcie i kodzie poprzez wykrywanie niedociągnięć takich jak: niespójności, niejednoznaczności, sprzeczności, nieścisłości, przeoczenia czy elementy nadmiarowe w wymaganiach

3. Testowanie statyczne
3.1.2 Zalety testowania statycznego

Dodatkowe korzyści:

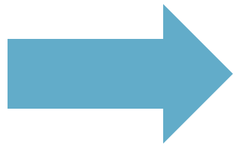
- zwiększenie wydajności prac programistycznych między innymi dzięki udoskonalonemu projektowaniu i utrzymywalności kodu
- obniżenie kosztów i zmniejszenie czasochłonności wytwarzania oprogramowania i testowania
- obniżenie łącznego kosztu jakości w całym cyklu życia oprogramowania poprzez zmniejszenie liczby awarii na późniejszych etapach tego cyklu lub po przekazaniu do eksploatacji
- usprawnienie komunikacji miedzy członkami zespołu uczestniczącymi w przeglądach

3. Testowanie statyczne

3.1.2 Zalety testowania statycznego

Ogromna zaleta testowania statycznego

Jako tester w zespole projektowym brałeś udział w testowaniu statycznym. Podczas przeglądu wymagań została wykryta niespójność z innymi wymaganiami. Pomogło to wyeliminować defekty w wymaganiach, które w przyszłości mogłyby spowodować awarię systemu.



Ponadto, podczas analizy statycznej kodu okazało się, że co prawda działa on poprawnie, ale zastosowano bardzo nieefektywny algorytm. Zdecydowano o poprawieniu algorytmu, co umożliwi usprawnienie działania systemu.

3. Testowanie statyczne

3.1.3 Różnice między testowaniem statycznym a dynamicznym

ten sam cel, ale inny rodzaj defektów

Testy statyczne i dynamiczne mają te same cele, ale znajdują różne typy defektów. Oba rodzaje testów mogą zmierzyć jakość oprogramowania poprzez identyfikację defektów we wczesnej fazie cyklu życia ale powinny być uważane jako komplementarne względem siebie.

Wynik testów dynamicznych koncentruje się na zachowaniu zewnętrznym systemu, podczas gdy testowanie statyczne może być wykorzystywane do poprawy ogólnej jakości produktów pracy.

3. Testowanie statyczne

3.1.3 Różnice między testowaniem statycznym a dynamicznym

ten sam cel, ale inny rodzaj defektów

Główną różnicą jest to, że testy statyczne raczej znajdują źródło przyszłych awarii, podczas gdy testy dynamiczne ujawniają awarie spowodowane defektami podczas wykonywania oprogramowania. W przypadku niedostatecznych testów statycznych defekt w oprogramowaniu może znajdować się przez bardzo długi czas - aż do momentu, gdy spowoduje awarię, np. mało realistyczny scenariusz lub rzadko używana funkcja.

Znajdowanie defektów w testach statycznych na ogół kosztuje mniej wysiłku niż w testowaniu dynamicznym.

3. Testowanie statyczne
3.1.3 Różnice między testowaniem statycznym a dynamicznym

Typowe defekty, które znajdziemy podczas testowania statycznego [1/2]:

- ✓ defekty w wymaganiach (takie jak: niespójności, niejednoznaczności, opuszczenia, sprzeczności, nieścisłości, przeoczenia, wewnętrzne sprzeczności czy elementy nadmiarowe w wymaganiach)
- ✓ defekty w projekcie np. nieefektywne algorytmy lub struktury baz danych, wysoki stopień sprzężenia (ang. coupling) czy mała spójność (ang. cohesion)
- ✓ defekty w kodzie (np. zmienne z niezdefiniowanymi wartościami, zmienne zadeklarowane lecz nigdy nie używane, niedostępny kod, powielony kod)

3. Testowanie statyczne
3.1.3 Różnice między testowaniem statycznym a dynamicznym

Typowe defekty, które znajdziemy podczas testowania statycznego [2/2]:

- ✓ odchylenia od standardów (np. brak zgodności ze standardami tworzenia kodu)
- ✓ niepoprawne specyfikacje interfejsów (np. użycie różnych jednostek miary w systemie wywołującym i systemie wywoływanym)
- ✓ wrażliwości zabezpieczeń (np. podatność na przepełnienie bufora)
- ✓ luki lub nieścisłości w zakresie śledzenia powiązań lub pokrycia (np. brak testów odpowiadających kryteriom akceptacji).

Łatwiej i taniej jest na ogół znaleźć i naprawić defekty znalezione podczas testów statycznych niż podczas testów dynamicznych.

3. Testowanie statyczne

3.1.3 Różnice między testowaniem statycznym a dynamicznym

Istotna różnica polega na tym, że podczas testowania statycznego możemy znaleźć takie defekty, które są niemal niemożliwe do wykrycia podczas testów dynamicznych

nieprawidłowa modularyzacja (niewystarczająca skalowalność)

niski poziom ponownego wykorzystania modułów

kod trudny do analizy i modyfikacji bez wprowadzania nowych defektów

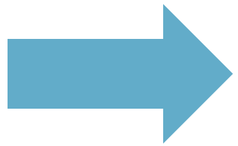
3. Testowanie statyczne

3.1.3 Różnice między testowaniem statycznym a dynamicznym

PRZYKŁAD

Scenariusz 1

System działał dobrze, dopóki nie zdecydowano się go rozszerzyć na większą skalę do kolejnych 5 dużych klientów. Okazało się wtedy, że kod jest trudny do analizy i modyfikacji bez wprowadzania nowych defektów. Podczas testów dynamicznych okazało się, że komponenty mają bardzo niską przydatność do ponownego użycia.



Scenariusz 2

Wymagania do nowej funkcjonalności zostały napisane przez osobę bez dostatecznej wiedzy o dotychczasowym działaniu systemu. Podczas przeglądu wymagań okazało się, że są one niespójne z dotychczasowymi. Testy statyczne pomogły wyeliminować defekty w wymaganiach co w przyszłości mogłyby spowodować awarię systemu.

Proces przeglądu

Bardziej szczegółowy opis procesu przeglądu produktów pracy (w tym ról i technik związanych z przeglądem) zawiera standard międzynarodowy **ISO/IEC 20246**.

3. Testowanie statyczne
3.2 Proces przeglądu

Wprowadzenie

Przeglądy to ręczna forma testowania statycznego. Ich forma może różnić się od bardzo nieformalnych do bardzo formalnych. Wszystko zależy od modelu cyklu życia oprogramowania, dojrzałości procesu wytwarzania oprogramowania, złożoności produktu pracy będącego przedmiotem przeglądu, wymogów prawnych czy konieczności prowadzenia ścieżki audytu. Nieformalne przeglądy nie są w ogóle dokumentowane, a bardziej formalne przeglądy są zgodne z określonymi procedurami i koncentrują się na dokumentowaniu wyników.

3. Testowanie statyczne
3.2 Proces przeglądu

Wprowadzenie

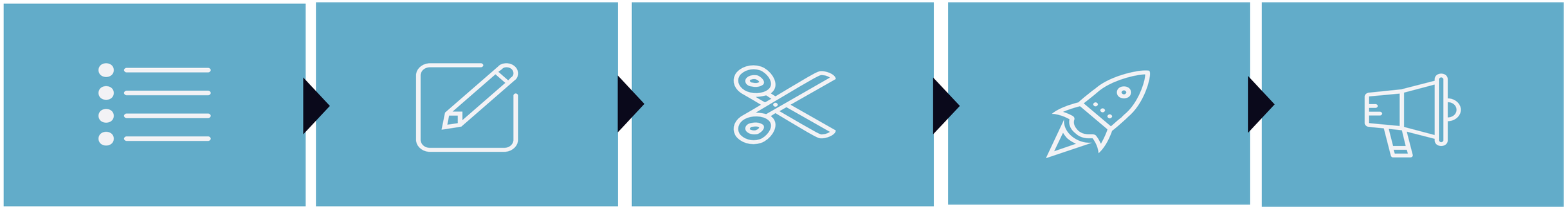
Celem przeglądu może być:

- szukanie defektów
- poszerzanie wiedzy
- edukowanie uczestników (np. członków zespołu)
- omawianie zagadnień i podejmowanie decyzji

3. Testowanie statyczne

3.2.1 Proces przeglądu produktów pracy

Proces przeglądu



Planowanie

Rozpoczęcie przeglądu

Przegląd indywidualny

Przekazanie informacji o problemach i analiza problemów

Usunięcie defektów i raportowanie

3. Testowanie statyczne

3.2.1 Proces przeglądu produktów pracy

1. Planowanie

- ✓ Określenie zakresu prac, w tym celu przeglądu i dokumentów (lub ich części).
- ✓ Oszacowanie nakładu pracy i ram czasowych.
- ✓ Wskazanie typu przeglądu, ról, czynności i list kontrolnych.
- ✓ Wybór osób, które mają wziąć udział w przeglądzie oraz wyznaczenie im ról.
- ✓ Określenie kryteriów wejścia i wyjścia.
- ✓ Sprawdzenie, czy zostały spełnione kryteria wejścia.

2. Rozpoczęcie przeglądu

- ✓ Rozesłanie produktu pracy (w formie fizycznej lub elektronicznej) oraz innych materiałów (np. formularzy dziennika problemów, list kontrolnych i innych powiązanych produktów pracy).
- ✓ Udzielenie uczestnikom wyjaśnień dotyczących zakresu, celów, procesu, ról i produktów pracy.
- ✓ Udzielenie odpowiedzi na ewentualne pytania uczestników dotyczące przeglądu.

3. Przegląd indywidualny

- ✓ Dokonanie przeglądu całości lub części produktu pracy.
- ✓ Odnotowanie potencjalnych defektów oraz zaleceń i pytań.

3. Testowanie statyczne
3.2.1 Proces przeglądu produktów pracy

4. Przekazanie informacji o problemach i analiza problemów

- ✓ Przekazanie informacji o zidentyfikowanych potencjalnych defektach (np. na spotkaniu przeglądowym).
- ✓ Przeanalizowanie defektów (w tym wyznaczenie osób za nie odpowiedzialnych oraz określenie statusu defektów).
- ✓ Dokonanie oceny i udokumentowanie charakterystyk jakościowych.
- ✓ Dokonanie oceny wniosków z przeglądu pod kątem kryteriów wyjścia w celu podjęcia decyzji (odrzucenie produktu pracy; wymagane poważne zmiany; akceptacja, być może z niewielkimi zmianami).

5. Usunięcie defektów i raportowanie

- ✓ Utworzenie raportów o defektach dotyczących wykrytych defektów wymagających wprowadzenia zmian.
- ✓ Usunięcie defektów wykrytych w produkcie pracy będącym przedmiotem przeglądu (czynność tę zwykle wykonuje autor).
- ✓ Poinformowanie odpowiedniej osoby lub odpowiedniego zespołu o defektach wykrytych w produkcie pracy związanym z produktem będącym przedmiotem przeglądu.
- ✓ Zarejestrowanie zaktualizowanego statusu defektów (w przypadku przeglądów formalnych), włączając w to potencjalną zgodę autora komentarza.
- ✓ Zebranie miar (w przypadku bardziej formalnych typów przeglądów).
- ✓ Sprawdzenie, czy zostały spełnione kryteria wyjścia (w przypadku bardziej formalnych typów przeglądów).
- ✓ Zaakceptowanie produktu pracy po spełnieniu kryteriów wyjścia.

3.2.1 Proces przeglądu produktów pracy

PRZYKŁAD [1/2]

Przegląd zestawu przypadków testowych

Jesteś testerem nowej funkcjonalności i właśnie utworzyłeś zestaw manualnych przypadków testowych, które wymagają sprawdzenia.

1. Zaczynasz od wybrania uczestników przeglądu i przygotowania listy kontrolnej dla tego działania.
2. Prosisz kolegów o przejrzenie, wysyłając e-maila do wszystkich uczestników przeglądu i odpowiednio udostępniasz przypadki testowe z listą kontrolną. Wyjaśniasz, co ma zostać poddane przeglądowi i w razie potrzeby odpowiadasz na wszelkie pytania.

3.2.1 Proces przeglądu produktów pracy

PRZYKŁAD [2/2]

Przegląd zestawu przypadków testowych

3. Wreszcie przegląd indywidualny ma miejsce - każdy uczestnik odnotowuje potencjalne defekty i ewentualne pytania.
4. Kiedy zakończy się przygotowanie indywidualne, potencjalne defekty (obserwacje) są komunikowane, analizowane.
5. Ostatnim krokiem jest naprawienie i zgłoszenie usterek znalezionych podczas sprawdzania i ostateczna akceptacja zestawu przypadków testowych, które zostaną dodane do głównego repozytorium.

3. Testowanie statyczne
3.2.2 Role i obowiązki w przeglądzie formalnym

Autor

- Tworzy produkt pracy będący przedmiotem przeglądu.
- Usuwa defekty w produkcie pracy będącym przedmiotem przeglądu (jeśli jest to konieczne).

Kierownictwo

- Odpowiada za zaplanowanie przeglądu.
- Podejmuje decyzję o przeprowadzeniu przeglądu.
- Wyznacza pracowników oraz określa budżet i ramy czasowe.
- Monitoruje na bieżąco opłacalność przeglądu.
- Wykonuje decyzje kontrolne w przypadku uzyskania niezadowalających wyników.

Facylitator (zwany często moderatorem)

- Dba o sprawny przebieg spotkań związanych z przeglądem (o ile je prowadzi).
- Występuje w roli mediatora, jeśli konieczne jest pogodzenie różnych punktów widzenia.
- Jest często osobą, od której zależy powodzenie przeglądu.

3. Testowanie statyczne
3.2.2 Role i obowiązki w przeglądzie formalnym

Lider przeglądu

- Ponosi ogólną odpowiedzialność za przegląd.
- Decyduje o tym, kto ma wziąć udział w przeglądzie oraz określa miejsce i termin przeglądu.

Protokolant

- Gromadzi potencjalne defekty wykryte w ramach przeglądu indywidualnego.
- Rejestruje nowe potencjalne defekty, otwarte punkty i decyzje podczas spotkania związanego z przeglądem (gdy ma ono miejsce).

Przeglądający

- Mogą być ekspertami merytorycznymi, osobami pracującymi przy projekcie, interesariuszami zainteresowanymi produktem pracy i/lub osobami mającymi określone doświadczenie techniczne lub biznesowe.
- Identyfikują potencjalne defekty w produkcie pracy będącym przedmiotem przeglądu.
- Mogą reprezentować różne punkty widzenia (np. punkt widzenia testera, programisty, użytkownika, operatora, analityka biznesowego, specjalisty ds. użyteczności itd.).

Może nie być potrzeby wyznaczania protokolanta ze względu na wykorzystanie narzędzi wspomagających proces przeglądu (zwłaszcza rejestrowanie defektów, otwartych punktów i decyzji)

3. Testowanie statyczne

3.2.3 Typy przeglądów

Który rodzaj przeglądu wybrać ?

Głównym celem przeprowadzenia przeglądu jest **wykrycie defektów.** Oprócz tego wybrany typ przeglądu powinien opierać się na potrzebach projektowych, dostępnych zasobach, rodzaju produktu i ryzyka, dziedzinie biznesowej, kulturze korporacyjnej i innych kryteriach.

Ten sylabus koncentruje się na 4 najczęściej spotykanych rodzajach przeglądów i ich atrybutach.

3. Testowanie statyczne
3.2.3 Typy przeglądów

Który rodzaj przeglądu wybrać ?

Przegląd nieformalny (np. sprawdzenie koleżeńskie, przegląd w parach)

- Główny cel: wykrycie potencjalnych defektów
- Brak formalnego procesu
- Może być przeprowadzony przez współpracownika
- Wyniki niekoniecznie są udokumentowane
- Powszechnie stosowany podczas zwinnego wytwarzania oprogramowania

Przejrzenie

- Główny cel: znaleźć usterki, rozważyć alternatywne wdrożenia
- Spotkanie przeglądowe zazwyczaj prowadzi autor
- Można użyć list kontrolnych
- Może przybrać formę scenariuszy, próbnych przebiegów lub symulacji
- Raporty przeglądowe mogą być tworzone
- Może się dość różnić w praktyce od nieformalnego do bardzo formalnego

3. Testowanie statyczne
3.2.3 Typy przeglądów

Który rodzaj przeglądu wybrać ?

Przegląd techniczny

- Główne cele: osiągnięcie konsensusu, wykrycie potencjalnych wad
- Twórz nowe pomysły, motywuj i umożliwiaj twórcom ulepszanie przyszłych produktów pracy,
- Recenzenci: techniczni współpracownicy lub eksperci techniczni w tej samej lub innej dyscyplinie
- Wymagane jest indywidualne przygotowanie przed spotkaniem przeglądowym
- Spotkanie przeglądowe powinno być prowadzone przez przeszkolonego facylitatora (nie autora)
- Protokolant jest obowiązkowy
- Tworzone są potencjalne raporty defektów i raporty z przeglądu

3. Testowanie statyczne
3.2.3 Typy przeglądów

Który rodzaj przeglądu wybrać ?

Inspekcja

- Główne cele: wykrywanie potencjalnych wad, ocena jakości i budowanie zaufania do przeglądanego produktu,
- Podąża za zdefiniowanym procesem z formalnie udokumentowanymi wynikami, w oparciu o reguły i listy kontrolne
- Jasno określone role, które są obowiązkowe
- Wymagane jest indywidualne przygotowanie przed spotkaniem przeglądowym
- Recenzentami są albo współpracownicy autora lub eksperci z innych dziedzin, które są istotne dla produktu roboczego
- Wykorzystywane są określone kryteria wejścia i wyjścia
- Protokolant jest obowiązkowy
- Autor nie może pełnić funkcji lidera przeglądu, czytającego ani protokolanta
- Potencjalne raporty błędów i raport przeglądu są zapisane

3. Testowanie statyczne
3.2.3 Typy przeglądów

Podsumowanie

- ✓ Możliwe, że **jeden produkt roboczy** jest poddawany **więcej niż jednemu przeglądowi**. Może się zdarzyć, gdy przed oficjalną inspekcją chcemy przeprowadzić nieformalny przegląd, aby upewnić się, że jesteśmy gotowi na ostateczny przegląd formalny.
- ✓ Wszystkie opisane przeglądy można wykonywać jako **przeglądy koleżeńskie** - współpracowników na podobnym szczeblu w organizacji. Rezultaty przeglądu produktu pracy mogą być różne w zależności od typu przeglądu oraz stopnia jego sformalizowania.

3. Testowanie statyczne

3.2.4 Stosowanie technik przeglądu

Techniki przeglądu

Podczas indywidualnego przygotowania ważne jest stosowanie różnych technik. Podane techniki mogą być stosowane w różnych typach poglądów, ale ich skuteczność może się różnić w zależności od wybranego typu przeglądu.

Przykłady indywidualnych technik przeglądu

- Ad hoc
- Oparty na liście kontrolnej
- Czytanie oparte na perspektywie
- Scenariusze i przebiegu próbne
- Oparty na rolach

3. Testowanie statyczne
3.2.4 Stosowanie technik przeglądu

Techniki przeglądu

01

Ad hoc

- niewiele lub brak wskazówek odnośnie sposobu przeglądu
- przeglądający czytają produkt roboczy, identyfikują i dokumentują stwierdzone problemy
- wymagane jest niewielkie przygotowanie
- wynik przeglądu zależy od umiejętności przeglądającego

02

Oparty na liście kontrolnej

- systematyczna technika
- przeglądający są w stanie wykryć defekty na podstawie listy kontrolnej, która wynika z doświadczenia przeglądających
- dostosowany do specyfiki produktu roboczego
- główną zaletą jest systematyczne pokrycie najczęściej występujących defektów
- recenzenci nie powinni ograniczać się do listy kontrolnej

03

Scenariusze i przebiegi próbne

- przeglądający otrzymują wskazówki dotyczące działania produktu roboczego
- mając przygotowane scenariusze, możemy wykonać "suche przebiegi" w oparciu o oczekiwane wykorzystanie produktu roboczego
- przeglądający nie powinni ograniczać się do podanych scenariuszy, ale także przeglądać nieszablonowo

3. Testowanie statyczne
3.2.4 Stosowanie technik przeglądu

Techniki przeglądu

Takie czytanie jest uznawane za najbardziej skuteczną technikę przeglądu wymagań

04

Oparty na rolach

- rozważamy **wykorzystanie produktu roboczego z perspektywy poszczególnych interesariuszy**
- typowa rola obejmuje określony typ użytkownika (np. bardzo doświadczony / niedoświadczony, dziecko / dorosły itp.) lub dedykowaną rolę w organizacji (np. administrator, tester użyteczności)

05

Czytanie oparte na perspektywie

- przyjmujemy **punkt widzenia różnych interesariuszy** podczas przeglądu, np. użytkownik końcowy, dział marketingu, projektant lub tester
- punkty widzenia różnych interesariuszy umożliwiają bardziej wnikliwe przeprowadzenie przeglądu indywidualnego
- zwykle **wymaga wykorzystania sprawdzanego produktu roboczego w celu wyprowadzenia z niego produktu końcowego**, np. tester próbowałby opracować projekt testów akceptacyjnych podczas czytania wymagań, żeby upewnić się, że wszystkie potrzebne informacje zostały uwzględnione

3. Testowanie statyczne

3.2.5 Czynniki powodzenia związane z przeglądami

Organizacyjne czynniki sukcesu

- Jasne i mierzalne kryteria wyjścia zdefiniowane podczas planowania
- Wybranie odpowiedniego rodzaju przeglądu
- Zastosowane są różne techniki przeglądu
- Używana lista kontrolna jest aktualna i uwzględnia główne czynniki ryzyka
- W przypadku dużych dokumentów są one pisane i przeglądane partiami
- Przeglądający mają wystarczająco dużo czasu na przygotowanie
- Przeglądy są planowane z wyprzedzeniem i mają pełne wsparcie kierownictwa

3. Testowanie statyczne

3.2.5 Czynniki powodzenia związane z przeglądami

Czynniki sukcesu związane z ludźmi [1/2]

- Zaangażowanie odpowiednich osób
- Testerzy to istotni uczestnicy przeglądu
- Uczestnicy przeznaczają odpowiednio dużo czasu i uwagi na przegląd
- Przeglądy są prowadzone na małych fragmentach (nie tracimy koncentracji)
- Wykryte defekty są przyjęte do wiadomości, potwierdzone i przetwarzane obiektywnie

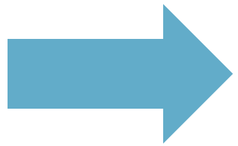
3.2.5 Czynniki powodzenia związane z przeglądami

Czynniki sukcesu związane z ludźmi [2/2]

- Dobre zarządzanie czasem przeglądu
- Atmosfera wzajemnego zaufania
- Unikanie negatywnych gestów i zachowań
- Zapewnienie uczestnikom należytego szkolenia
- Pozytywna atmosfera sprzyjająca dzieleniu się wiedzą

3.2.5 Czynniki powodzenia związane z przeglądami

PRZYKŁAD



Nie zawsze jest to historia sukcesu

Termin realizacji projektu upływa za 5 dni. Jesteś jedynym testerem w zespole. Niestety wprowadzono nową zmianę w wymaganiach i na podstawie analizy wpływu należy wykonać zestaw testów regresji (szacowany na 4 dni robocze). Wymagane jest dokonanie przeglądu analizy wpływu przez rozpoczęciem testów, ale z powodu napiętego harmonogramu i okresu wakacyjnego selekcja uczestników nie była dobrze wykonana. Zespół miał jedynie godzinę na zapoznanie się z wynikiem analizy wpływu. Jak można sobie wyobrazić, nie znaleźli żadnych poważniejszych problemów, a wykonywanie testów rozpoczęto w następnym dniu zgodnie z planem.

Co można było zrobić lepiej w takim przypadku?

3. Testowanie statyczne

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

04

Techniki testowania.

4. Techniki testowania
Cele Nauczania

4.1 Kategorie technik testowania

FL-4.1.1 (K2) Kandydat potrafi wyjaśnić cechy charakterystyczne i elementy wspólne czarnoskrzynkowych technik testowania, białoskrzynkowych technik testowania oraz technik testowania opartych na doświadczeniu, a także różnice między nimi

4.2 Czarnoskrzynkowe techniki testowania

FL-4.2.1 (K3) Kandydat potrafi zaprojektować przypadki testowe z podanych wymagań metodą podziału na klasy równoważności

FL-4.2.2 (K3) Kandydat potrafi zaprojektować przypadki testowe z podanych wymagań metodą analizy wartości brzegowych

FL-4.2.3 (K3) Kandydat potrafi zaprojektować przypadki testowe z podanych wymagań metodą testowania w oparciu o tablicę decyzyjną

FL-4.2.4 (K3) Kandydat potrafi zaprojektować przypadki testowe z podanych wymagań metodą testowania przejść pomiędzy stanami

4. Techniki testowania
Cele Nauczania

FL-4.2.5 (K2) Kandydat potrafi wyjaśnić, w jaki sposób można wyprowadzać przypadki testowe z przypadku użycia

4.3 Białoskrzynkowe techniki testowania

FL-4.3.1 (K2) Kandydat potrafi wyjaśnić pojęcie pokrycia instrukcji kodu.

FL-4.3.2 (K2) Kandydat potrafi wyjaśnić pojęcie pokrycia decyzji.

FL-4.3.3 (K2) Kandydat potrafi wyjaśnić korzyści wynikające z pokrycia instrukcji kodu i pokrycia decyzji

4.4 Techniki testowania oparte na doświadczeniu

FL-4.4.1 (K2) Kandydat potrafi wyjaśnić pojęcie zgadywania błędów

FL-4.4.2 (K2) Kandydat potrafi wyjaśnić pojęcie testowania eksploracyjnego

FL-4.4.3 (K2) Kandydat potrafi wyjaśnić pojęcie testowania w oparciu o listę kontrolną


4.1
Kategorie **technik testowania**

4. Techniki testowania
4.1 Wprowadzenie

Kategorie technik testowania

Techniki testowania są pomocne podczas identyfikacji warunków testowych, przypadków testowych i danych testowych.
Nie zawsze wykorzystujemy wszystkie techniki.
Wybór techniki zależy od wielu czynników.

4. Techniki testowania

Co wziąć pod uwagę przy doborze właściwej techniki ?

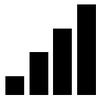
- ✓ typ modułu lub systemu
- ✓ złożoność modułu lub systemu
- ✓ obowiązujące przepisy i normy
- ✓ wymagania klienta lub wymagania wynikające z umów
- ✓ poziomy i typy ryzyka
- ✓ cele testów
- ✓ dostępną dokumentację
- ✓ wiedzę i umiejętności testerów
- ✓ dostępne narzędzia, czas i budżet
- ✓ model cyklu życia oprogramowania
- ✓ przewidywany sposób korzystania z oprogramowania
- ✓ dotychczasowe doświadczenie
- ✓ typy defektów spodziewane w modułach i systemach

4. Techniki testowania

Czy to jest właściwa technika testowania ?

Poziom testów

Ważne jest, aby wiedzieć, że niektóre techniki opisane w tym rozdziale mają zastosowanie tylko na określonym poziomie, podczas gdy inne mogą być stosowane na wszystkich poziomach testów. Generalnie w celu osiągnięcia najlepszych wyników używamy kombinacji różnych technik testowania.



Poziom formalności

Techniki testowania są zróżnicowane od bardzo nieformalnych do bardzo formalnych, które są dobrze udokumentowane. Poziom formalności zależy od kontekstu testowania, tego czy pracujemy w dojrzałym modelu wytwarzania oprogramowania, od ograniczeń czasowych, tego kto jest zaangażowany w testowanie i jak jesteśmy wykwalifikowani.

4. Techniki testowania

4.1.1 Kategorie technik testowania i ich cechy charakterystyczne

Kategoryzacja technik testowania

Techniki czarnoskrzynkowe

(behawioralne, oparte na specyfikacji)

- ✓ w oparciu o podstawę testową, która jest dowolną wyrocznią testową (np. specyfikacja przypadków użycia, formalna dokumentacja, historyjki użytkowników, procesy biznesowe)
- ✓ testy funkcjonalne i niefunkcjonalne
- ✓ koncentrują się na danych wejściowych i wyjściowych obiektu testowego

Techniki białoskrzynkowe

(strukturalne, oparte na strukturze)

- ✓ w oparciu o szczegółową analizę struktury wewnętrznej (architektury, projektu, kodu)
- ✓ koncentrują się na strukturze i przetwarzaniu wewnątrz obiektu testowego

Techniki oparte na doświadczeniu

- ✓ w oparciu o doświadczenie i wiedzę programistów, testerów
- ✓ równoczesne projektowanie, implementacja i wykonanie testu
- ✓ zwykle połączone z bardziej systematycznymi technikami i uważane są za uzupełnienie tych technik

4. Techniki testowania

4.1.1 Kategorie technik testowania i ich cechy charakterystyczne

Cechy charakterystyczne

Czarnoskrzynkowe techniki testowania

- Warunki testowe, przypadki testowe i dane testowe pochodzą z podstawy testowania, która może obejmować wymagania oprogramowania, specyfikacje, przypadki użycia i historyjki użytkowników
- Przypadki testowe mogą być wykorzystywane do wykrycia luk pomiędzy wymaganiami a implementacją wymagań, a także odstępstw od wymagań
- Pokrycie jest mierzone na podstawie przetestowanych elementów podstawy testów i techniki zastosowanej do podstawy testu

4. Techniki testowania

4.1.1 Kategorie technik testowania i ich cechy charakterystyczne

Cechy charakterystyczne

Białoskrzynkowe techniki testowania

- Warunki testowe, przypadki testowe i dane testowe pochodzą z podstawy testowania, która może obejmować kod, architekturę oprogramowania, szczegółowy projekt lub dowolne inne źródło informacji dotyczących struktury oprogramowania
- Pokrycie jest mierzone na podstawie elementów testowanych w ramach wybranej struktury (np. kodu lub interfejsów)
- Specyfikacje często są używane jako dodatkowe źródło informacji w celu określenia oczekiwanego wyniku przypadków testowych

4. Techniki testowania

4.1.1 Kategorie technik testowania i ich cechy charakterystyczne

Cechy charakterystyczne

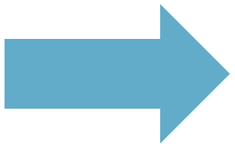
Techniki testowania oparte na doświadczeniu

- Warunki testowe, przypadki testowe i dane testowe pochodzą z podstawy testów, która może obejmować wiedzę i doświadczenie testerów, programistów, użytkowników i innych interesariuszy
- Taka wiedza i doświadczenie obejmuje oczekiwane użycie oprogramowania, jego środowisko, prawdopodobne defekty i ich rozkład

4. Techniki testowania

4.1.1 Kategorie technik testowania i ich cechy charakterystyczne

PRZYKŁAD



Kiedy powinniśmy używać danej techniki?

Jesteś odpowiedzialny za testowanie rozwiązania e-commerce. Pokrycie decyzji będzie najprawdopodobniej zastosowane na poziomie testów modułowych oraz być może na poziomie testów integracyjnych, gdy rozważa się proste komponenty internetowe, ale nie na wyższych poziomach. Techniki czarnoskrzynkowe, takie jak analiza wartości brzegowych lub podział na klasy równoważności czy tablice decyzyjne, będą najprawdopodobniej zastosowane na poziomach testów systemowych lub testach akceptacyjnych, gdy jesteśmy bardziej zainteresowani tym, jak produkt działa, a nie tym w jaki sposób przetwarza decyzje, i jak został do tego zaprojektowany.


4.2
Czarnoskrzynkowe
techniki testowania

4. Techniki testowania
4.2.1 Podział na klasy równoważności

Podział na klasy równoważności - charakterystyka

- Dostępne dane są podzielone na osobne klasy (partycje) w taki sam sposób, żeby każda z nich zawierała elementy przetwarzane w ten sam sposób
- Poprawne i niepoprawne klasy równoważności
- Klasa może być identyfikowana na podstawie wartości wejściowych i wyjściowych, wartości wewnętrznych, wartości zależnych od czasu
- Jedna wartość może należeć tylko do jednej klasy
- Nieprawidłowe klasy należy testować indywidualnie, aby pokryć wszystkie i uniknąć maskowania awarii

4.2.1 Podział na klasy równoważności

TERMINY

maskowanie awarii: sytuacja, w której występowanie jednego defektu uniemożliwia wykrycie innego.

4.2.1 Podział na klasy równoważności

Podział na klasy równoważności PRZYKŁAD

Pokrycie: liczba klas równoważności przetestowanych za pomocą co najmniej jednej wartości, podzielona przez całkowitą liczbę zidentyfikowanych klas równoważności.

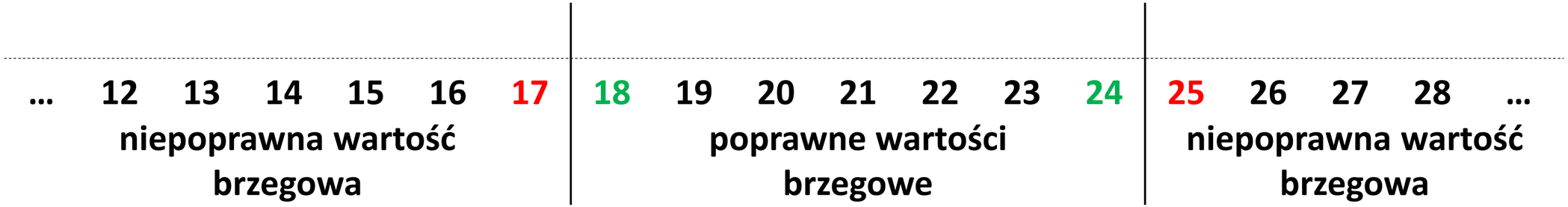
4. Techniki testowania
4.2.2 Analiza wartości brzegowych

Analiza wartości brzegowych - charakterystyka

- rozszerzenie techniki podziału na klasy równoważności (najczęściej stosowane razem z tą techniką)
- może być użyta, gdy klasa składa się z elementów, które są uporządkowane (np. dane numeryczne, porządek leksykograficzny)
- wartości brzegowe to wartości minimalne i maksymalne w danej klasie
- technika służy do testowania wymaganego zakresu liczb (danych lub czasu)

4. Techniki testowania
4.2.2 Analiza wartości brzegowych

Analiza wartości brzegowych PRZYKŁAD

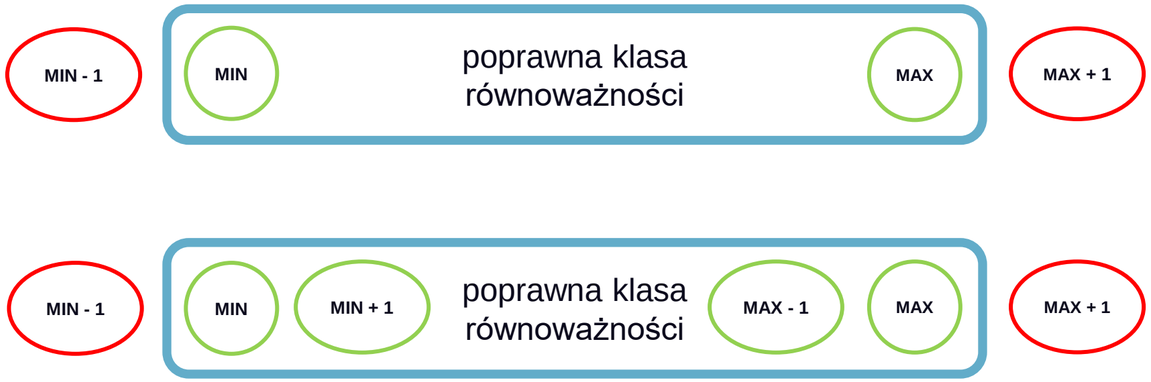


Pokrycie: liczba przetestowanych wartości brzegowych podzielona przez całkowitą liczbę zidentyfikowanych wartości brzegowych.

Często w specyfikacji stosuje się pojęcia "nie mniejszy niż" lub "większy niż”. Niestety podczas implementacji dochodzi do usterek, które wynikają z błędnego rozróżnienia „<" od ">=" pisanych również jako "(" lub ">"

4. Techniki testowania
4.2.2 Analiza wartości brzegowych

Analiza wartości brzegowych PRZYKŁAD



Dwupunktowa analiza wartości brzegowych:
- MIN oraz MAX to poprawne wartości brzegowe.
- MIN – 1 oraz MAX + 1 to niepoprawne wartości brzegowe

Trzypunktowa (pełna) analiza wartości brzegowych:
- MIN oraz MAX to poprawne wartości brzegowe
- MIN – 1 oraz MAX + 1 to niepoprawne wartości brzegowe
- MIN + 1 oraz MAX – 1 to poprawne wartości brzegowe

Klasy równoważności oraz analiza wartości brzegowych PRZYKŁAD

Rabat na zakupy w sklepie internetowym jest przyznawany

jeśli wartość koszyka wynosi od 1200 zł do 2500 zł.

Na podstawie tych informacji:

- ✓ Określ klasy równoważności oraz wartości dla przypadków testowych sprawdzających te klasy.
- ✓ Oblicz ile testów trzeba wykonać w celu uzyskania 100% pokrycia testami.
- ✓ Określ wartości brzegowe i przypadki testowe sprawdzające poprawne oraz niepoprawne wartości brzegowe.
- ✓ Oblicz pokrycie wymagane do pokrycia poprawnych, niepoprawnych wartości brzegowych i do pełnego pokrycia wartości brzegowych.

Klasy równoważności oraz analiza wartości brzegowych PRZYKŁAD

Rabat na zakupy w sklepie internetowym jest przyznawany

jeśli wartość koszyka wynosi od 1200 zł do 2500 zł.

- ✓ Klasy równoważności:
  - niepoprawne (0;1200) i (2500; +∞)
  - poprawne <1200;2500>
- ✓ Wartości dla przypadków testowych:
  - testy niepoprawnych klas równoważności 1000zł i 2700zł
  - test poprawnej klasy równoważności 1700zł

- ✓ Obliczenie pokrycia:
  - potrzeba 1 przypadek testowy dla pokrycia poprawnej klasy równoważności
  - potrzeba 2 przypadków testowych dla pokrycia niepoprawnych klas równoważności

4. Techniki testowania

Klasy równoważności oraz analiza wartości brzegowych PRZYKŁAD

Rabat na zakupy w sklepie internetowym jest przyznawany

jeśli wartość koszyka wynosi od 1200 zł do 2500 zł.

- ✓ Wartości brzegowe: 1200 i 2500
- ✓ Wartości dla przypadków testowych:
  - poprawne wartości brzegowe: 1200 i 2500
  - niepoprawne wartości brzegowe: 1199 i 2501
  - pełne pokrycie wartości brzegowych: 1199, 1200, 1201, 2499, 2500, 2501
- ✓ Obliczenie pokrycia:
  - potrzeba 2 przypadków testowych do pokrycia poprawnych wartości brzegowych
  - potrzeba 2 przypadków testowych do pokrycia niepoprawnych wartości brzegowych
  - potrzeba 6 przypadków testowych do pełnego pokrycia wartości brzegowych

4. Techniki testowania
4.2.3 Testowanie w oparciu o tablicę decyzyjną

Tablica decyzyjna

- kombinatoryczna technika testowa stosowana do sprawdzania, w jaki sposób różne kombinacje warunków powodują uzyskanie różnych wyników
- reprezentacja złożonych reguł biznesowych
- tabela zazwyczaj składa się z wartości logicznych (prawda/fałsz) lub dyskretnych (czerwony/zielony/niebieski)
- warunki są oznaczone jako Y jako prawdziwe (może być również pokazane jako T lub 1) lub N dla fałszu (może również być pokazane jako F lub 0)
- "-" oznacza, że wartość warunku nie ma znaczenia (może być również pokazywana jako N / A)
- oznaczenie X mówi o tym, że akcja zostanie wykonana (może być również pokazane jako Y lub T lub 1)
- puste pole oznacza, że akcja nie powinna zostać wykonana (może być również pokazana jako - lub N lub F lub 0)

4.2.3 Testowanie w oparciu o tablicę decyzyjną

Tablica decyzyjna PRZYKŁAD

	TC 1	TC 2	TC 3	TC 4	TC 5	TC 6	TC 7	TC 8
Warunek								
W1: Osoba pełnoletnia?	T	T	T	T	N	N	N	N
W2: Stały klient	T	T	N	N	T	T	N	N
W3: Masz dzisiaj urodziny?	T	N	T	N	T	N	T	N
Akcja								
A1: Bilet normalny			T	T				
A2: Bilet ulgowy	T	T			T	T	T	T
A3: Kolejne wejście gratis	T		T		T		T	

4.2.3 Testowanie w oparciu o tablicę decyzyjną

Zminimalizowana tablica decyzyjna

- ✓ Tablica decyzyjna może zostać zredukowana. W takiej sytuacji usuwa się kolumny, zawierające kombinacje warunków, które są niemożliwe do spełnienia lub kolumny zawierające możliwe ale niewykonalne kombinacje warunków, a także kolumny testujące kombinacje warunków, które nie mają wpływu na wynik.

4. Techniki testowania

4.2.3 Testowanie w oparciu o tablicę decyzyjną

Minimalizowanie tablicy decyzyjnej PRZYKŁAD 1

	1	2	3	4	5	6	7	8
Warunki:								
W1: Transakcja powyżej limitu	T	T	T	T	F	F	F	F
W2: Wiecej niż jedna karta do konta	T	F	T	F	T	T	F	F
W3: Kradzież karty płatniczej	T	T	F	F	T	F	T	F
Akcje:								
A1: Podwyższenie limitu			T	T				
A2: Prośba o kontakt z bankiem			T			T		
A3: Procedura w razie kradzieży karty	T	T			T		T	
A4: Umożliwienie transakcji			T	T		T		T

- ✓ Redukcję kolumn można zrobić wtedy, gdy w każdej z nich jest taki sam zestaw akcji, a w warunkach jeden lub więcej warunków nie mają wpływu na podjęcie tych akcji
- ✓ W tym przykładzie minimalizacji podlegają cztery kolumny: 1, 2, 5 i 7
- ✓ Czy możemy wyeliminować przypadek 1, 2, 5 lub 7?
  - Warunki 1 i 2 nie mają tutaj wpływu na wykonywaną akcję A3.

4. Techniki testowania

4.2.3 Testowanie w oparciu o tablicę decyzyjną

Minimalizowanie tablicy decyzyjnej PRZYKŁAD 2

	1	2	3	4	5	6	7	8
Warunki:								
W1: Pracownik etatowy ?	T	T	T	T	F	F	F	F
W2: Klient pełnoletni	T	F	T	F	T	T	F	F
W3: Klient poniżej 18 lat	T	T	F	F	T	F	T	F
Akcje:								
A1: Wypłata przelewem na konto	T	T	T	T				
A2: Wypłata wynagrodzenia w kasie					T	T	T	T
A3: Wypełnienie formularza J-48	T	T			T		T	

- ✓ Tablica może zawierać także warunki sprzeczne.
- ✓ Mówimy tu o przypadkach, które zawierają możliwe, ale niewykonalne kombinacje warunków np. 1, 4, 5 i 8. Warunek W2 i W3 wykluczają się nawzajem.

4.2.3 Testowanie w oparciu o tablicę decyzyjną

Zminimalizowana tablica decyzyjna

- ✓ Przy stosowaniu zminimalizowanych tablic decyzyjnych należy pamiętać, że jest to pewien kompromis (trade-off) pomiędzy mniejszą liczbą przypadków testowych, a ryzykiem pominięcia interesującego przypadku mogącego ujawnić awarię.
- ✓ Pokrycie tablicy decyzyjnej zależy od liczby kolumn, a nie liczby możliwych kombinacji warunków.

4. Techniki testowania

4.2.4 Testowanie przejść pomiędzy stanami

Przejścia między stanami

- systemy reagują inaczej na dane wydarzenie w zależności od aktualnych warunków lub warunków historycznych
- takie zdarzenia, które wystąpiły od momentu zainicjowania systemu, można podsumować za pomocą pojęcia stanów
- pojęcie stanu ma charakter abstrakcyjny i może odpowiadać kilku wierszom kodu lub całemu procesowi biznesowemu
- diagram przejścia stanu pokazuje wszystkie możliwe przejścia między stanami
- tabela przejść między stanami pokazuje również poprawne i nieprawidłowe przejścia

4. Techniki testowania

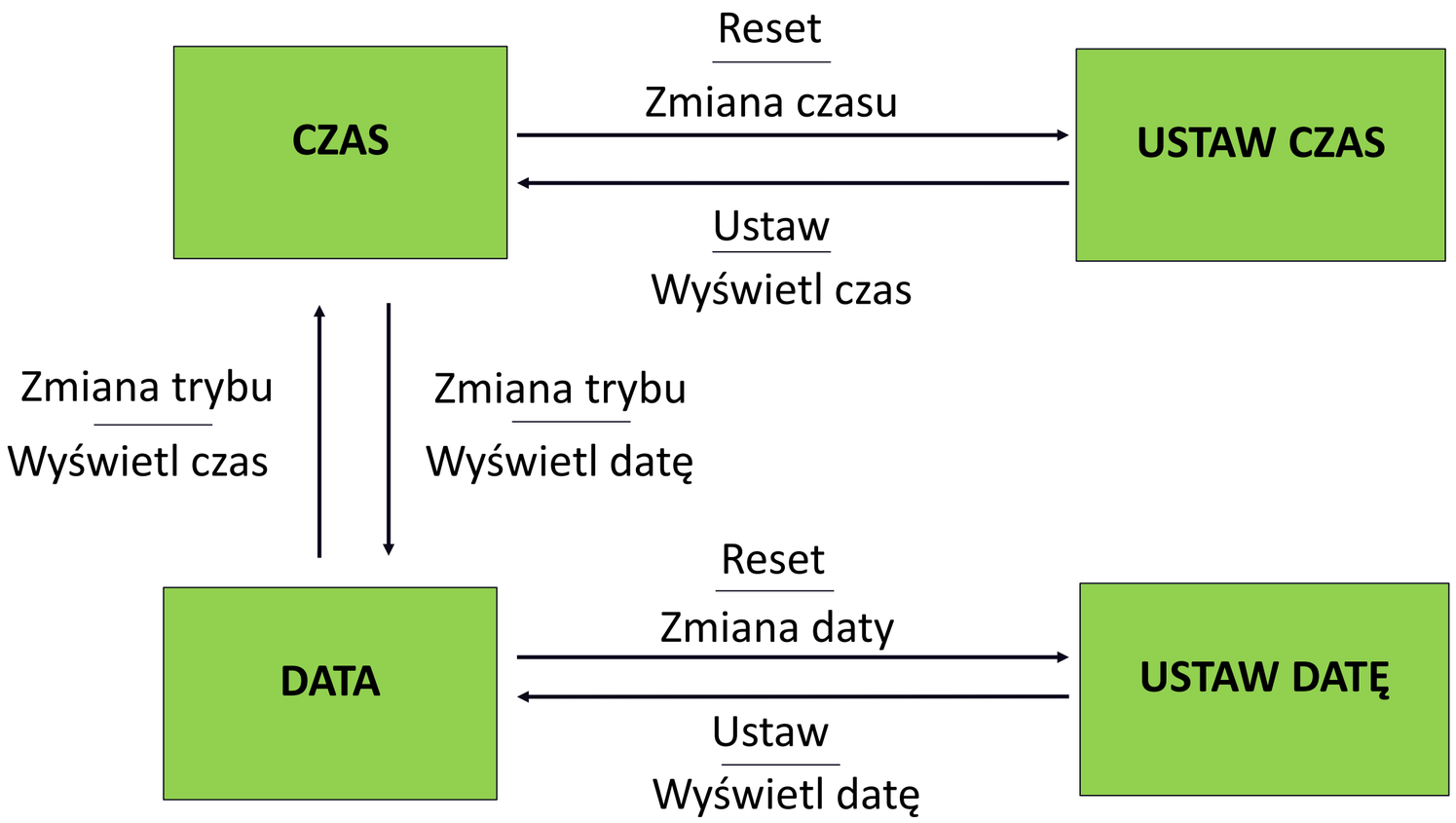
4.2.4 Testowanie przejść pomiędzy stanami

Przejścia między stanami

- technika używana w aplikacjach opartych na menu lub IVR (interaktywna odpowiedź głosowa)
- szeroko stosowana w branży oprogramowania wbudowanego
- technika stosowana także do testowania nawigacji ekranowej
- wykorzystywana do modelowania scenariuszy biznesowych

4.2.4 Testowanie przejść pomiędzy stanami

Przejścia między stanami PRZYKŁAD



Pokrycie: liczba przetestowanych stanów lub przejść między stanami, podzielona przez całkowitą liczbę zidentyfikowanych stanów lub przejść w obiekcie testowym.

4.2.4 Testowanie przejść pomiędzy stanami

Przejścia między stanami PRZYKŁAD

- ✓ Przejścia pomiędzy stanami mogą być przedstawione także w postaci tablicy.
- ✓ Warto zwrócić uwagę na to, że przejścia między stanami w postaci tablicy zawierają zarówno prawidłowe jak i nieprawidłowe przejścia. Natomiast diagram zawiera jedynie prawidłowe przejścia między stanami.

ZDARZENIE / STAN	CZAS	DATA	USTAW CZAS	USTAW DATĘ
ZMIANA TRYBU	DATA	CZAS	—	—
RESET	ZMIANA CZASU	ZMIANA DATY	—	—
USTAW	—	—	CZAS	DATA

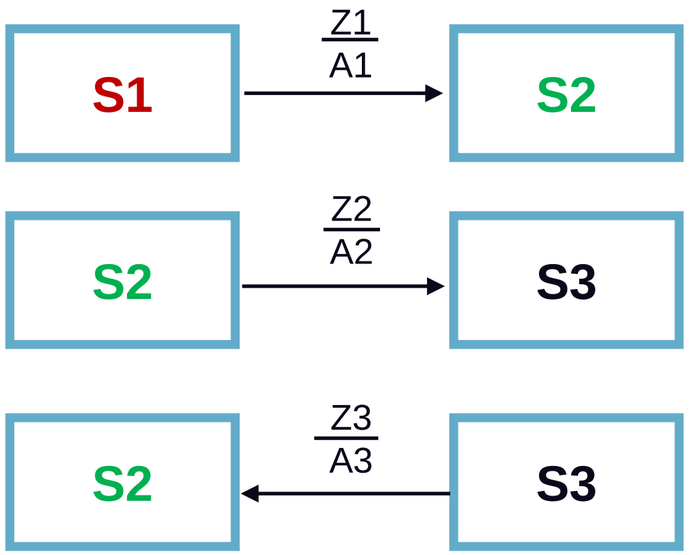
4.2.4 Testowanie przejść pomiędzy stanami

Maszyny stanowe [1/2]

pojęcie N-przełączeń (N-switch): forma testowania przejść pomiędzy stanami, gdzie przypadki testowe są zaprojektowane tak, aby wykonać wszystkie poprawne sekwencje N+1 przejść.

Przypadki testowe dla pojedynczych przejść:

S – stan
Z – zdarzenie
A – akcja



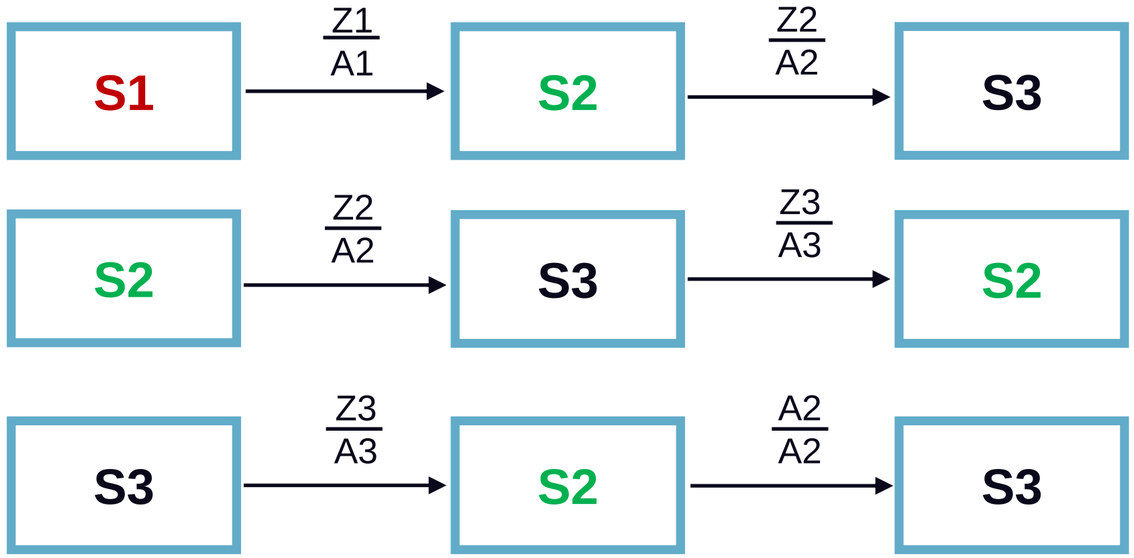
4. Techniki testowania

4.2.4 Testowanie przejść pomiędzy stanami

Maszyny stanowe [2/2]

Przypadki testowe dla przejść ze stanem pośrednim:

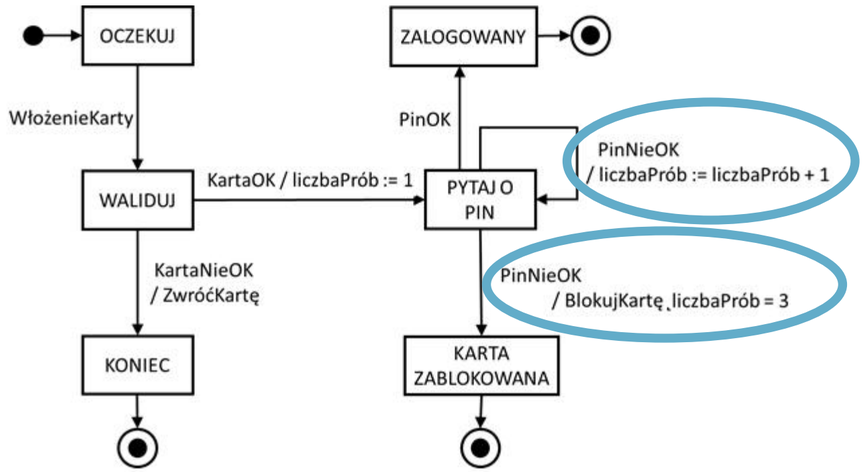
S – stan
Z – zdarzenie
A – akcja



4. Techniki testowania
4.2.4 Testowanie przejść pomiędzy stanami

Zdarzenie / Akcja

- ✓ To samo zdarzenie może skutkować dwoma lub więcej przejściami z tego samego stanu.



Pełny opis przejścia w maszynie stanów:
ZDARZENIE / AKCJA

4. Techniki testowania

4.2.5 Testowanie oparte na przypadkach użycia

Przypadek użycia

przypadek użycia to mały scenariusz biznesowy, który jest ściśle powiązany z aktorami, którymi mogą być użytkownicy końcowi, sprzęt zewnętrzny, administratorzy i przedmioty, takie jak testowane komponenty lub systemy

składa się z opisanych interakcji i działań, warunków wstępnych i wyjściowych

przypadek użycia służy do sprawdzania określonych zachowań systemu, w tym alternatywnych ścieżek czyli mechanizmów obsługi błędów

4.2.5 Testowanie oparte na przypadkach użycia

Przypadek użycia PRZYKŁAD

Nazwa	Kupowanie napoju z automatu	Nr 82
Cel	Klient kupuje napój z automatu za pomocą karty płatniczej	
Referencje do innych przypadków użycia	Procedura postępowania w przypadku niesprawnego urządzenia	
Wiadomości	Wiadomość nr 12 Wiadomość nr 7	
Warunki wstępne	Maszyna jest podłączona do Internetu i posiada napoje, które klient może kupić.	
Warunki wyjściowe	Klient kupuje napój. Jego konto zostaje obciążone	
Aktorzy	Klient	

4.2.5 Testowanie oparte na przypadkach użycia

Przypadek użycia PRZYKŁAD

Kroki	Akcje
1	Klient podchodzi do automatu i wybiera napój za pomocą klawiatury
2	Klient potwierdza swój wybór
3	System wyświetla okno wyboru typu płatności
4	Klient wybiera płatność za pomocą karty płatniczej
5	System prosi o zbliżenie karty płatniczej do czytnika
6	Klient zbliża kartę płatniczą do czytnika
7	System informuje o nawiązywaniu połączenia z bankiem
8	Klient jest poinformowany, że jego konto zostanie obciążone (Wiadomość nr 12)
9	Maszyna wydaje klientowi napój i drukuje pokwitowanie

4.2.5 Testowanie oparte na przypadkach użycia

Przypadek użycia PRZYKŁAD

Kroki	Ścieżki alternatywne
2a	Brak potwierdzenia wyboru klienta. Prośba o powtórny wybór napoju (Wiadomość nr 7)
8a	Klient nie posiada wystarczających środków na koncie
8b	Problemy z nawiązaniem połączenia z bankiem

4.2.5 Testowanie oparte na przypadkach użycia

Przypadek użycia - pokrycie

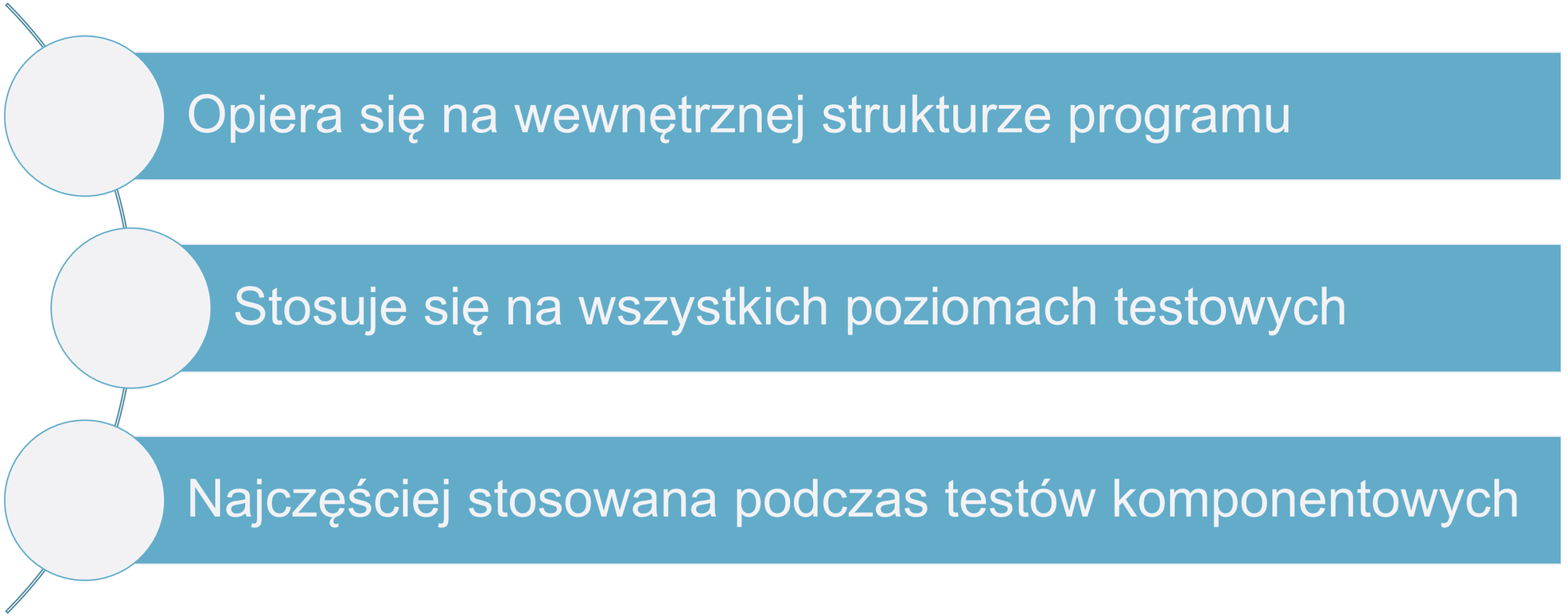
Pokrycie: procent testowanych zachowań przypadków użycia podzielony przez całkowitą liczbę zachowań przypadków użycia.

W omawianym przypadku użycia mamy jedną ścieżkę główną oraz 3 alternatywne czyli w celu osiągnięcia 100% pokrycia musimy wykonać 4 testy.


4.3
Białoskrzynkowe
techniki testowania

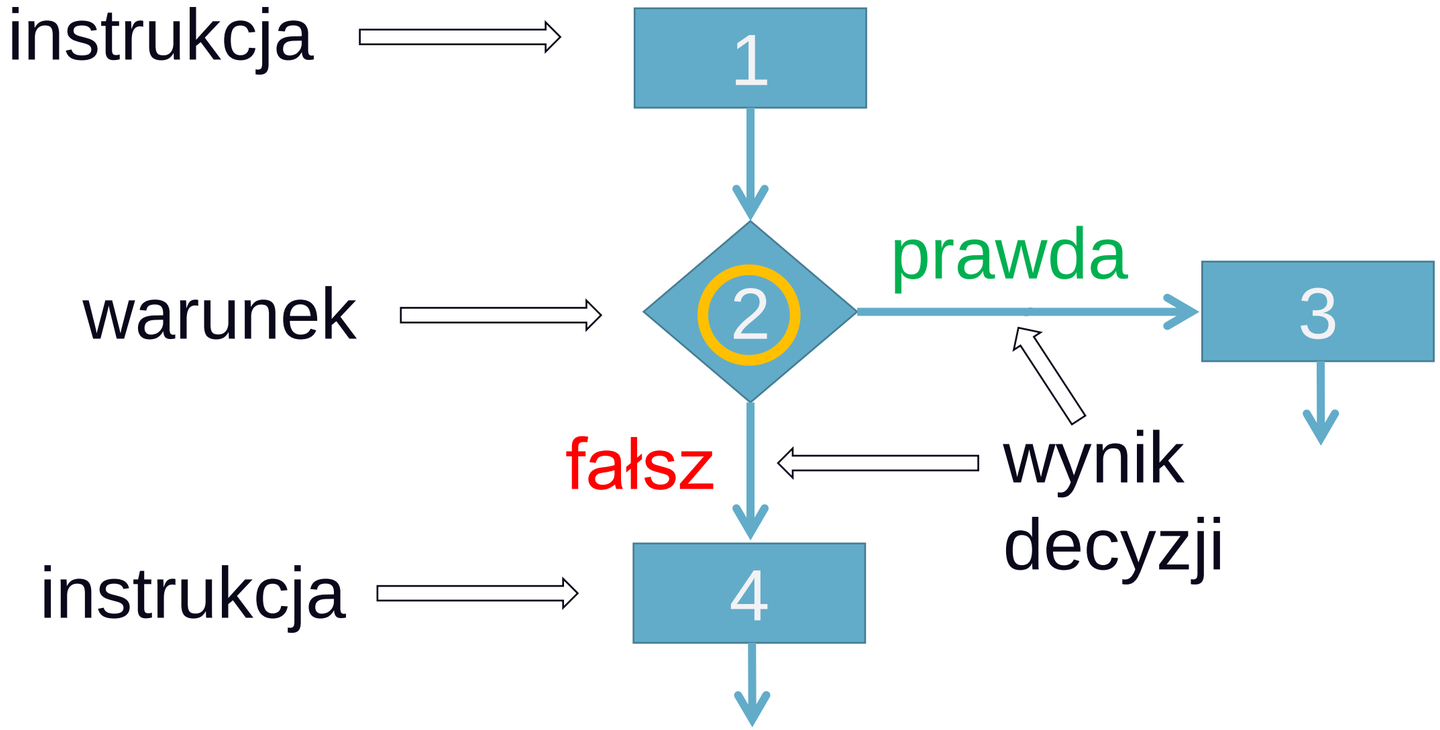
4. Techniki testowania
4.3.1 Testowanie i pokrycie instrukcji kodu

Testowanie i pokrycie instrukcji kodu



4.3.1 Testowanie i pokrycie instrukcji kodu

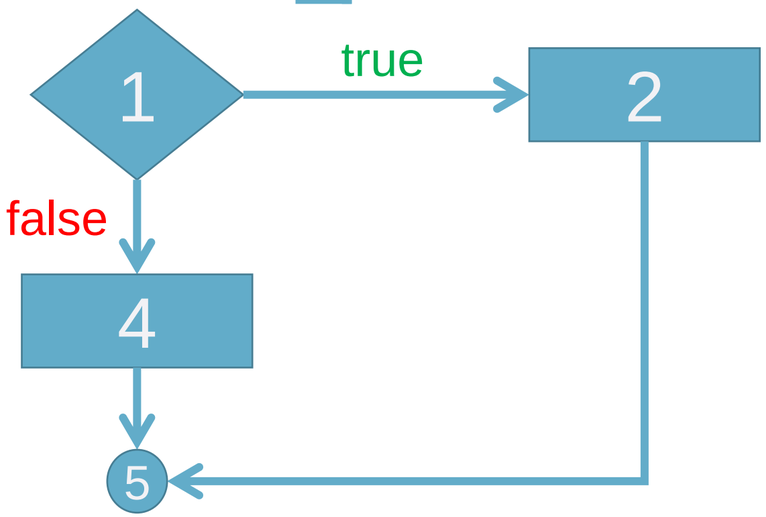
Schemat blokowy



4.3.1 Testowanie i pokrycie instrukcji kodu

Schemat blokowy

```
1 If (hungry)
2         Eat sandwich
3 Else
4         Sleep
5 End If
```



- END IF oznacza, że instrukcja IF została zakończona -> Prawdziwa i Fałszywa gałąź są ponownie połączone
- ELSE wskazuje, że następująca linia (linie) występuje, jeśli decyzja jest fałszywa

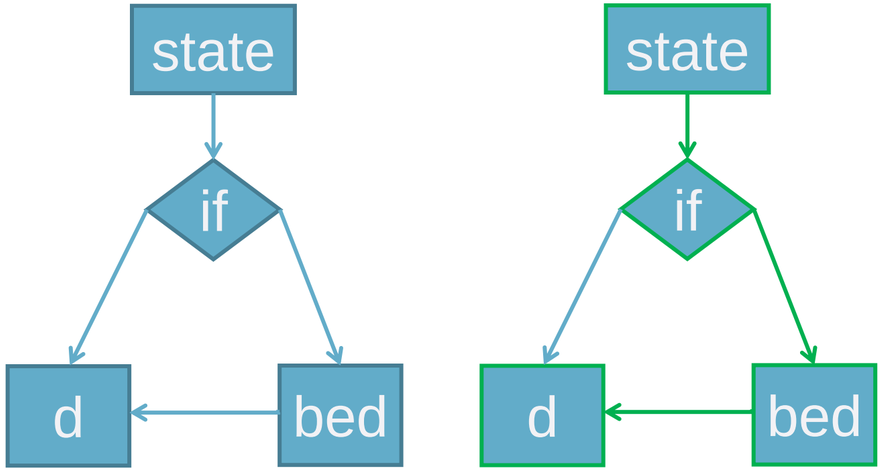
4.3.1 Testowanie i pokrycie instrukcji kodu

Pokrycie instrukcji PRZYKŁAD

Kod

```
Read state;
        if (Sleepy);
                go to bed;
        end if;
d;
```

Przepływ sterowania



Sleepy = TRUE

1 przypadek testowy dla 100% pokrycia instrukcji

4. Techniki testowania

4.3.1 Testowanie i pokrycie instrukcji kodu

Pokrycie instrukcji

Pokrycie instrukcji mierzy się procentowo jako iloraz liczby wykonywalnych instrukcji wykonanych przez testy i łącznej liczby instrukcji wykonywalnych w przedmiocie testów, zazwyczaj wyrażany w procentach.

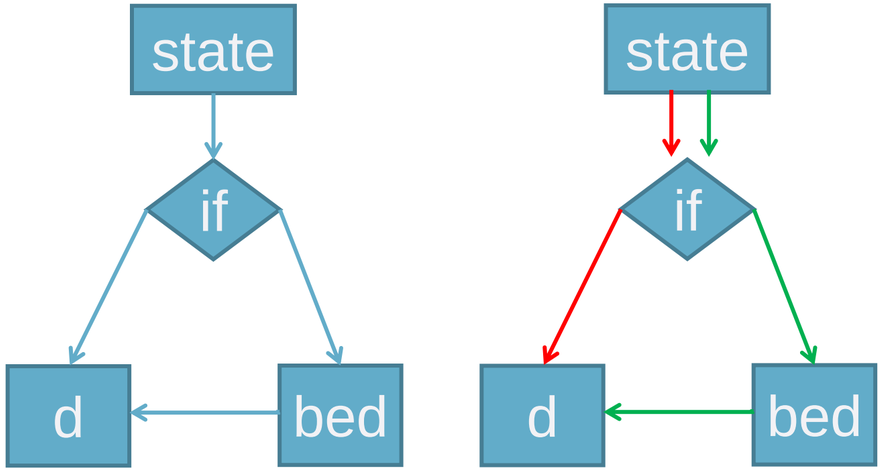
4.3.2 Testowanie i pokrycie decyzji

Pokrycie decyzji PRZYKŁAD

Kod

```
Read state;
        if (Sleepy);
                go to bed;
        end if;
d;
```

Przepływ sterowania



Sleepy = TRUE and Sleepy = FALSE
2 przypadki testowe dla 100% pokrycia decyzji

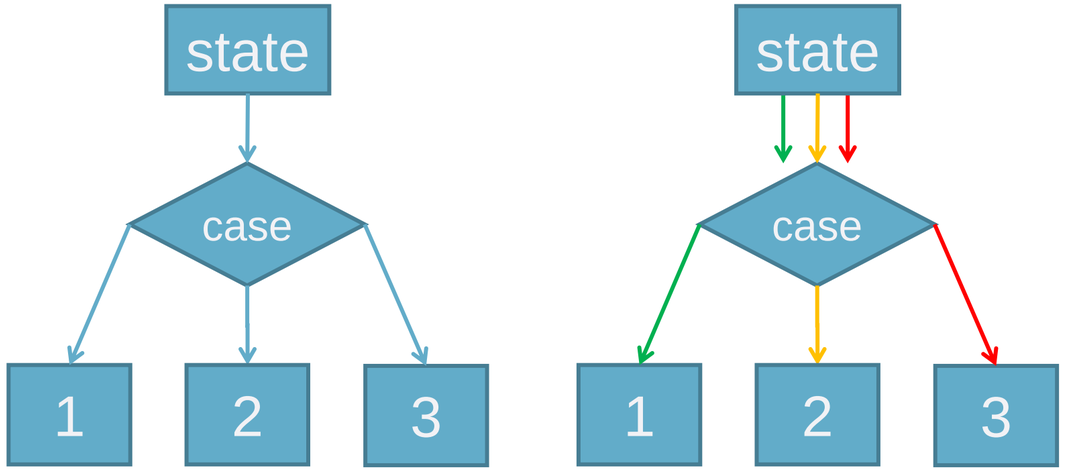
4.3.2 Testowanie i pokrycie decyzji

Pokrycie decyzji PRZYKŁAD

Kod

```
switch (State);
    case Sleepy;
        Instruction 1;
        break;
    case Tired;
        Instruction 2;
        break;
    default;
        Instruction 3;
```

Przepływ sterowania



State = Sleepy, State = Tired, State = default

3 przypadki testowe dla 100% pokrycia decyzji

4. Techniki testowania
4.3.2 Testowanie i pokrycie decyzji

Pokrycie decyzji

Pokrycie decyzji mierzy się procentowo jako iloraz liczby wyników decyzji wykonanych przez testy i łącznej liczby możliwych wyników decyzji w przedmiocie testów, zazwyczaj wyrażany w procentach.

4. Techniki testowania

4.3.3 Korzyści wynikające z testowania instrukcji i testowania decyzji

Wartość testowania instrukcji i decyzji

100% pokrycia instrukcji

- ✓ oznacza, że wszystkie wykonywalne instrukcje zostały wykonane co najmniej raz, ale nie oznacza to, że logika decyzji została przetestowana

100% pokrycia decyzji

- ✓ oznacza, że wszystkie wyniki decyzji (fałszywe i prawdziwe) zostały wykonane
- ✓ pomaga znaleźć błędy w kodzie, tam gdzie inne testy nie uwzględniają zarówno prawdziwych, jak i fałszywych wyników

100% pokrycia decyzji gwarantuje 100% pokrycia instrukcji

100% pokrycia instrukcji NIE gwarantuje 100% pokrycia decyzji

Warunkami testowymi są wyniki decyzji, a nie „pełne przetestowanie decyzji”

Techniki testowania **oparte na doświadczeniu**

4. Techniki testowania

4.4 Techniki testowania oparte na doświadczeniu

Przypadki testowe bazują na umiejętnościach testera, jego doświadczeniu, intuicji i znajomości podobnych produktów oraz technologii

Może odkryć defekty, które nie są łatwo identyfikowane za pomocą bardziej systematycznych technik

Trudno jest zdefiniować pokrycie

4. Techniki testowania
4.4.1 Zgadywanie błędów

Zgadnij co jest źle

- Najpierw wymień możliwe błędy, usterki lub awarie i zaprojektuj odpowiednie testy
- Następnie sprawdź, czy taki błąd występuje w oprogramowaniu
- Lista możliwych defektów może być oparta na doświadczeniu, danych dotyczących usterek i awarii lub z ogólnodostępnej wiedzy o systemie

Należy wziąć pod uwagę przy stosowaniu tej techniki

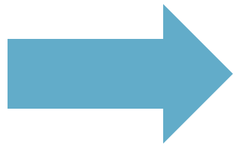
- Jak aplikacja działała w przeszłości?
- Jakie typy błędów popełniają deweloperzy?
- Awarie, które wystąpiły w innych aplikacjach

4. Techniki testowania
4.4.1 Zgadywanie błędów

PRZYKŁAD

Testowanie pola daty daty urodzenia

Aby użyć techniki zgadywania błędów podczas testowania danego pola tekstowego możesz spróbować wpisać zestaw wartości liczb niecałkowitych lub datę urodzenia osoby, która jeszcze się nie urodziła, lub bardzo długi zestaw znaków przez Ctrl + C, Ctrl + V.



Kiedy lista inspiracji jest gotowa próbujesz ją przetestować na działającym oprogramowaniu i zarejestrować defekty, jeśli są obecne.

4. Techniki testowania
4.4.2 Testowanie eksploracyjne

może wykorzystywać techniki czarno- i białoskrzynkowe oraz te oparte na doświadczeniu

Eksploracja

- nieformalna technika, w której testy są projektowane, wdrażane i rejestrowane podczas wykonywania testu
- testy oparte na sesjach
- karta testu służy do rejestrowania wyników sesji
- wynik może obejmować obszary, które mogą wymagać więcej testów
- najbardziej przydatne w przypadku słabej jakości specyfikacji (jej braku) lub presji czasu
- technika uzupełniająca bardziej formalne testowanie
- mocno związana ze strategią testów reaktywnych (rozdział 5.2.2)

4.4.2 Testowanie eksploracyjne

PRZYKŁAD

Eksploracyjne sesje testowe

Projekt, w którym obecnie pracujesz, niedługo dobiegnie końca. Jest jeszcze 4 innych testerów znających funkcjonalność produktu, nad którym pracowaliście. Spełnione zostało kryterium wyjścia, nie znaleziono żadnych poważniejszych problemów i mamy jeszcze kilka dni do zakończenia projektu. W oparciu o waszą wiedzę i doświadczenie decydujecie się skorzystać z sesji testów eksploracyjnych (nie więcej niż 1,5 - 2h każda), które odbywać się będą równolegle.

Pomoże Wam to wykonać bardziej złożone scenariusze, niż te już wykonane wcześniej podczas osobnego testowania każdej funkcjonalności.

4. Techniki testowania

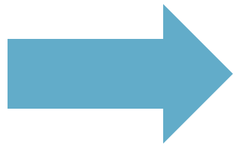
4.4.3 Testowanie w oparciu o listę kontrolną

Lista kontrolna

- testy przeprowadzane w celu uwzględnienia warunków testowych znajdujących się na liście kontrolnej
- analiza i projektowanie jest uważane za tworzenie lub aktualizowanie istniejącej listy kontrolnej
- listy kontrolne najczęściej bazują na doświadczeniu, znajomości oczekiwań użytkowników i zrozumieniu, dlaczego oprogramowanie nie działa
- wsparcie dla testów funkcjonalnych i niefunkcjonalnych o charakterze wysokopoziomowym
- zwykle zapewniają lepsze pokrycie, ale potencjalnie mniejszą powtarzalność

4. Techniki testowania
4.4.3 Testowanie w oparciu o listę kontrolną

PRZYKŁAD



Czy możemy użyć listy kontrolnej do przeglądu przypadków testowych?

Tak, na przykład ten zestaw warunków może być użyty:

- ✓ Sprawdź, czy wszystkie potrzebne informacje są zawarte
- ✓ Sprawdź, czy docelowe wymagania są objęte przypadkami testowymi
- ✓ Sprawdź, czy przypadek testowy jest logiczny, jeśli chodzi o kolejność kroków, warunki wstępne, informacje w oczekiwanych wynikach
- ✓ Sprawdź, czy przypadek testowy jest jasny i zrozumiały

4. Techniki testowania

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

05

Zarządzanie testami.

5. Zarządzanie testami
Cele Nauczania

5.1 Organizacja testów
- FL-5.1.1 (K2) Kandydat potrafi omówić zalety i wady niezależnego testowania
- FL-5.1.2 (K1) Kandydat potrafi wskazać zadania kierownika testów i testera

5.2 Planowanie i szacowanie testów
- FL-5.2.1 (K2) Kandydat potrafi podsumować cel i treść planu testów
- FL-5.2.2 (K2) Kandydat potrafi rozróżnić poszczególne strategie testowania
- FL-5.2.3 (K2) Kandydat potrafi podać przykłady potencjalnych kryteriów wejścia i wyjścia
- FL-5.2.4 (K3) Kandydat potrafi wykorzystać wiedzę na temat ustalania priorytetów oraz zależności technicznych i logicznych do zaplanowania wykonania określonego zbioru przypadków testowych

5. Zarządzanie testami

Cele Nauczania

FL-5.2.5 (K1) Kandydat potrafi wskazać czynniki wpływające na pracochłonność testowania

FL-5.2.6 (K2) Kandydat potrafi wyjaśnić różnicę między dwiema technikami szacowania: techniką opartą na miarach i techniką ekspercką

5.3 Monitorowanie testów i nadzór nad testami

FL-5.3.1 (K1) Kandydat pamięta miary stosowane w odniesieniu do testowania

FL-5.3.2 (K2) Kandydat potrafi podsumować cele i treść raportów z testów oraz wskazać ich odbiorców

5.4 Zarządzanie konfiguracją

FL-5.4.1 (K2) Kandydat potrafi podsumować, w jaki sposób zarządzanie konfiguracją wspomaga testowanie

5. Zarządzanie testami
Cele Nauczania

5.5 Czynniki ryzyka a testowanie

FL-5.5.1 (K1) Kandydat potrafi określić poziom ryzyka na podstawie prawdopodobieństwa i wpływu

FL-5.5.2 (K2) Kandydat potrafi rozróżnić czynniki ryzyka projektowego i produktowego

FL-5.5.3 (K2) Kandydat potrafi opisać na przykładach potencjalny wpływ analizy ryzyka produktowego na staranność i zakres testowania

5.6 Zarządzanie defektami

FL-5.6.1 (K3) Kandydat potrafi sporządzać raporty o defektach zawierające informacje na temat defektów wykrytych podczas testowania


5.1
Organizacja testów

5.1.1 Niezależne testowanie

Co to jest niezależne testowanie?

Testowanie wcale nie musi być wykonywane przez testerów. Może to być dowolna osoba np. klient lub analityk biznesowy. Zmiana ról sprawia, że testowanie może być jeszcze skuteczniejsze w wykrywaniu defektów ze względu na inne spojrzenie drugiej osoby. Niezależne testowanie wcale jednak nie oznacza, że osoba, która testuje produkt nie może go dobrze znać. Często to programiści, którzy bardzo dobrze znają produkt mogą łatwo znaleźć wiele błędów w napisanym przez siebie kodzie.

5. Zarządzanie testami
5.1.1 Niezależne testowanie

Poziomy niezależności testowania

Niezależni testerzy spoza organizacji

Praca u klienta (on-site) lub poza siedzibą firmy (outsourcing)

Niezależny zespół testowy

Grupa w organizacji, raportująca do kierownictwa projektu

Brak niezależnych testerów

Programiści testują swój własny kod

Społeczność użytkowników / testerzy z działów biznesowych

Testerzy specjalizujący się w określonych typach testów, takich jak użyteczność, bezpieczeństwo, wydajność, zgodność z przepisami lub przenaszalność

Niezależni programiści lub testerzy w zespole programistów

Programiści testujący produkty swoich kolegów

5. Zarządzanie testami
5.1.1 Niezależne testowanie

Poziomy niezależności testowania

Na każdym poziomie testujemy inaczej

Programiści mogą łatwo testować na niższych poziomach, podczas gdy więcej niezależnych testerów powinno brać udział w testach na wyższych poziomach

Cykl rozwoju oprogramowania

Niezależność testowania zależy od cyklu rozwoju oprogramowania, np. testerzy mogą być częścią osobnego zespołu testowego lub w Agile mogą być częścią zespołu programistycznego

Agile – historyjki użytkowników

W projektach Agile każda z historyjek użytkownika może zostać zweryfikowana na końcu danej iteracji

5. Zarządzanie testami
5.1.1 Niezależne testowanie

Potencjalne korzyścią i wady niezależnego testowania

Niezależni testerzy mają różne doświadczenie, techniczny punkt widzenia i są bezstronni. Prawdopodobnie rozpoznają innego rodzaju awarie w porównaniu do deweloperów.

Założenia przyjęte przez interesariuszy podczas specyfikowania i implementacji systemu można zweryfikować, zakwestionować lub obalić.

- Izolacja od zespołu programistów (brak współpracy, opóźnienia w przekazywaniu informacji zwrotnych do zespołu programistów, złe relacje z zespołem programistów).
- Niezależni testerzy mogą być uznani za wąskie gardło lub obwiniani za opóźnienia we wdrożeniu.
- Deweloperzy mogą utracić poczucie odpowiedzialności za jakość.
- Brak dostępu do istotnych informacji (np. o testowanym produkcie).

5.1.1 Niezależne testowanie

PRZYKŁAD

Jak działa niezależne testowanie ?

Jesteś częścią międzynarodowego software house`u, który testuje różne produkty. Zostałeś poproszony o przetestowanie produktu dla amerykańskiego klienta. Nie masz doświadczenia z nowym produktem, ale wcześniej pracowałeś przy podobnych projektach. Twój zespół jest odpowiedzialny za testy alfa. Podczas testowania zdajesz sobie sprawę, że początkowe założenie interesariuszy nie jest realistycznym scenariuszem i nie spełni oczekiwań użytkownika końcowego.

Gdybyś był częścią zespołu opracowującego ten produkt od samego początku, najprawdopodobniej nie znalazłbyś tego defektu, ponieważ od samego początku byłby to ’dobry pomysł’. W takim przypadku jako niezależny tester możesz zweryfikować wstępne założenia poczynione przez interesariuszy.

5. Zarządzanie testami

5.1.2 Zadania kierownika testów i testera

Kierownik testów i tester

Zupełnie inne role, które zależą od:

- ✓ kontekstu projektu i produktu
- ✓ umiejętności zaangażowanych osób
- ✓ organizacji

Kierownik testów

Kierownik projektu, profesjonalny kierownik testów, kierownik rozwoju lub kierownik ds. zapewnienia jakości.

Odpowiedzialność

Ogólna odpowiedzialność za proces testowania i skuteczne kierowanie działaniami testowymi.

Duże projekty

Rozważając większe projekty, kilka zespołów może raportować do kierownika testowego / lidera testów / koordynatora testów / testera wiodącego.

5.1.2 Zadania kierownika testów i testera

Zadania kierownika testów [1/4]

- ✓ **opracowywanie** lub dokonywanie przeglądu **strategii testów i polityki testów** danej organizacji
- ✓ **planowanie** projektu **testowania**, w tym uwzględnienie kontekstu oraz zapoznanie się z celami testów i czynnikami ryzyka w zakresie testowania
- ✓ sporządzanie i **aktualizowanie planu testów**
- ✓ **koordynowanie strategii testów i planu testów** z kierownikami projektu i innymi osobami
- ✓ **prezentowanie punktu widzenia** testerów w ramach innych czynności projektowych

5.1.2 Zadania kierownika testów i testera

Zadania kierownika testów [2/4]

- ✓ **inicjowanie procesów** analizy, projektowania, implementacji i wykonywania testów, monitorowanie rezultatów testów oraz sprawdzanie statusu kryteriów wyjścia (definicji ukończenia)
- ✓ **przygotowanie i dostarczenie raportu z postępu testów** i raportu sumarycznego z testów na podstawie zgromadzonych informacji
- ✓ **dostosowywanie planów do rezultatów i postępu testów** (dokumentowanych niekiedy w raportach o statusie testów i raportach z ukończenia testów) oraz podejmowanie niezbędnych działań w zakresie nadzoru nad testami

5.1.2 Zadania kierownika testów i testera

Zadania kierownika testów [3/4]

- ✓ **udzielanie pomocy w tworzeniu** należytego mechanizmu **zarządzania konfiguracją** testaliów i systemu zarządzania defektami
- ✓ **wprowadzanie odpowiednich miar** służących do mierzenia postępu testów oraz oceniania jakości testowania i produktu
- ✓ **udzielanie pomocy przy wyborze i implementacji narzędzi** służących do realizacji procesu testowego, w tym przy określaniu budżetu na wybór narzędzi oraz wyznaczaniu czasu na realizację projektów pilotażowych, a także udzielanie dalszej pomocy w zakresie korzystania z narzędzi

5.1.2 Zadania kierownika testów i testera

Zadania kierownika testów [4/4]

- ✓ **podejmowanie decyzji** o implementacji środowisk testowych
- ✓ **monitorowanie procesu testowania** oraz sporządzanie i przedstawianie raportów z testów na podstawie zgromadzonych informacji
- ✓ **promowanie testerów i zespołu testowego** oraz reprezentowanie ich punktu widzenia w organizacji
- ✓ **rozwijanie umiejętności** i kariery testerów (np. poprzez plan szkoleń, ewaluacje osiągnięć, coaching)

5. Zarządzanie testami
5.1.2 Zadania kierownika testów i testera

Rola kierownika testów zależy od używanego przez nas modelu rozwoju oprogramowania, np. w modelach zwinnych pewne obowiązki kierownika są przejmowane przez zespół, ale w przypadku dużych projektów, takie działania kierownika mogą być wyjęte poza zespół do dedykowanych kierowników testów.

5. Zarządzanie testami
5.1.2 Zadania kierownika testów i testera

Zadania testera [1/3]

- ✓ dokonywanie przeglądu planów testów i uczestniczenie w ich opracowywaniu
- ✓ analizowanie, dokonywanie przeglądu i ocenianie podstawy testów (tj. historyjek użytkownika, specyfikacji) pod kątem testowalności
- ✓ identyfikowanie i dokumentowanie warunków testowych oraz rejestrowanie powiązań między przypadkami, warunkami testowymi i podstawą testów
- ✓ projektowanie, konfigurowanie i weryfikowanie środowisk testowych (często w porozumieniu z administratorami systemu i sieci)

5.1.2 Zadania kierownika testów i testera

Zadania testera [2/3]

- ✓ projektowanie i implementowanie przypadków testowych i skryptów testowych
- ✓ przygotowywanie i pozyskiwanie danych testowych
- ✓ tworzenie harmonogramu wykonywania testów
- ✓ wykonywanie testów, ocenianie rezultatów i dokumentowanie odchyleń od oczekiwanych rezultatów
- ✓ korzystanie z odpowiednich narzędzi usprawniających proces testowy

5. Zarządzanie testami
5.1.2 Zadania kierownika testów i testera

Zadania testera [3/3]

- ✓ automatyzowanie testowania w zależności od potrzeb (zadanie to może być wykonywane z pomocą programisty lub eksperta ds. automatyzacji testowania)
- ✓ ewaluacja charakterystyk niefunkcjonalnych, takich jak: wydajność, niezawodność, użyteczność, zabezpieczenia, zgodność, przenaszalność
- ✓ dokonywanie przeglądu testów opracowanych przez inne osoby

Rola testera zależy od kontekstu

Specjalista w danej domenie, np. analityk testów, projektant specyficznych typów testów jak np. wydajność, użyteczność lub automatyzacja testów

5. Zarządzanie testami
5.1.2 Zadania kierownika testów i testera

Testuje nie tylko tester. Inne osoby mogą przejmować jego odpowiedzialność na różnych poziomach

Programista
Testowanie komponentów
i integracji komponentów

Niezależny zespól testowy
Testowanie systemowe
i integracji systemów

Analityk biznesowy, ekspert
Testowanie akceptacyjne

Administratorzy systemu
Operacyjne testy akceptacyjne


5.2
Planowanie i
szacowanie testów

5. Zarządzanie testami
5.2.1 Wprowadzenie

Cel i treść planu testów

plan testów: dokumentacja opisująca cele testowe do osiągnięcia oraz środki i harmonogram ich realizacji, zorganizowane tak, by koordynować czynności testowe.

główny plan testów: plan testów, który odnosi się do wielu poziomów testów.

plan testów jednego poziomu: plan testu, który odnosi się do jednego poziomu testowania.

- ma wpływ na politykę testów i strategię testowania oraz nakreśla działania testowe
- określa zakres testowania, cele, ryzyko, ograniczenia, krytyczność, testowalność, dostępność zasobów

5. Zarządzanie testami
5.2.1 Cel i treść planu testów

Charakterystyka planu testów

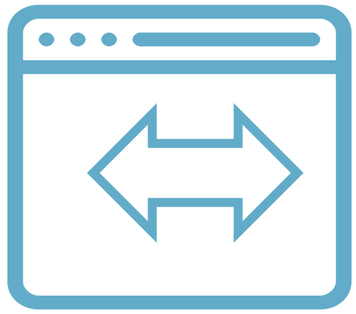
- Plan testów nie jest ’zamrożonym’ dokumentem.
- Jest aktualizowany razem z postępem projektu.
- Jest to ciągła aktywność polegająca na aktualizacji planu testów.
- Faza pielęgnacyjna może również zostać włączona do planu testów, ponieważ plan obejmuje czynności testowe w całym cyklu życia produktu.
- Należy zdefiniować główny plan testów. Każdy poziom testowy może mieć oddzielny plan testów.

5. Zarządzanie testami
5.2.1 Cel i treść planu testów

Zawartość planu testów

CO ?	KTO ?	KIEDY ?	JAK ?
• Określa zakres, cele i ryzyko testowania.	• Pomaga w podejmowaniu decyzji o tym, co należy przetestować, ludziach i innych zasobach wymaganych do wykonywania różnych czynności testowych oraz w jaki sposób będą przeprowadzane czynności testowe.	• Planuje analizę, projektowanie, implementację, wykonanie i ocenę testów w określonych datach (np. w modelu sekwencyjnym) lub w kontekście każdej iteracji (np. w modelu iteracyjnym).	• Określa ogólne podejście do testowania. • Integracja i koordynacja działań testowych w ramach działań związanych z cyklem życia oprogramowania. • Wybór danych do monitorowania testów i nadzoru nad testami. • Budżetowanie czynności testowych. • Określanie poziomu szczegółowości i struktury dokumentacji testowej (np. poprzez dostarczanie szablonów lub przykładowych dokumentów).

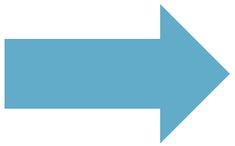
5. Zarządzanie testami
5.2.1 Cel i treść planu testów



PRZYKŁAD

Trzymamy się planu bez względu na wszystko?

Często zakłada się, że to, co jest planowane na samym początku projektu, najprawdopodobniej nie zostanie zaktualizowane, ponieważ przecież musimy trzymać się tego planu. Ale…



Jest to tylko omówienie np. zakresu, zidentyfikowanych zagrożeń i dostępności zasobów, które mogą w międzyczasie ulec zmianie. Zawsze możemy mieć do wykonania więcej testów, więcej zidentyfikowanych zagrożeń i więcej zasobów w miarę postępu projektu i jest to absolutnie normalne. Co więcej, konieczne jest uaktualnienie planu testów, jeśli zajdzie taka potrzeba.

5. Zarządzanie testami
5.2.2 Strategie testów i podejście do testowania

TERMINY

strategia testów: dokumentacja, która wyraża ogólne wymagania dotyczące testowania jednego lub większej liczby projektów prowadzonych w organizacji, dostarczająca szczegółowych informacji na temat sposobu przeprowadzania testów i zgodna z polityką testów.

Wysokopoziomowy opis procesu testowego (poziom produktu / organizacji).

5. Zarządzanie testami
5.2.2 Strategie testów i podejście do testowania

Strategie testów [1/2]

Analityczna	strategia oparta na analizie wybranego czynnika, np. ryzyko, w którym testy są projektowane i ustalane według priorytetów na podstawie poziomu ryzyka
Oparta na modelu	strategia, w której testy są projektowane w oparciu o model danego aspektu produktu, np. niezawodność, gdzie testy opierają się na modelu wzrostu niezawodności
Metodyczna	systematyczne korzystanie z niektórych wstępnie zdefiniowanych zestawów testów lub warunków testowych (taksonomia defektów, cechy jakościowe, standardy firmy)

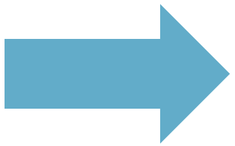
5.2.2 Strategie testów i podejście do testowania

Strategie testów [2/2]

Zgodna z procesem	wdrażanie testów w oparciu o zewnętrzne zasady i standardy (standardy branżowe, dokumentacja procesowa)
Kierowana (konsultatywna)	w oparciu o porady lub wskazówki od ekspertów z dziedziny biznesu lub ekspertów ds. technologii
Minimalizująca regresję	aby uniknąć regresji istniejących funkcjonalności; ponowne wykorzystanie istniejącego oprogramowania testowego z rozbudowaną automatyzacją testów regresji
Reaktywna	testowanie jest ukierunkowane na reagowanie niż na realizację z góry ustalonego planu; testy są projektowane i mogą być wdrożone natychmiast wykonane w odpowiedzi na wiedzę uzyskaną z wcześniejszych wyników testów (np. testy eksploracyjne)

5.2.2 Strategie testów i podejście do testowania

PRZYKŁAD



Która strategia jest najlepsza?

Produkt jest opracowywany w ramach długoterminowego projektu. Możesz ponownie wykorzystać istniejący zestaw testów, ale wymagane jest również napisanie nowych dla danej wersji. Priorytetyzujesz swoje testy w oparciu o ryzyko. Nie znasz obszaru, którego dotyczy produkt, więc masz spotkania koordynacyjne z zewnętrznymi konsultantami. Oprócz zbierania wyników testów, kierownik testu musi zająć się dokumentacją prawną do celów FDA. Ponieważ jest to projekt długoterminowy, posiadasz gotową do użycia taksonomię defektów.

Jakie strategie rozpoznajesz w tym przykładzie?

5. Zarządzanie testami
5.2.2 Strategie testów i podejście do testowania

Nie ma jednego podejścia do strategii

Strategia testowa jest zazwyczaj połączeniem
kilka typów razem, np. testy oparte na ryzyku
jako strategia analityczna i eksploracja
jako strategia reaktywna.

Wymienione strategie uzupełniają się nawzajem
i bardzo skuteczne jest użycie kilku strategii naraz.

5. Zarządzanie testami
5.2.2 Strategie testów i podejście do testowania

TERMINY

podejście do testowania: implementacja strategii testowej w określonym projekcie.

Pomaga w doborze techniki testowej, poziomów i typów testów oraz zdefiniowania kryteriów wejścia i wyjścia. Sposób wdrożenia podejścia testowego opiera się na decyzjach projektowych (rodzaj produktu, analiza ryzyka itp.). Zależy to od poziomu ryzyka, bezpieczeństwa, dostępnych zasobów i umiejętności, technologii, charakteru systemu (np. system na zamówienie lub do powszechnej sprzedaży), celów testów i przepisów.

5. Zarządzanie testami
5.2.3 Kryteria wejścia i wyjścia

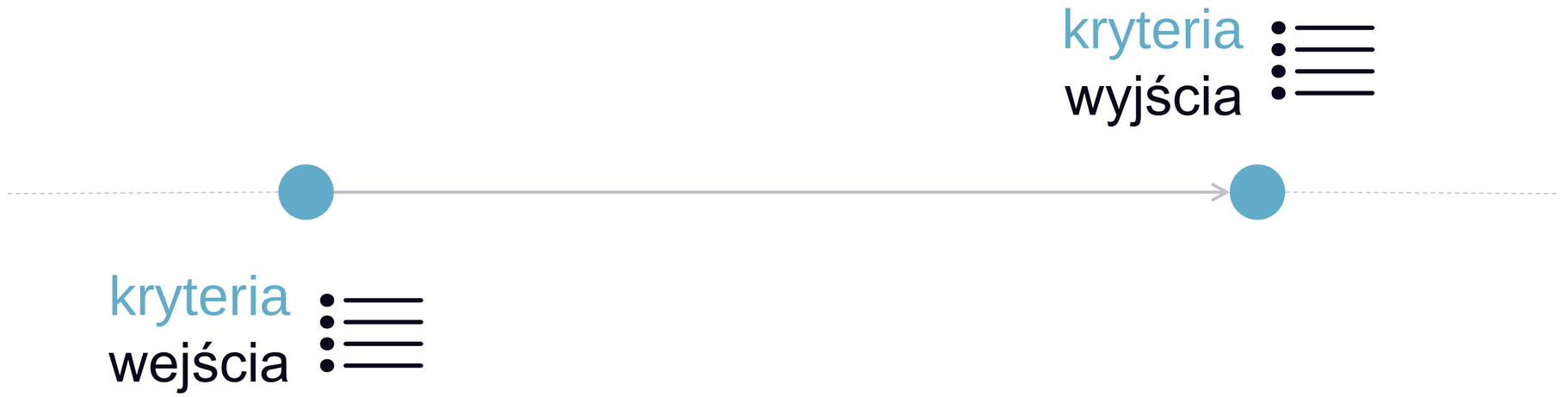
Definicja gotowości i Definicja ukończenia

kryteria wejścia (definicja gotowości):

zestaw warunków do oficjalnego rozpoczęcia określonego zadania.

kryteria wyjścia (definicja ukończenia):

zestaw warunków do oficjalnego zakończenia określonego zadania.



5. Zarządzanie testami
5.2.3 Kryteria wejścia i wyjścia

Kryteria wejścia

Kryteria wejścia są związane z dostępnością:

- testowalnych wymagań, historyjek użytkownika i/lub modeli (np. w przypadku stosowania strategii testowej opartej na modelu)
- elementów testowych, które spełniły kryteria wyjścia obowiązujące na wcześniejszych poziomach testowania
- środowiska testowego
- niezbędnych narzędzi testowych
- danych testowych i innych niezbędnych zasobów

5. Zarządzanie testami
5.2.3 Kryteria wejścia i wyjścia

Kryteria wyjścia

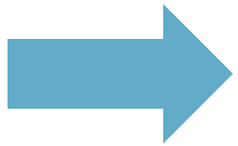
Kryteria wyjścia to np. :

- zakończenie wykonywania zaplanowanych testów
- osiągnięcie należytego poziomu pokrycia (tj. pokrycia wymagań, historyjek użytkownika i kryteriów akceptacji oraz kodu)
- nieprzekroczenie uzgodnionego limitu nieusuniętych defektów
- uzyskanie wystarczająco niskiej szacowanej gęstości defektów bądź wystarczająco wysokich wskaźników niezawodności

5. Zarządzanie testami
5.2.3 Kryteria wejścia i wyjścia

Przykładowe kryteria wejścia

- Testy komponentów są zakończone i możemy rozpocząć wykonywanie testów integracji
- 90% wymagań zostało sprawdzonych i zatwierdzonych
- Zgromadziliśmy dane testowe niezbędne do rozpoczęcia testów



Przykładowe kryteria wyjścia

- Wszystkie zaplanowane testy zostały wykonywane
- 95% planowanych testów przeszło poprawnie
- Żadne poważne usterki nie zostały nienaprawione
- Wszystkie zidentyfikowane ryzyka zostały pokryte testami

5. Zarządzanie testami
5.2.3 Kryteria wejścia i wyjścia

Co jeśli nie możemy spełnić kryteriów wyjścia?

- Powszechnym działaniem jest redukcja kryteriów wyjścia, aby nie przekroczyć budżetu, ukończyć testy w czasie i wdrożyć produkt.
- Pamiętajmy, że niektóre ryzyka są akceptowalne – zagrożenia zostały zdefiniowane na początku projektu, staraliśmy się je złagodzić, ale ostatecznie produkt został wydany ze znanymi problemami.

5. Zarządzanie testami
5.2.4 Harmonogram wykonywania testów

TERMINY

harmonogram wykonywania testu: harmonogram opisujący kolejność wykonywania zestawów testowych w ramach cyklu testowego.

harmonogram testów: lista aktywności, zadań lub zdarzeń z procesu testowego, określająca ich zamierzoną datę rozpoczęcia i zakończenia i/lub czas realizacji oraz ich współzależności.

5. Zarządzanie testami
5.2.4 Harmonogram wykonywania testów

Zróbmy to książkowo

- Zazwyczaj przypadki testowe są uporządkowane według priorytetów (najwyższy priorytet jest wykonywany jako pierwszy).
- W rzeczywistości istnieją zależności z innymi testami / funkcjonalnościami o niższym priorytecie, ale muszą one zostać wykonane jako pierwsze.
- Egzekucje testowe należy odpowiednio uporządkować, niezależnie od ich priorytetów.
- Testy regresji muszą również mieć swój priorytet w zależności od tego jak szybko chcemy uzyskać informację zwrotną na temat zmian.

5.2.4 Harmonogram wykonywania testów

Wykorzystanie Diagramu Gantta [1/5]

- Harmonogram testów można przedstawić w postaci diagramu Gantta.
- Uwzględnia podział przedsięwzięcia na poszczególne zadania a także rozplanowanie ich w czasie.
- Jest przydatny do bieżącego monitorowania i kontroli.
- Niestety pokazuje jedynie kolejność zadań i nie zawiera opisu.
- Jest to zobrazowanie planu pracy a nie pokazanie najkrótszej możliwej drogi do ukończenia pracy.

5. Zarządzanie testami

5.2.4 Harmonogram wykonywania testów

Wykorzystanie Diagramu Gantta [2/5]

Przedstawienie zadań testowych wraz z ich zależnościami, które pokazują kolejność wykonania poszczególnych testów:

- Test 2,3,8 ⇨ Test 9
- Test 5 ⇨ Test 3
- Test 6,7 ⇨ Test 8
- Test 8 ⇨ Test 9
- Test 12 ⇨ Test 11
- Test 4 ⇨ Test 10

Nazwa	DZIEŃ 1	DZIEŃ 2	DZIEŃ 3	DZIEŃ 4	DZIEŃ 5	DZIEŃ 6	DZIEŃ 7	DZIEŃ 8	DZIEŃ 9	DZIEŃ 10
TEST 1										
TEST 2										
TEST 3										
TEST 4										
TEST 5										
TEST 6										
TEST 7										
TEST 8										
TEST 9										
TEST 10										
TEST 11										
TEST 12										

5. Zarządzanie testami

5.2.4 Harmonogram wykonywania testów

Wykorzystanie Diagramu Gantta [3/5]

Przedstawienie zadań testowych wraz z ich zależnościami oraz priorytetem, które pokazują kolejność wykonania poszczególnych testów.

Priorytet	Nazwa	DZIEŃ 1	DZIEŃ 2	DZIEŃ 3	DZIEŃ 4	DZIEŃ 5	DZIEŃ 6	DZIEŃ 7	DZIEŃ 8	DZIEŃ 9	DZIEŃ 10
1	TEST 1										
3	TEST 2										
6	TEST 3										
8	TEST 4										
2	TEST 5										
4	TEST 6										
5	TEST 7										
7	TEST 8										
9	TEST 9										
12	TEST 10										
11	TEST 11										
10	TEST 12										

5. Zarządzanie testami

5.2.4 Harmonogram wykonywania testów

Wykorzystanie Diagramu Gantta [4/5]

Przedstawienie zadań testowych wraz z ich zależnościami, ważnością oraz priorytetem, które pokazują kolejność wykonania poszczególnych testów.

Ważność:
1 – wysoka,
2 – średnia,
3 – niska.

Ważność	Priorytet	Nazwa	DZIEŃ 1	DZIEŃ 2	DZIEŃ 3	DZIEŃ 4	DZIEŃ 5	DZIEŃ 6	DZIEŃ 7	DZIEŃ 8	DZIEŃ 9	DZIEŃ 10
1	1	TEST 1										
2	3	TEST 2										
2	6	TEST 3										
3	8	TEST 4										
2	2	TEST 5										
3	4	TEST 6										
3	5	TEST 7										
1	7	TEST 8										
2	9	TEST 9										
1	12	TEST 10										
2	11	TEST 11										
3	10	TEST 12										

5.2.4 Harmonogram wykonywania testów

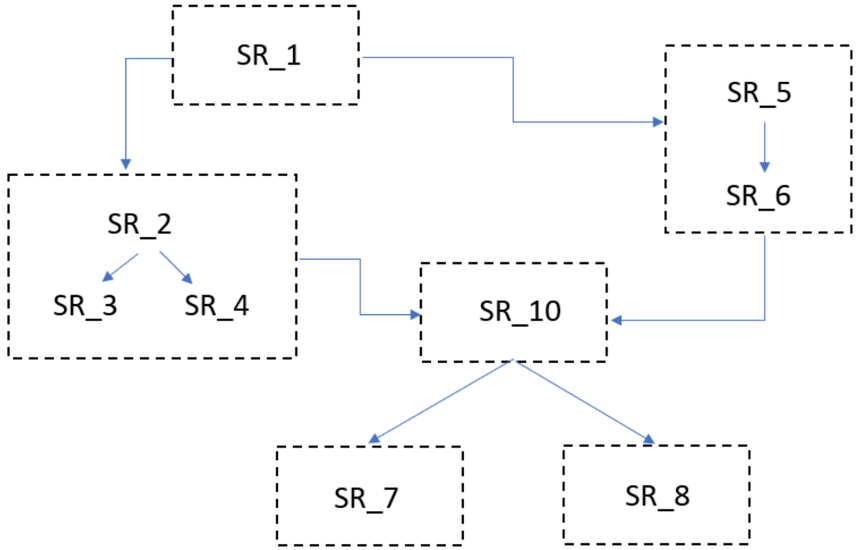
Wykorzystanie Diagramu Gantta [5/5]

- W rzeczywistości konieczne będzie także wykonanie ewentualnych retestów oraz testów regresji która nie jest ujęta w przedstawionym wcześniej harmonogramie.
- Zazwyczaj retesty i testy regresji mają charakter priorytetowy – szybka informacja na temat zmian.
- Planując sekwencje wykonania testów zawsze trzeba wziąć pod uwagę wszelkie możliwe aspekty. Być może wykonamy te same testy szybciej. Czasem różne sekwencje testów mogą mieć różną efektywność.

5. Zarządzanie testami
5.2.4 Harmonogram wykonywania testów

PRZYKŁAD

W jaki sposób powinniśmy używać priorytetów i logicznych zależności podczas planowania testów ?



SR_xx to test, który sprawdza wymagania systemowe.

Biorąc pod uwagę logiczne zależności, na przykład SR_01 -> SR_02, co oznacza, że SR_02 zależy od SR_01 i złożoność wymagań (wszystkie z nich mają niewielką złożoność, z wyjątkiem SR_2 i SR_7, które są wysokie), w jaki sposób zaplanowałbyś ten zestaw testów?

5. Zarządzanie testami

5.2.5 Czynniki wpływające na pracochłonność testowania

Jak przewidzieć wysiłek testowy?

Wysiłek testowy to ilość prac związanych z testowaniem, które należy ukończyć, aby osiągnąć cele testowania. Do czynników wpływających na pracochłonność testowania zaliczamy czynnik ludzki, charakterystykę produktu, procesu i rezultaty testu.

Czynnik ludzki

- umiejętności i doświadczenie zaangażowanych osób, zwłaszcza doświadczenie z podobnymi projektami/produktami (np. doświadczenie branżowe)
- spójność zespołu i umiejętności kierownika

Rezultaty testów

- liczba i ważność wykrytych defektów
- liczba wymaganych poprawek

5. Zarządzanie testami

5.2.5 Czynniki wpływające na pracochłonność testowania

Jak przewidzieć wysiłek testowy?

Charakterystyka produktu

- czynniki ryzyka związane z produktem
- jakość specyfikacji (tj. podstawy testów)
- wielkość produktu
- złożoność dziedziny produktu
- wymagania dotyczące charakterystyk jakościowych (np. bezpieczeństwa i niezawodności)
- wymagany poziom szczegółowości dokumentacji testów
- wymagania dotyczące zgodności z przepisami

Charakterystyka procesu wytwarzania oprogramowania

- stabilność i dojrzałość organizacji
- stosowany model wytwarzania oprogramowania (np. model kaskadowy, model zwinny)
- podejście do testowania
- używane narzędzia
- proces testowy
- presja czasu

5. Zarządzanie testami
5.2.6 Techniki szacowania testów

Estymata

szacowanie testów: obliczona aproksymacja wyniku związana z różnymi aspektami testowania (takimi jak wysiłek, data zakończenia, poniesione koszty, ilość przypadków testowych itp.), która jest użyteczna, nawet gdy dane wejściowe są niekompletne, niepewne lub zakłócone.

- techniki oparta na miarach
- techniki eksperckie

5. Zarządzanie testami
5.2.6 Techniki szacowania testów

Estymata

techniki oparte na miarach

- Wykres spalania w kontekście zwinnych modeli w celu określenia ilości pracy, jaką można wykonać w danym okresie czasu.
- Modele usuwania defektów, czas pomiędzy zgłoszeniem, a naprawą defektu.

PRZYKŁADY

techniki eksperckie

- Poker planistyczny, w oparciu o doświadczenia członków zespołu.
- Technika Delficka, szacunek oparty na doświadczeniu ekspertów biznesowych.

Monitorowanie testów i **nadzór** nad testami

5. Zarządzanie testami
5.3 Monitorowanie testów i nadzór nad testami

TERMINY

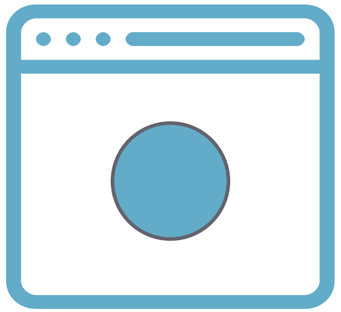
monitorowanie testów: zadanie zarządzania testami, które zajmuje się działaniami związanymi z okresowym monitorowaniem statusu aktywności testowych, identyfikowaniem odchyleń od planu lub oczekiwanego statusu oraz raportowaniem statusu do interesariuszy.

zbierz informacje i uzyskaj opinie

nadzór nad testami: zadanie z zakresu zarządzania testem, którego celem jest opracowanie i zastosowanie działań korygujących projekt testowy, kiedy monitowanie pokazuje odchylenie od planu.

działania naprawcze

5. Zarządzanie testami
5.3 Monitorowanie testów i nadzór nad testami



PRZYKŁAD

Jakie działania naprawcze można zastosować?

- Ponowne ustalanie priorytetów testów w przypadku stwierdzenia określonego ryzyka (np. opóźnione dostarczenie oprogramowania)
- Zmiana harmonogramu testów ze względu na dostępność lub niedostępność środowiska testowego lub innych zasobów
- Ponowna ocena, czy element testowy spełnia kryterium wejścia lub wyjścia z powodu poprawek

5. Zarządzanie testami

5.3.1 Miary stosowane w odniesieniu do testowania

Dane uzyskane podczas czynności testowych mają istotne znaczenie dla nadzoru nad testami

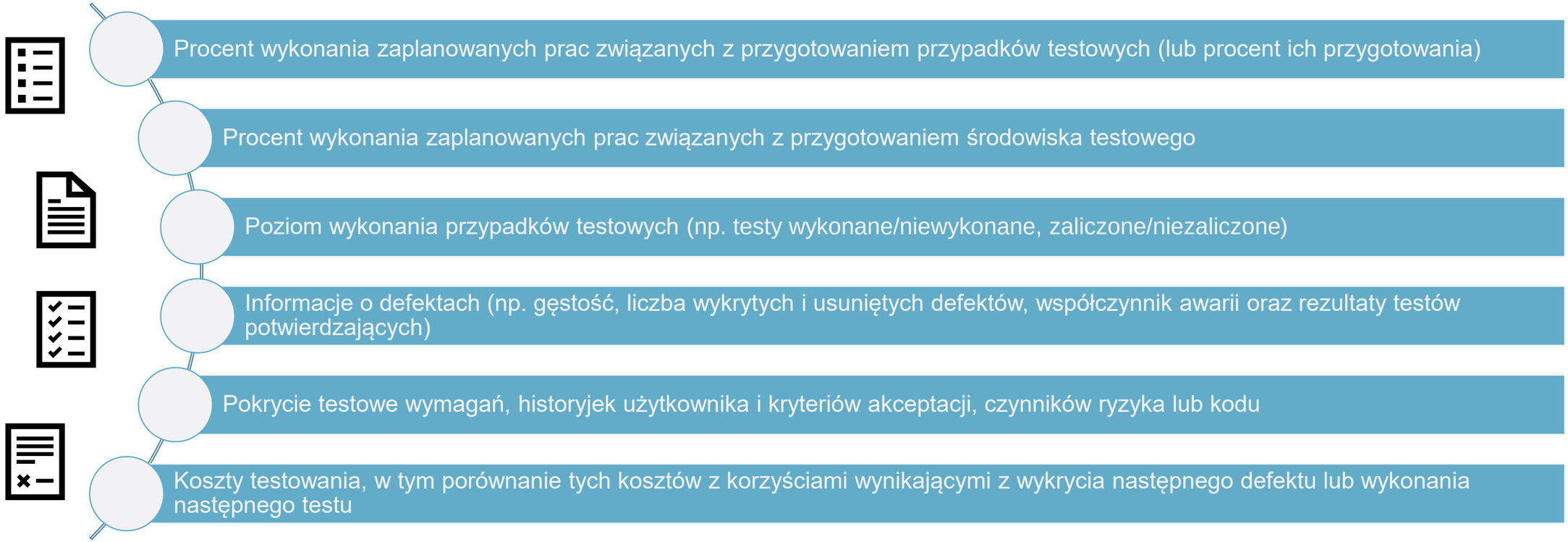
Metryki testowania pozwalają ocenić:

- postęp realizacji harmonogramu i budżetu
- bieżącą jakość przedmiotu testów
- adekwatność wybranego podejścia do testowania
- skuteczność czynności testowych z punktu widzenia realizacji celów

5. Zarządzanie testami

5.3.1 Miary stosowane w odniesieniu do testowania

Powszechne miary dotyczące testów



5. Zarządzanie testami
5.3.2 Cel, treść i odbiorcy raportów z testów

Dlaczego, co i komu raportujemy ?

- Podsumowanie i komunikacja statusu testów
- Może być traktowany jako raport z przebiegu testu przygotowany w trakcie testów lub raport sumaryczny po wykonaniu czynności testowych

To kierownik testów przygotowuje takie raporty dla zainteresowanych stron na podstawie informacji od testerów, analityków testów lub technicznych analityków testów.

5. Zarządzanie testami
5.3.2 Cel, treść i odbiorcy raportów z testów

Zawartość raportów testowych

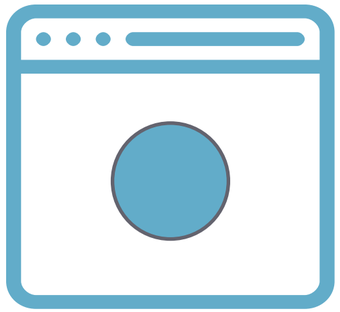
Raport z postępu testów może zawierać:

- Status działań testowych i postęp w stosunku do planu testów
- Czynniki zakłócające postęp w testach
- Testy zaplanowane na następny okres raportowania
- Jakość przedmiotu testów

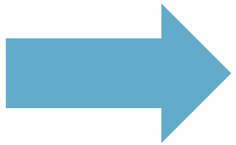
Raport sumaryczny z testów może zawierać:

- podsumowanie wykonanych testów
- informacje na temat zdarzeń, które miały miejsce w okresie testowania
- informacje o odstępstwach od planu, włączając odstępstwa od harmonogramu, czasu trwania, lub wysiłku związanego z testowaniem
- informacje o statusie testowania i jakości produktu z punktu widzenia kryteriów wyjścia (lub definicji ukończenia)
- informacje o czynnikach, które blokowały lub nadal blokują przebieg testów
- miary związane z defektami, przypadkami testowymi, pokryciem testowym, postępem prac, oraz wykorzystaniem zasobów
- informacje na temat ryzyka rezydualnego
- informacje o wytworzonych produktach pracy związanych z testowaniem, które mogą zostać ponownie wykorzystane

5. Zarządzanie testami
5.3.2 Cel, treść i odbiorcy raportów z testów



PRZYKŁAD



Podsumowanie testów

Zgodziłeś się z interesariuszami, aby dostarczać im raport z przebiegu testu w każdy piątek do końca dnia roboczego zgodnie z wcześniej uzgodnionymi wytycznymi. Ważne jest, aby można było usunąć wszelkie niedogodności, które uniemożliwiają wykonanie testów. Pod koniec każdego z okresów wykonywania testów wymagane jest dostarczenie raportu z podsumowaniem testu zawierającego wszystkie potrzebne informacje o tym, co zostało przetestowane, czy spełniamy definicję ukończenia i czy jakiekolwiek ryzyko pozostanie niezmienione.

5. Zarządzanie testami

5.3.2 Cel, treść i odbiorcy raportów z testów

Raport testowy zależy od kontekstu

Zawartość raportu testowego zależy od kontekstu projektu, organizacji i cyklu życia oprogramowania. W przypadku skomplikowanego projektu podlegającemu określonym regulacjom prawnym możemy oczekiwać, że raport z testu będzie bardzo szczegółowy, podczas gdy w raportach projektów zwinnych można je zwizualizować na tablicach Kanban, wykresach spalania i omawiać na codziennych spotkaniach scrumowych.

Raport z testu powinien być dopasowany do odbiorców. Biorąc pod uwagę, czy jest to grupa techniczna czy nietechniczna, powinniśmy podawać jedynie niezbędne informacje.


5.4
Zarządzanie
konfiguracją

5. Zarządzanie testami
5.4 Zarządzanie konfiguracją

Do czego służy konfiguracja?

Korzystamy z zarządzania konfiguracją w celu ustanowienia i utrzymania integralności komponentów, systemu lub testaliów oraz ich relacji w całym cyklu życia produktu.

Procedury dotyczące zarządzania konfiguracją należy określić podczas planowania testów.

5. Zarządzanie testami
5.4 Zarządzanie konfiguracją

Do czego służy konfiguracja?

Zarządzanie konfiguracją gwarantuje nam, że:

- Wszystkie przedmioty testów zostały zidentyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane ze sobą wzajemnie.
- Wszystkie testalia zostały zidentyfikowane, objęte kontrolą wersji i śledzeniem zmian oraz powiązane ze sobą wzajemnie i z wersjami przedmiotu testów w sposób pozwalający utrzymać możliwość śledzenia na wszystkich etapach procesu testowego.
- Wszystkie zidentyfikowane dokumenty i elementy oprogramowania były przywoływane w sposób jednoznaczny w dokumentacji testów.

5. Zarządzanie testami
5.4 Zarządzanie konfiguracją

Jak konfiguracja wspomaga testowanie?

Wyobraź sobie, że znalazłeś defekt wykonując test na wersji 1.382. Przed utworzeniem raportu upewniasz się, czy nie został on wcześniej utworzony. Zdajesz sobie sprawę, że został już zgłoszony, ale jest ustawiony jako zamknięty (naprawiony) w wersji 1.370. Niestety defekt nadal istnieje i musi zostać ponownie otwarty i przypisany do naprawy.

W dalszym etapie sprawdzania zostałeś poproszony o upewnienie się, czy wersja współpracującego z nami oprogramowania wbudowanego jest w wersji 3.5 lub nowszej, ale okazało się, że korzystasz z wersji 3.8, która jest poprawna.

Czynniki **ryzyka**, a **testowanie**

5. Zarządzanie testami
5.5.1 Definicja ryzyka

TERMINY

ryzyko: czynnik, który w przyszłości może skutkować negatywnymi konsekwencjami; zazwyczaj opisywany poprzez wpływ oraz prawdopodobieństwo.

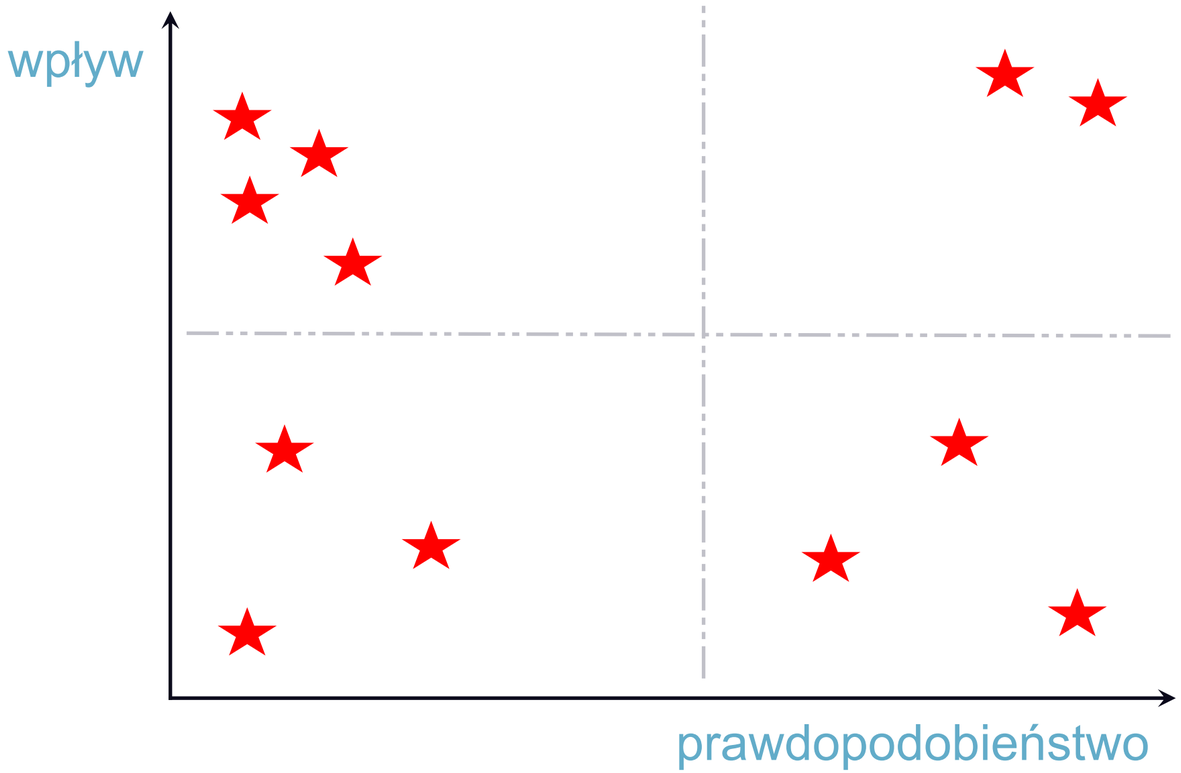
poziom ryzyka (eksponowanie ryzyka): jakościowa lub ilościowa miara ryzyka zdefiniowana przez wpływ i prawdopodobieństwo.

wpływ ryzyka: szkoda, jaka powstanie jeżeli ryzyko zmaterializuje się jako rzeczywisty skutek lub zdarzenie.

prawdopodobieństwo ryzyka: oszacowane prawdopodobieństwo, że ryzyko wystąpi jako wynik rzeczywisty lub zdarzenie.

5. Zarządzanie testami
5.5.1 Definicja ryzyka

Ryzyko na pierwszy rzut oka



5. Zarządzanie testami

5.5.2 Czynniki ryzyka produktowego i projektowego

TERMINY

ryzyko produktowe: ryzyko wpływające na jakość produktu.

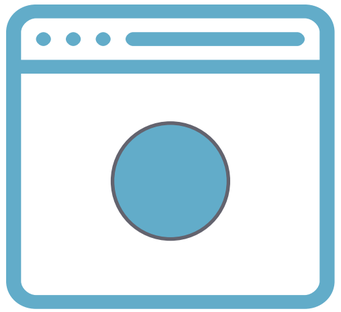
słaba charakterystyka produktu

produkt nie spełnia wymagań użytkownika końcowego

ryzyko projektowe: ryzyko wpływające na sukces projektu.

negatywny wpływ na projekt - osiągnięcie jego celów

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego

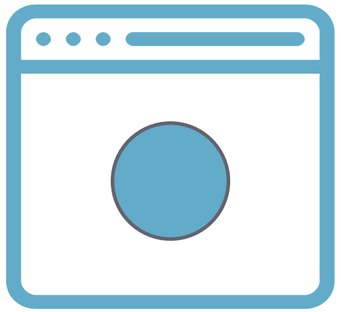


PRZYKŁAD

Ryzyko Produktowe

- niewykonywanie zakładanych funkcji oprogramowania zgodnie ze specyfikacją
- niewykonywanie zakładanych funkcji oprogramowania zgodnie z potrzebami użytkowników, klientów i/lub interesariuszy
- niedostateczne spełnienie przez architekturę systemu określonych wymagań niefunkcjonalnych
- niepoprawne wykonywanie konkretnych obliczeń w niektórych okolicznościach
- błędy w kodzie struktury sterowania pętlą
- zbyt długi czas odpowiedzi w systemie transakcyjnym wysokiej wydajności
- niezgodność informacji zwrotnych na temat doświadczenia użytkownika z oczekiwaniami dotyczącymi produktu

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego



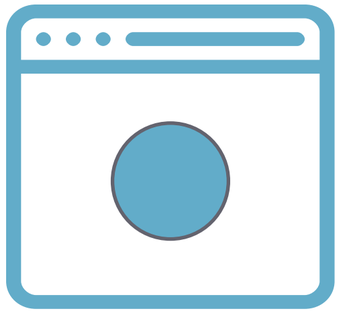
PRZYKŁAD

Ryzyko Projektowe

Problemy związane z projektem

- potencjalne opóźnienia w dostawach, ukończeniu zadań bądź spełnieniu kryteriów wyjścia
- niedokładne oszacowanie, realokacja środków do projektów o wyższym priorytecie lub ogólne cięcia kosztów w całej organizacji, które mogą skutkować niewystarczającym finansowaniem
- wprowadzenie w ostatniej chwili zmian wymagających dokonania licznych przeróbek

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego



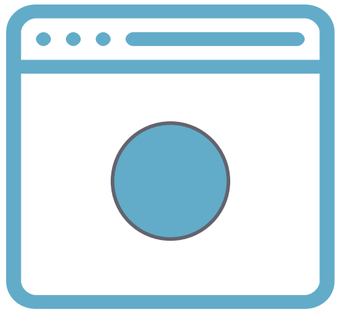
PRZYKŁAD

Ryzyko Projektowe

Problemy organizacyjne

- niewystarczające kwalifikacje lub przeszkolenie pracowników bądź braki kadrowe
- konflikty i problemy wynikające z doboru personelu
- brak dostępności użytkowników, pracowników struktur biznesowych lub ekspertów merytorycznych z powodu sprzecznych priorytetów biznesowych

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego



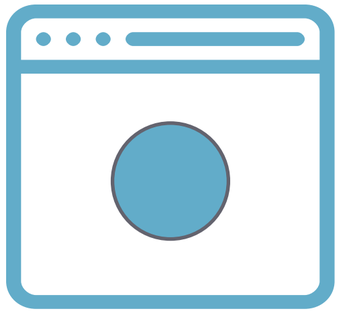
PRZYKŁAD

Ryzyko Projektowe

Problemy natury politycznej

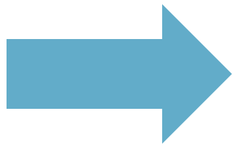
- niewłaściwe przekazywanie przez testerów informacji na temat ich potrzeb i/lub rezultatów testów
- niepodjęcie przez programistów/testerów dalszych działań na podstawie informacji uzyskanych w wyniku testowania i przeglądów (np. niewprowadzenie udoskonaleń w procedurach wytwarzania i testowania oprogramowania)
- niewłaściwe podejście do testowania lub oczekiwania związane z testowaniem (np. niedocenianie korzyści wynikających z wykrycia defektów podczas testowania)

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego



PRZYKŁAD

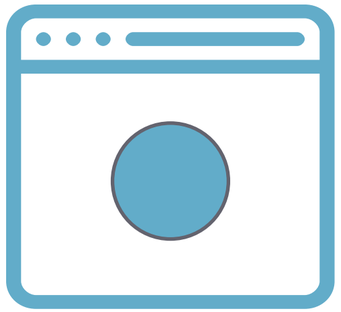
Ryzyko Projektowe



Problemy techniczne

- niedostateczne doprecyzowanie wymagań
- brak możliwości spełnienia wymagań z uwagi na ograniczenia czasowe
- nieudostępnienie na czas środowiska testowego
- zbyt późne przeprowadzenie konwersji danych, zaplanowanie migracji lub udostępnienie potrzebnych do tego narzędzi
- wady w procesie wytwórczym mogące wpływać na spójność lub jakość produktów prac projektowych, kodu, konfiguracji, danych testowych i przypadków testowych na różnych etapach procesu wytwarzania oprogramowania
- kumulacja defektów lub innego rodzaju dług techniczny powstały na skutek problemów z zarządzaniem defektami lub innych podobnych problemów

5. Zarządzanie testami
5.5.2 Czynniki ryzyka produktowego i projektowego



PRZYKŁAD

Ryzyko Projektowe

Problemy związane z dostawcami

- niedostarczenie niezbędnego produktu lub usługi przez zewnętrznego dostawcę bądź ogłoszenie przez niego upadłości
- utrudnienia w realizacji projektu wynikające z problemów związanych z umowami

5. Zarządzanie testami

5.5.2 Czynniki ryzyka produktowego i projektowego

Kto jest odpowiedzialny za zarządzanie ryzykiem projektowym?

…może to być kierownik projektu, ale także kierownik testów dla obszarów ryzyka związanych z testami.

5. Zarządzanie testami

5.5.3 Testowanie oparte na ryzyku a jakość produktu

TERMINY

testowanie oparte na ryzyku: testowanie, w którym zarządzanie, wybór, priorytetyzacja i wykorzystanie działań testowych i zasobów są oparte na odpowiednich typach i poziomach ryzyka.

Zarządzaj, Wybieraj, określ priorytety czynności testowych

ryzyko rezydualne: ryzyko lub niebezpieczeństwo zdarzenia, zjawiska lub okoliczności, które po zastosowaniu wszelkich możliwych, bądź częściowych środków kontroli oraz najlepszych praktyk w postępowaniu z nim nadal pozostaje.

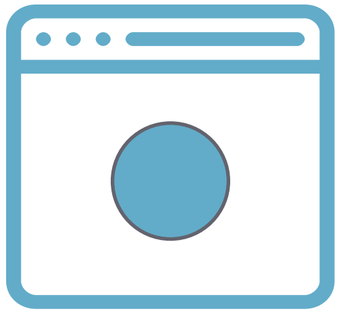
5. Zarządzanie testami

5.5.3 Testowanie oparte na ryzyku a jakość produktu

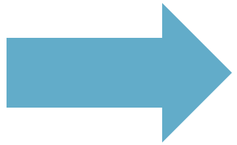
Wykorzystanie testowania opartego na ryzyku

Używamy tego podejścia, aby móc zdecydować **kiedy zacząć testy** i gdzie powinniśmy zwrócić większą uwagę na testowanie. Testujemy w celu **zmniejszenia prawdopodobieństwa** wystąpienia niepożądanego zdarzenia. Testowanie jest uważane za **działanie łagodzące ryzyko**, które dostarcza informacji zwrotnej na temat zidentyfikowanych zagrożeń. Nierozwiązane ryzyko jest opisane jako **ryzyko rezydualne**.

5. Zarządzanie testami
5.5.3 Testowanie oparte na ryzyku a jakość produktu



PRZYKŁAD



Jak analiza ryzyka wpływa na testowanie?

Zakres testów opiera się także na rozpoznanych ryzykach. Jeśli analiza ryzyka wskazuje na wydajność, użyteczność lub inne obszary, które mogą nie działać zgodnie z oczekiwaniami, należy to zweryfikować przed wydaniem produktu. Testowanie jest w tym przypadku traktowane jako działanie łagodzące, które wymaga zaprojektowania, wdrożenia i wykonania testów pokrywające dane ryzyko.

5. Zarządzanie testami

5.5.3 Testowanie oparte na ryzyku a jakość produktu

Analiza Ryzyka Produktowego

- identyfikacja ryzyka produktowego
- ocena każdego ryzyka (prawdopodobieństwo i wpływ)
- łagodzenie ryzyka (informacje o ryzyku produktu wykorzystywane są do planowania testów, specyfikacji, przygotowania i wykonywania przypadków testowych itp.)

5. Zarządzanie testami

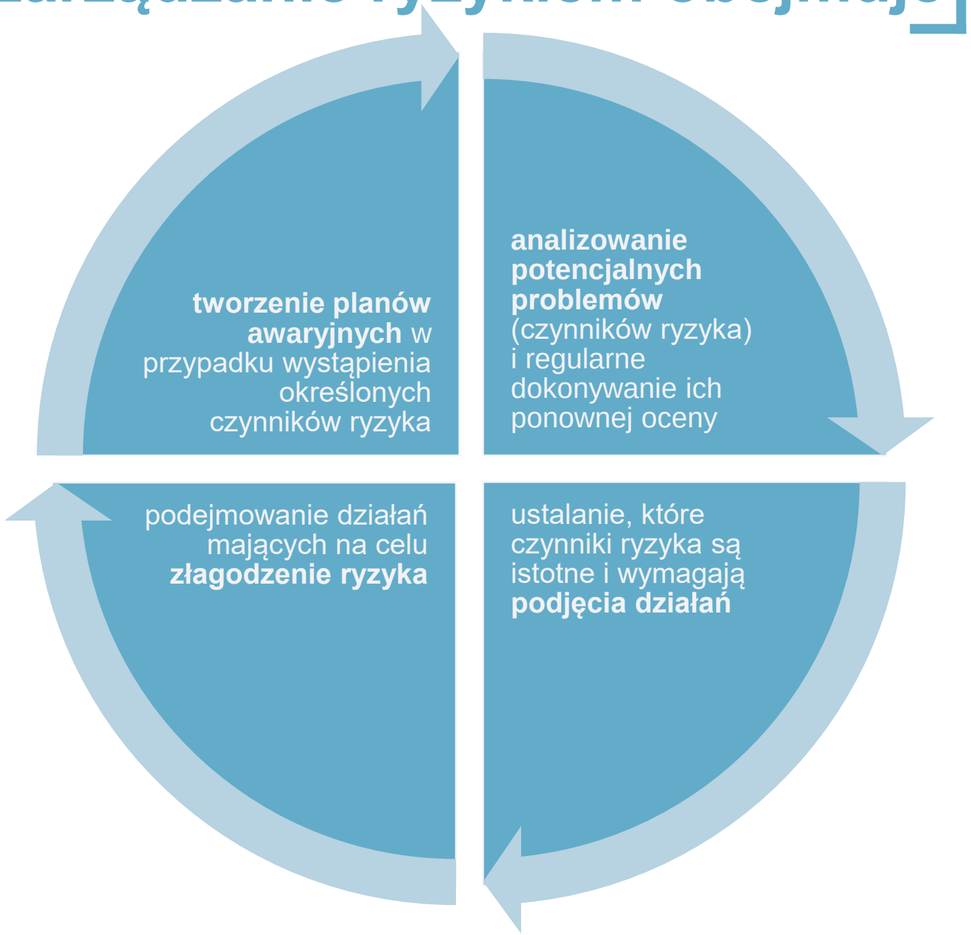
5.5.3 Testowanie oparte na ryzyku a jakość produktu

Co umożliwia analiza ryzyka produktowego ?

- wskazanie odpowiednich technik testowania
- określenie konkretnych typów testów, które należy wykonać (np. testowanie zabezpieczeń, testowanie dostępności)
- określenie zakresu wykonywanych testów
- ustalenie priorytetów testowania w sposób sprzyjający jak najwcześniejszemu wykryciu defektów krytycznych
- ustalenie, czy w celu zmniejszenia ryzyka należy wykonać inne czynności niezwiązane z testowaniem

5. Zarządzanie testami
5.5.3 Testowanie oparte na ryzyku a jakość produktu

Zarządzanie ryzykiem obejmuje



Zarządzanie defektami

5. Zarządzanie testami
5.6 Zarządzanie defektami

Wprowadzenie [1/2]

- Wykrywanie defektów (usterek) to jeden z opisywanych celów testowania. Są to wszelkie rozbieżności pomiędzy oczekiwanym i rzeczywistym rezultatem.
- To w jaki sposób należy udokumentować i opisać defekt powinno być opisane w planie testów.
- Defekty mogą dotyczyć np. wszelkiego rodzaju dokumentacji, działającego systemu, testowania, a także użytkowania produktu.

5. Zarządzanie testami
5.6 Zarządzanie defektami

Wprowadzenie [2/2]

- Defekty są śledzone od momentu wykrycia do stanu ostatecznej weryfikacji naprawy.
- Proces zarządzania defektami musi zostać ustalony, aby dana organizacja podążała zgodnie z przepływem pracy i ustalonymi zasadami.
- Proces może się różnić w zależności od kontekstu systemu, poziomu testu, cyklu życia oprogramowania i może być bardzo nieformalny.

5. Zarządzanie testami
5.6 Zarządzanie defektami

TERMINY

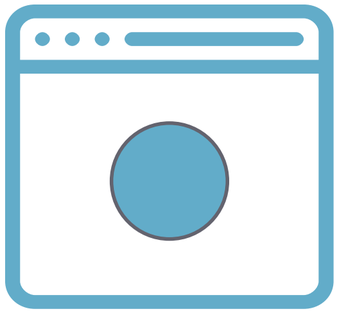
rezultat fałszywie pozytywny (fałszywie niezaliczony): test, w którym defekt został zaraportowany, chociaż defekt ten nie występuje.

Wykonanie testu kończy się defektem, ale tak naprawdę to nie jest defekt

rezultat fałszywie negatywny (fałszywie zaliczony): test, w którym nie zidentyfikowano obecności usterki występującej w testowanym obiekcie.

Test kończy się powodzeniem i nie identyfikujemy prawdziwego defektu

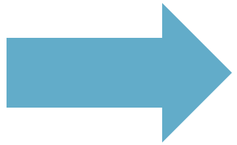
5. Zarządzanie testami
5.6 Zarządzanie defektami



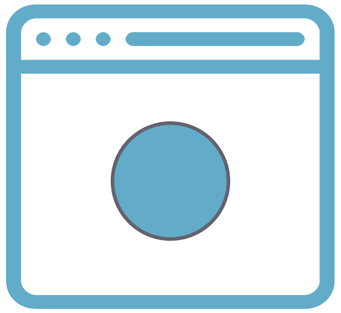
PRZYKŁAD

Rezultat fałszywie pozytywny (false-positive)

Kod został niedawno zmieniony ze względu na ostateczne uzgodnienia z naszym klientem. Tester i analityk biznesowy nie są o tym informowani. Test jest wykonany i kończy się niepowodzeniem. Ostatecznie powstaje defekt, ale oprogramowanie działa zgodnie z oczekiwaniami.



5.6 Zarządzanie defektami



PRZYKŁAD

Rezultat fałszywie negatywny (false-negative)

Przypadek testowy wymaga zweryfikowania wielu elementów na poziomie interfejsu użytkownika. Niestety tester przeoczył, aby zweryfikować jeden element interfejsu użytkownika. Test został zaliczony, ale element interfejsu użytkownika, którego tester nie zweryfikował jest niewłaściwy. Defekt powinien zostać utworzony, ale tester nie zidentyfikował obecności tej usterki.

5. Zarządzanie testami
5.6 Zarządzanie defektami

Dlaczego tworzymy raporty o defektach ?

Defekty:
- w kodzie, systemie, dowolnej dokumentacji
- na dowolnym poziom testu
- zgłaszane problemy

Dlaczego?	
	informujemy o stanie systemu
	identyfikujemy i rozwiązujemy potencjalne defekty
	śledzimy jakość produktu roboczego i postęp testowania
	zbieramy ew. pomysły na doskonalenie procesu

5. Zarządzanie testami
5.6 Zarządzanie defektami

Treść raportu o defekcie

- **identyfikator**, tytuł i krótkie podsumowanie zgłaszanego defektu
- data i autor
- **konfiguracja** i **środowisko** testowanego elementu
- faza cyklu życia, w której zaobserwowano defekt
- **opis defektu** umożliwiający jego odtworzenie i usunięcie (w tym ewentualne zrzuty ekranu lub nagrania)
- **oczekiwane i rzeczywiste rezultaty**
- zakres lub stopień wpływu (**ważność**) defektu z punktu widzenia interesariuszy
- **priorytet** usunięcia defektu
- **stan** raportu o defekcie (np. otwarty)
- wnioski, zalecenia i zgody
- kwestie globalne, takie jak wpływ zmiany wynikającej z defektu na inne obszary
- historia zmian, w tym sekwencja działań podjętych przez członków zespołu projektowego w celu zlokalizowania, usunięcia i potwierdzenia usunięcia defektu
- odwołania do innych elementów, w tym do przypadku testowego, dzięki któremu problem został ujawniony

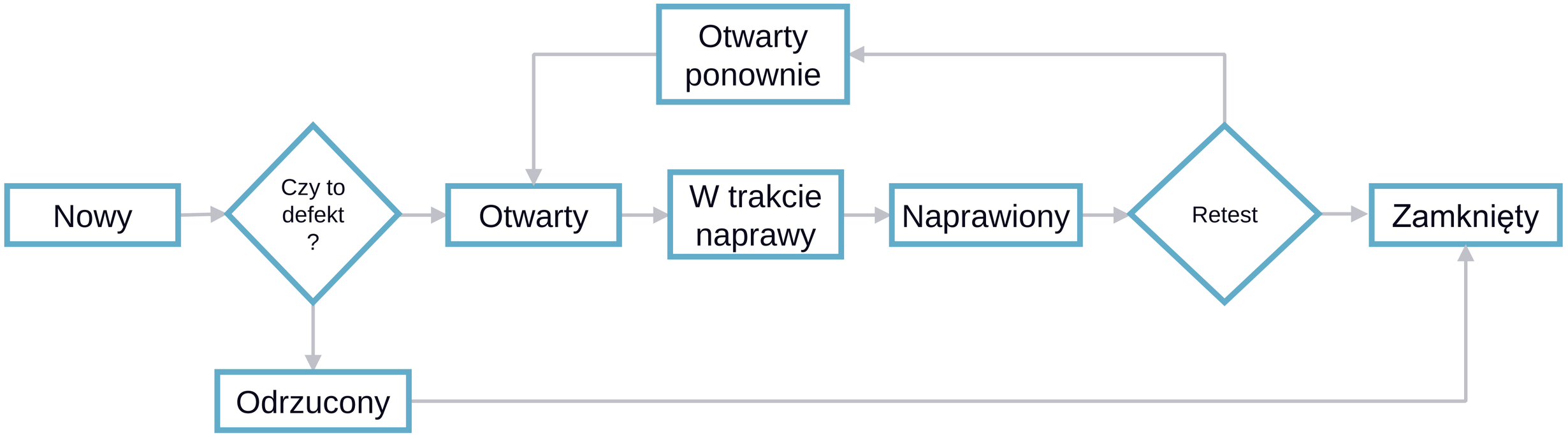
5. Zarządzanie testami
5.6 Zarządzanie defektami

Pamiętaj, że…

- Identyfikator jest zwykle przypisywany automatycznie, jeśli używane jest dedykowane narzędzie.
- Początkowo podaje się stan defektu (np. "NOWY").
- W przypadku przeprowadzania przeglądu, wady mogą być udokumentowane w formie notatek ze spotkania przeglądowego.
- Czasami nieudokumentowane, np. podczas tworzenia nowych funkcji.

5. Zarządzanie testami
5.6 Zarządzanie defektami

Przykładowy cykl życia defektu



5.6 Zarządzanie defektami

Możliwe klasyfikacje defektu [1/2]

Nowy	pierwszy, początkowy stan defektu
Otwarty	zgłoszenie uznane za defekt
W trakcie naprawy	zgłoszenie, które jest przypisane do programisty, który analizuje przyczynę defektu i go usuwa
Naprawiony	defekt, który jest usunięty i może zostać zweryfikowany
Zamknięty	defekt, który jest poprawnie zweryfikowany – został usunięty
Odrzucony	defekt, który jest nie możliwy do odtworzenia („cannot reproduce”) lub np. nie jest błędem („not a defect”)

5.6 Zarządzanie defektami

Możliwe klasyfikacje defektu [2/2]

Wymaga więcej informacji	zgłoszenie, które nie może być przekazane do naprawy, ponieważ wymaga więcej szczegółów
Do udoskonalenia („refine”)	zgłoszenie, które nie ma jasno określonego oczekiwanego rezultatu
Odroczony	zgłoszenie, które jest uznane jako defekt, ale nie będzie teraz przekazane do naprawy
Duplikat	zgłoszenie, które istnieje już w systemie
Zduplikowany	zgłoszenie, które posiada duplikat w systemie
Nie będzie naprawiany	zgłoszenie, które nie będzie naprawiane pomimo, że jest uzasadnione

5.6 Zarządzanie defektami

Przykładowy raport o defekcie

Raport o defekcie			
ID:	Sprint15_Defekt_04	**Produkt:**	Webserwis
Tytuł:	Problem z logowaniem	**Status:**	Otwarty
Autor:	John	**Data:**	1/10/2012
Opis:	1. Wprowadź login 2. Wprowadź hasło **Aktualny rezultat:** Komunikat 'Null exception' **Oczekiwany rezultat:** Poprawne zalogowanie do systemu		

5. Zarządzanie testami
5.6 Zarządzanie defektami

Czego najczęściej brakuje w niepoprawnym raporcie o defekcie?

- Brak informacji o środowisku, na którym występuje defekt
- Niepełne informacje na temat konfiguracji (np. wersja aplikacji)
- Niejasne kroki do wykonania w celu odtworzenia defektu
- Brak oczekiwanego rezultatu lub referencji do podstawy testów

5. Zarządzanie testami

Projekt współfinansowany ze środków Europejskiego Funduszu Społecznego
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020

06

Narzędzia wspomagające testowanie.

6. Narzędzia wspomagające testowanie
Cele Nauczania

6.1 Uwarunkowania związane z narzędziami testowymi

FL-6.1.1 (K2) Kandydat potrafi sklasyfikować narzędzia testowe według przeznaczenia i obsługiwanych czynności testowych

FL-6.1.2 (K1) Kandydat potrafi zidentyfikować korzyści i czynniki ryzyka związane z automatyzacją testowania

FL-6.1.3 (K1) Kandydat pamięta o szczególnych uwarunkowaniach związanych z narzędziami do wykonywania testów i zarządzania testami

6. Narzędzia wspomagające testowanie
Cele Nauczania

6.2 Skuteczne korzystanie z narzędzi

- FL-6.2.1 (K1) Kandydat potrafi wskazać główne zasady wyboru narzędzi
- FL-6.2.2 (K1) Kandydat pamięta cele stosowania projektów pilotażowych wprowadzających narzędzia
- FL-6.2.3 (K1) Kandydat potrafi wskazać czynniki sukcesu związane z oceną, implementacją, wdrożeniem i bieżącą obsługą narzędzi testowych w organizacji

Uwarunkowania związane z narzędziami testowymi

6. Narzędzia wspomagające testowanie
Wprowadzenie

Jakich narzędzi testowych używamy?

	Narzędzia używane do testów pomagają nam przeprowadzać testy i przygotowywać dane testowe
	Narzędzia pomagają zarządzać wymaganiami, przypadkami testowymi, procedurami testowymi, zautomatyzowanymi skryptami testowymi, wynikami testów, danymi testowymi i defektami oraz do raportowania i monitorowania wykonywania testów
	Narzędzia używane są do badania i oceny, np. wyników testów
	Narzędzia wspomagają testowanie (arkusz kalkulacyjny jest w tym znaczeniu również narzędziem testowym)

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Jak wspieramy testowanie za pomocą narzędzi ?

- ✓ Poprawa skuteczności działań testowych
- ✓ Automatyzacja powtarzalnych zadań (np. testowanie regresji)
- ✓ Obsługa ręcznie wykonywanych czynności podczas całego procesu testowego
- ✓ Poprawa jakości działań testowych
- ✓ Bardziej konsekwentne testowanie i wyższy poziom powtarzalności defektów
- ✓ Automatyzowanie czynności, których nie można wykonać ręcznie (np. testowanie wydajności na dużą skalę)
- ✓ Zwiększanie wiarygodności testowania (np. symulacja zachowania)

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Narzędzia mają znaczenie

Klasyfikacja narzędzi

Narzędzia są klasyfikowane w tym sylabusie zgodnie z działaniami, które obsługują. Niektóre narzędzia obsługują tylko lub głównie jedną czynność, natomiast inne pozwalają wykonywać kilka czynności.

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

DEFINICJA

efekt próbnika: efekt wpływu urządzenia pomiarowego na moduł lub system podczas dokonywania pomiaru, np. poprzez narzędzie do testów wydajnościowych. Przykładowo wydajność testowanego oprogramowania może być nieznacznie gorsza, kiedy stosowane jest narzędzie do testów wydajnościowych.

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Klasyfikacja narzędzi testowych

Klasyfikacja narzędzi testowych

Niektóre narzędzia posiadają funkcje, które są bardziej odpowiednie dla deweloperów.

Te dedykowane programistom są oznaczone literą D

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Klasyfikacja narzędzi testowych

Narzędzia wspomagające zarządzanie testowaniem i testaliami

- narzędzia do zarządzania testami i narzędzia do zarządzania cyklem życia aplikacji
- narzędzia do zarządzania wymaganiami (np. śledzenie do obiektów testów)
- narzędzia do zarządzania defektami
- narzędzia do zarządzania konfiguracją
- narzędzia do ciągłej integracji (D)

Narzędzia wspomagające testowanie statyczne

- narzędzia do analizy statycznej (D)

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Klasyfikacja narzędzi testowych

Narzędzia wspomagające projektowanie i implementację testów

- narzędzia do testowania opartego na modelu
- narzędzia do przygotowywania danych testowych

Narzędzia wspomagające wykonywanie i logowanie testów

- narzędzia do wykonywania testów (np. do uruchamiania testów regresji)
- narzędzia mierzące pokrycie (np. pokrycie wymagań lub kodu) (D)
- jarzma testowe (D)

6. Narzędzia wspomagające testowanie
6.1.1 Klasyfikacja narzędzi testowych

Klasyfikacja narzędzi testowych

Narzędzia wspomagające pomiar wydajności i analizę dynamiczną

- narzędzia do testów wydajnościowych
- narzędzia do analizy dynamicznej (D)

Narzędzia wspomagające wyspecjalizowane czynności testowe

- w tej grupie znajdują się narzędzia pomocne w bardziej szczegółowym testowaniu charakterystyk niefunkcjonalnych np. testowanie użyteczności, zabezpieczeń lub przenaszalności

6. Narzędzia wspomagające testowanie

6.1.2 Korzyści i czynniki ryzyka związane z automatyzacją testów

Korzyści

- **ograniczenie powtarzalnych czynności** wykonywanych ręcznie (uruchamianie testów regresji, wykonywanie zadań związanych z konfigurowaniem środowiska, wielokrotne wprowadzanie tych samych danych testowych)
- **zwiększenie spójności i powtarzalności** (np. poprzez spójne tworzenie danych testowych, wykonywanie testów przy użyciu narzędzia w tej samej kolejności i z tą samą częstotliwością bądź wyprowadzanie testów w spójny sposób z wymagań)
- bardziej **obiektywna ocena** (np. w zakresie miar statycznych czy pokrycia)
- **łatwiejszy dostęp do informacji** na temat testowania (np. danych statystycznych i wykresów obrazujących postęp testów, współczynników występowania defektów oraz danych dotyczących wydajności).

6. Narzędzia wspomagające testowanie

6.1.2 Korzyści i czynniki ryzyka związane z automatyzacją testów

Czynniki ryzyka 1/2

- **nierealistyczne oczekiwania** wobec narzędzia (dotyczące między innymi funkcjonalności i łatwości obsługi)
- **niedoszacowanie czaso- i pracochłonności oraz kosztów** początkowego wdrożenia narzędzia (w tym szkoleń i korzystania z usług zewnętrznych ekspertów)
- **niedoszacowanie czaso- i pracochłonności działań** niezbędnych do osiągnięcia znaczących i trwałych korzyści z tytułu używania narzędzia (nakładów niezbędnych do wprowadzenia zmian w procesie testowym)
- **niedoszacowanie nakładów pracy** związanych z utrzymaniem artefaktów testowych generowanych przez narzędzie
- **nadmierne uzależnienie** od narzędzia (nie można wszystkiego zautomatyzować)
- **zaniedbanie kontroli wersji** zasobów testowych

6. Narzędzia wspomagające testowanie

6.1.2 Korzyści i czynniki ryzyka związane z automatyzacją testów

Czynniki ryzyka 2/2

- **zlekceważenie zależności** między newralgicznymi narzędziami (np. zarządzanie wymaganiami, testami i defektami)
- **niska jakość usług dostawcy** w zakresie pomocy technicznej bądź dostarczania uaktualnień lub poprawek usuwających defekty
- **możliwość wstrzymania realizacji projektu** w przypadku narzędzia typu open source
- **możliwość braku współpracy** narzędzia z nowymi platformami lub technologiami
- możliwy **brak jednoznacznego określenia odpowiedzialności** za narzędzie (np. w zakresie mentoringu, aktualizacji itd.)

6. Narzędzia wspomagające testowanie

6.1.2 Korzyści i czynniki ryzyka związane z automatyzacją testów

Nie każde narzędzie to gwarancja sukcesu

Użycie każdego narzędzia wymaga pewnego wysiłku (głównie czas i zasoby)

6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Narzędzia do wykonywania testów

- Zautomatyzowane skrypty testowe wymagają **znacznego wysiłku**, aby osiągnąć **znaczące korzyści**
- Narzędzia do testowania automatyzacji wykonują testy, stosując różne podejścia:
  - Rejestrowanie działań wykonywanych ręcznie (capture-playback)
  - Testowanie sterowane danymi (data-driven)
  - Testowanie oparte na słowach kluczowych (keyword-driven)
  - Testowanie oparte na modelu (MBT)

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

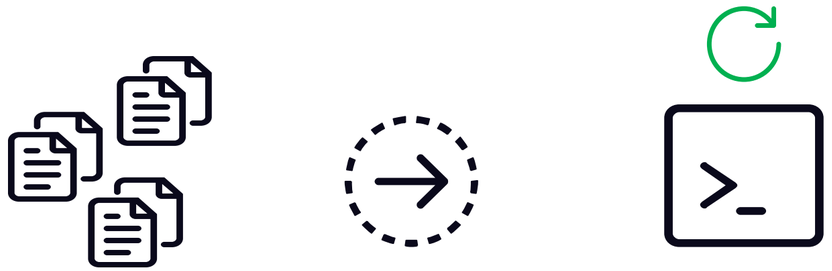
Rejestrowanie działań wykonywanych ręcznie

- ręczne akcje są rejestrowane, a następnie w razie potrzeby odtwarzane
- mała skalowalność w dużych projektach
- przechwycony skrypt jest liniową reprezentacją akcji i określonych danych
- technika ta poprawiła się w ostatnich latach, ale nadal wymaga dużego nakładu pracy związanego z konserwacją, by dostosowywać testy do zmian w interfejsie

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Testowanie sterowane danymi

- wejście i oczekiwany wynik są oddzielone od testu
- zwykle przechowywane w arkuszu kalkulacyjnym i wykonywane w iteracjach
- stosuje się bardziej generyczny skrypt testowy, który odczytuje dane wejściowe i wykonuje testy przy użyciu różnych danych



6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Testowanie oparte na słowach kluczowych

- stosuje się generyczny skrypt, który przetwarza słowa kluczowe opisujące wykonywane akcje (zwane także słowami akcji)
- następnie wywołuje się odpowiednie skrypty przetwarzające dostarczone dane testowe.

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Testowanie oparte na słowach kluczowych

- Skalowalne dla dużych projektów.
- Dane testowe mogą być przygotowane przez testerów
- Wymagana jest ekspercka znajomość języka skryptowego (przez testerów, deweloperów lub specjalistów z dziedziny automatyzacji testowania)
- Słowo kluczowe jest traktowane jako zestaw działań opisanych w danej metodzie / funkcji, która jest wywoływana w sekwencji zdarzeń przez skrypt kontrolny (główny).

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Testowanie oparte na słowach kluczowych

- Główną korzyścią jest relatywnie niewielki nakład pracy w przypadku zmian w oprogramowaniu.
- Nie ma potrzeby aktualizowania wszystkich testów, które używają danego słowa kluczowego. Zmiana słowa kluczowego pociąga za sobą zmiany tych skryptów testowych, które zawierały "stare" słowo kluczowe i tych które powinny obsługiwać nowe słowo kluczowe (słowo akcji).

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Testowanie oparte na modelu

- Model jest tworzony w oparciu o specyfikację funkcjonalną, np. jako diagram działań.
- Podejście obsługiwane przez narzędzie, które generuje sekwencje zdarzeń do wykonania przez narzędzie do wykonywania testów.

Wszystkie wymienione podejścia wymagają wiedzy z zakresu programowania. Rezultat testu jest porównywany z wynikiem oczekiwanym. Głównym zyskiem jest to, że wykonuje to automat, a nie tester.

6. Narzędzia wspomagające testowanie
6.1.3 Narzędzia do wykonywania testów i zarządzania testami

Narzędzia do zarządzania testami

- Generujemy przydatne informacje w formacie zgodnym z potrzebami organizacji
- Wymagane jest utrzymanie spójności śledzenia wymagań z wymaganiami w narzędziu do zarządzania wymaganiami
- Tworzymy powiązania z informacjami o wersji przedmiotu testów w narzędziu do zarządzania konfiguracją

Istotne jest rozważenie zintegrowanego narzędzia, które obejmuje zarówno moduł zarządzania jak i inne moduły dostarczające np. informacje na temat harmonogramu i budżetu projektu, które mogą być używane w całej organizacji

Skuteczne **korzystanie** z narzędzi

6. Narzędzia wspomagające testowanie
6.2.1 Główne zasady wyboru narzędzi

DEFINICJA

Zwrot z inwestycji (ROI od ang. Return On Investment): Wskaźnik rentowności stosowany w celu zmierzenia efektywności przedsięwzięcia. Służy do pomiaru bezwzględnej opłacalności inwestycji.

6. Narzędzia wspomagające testowanie
6.2.1 Główne zasady wyboru narzędzi

Co uwzględnić przy wyborze narzędzia ? [1/2]

- Dojrzałość organizacji
- Uwzględnienie dostępności bezpłatnego okresu sprawdzania narzędzia
- Zrozumienie technologii (zgodność z innymi narzędziami)
- Zrozumienie działania narzędzi do budowania wersji i ciągłej integracji, które są już używane w danej organizacji, w celu zapewnienia kompatybilności i integracji
- Ocena narzędzia pod względem jasnych wymagań i obiektywnych kryteriów

6. Narzędzia wspomagające testowanie
6.2.1 Główne zasady wyboru narzędzi

Co uwzględnić przy wyborze narzędzia ? [2/2]

- Możliwości udoskonalania procesu testowego dzięki wykorzystaniu narzędzia
- Ocena sprzedawcy i ew. obsługi narzędzia (np. open source)
- Identyfikacja wymagań wewnętrznych (coaching i mentoring)
- Ocena potrzeb szkoleniowych z uwzględnieniem umiejętności testowania (i automatyzacji testów)
- Uwzględnianie modeli licencjonowania (np. komercyjne i open source)
- Wskaźnik ROI

6. Narzędzia wspomagające testowanie
6.2.1 Główne zasady wyboru narzędzi

Dowód słuszności koncepcji

Wykonujemy w celu ustalenia czy narzędzie działa efektywnie z naszym oprogramowaniem

Upewniamy się, że narzędzie działa poprawnie z naszą wewnętrzną infrastrukturą

Identyfikujemy potrzebne zmiany, aby działać skutecznie z tym narzędziem (jeśli potrzebne)

6. Narzędzia wspomagające testowanie
6.2.2 Projekt pilotażowy wprowadzający narzędzie

Cele projektu pilotażowego

Wybór narzędzia **TAK** → Dowód słuszności projektu (PoC) **ZAKOŃCZONY** → Rozpoczęcie projektu pilotażowego

- Zapozanie się z narzędziem (zrozumienie mocnych i słabych stron)
- Ocena, czy narzędzie pasuje do istniejących procesów i praktyk (co zmienić - w razie potrzeby)
- Decyzja o standardowych sposobach używania, zarządzania, przechowywania, utrzymania narzędzia i zasobów testowych
- Oszacowanie kosztów i korzyści
- Konfiguracja narzędzia pod kątem zbierania i raportowania potrzebnych danych

6. Narzędzia wspomagające testowanie
6.2.3 Czynniki sukcesu wdrażania narzędzi testowych

Czynniki sukcesu związane z narzędziami

- Przyrostowe wdrażanie narzędzia w organizacji
- Adaptacja i doskonalenie procesów w celu dopasowania do użycia narzędzia
- Zapewnienie użytkownikom szkoleń i mentoringu
- Definiowanie wytycznych dotyczących użycia narzędzia
- Wdrożenie mechanizmu zbierania informacji na temat użycia na podstawie rzeczywistego wykorzystania narzędzia
- Monitorowanie użycia narzędzia i udzielanie wsparcia
- Retrospekcje (lessons learned) ze wszystkimi użytkownikami
- Łatwa integracja narzędzia z cyklem życia oprogramowania

6. Narzędzia wspomagające testowanie

Polecane książki i matariały

- Normy:
  - ISO/IEC/IEEE 29119: Software Testing Standard
  - ISO/IEC 25010: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models
  - ISO/IEC 20246: Software and systems engineering — Work product reviews

- Strony internetowe:
  - http://satisfice.com
  - http://sjsi.org
  - http://glossary.istqb.org

Polecane książki i matariały

- Książki:
  - Sylabus poziomu podstawowego (wersja 3.1 2018)
  - Tester oprogramowania przygotowanie do egzaminu z testowania oprogramowania (2015), Karolina Zmitrowicz
  - Testowanie i jakość oprogramowania. Modele, techniki, narzędzia (2017), Adam Roman
  - Testowanie oprogramowania w praktyce (2017), Adam Roman, Karolina Zmitrowicz
  - Testowanie w procesie Scrum (2016), Tilo Linz
  - Jakość projektów informatycznych (2015), Karolina Zmitrowicz