CERTYFIKOWANY TESTER

NOTATKI DO SYLABUSA ISTQB

WERSJA 4.0

Notatki opracowała: Aleksandra Broda

SPIS TREŚCI

ROZDZIAŁ 1. PODSTAWY TESTOWANIA

1.1 CO TO JEST TESTOWANIE?

Typowe cele testów

- **dokonywanie oceny wymagań, historyjek użytkownika, projektów, kodu;**
- **powodowanie awarii i znajdowanie defektów;**
- **zapewnienie wymaganego pokrycia przedmiotu testów** *(chodzi o to, aby dostosować plan testów tak, aby wszystkie istotne elementy systemu były odpowiednio przetestowane, zgodnie z dokumentacją projektową i wymaganiami klienta)***;**
- **obniżanie poziomu ryzyka związanego z niedostateczną jakością oprogramowania;**
- **sprawdzanie, czy zostały spełnione wyspecyfikowane wymagania;**
- **sprawdzanie, czy przedmiot testów spełnia wymagania umowne, prawne i regulacyjne;**
- **dostarczanie interesariuszom informacji niezbędnych do podejmowania świadomych decyzji** *(zespół testujący dostarcza interesariuszom, czyli wszystkim zaangażowanym stronom w projekcie (np. kierownictwu, klientom, deweloperom), informacje na temat stanu i jakości produktu poddanego testom);*
- **budowanie zaufania do jakości przedmiotu testów;**
- **sprawdzanie, czy przedmiot testów jest kompletny i działa zgodnie z oczekiwaniami interesariuszy**

Różnica między testowaniem a debugowaniem

Testowanie	Debugowanie
• *pozwala wywołać awarie, które są skutkiem defektów w oprogramowaniu (testowanie dynamiczne)* • *lub znaleźć defekty bezpośrednio w przedmiocie testów (testowanie statyczne).*	• *po wywołaniu awarii* • *znalezienie przyczyn danej awarii (defektów), a następnie ich przeanalizowanie i wyeliminowanie* • *odtworzenie awarii -> znalezienie przyczyny -> usunięcie przyczyny*

1.2 DLACZEGO TESTOWANIE JEST NIEZBĘDNE?

Przykłady wskazujące dlaczego testowanie jest niezbędne

Testowanie pomaga osiągnąć uzgodnione cele w wyznaczonym zakresie i czasie, z zachowaniem ustalonego poziomu jakości oraz w granicach przyjętego budżetu poprzez:

- wykrywanie defektów, a następnie ich usuwanie (debugowanie)
- bezpośrednią ocenę jakości przedmiotu testów na różnych etapach cyklu wytwarzania oprogramowania (pomaga w podejmowaniu decyzji o przejściu do kolejnego etapu)
- wpływ na przebieg projektu wytwarzania oprogramowania, ponieważ testerzy dbają o to aby ich znajomość potrzeb użytkowników była uwzględniana na wszystkich etapach cyklu wytwarzania

Relacja między testowaniem a zapewnieniem jakości

Testowanie jest formą kontroli jakości.

Testowanie skupia się na identyfikowaniu problemów w produkcie, podczas gdy **zapewnianie jakości** koncentruje się na poprawie procesów, aby uniknąć powstawania tych problemów.

Pomyłka, defekt, awaria i ich podstawowe przyczyny

Pomyłka (błąd człowieka) → **Defekt** (usterka, pluskwa) → **Awaria**

Przyczyny:

- presja czasu
- złożoność produktów pracy, procesów, infrastruktury lub interakcji
- zmęczenie lub brak odpowiedniego przeszkolenia

Zasadniczy czynnik powodujący wystąpienie problemu (np. sytuacja prowadząca do pomyłki) jest nazywany podstawową przyczyną. Uważa się, że podjęcie odpowiednich działań w stosunku do podstawowej przyczyny (na przykład jej usunięcie) pozwala zapobiec powstaniu kolejnych podobnych awarii lub defektów, a przynajmniej zmniejszyć ich częstotliwość.

1.3 ZASADY TESTOWANIA

1. **Testowanie ujawnia defekty, ale nie może dowieść ich braku.**
2. **Testowanie gruntowne nie jest możliwe.**
3. **Wczesne testowanie oszczędza czas i pieniądze.**
4. **Defekty mogą się kumulować.**
5. **Testy ulegają zużyciu.** Wielokrotne powtarzanie tych samych testów prowadzi do spadku ich skuteczności w wykrywaniu nowych defektów.
6. **Testowanie zależy od kontekstu.** Nie ma jednego uniwersalnego podejścia do testowania.
7. **Przekonanie o braku defektów jest błędem.***

*Nawet bardzo dokładne przetestowanie wszystkich wyspecyfikowanych wymagań i usunięcie wszystkich znalezionych defektów nie chronią przed zbudowaniem systemu, który nie zaspokoi potrzeb użytkowników i nie spełni ich oczekiwań, nie pomoże klientowi w osiągnięciu celów biznesowych oraz będzie miał gorsze parametry od konkurencyjnych rozwiązań. Dlatego oprócz weryfikacji należy również przeprowadzać walidację. ***(Weryfikacja** *to proces sprawdzania, czy produkt jest zbudowany zgodnie z wymaganiami, podczas gdy* ***walidacja*** *to proces sprawdzania, czy produkt spełnia rzeczywiste potrzeby i oczekiwania użytkowników).*

1.4 CZYNNOŚCI TESTOWE

Poszczególne czynności i zadania testowe

- Planowanie testów polega na zdefiniowaniu celów testów, a następnie dokonaniu wyboru podejścia, które pozwoli najskuteczniej osiągnąć te cele
- Monitorowanie testów i nadzór nad testami. Monitorowanie testów polega na ciągłym sprawdzaniu wszystkich czynności testowych i porównywaniu rzeczywistego postępu z założeniami przyjętymi w planie, a nadzór nad testami polega na podejmowaniu działań, które są niezbędne do osiągnięcia celów testowania.
- Analiza testów służy do ustalenia tego “ co należy” przetestować, polega na przeanalizowaniu podstawy testów w celu zidentyfikowania testowalnych cech oraz zdefiniowania i określenia priorytetów związanych z nimi warunków testowych, z uwzględnieniem występujących w danym przypadku ryzyk i poziomów ryzyka.
- Projektowanie testów służy do ustalenia “jak należy testować”, polega na przekształceniu warunków testowych w przypadki testowe i inne testalia
- Implementacja testów polega na utworzeniu lub pozyskaniu testaliów niezbędnych do wykonywania testów
- Wykonywanie testów polega na uruchamianiu testów zgodnie z harmonogramem wykonywania testów (czyli na wykonywaniu przebiegów testów), przy czym może się to odbywać manualnie lub automatycznie.
- Ukończenie testów obejmuje czynności, które są zwykle wykonywane w momencie osiągnięcia kamieni milowych projektu (takich jak przekazanie do eksploatacji, zakończenie iteracji lub ukończenie testów danego poziomu).

Wpływ kontekstu na proces testowy

Sposób, w jaki wykonywane jest testowanie, zależy od kontekstu, na który składają się między innymi następujące czynniki:

- interesariusze (w tym ich potrzeby, oczekiwania, wymagania, gotowość do współpracy itd.);
- członkowie zespołu (w tym ich umiejętności, wiedza, doświadczenie, dostępność, potrzeby w zakresie szkoleń itd.);
- dziedzina biznesowa (krytyczność przedmiotu testów, zidentyfikowane ryzyka, potrzeby rynku, konkretne uregulowania prawne itd.);
- czynniki techniczne (rodzaj oprogramowania, architektura produktu, zastosowana technologia itd.);
- ograniczenia związane z projektem (zakres, terminy, budżet, zasoby itd.);
- czynniki organizacyjne (struktura organizacyjna, istniejące polityki, przyjęte praktyki itd.);
- cykl wytwarzania oprogramowania (praktyki w dziedzinie inżynierii oprogramowania, metody wytwarzania itd.);
- narzędzia (dostępność, łatwość używania, zgodność itd.).

Testalia wspomagające czynności testowe

Testalia - produkty pracy powstałe w wyniku wykonywania czynności testowych, dzielimy na:

- Produkty pracy związane z planowaniem testów: plan testów, harmonogram testów, rejestr ryzyk oraz kryteria wejścia i wyjścia. *Rejestr ryzyk zawiera wykaz ryzyk wraz z informacjami o prawdopodobieństwie, wpływie i sposobach łagodzenia każdego z nich.*
- Produkty pracy związane z monitorowaniem testów i nadzorem nad testami: raporty o postępie testów, dokumentację dotyczącą dyrektyw nadzoru oraz informacje o ryzyku.
- Produkty pracy związane z analizą testów: uszeregowane według priorytetów warunki testowe (np. kryteria akceptacji) i raporty o defektach dotyczące defektów w podstawie testów.
- Produkty pracy związane z projektowaniem testów: uszeregowane według priorytetów przypadki testowe, karty opisu testów, elementy pokrycia oraz wymagania dotyczące danych testowych i środowiska testowego.
- Produkty pracy związane z implementacją testów: procedury testowe, skrypty testów automatycznych, zestawy testowe, dane testowe, harmonogram wykonywania testów oraz elementy środowiska testowego (zaślepki, sterowniki, symulatory i wirtualizacja usług).
- Produkty pracy związane z wykonywaniem testów: dzienniki testów i raporty o defektach.
- Produkty pracy związane z ukończeniem testów: sumaryczny raport z testów, lista czynności do wykonania mających na celu wprowadzenie udoskonaleń w kolejnych projektach lub iteracjach, udokumentowane wnioski oraz żądania zmian (np. w formie backlogu produktu).

Korzyści wynikające ze śledzenia powiązań

Śledzenie powiązań oznacza monitorowanie i utrzymywanie relacji między różnymi elementami procesu testowego. Korzyści:

- ułatwia ocenę pokrycia testowego (np. śledzenie powiązań między przypadkami testowymi a wymaganiami pozwala potwierdzić, że wymagania zostały pokryte przez przypadki testowe),
- umożliwia ocenę poziomu ryzyka (śledzenie powiązań między wynikami testów a ryzykami),
- pozwala ustalać wpływ zmian oraz ułatwia przeprowadzanie audytów testów i spełnianie kryteriów związanych z zarządzaniem w obszarze IT,
- umożliwia tworzenie bardziej zrozumiałych raportów o postępie testów i sumarycznych raportów z testów,
- możliwość udzielania informacji potrzebnych do oceny jakości produktów, wydajności procesów i postępu w realizacji projektu z punktu widzenia celów biznesowych.

Role w testowaniu

- **Osoba pełniąca rolę związaną z zarządzaniem testami** ponosi ogólną odpowiedzialność za proces testowy i pracę zespołu testowego oraz za kierowanie przebiegiem czynności testowych. Rola ta polega głównie na wykonywaniu czynności związanych z planowaniem testów, monitorowaniem testów, nadzorem nad testami oraz ukończeniem testów.
- **Osoba pełniąca rolę związaną z testowaniem** ponosi ogólną odpowiedzialność za aspekty techniczne testowania. Rola ta polega głównie na podejmowaniu działań związanych z analizą, projektowaniem, implementacją i wykonywaniem testów.

1.5 NIEZBĘDNE UMIEJĘTNOŚCI I DOBRE PRAKTYKI W DZIEDZINIE TESTOWANIA

Przykłady ogólnych umiejętności wymaganych w testowaniu

- wiedza w dziedzinie testowania (umożliwiająca zwiększanie skuteczności testowania np. poprzez zastosowanie technik testowania);
- staranność, ostrożność, ciekawość, dbałość o szczegóły i metodyczność (niezbędne do identyfikowania defektów, zwłaszcza defektów trudnych do wykrycia);
- umiejętności komunikacyjne oraz umiejętność aktywnego słuchania i pracy w zespole (pozwalające sprawnie komunikować się ze wszystkimi interesariuszami, w zrozumiały sposób przekazywać informacje innym osobom oraz zgłaszać i omawiać defekty);
- umiejętność analitycznego i krytycznego myślenia oraz kreatywność (umożliwiające zwiększanie skuteczności testowania);
- wiedza techniczna (pozwalająca zwiększyć efektywność testowania np. poprzez korzystanie z odpowiednich narzędzi testowych);

- wiedza merytoryczna (pozwalająca zrozumieć użytkowników i przedstawicieli jednostek biznesowych oraz sprawnie się z nimi porozumiewać).

Zalety podejścia "cały zespół"

Podejście „cały zespół" *(ang. whole-team approach)* opiera się na umiejętności sprawnej pracy zespołowej i działania na rzecz realizacji celów zespołu. Zalety:

- zwiększa dynamikę pracy zespołowej, usprawnia wymianę informacji i współpracę w zespole oraz tworzy efekt synergii, ponieważ umożliwia wykorzystanie różnych kombinacji umiejętności na rzecz realizacji projektu,
- w ten sposób testerzy mogą przekazywać wiedzę w dziedzinie testowania innym członkom zespołu oraz wpływać na proces wytwarzania produktu.

Korzyści i wady niezależności testowania

Produkty pracy mogą być testowane przez

- autora (brak niezależności),
- innych członków zespołu autora (pewien stopień niezależności),
- testerów spoza zespołu autora w obrębie danej organizacji (wysoki poziom niezależności)
- testerów spoza organizacji (bardzo wysoki poziom niezależności).

W większości projektów najlepiej sprawdza się przeprowadzanie testów na wielu poziomach niezależności

KORZYŚCI	WADY
• prawdopodobieństwo wykrycia przez niezależnych testerów innego rodzaju awarii i defektów niż te wykryte przez programistów ze względu na różne doświadczenia, techniczne punkty widzenia i błędy poznawcze, • niezależny tester może zweryfikować, zakwestionować lub obalić założenia przyjęte przez interesariuszy na etapie specyfikowania i implementacji systemu.	• Odizolowanie niezależnych testerów od zespołu deweloperskiego może prowadzić do braku współpracy, problemów z wymianą informacji, a nawet konfliktu z tym zespołem, • niebezpieczeństwo utraty przez programistów poczucia odpowiedzialności za jakość oraz stwarza ryzyko, że niezależni testerzy zostaną potraktowani jako wąskie gardło i obarczeni winą za nieterminowe przekazanie produktu do eksploatacji.

ROZDZIAŁ 2. TESTOWANIE W CYKLU WYTWARZANIA OPROGRAMOWANIA

2.1 TESTOWANIE W KONTEKŚCIE CYKLU WYTWARZANIA OPROGRAMOWANIA

Model cyklu wytwarzania oprogramowania (ang. software development lifecycle — SDLC) określa wzajemne relacje między poszczególnymi fazami wytwarzania oprogramowania oraz rodzajami czynności wykonywanych w ramach tego procesu.

Przykładami modeli cyklu wytwarzania oprogramowania są między innymi:

- sekwencyjne modele wytwarzania oprogramowania (np. model kaskadowy - ang. waterfall - lub model V),
- iteracyjne modele wytwarzania oprogramowania (np. model spiralny lub prototypowanie)
- przyrostowe modele wytwarzania oprogramowania (np. model Unified Process).

Wpływ wybranego modelu cyklu wytwarzania oprogramowania na testowanie

Warunkiem powodzenia procesu testowania jest dopasowanie go do przyjętego cyklu wytwarzania oprogramowania.

Wybór modelu cyklu wytwarzania oprogramowania wpływa na:

- zakres i czas wykonywania czynności testowych (np. poziomy testów i typy testów);
- szczegółowość dokumentacji testów;
- wybór technik testowania i podejścia do testowania;
- zakres automatyzacji testów;
- role i obowiązki testera.

Dobre praktyki testowania mające zastosowanie do wszystkich modeli cyklu wytwarzania oprogramowania

- Do każdej czynności związanej z wytwarzaniem oprogramowania powinna być przypisana odpowiadająca jej czynność testowa, tak aby wszystkie czynności związane z wytwarzaniem oprogramowania podlegały kontroli jakości.
- Poszczególnym poziomom testów powinny odpowiadać konkretne i różne cele testów, co pozwoli zapewnić odpowiednio szeroki zakres testowania, a przy tym uniknąć nadmiarowości.
- Aby zapewnić zgodność z zasadą wczesnego testowania, analizę i projektowanie testów na potrzeby danego poziomu testów należy rozpocząć w odpowiadającej temu poziomowi fazie cyklu wytwarzania oprogramowania.

- Testerzy powinni uczestniczyć w przeglądach produktów pracy natychmiast po udostępnieniu wersji roboczych odpowiednich dokumentów, tak aby wcześniejsze testowanie i wykrywanie defektów pomagało w realizacji podejścia przesunięcie w lewo.

Przykłady podejść typu „najpierw test" w kontekście wytwarzania oprogramowania

"**Najpierw test"** - zasada wczesnego testowania → testy są definiowane przed rozpoczęciem pisania kodu, przykłady:

Wytwarzanie sterowane testami (TDD):

- Zamiast rozbudowanych mechanizmów projektowania oprogramowania do określania sposobu tworzenia kodu wykorzystywane są przypadki testowe.
- Najpierw pisane są testy, a dopiero potem powstaje kod umożliwiający ich pomyślne przejście. Następnie testy i kod podlegają refaktoryzacji.

Wytwarzanie sterowane testami akceptacyjnymi (ATDD):

- Testy są tworzone na podstawie kryteriów akceptacji w ramach procesu projektowania systemu.
- Testy są pisane przed wytworzeniem części aplikacji, która ma je pomyślnie przejść.

Wytwarzanie sterowane zachowaniem (BDD):

- Pożądane zachowanie aplikacji wyraża się w postaci przypadków testowych napisanych w prostej formie języka naturalnego, która jest zrozumiała dla interesariuszy — zwykle w formacie Given/When/Then (Mając/Kiedy/Wtedy).
- Następnie przypadki testowe są automatycznie przekładane na możliwe do wykonania testy.

W jaki sposób metodyka DevOps może wpłynąć na testowanie

- szybkie otrzymywanie informacji zwrotnych na temat jakości kodu oraz ewentualnego niekorzystnego wpływu zmian na dotychczasowy kod;
- ciągłą integrację, która sprzyja przesunięciu w lewo w obszarze testowania (patrz sekcja 2.1.5) poprzez zachęcenie programistów do dostarczania wysokiej jakości kodu sprawdzonego testami modułowymi i analizą statyczną;
- promowanie zautomatyzowanych procesów takich jak ciągła integracja i ciągłe dostarczanie, które ułatwiają tworzenie stabilnych środowisk testowych;
- zwiększenie widoczności niefunkcjonalnych charakterystyk jakościowych (np. wydajności lub niezawodności);
- zmniejszenie zapotrzebowania na powtarzalne testowanie manualne dzięki automatyzacji, jaką zapewnia potok dostarczania;

- zmniejszenie ryzyka związanego z regresją z uwagi na skalę i zasięg automatycznych testów regresji.

Na czym polega przesunięcie w lewo

Zasada wczesnego testowania jest niekiedy nazywana „przesunięciem w lewo”.

Przesunięcie w lewo sugeruje, że testowanie powinno odbywać się wcześniej (np. bez czekania na implementację kodu bądź integrację modułów).

Sposoby w jakie można wdrożyć podejście przesunięcie w lewo w testowaniu:

- dokonywanie przeglądu specyfikacji (pozwala znaleźć potencjalne defekty, takie jak niejasności, braki czy niespójności);
- pisanie przypadków testowych przed rozpoczęciem pisania kodu i uruchamianie kodu w jarzmie testowym podczas implementacji;
- korzystanie z mechanizmu ciągłej integracji (a jeszcze lepiej — ciągłego dostarczania), ponieważ mechanizm ten pozwala szybko uzyskiwać informacje zwrotne i wykonywać automatyczne testy modułowe w odniesieniu do kodu źródłowego przekazywanego do repozytorium kodu;
- przeprowadzanie analizy statycznej kodu źródłowego przed rozpoczęciem testowania dynamicznego lub w ramach zautomatyzowanego procesu;
- wykonywanie testowania niefunkcjonalnego już od poziomu testów modułowych wszędzie tam, gdzie jest to możliwe (jest to forma przesunięcia w lewo, ponieważ testy niefunkcjonalne są często wykonywane na późniejszych etapach cyklu wytwarzania oprogramowania, gdy jest już dostępny kompletny system wraz z reprezentatywnym środowiskiem testowym).

W jaki sposób retrospektywy mogą posłużyć jako mechanizmy doskonalenia procesów

Poprzez:

- zwiększenie skuteczności i efektywności testów (np. poprzez wprowadzenie w życie sugestii dotyczących doskonalenia procesów);
- podniesienie jakości testaliów (np. w wyniku wspólnego przeglądu procesów testowych);
- zacieśnienie więzi w zespole i wspólne uczenie się (np. dzięki możliwości zgłaszania problemów i proponowania usprawnień);
- podniesienie jakości podstawy testów (np. dzięki możliwości identyfikowania i usuwania niedociągnięć związanych z jakością i zakresem wymagań);
- usprawnienie współpracy między programistami a testerami (np. dzięki regularnemu weryfikowaniu i optymalizowaniu zasad współpracy).

Retrospektywy (zwane także spotkaniami poprojektowymi lub retrospektywami projektu) są często organizowane po zakończeniu projektu lub iteracji bądź osiągnięciu kamienia milowego związanego z przekazaniem do eksploatacji.

2.2 POZIOMY TESTÓW I TYPY TESTÓW

Poziomy testów

Poziomy testów to grupy czynności testowych. Wyróżniamy:

- **Testowanie modułowe** (jednostkowe lub testowanie komponentów) skupia się na oddzielnym testowaniu poszczególnych modułów. Często wymaga ono stosowania określonych elementów pomocniczych, takich jak jarzma testowe lub struktury do testów jednostkowych (ang. frameworks). Testowanie modułowe jest zwykle wykonywane przez programistów w środowiskach tworzenia oprogramowania.
- **Testowanie integracji modułów** (testowanie połączenia) skupia się na interfejsach i interakcjach między modułami. Sposób testowania integracji modułów zależy w dużej mierze od strategii integracji — może to być na przykład strategia zstępująca, strategia wstępująca bądź strategia typu „wielki wybuch" (ang. big bang).
- **Testowanie systemowe** skupia się na ogólnym zachowaniu i możliwościach całego systemu lub produktu. Często obejmuje również kompleksowe testowanie funkcjonalne wszystkich zadań, jakie system ten może wykonywać, oraz testowanie niefunkcjonalne charakterystyk jakościowych. W przypadku niektórych niefunkcjonalnych charakterystyk jakościowych wskazane jest przeprowadzanie testów kompletnego systemu w reprezentatywnym środowisku testowym (np. testowanie użyteczności), ale można również zastosować symulacje podsystemów. Testowanie systemowe może być wykonywane przez niezależny zespół testowy i jest powiązane ze specyfikacjami systemu.
- **Testowanie integracji systemów** skupia się na interfejsach łączących system podlegający testowaniu z innymi systemami oraz usługami zewnętrznymi. Do testowania integracji systemów niezbędne są odpowiednie środowiska testowe — w miarę możliwości zbliżone do środowiska produkcyjnego.
- **Testowanie akceptacyjne** skupia się na przeprowadzeniu walidacji i wykazaniu, że system jest gotowy do wdrożenia (tzn. zaspokaja potrzeby biznesowe użytkownika). W idealnych warunkach testowanie akceptacyjne powinni przeprowadzać docelowi użytkownicy. Najważniejsze formy testowania akceptacyjnego to: testowanie akceptacyjne przez użytkownika, operacyjne testy akceptacyjne, testowanie akceptacyjne zgodności z umową i testowanie akceptacyjne zgodności z prawem, testowanie alfa oraz testowanie beta.

Typy testów

- **Testowanie funkcjonalne** polega na dokonaniu oceny funkcji, które powinien realizować dany moduł lub system. Funkcje opisują to, „co" powinien robić dany przedmiot testów. Głównym celem testowania funkcjonalnego jest sprawdzenie kompletności funkcjonalnej, poprawności funkcjonalnej oraz adekwatności funkcjonalnej.
- **Testowanie niefunkcjonalne** ma na celu dokonanie oceny atrybutów innych niż charakterystyki funkcjonalne modułu lub systemu. Testowanie niefunkcjonalne pozwala sprawdzić to, „jak dobrze"

zachowuje się dany system. Głównym celem testowania niefunkcjonalnego jest sprawdzenie niefunkcjonalnych charakterystyk jakościowych oprogramowania czyli użyteczności, wydajności, kompatybilności, niezawodności, zabezpieczenia, utrzymywalności, przenaszalności
- **Testowanie czarnoskrzynkowe** (patrz podrozdział 4.2) opiera się na specyfikacjach, a testy wyprowadza się na podstawie dokumentacji zewnętrznej wobec przedmiotu testów. Głównym celem testowania czarnoskrzynkowego jest sprawdzenie zachowania systemu pod kątem zgodności ze specyfikacją.
- **Testowanie białoskrzynkowe** (patrz podrozdział 4.3) ma charakter strukturalny, a testy wyprowadza się na podstawie implementacji lub struktury wewnętrznej danego systemu (np. kodu, architektury, przepływów pracy i przepływów danych). Głównym celem testowania białoskrzynkowego jest uzyskanie akceptowalnego poziomu pokrycia testowego bazowej struktury systemu.

Wszystkie cztery typy testów powyżej można stosować na wszystkich poziomach testów

Testowanie potwierdzające a testowanie regresji

Testowanie potwierdzające ma na celu sprawdzenie, czy pierwotny defekt został pomyślnie usunięty. Zależnie od poziomu ryzyka skorygowaną wersję oprogramowania można przetestować na kilka sposobów, w tym poprzez:

- wykonanie wszystkich przypadków testowych, które wcześniej nie zostały zaliczone z powodu defektu lub
- dodanie nowych testów w celu pokrycia ewentualnych zmian, które były niezbędne do usunięcia defektu.

Jeśli jednak przy usuwaniu defektów brakuje czasu lub środków finansowych, testowanie potwierdzające można ograniczyć do wykonania kroków potrzebnych do odtworzenia awarii spowodowanej defektem i sprawdzenia, czy tym razem ona nie występuje.

Testowanie regresji pozwala sprawdzić, czy wprowadzona zmiana (w tym także poprawka, która była już przedmiotem testowania potwierdzającego) nie spowodowała negatywnych konsekwencji. Konsekwencje takie mogą dotyczyć modułu, w którym wprowadzono zmianę, innych modułów tego samego systemu, a nawet innych podłączonych systemów, w związku z czym testowanie regresji może wykraczać poza przedmiot testów i obejmować również środowisko, w którym się on znajduje.

Zestawy testów regresji są wykonywane wielokrotnie, a liczba związanych z nimi przypadków testowych rośnie z każdą iteracją lub każdą wersją, w związku z czym testowanie regresji świetnie nadaje się do automatyzacji.

2.3 TESTOWANIE PIELĘGNACYJNE I ZDARZENIA JE WYZWALAJĄCE

Pielęgnacja może obejmować między innymi działania naprawcze, działania wynikające z konieczności dostosowania oprogramowania do zmian w środowisku oraz działania mające na celu zwiększenie

wydajności lub utrzymywalności

Zdarzenia wywołujące pielęgnację i testowanie pielęgnacyjne można podzielić na następujące kategorie:

- Modyfikacje. Ta kategoria obejmuje między innymi zaplanowane udoskonalenia (wprowadzane w postaci nowych wersji oprogramowania), zmiany korekcyjne i poprawki doraźne.
- Uaktualnienia lub migracje środowiska produkcyjnego. Ta kategoria obejmuje między innymi przejście z jednej platformy na inną, co może wiązać się z koniecznością przeprowadzenia testów związanych z nowym środowiskiem i zmienionym oprogramowaniem bądź testów konwersji danych (w przypadku migracji danych z innej aplikacji do pielęgnowanego systemu).
- Wycofanie. Ta kategoria dotyczy sytuacji, w której okres użytkowania aplikacji dobiega końca. W przypadku wycofywania systemu może być konieczne przetestowanie archiwizacji danych, jeśli zachodzi potrzeba ich przechowywania przez dłuższy czas. Ponadto jeśli w okresie archiwizacji będzie wymagany dostęp do niektórych danych, może być konieczne przetestowanie procedur przywracania i odtwarzania danych po archiwizacji.

ROZDZIAŁ 3. TESTOWANIE STATYCZNE

3.1 PODSTAWY TESTOWANIA STATYCZNEGO

W przeciwieństwie do testowania dynamicznego testowanie statyczne nie wymaga uruchamiania testowanego oprogramowania. W testowaniu statycznym oceny kodu, specyfikacji procesów, specyfikacji architektury systemu i innych produktów pracy dokonuje się poprzez ich manualne zbadanie (np. w ramach przeglądu) lub poprzez zastosowanie odpowiedniego narzędzia (np. w ramach analizy statycznej).

Analiza statyczna pozwala rozpoznać problemy przed rozpoczęciem testowania dynamicznego, a przy tym jest często mniej pracochłonna, ponieważ nie wymaga tworzenia przypadków testowych i odbywa się zwykle z wykorzystaniem narzędzi.

Typy produktów, które mogą być badane przy użyciu poszczególnych technik testowania statycznego.

Przy użyciu technik testowania statycznego można zbadać niemal wszystkie produkty pracy, np.:

- dokumenty zawierające specyfikacje wymagań,
- kod źródłowy,
- plany testów,
- przypadki testowe,
- pozycje backlogu (ang. product backlog items),

- karty opisu testów,
- dokumentację projektu,
- umowy oraz modele.

Korzyści wynikające z testowania statycznego

Wczesne wykrywanie defektów:

- pozwala na identyfikację defektów już we wczesnych fazach cyklu wytwarzania oprogramowania, dzięki temu możliwe jest szybkie reagowanie i naprawa problemów na etapie, gdy są one jeszcze stosunkowo łatwe do poprawienia.

Identyfikacja defektów niewykrywalnych przez testowanie dynamiczne:

- umożliwia wykrywanie defektów, których nie da się łatwo zidentyfikować podczas testowania dynamicznego, takich jak nieosiągalny kod, niewłaściwie zaimplementowane wzorce projektowe czy defekty w niewykonywalnych produktach pracy.

Ocena jakości produktów pracy:

- umożliwia dokładną ocenę jakości produktów pracy, co pomaga w budowaniu zaufania do nich.

Weryfikacja udokumentowanych wymagań:

- umożliwia interesariuszom upewnienie się, że opisują one ich rzeczywiste potrzeby.

Wspólne zrozumienie i punkt widzenia:

- pozwala zaangażowanym interesariuszom wypracować wspólny punkt widzenia, co ułatwia późniejsze etapy projektu.

Usprawnienie wymiany informacji:

- przyczynia się do usprawnienia wymiany informacji między interesariuszami, co jest kluczowe dla efektywnej komunikacji w zespole projektowym.

Koszty projektu:

- Pomimo kosztów przeglądów, łączne koszty projektu są zwykle niższe niż w przypadku zaniedbania tego procesu. Przeglądy pomagają zmniejszyć czasochłonność i pracochłonność usuwania defektów na późniejszych etapach projektu.

Efektywniejsze wykrywanie defektów kodu:

- Analiza statyczna pozwala efektywniej wykrywać defekty kodu niż testowanie dynamiczne, co przekłada się na zmniejszenie liczby tego rodzaju defektów i obniżenie ogólnych nakładów pracy związanych z wytwarzaniem oprogramowania.

Testowanie statyczne i dynamiczne

STATYCZNE	DYNAMICZNE
Zarówno testowanie statyczne, jak i testowanie dynamiczne może prowadzić do wykrycia defektów, jednak istnieją pewne rodzaje defektów, które można wykryć tylko jedną z powyższych metod.	
umożliwia bezpośrednie wykrywanie defektów	powoduje występowanie awarii, które są następnie analizowane w celu zidentyfikowania związanych z nimi defektów
Testowanie statyczne pozwala łatwiej wykryć defekty, które znajdują się na rzadko wykonywanych ścieżkach w kodzie lub w miejscach trudno dostępnych podczas testowania dynamicznego.	
można stosować do niewykonywalnych produktów pracy	tylko do produktów wykonywalnych
może służyć do mierzenia charakterystyk jakościowych, które nie są zależne od wykonywania kodu (takich jak utrzymywalność),	**może służyć do mierzenia charakterystyk jakościowych zależnych od wykonywania kodu (takich jak wydajność)**

3.2 INFORMACJE ZWROTNE I PROCES PRZEGLĄDU

Korzyści wynikające z wczesnego i częstego otrzymywania informacji zwrotnych od interesariuszy

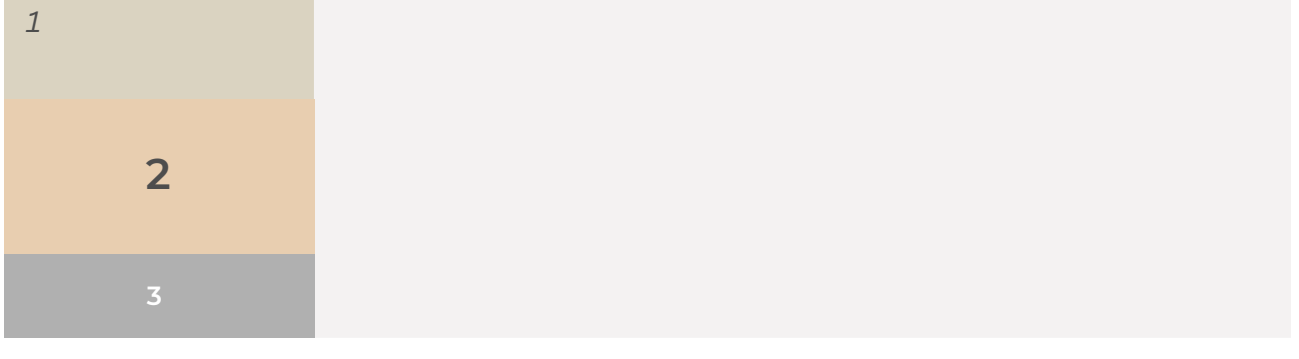
Czynności wykonywane w ramach procesu przeglądu

Obowiązki są przypisane do najważniejszych ról w trakcie wykonywania przeglądów

Typy przeglądów

Czynniki decydujące o powodzeniu przeglądu

1	2	3



2

1

1.1 Co to jest testowanie?

1.1.1 Typowe cele testów

1.1.2 Różnica między debugowaniem a testowaniem

”

2

1 **2**

1
2
3

1	2	3	4
a			
b			
c			