1.1 Co to jest testowanie?

Testowanie to proces oceny oprogramowania w celu identyfikacji błędów, zapewnienia zgodności z wymaganiami oraz zapewnienia jakości końcowego produktu. Polega na uruchamianiu systemu lub jego części w celu zidentyfikowania różnic między rzeczywistymi a oczekiwanymi wynikami.

- ważne – testy nigdy się nie kończą! Defekty zawsze występują!

1.2 Dlaczego testowanie jest niezbędne?

Testowanie jest kluczowe, aby:

- **Wykrywać i zapobiegać błędom w oprogramowaniu, które mogą prowadzić do awarii.**
- **Zapewniać jakość i zgodność z wymaganiami.**
- **Chronić przed stratami finansowymi, reputacyjnymi oraz prawnymi.**

1.3 Zasady testowania

- **Testowanie pokazuje obecność błędów, a nie ich brak.**
- **Wczesne testowanie oszczędza czas i pieniądze.**
- **Testowanie zorientowane na kontekst – różne podejścia dla różnych projektów.** ***(inaczej testujemy aplikacje webową dla kolei a inaczej dla szpitala! Tu też mówiliśmy o DDD – Domain Driven Design/Development czyli używanie języka związanego z aplikacją, którą testujemy!)***
- **Paradoks pestycydów – te same testy przestają wykrywać nowe błędy, więc muszą być regularnie przeglądane i aktualizowane.**
- ***(Czyli często zawężamy myślenie do aktualnego stanu aplikacji i jak ona działa TERAZ, przez co nasze testy nie wykryją błędów, które pojawią się w przyszłości!!)***
- **Testowanie zależy od jakości wstępnych informacji – brak błędów w wymaganiach, projektach itp.**
- ***(Tak, wymagania i User Stories też mogą mieć błędy, które testerzy mogą wychwycić już na etapie statycznej analizy wymagań!)***

1.4 Czynności testowe, testalia i role związane z testami

Czynności testowe obejmują: planowanie, projektowanie testów, wykonanie testów, raportowanie wyników i zamknięcie testów. Testalia to dokumentacja

związana z testowaniem, np. plany testów, przypadki testowe, raporty z testów. Role obejmują testerów, liderów testów, analityków testów i menedżerów testów.

- ***(Dużo teorii i słownictwa, a w praktyce – dostajesz wymaganie, robisz z niego (albo cały zespół robi) Test Suit i kilka Test Case'ów. Wykonujesz testy, sprawdzasz wyniki, zapisujesz je w systemie, koniec.***

1.5 Niezbędne umiejętności i dobre praktyki w dziedzinie testowania

Kluczowe umiejętności to: analityczne myślenie, znajomość narzędzi testowych, umiejętność komunikacji oraz znajomość procesu wytwarzania oprogramowania. Dobre praktyki obejmują: tworzenie testów wcześnie w cyklu życia projektu, regularne przeglądy testów oraz ciągłe doskonalenie umiejętności testowych.

- ***(tzw. Umiejętności miękkie, w pracy testera musisz być komunikatywny, otwarty na współpracę z programistami i być cały czas nieufny jeśli chodzi o program, który testujesz)***

2.1 Testowanie w kontekście cyklu wytwarzania oprogramowania

Testowanie integruje się z każdym etapem cyklu życia oprogramowania, od analizy wymagań przez implementację, aż po wdrożenie. Jest nieodłącznym elementem takich modeli jak Agile, Waterfall.

- ***(Czyli każda firma ma swój system testowania. W Agile – Scrum też testuje się w sprintach, ale każda firma inaczej organizuje zespoły. Czasami tester jest członkiem zespołu programistów, a czasami jest osobny zespół testerów, który sobie skacze pomiędzy zespołami)***

2.2 Poziomy testów i typy testów

- **Poziomy testów: jednostkowe, integracyjne, systemowe, akceptacyjne.**
- **Typy testów: funkcjonalne, niefunkcjonalne (np. wydajnościowe), regresyjne, testy bezpieczeństwa, użyteczności.**

2.3 Testowanie pielęgnacyjne

Testowanie pielęgnacyjne to testowanie oprogramowania po jego modyfikacji, np. po naprawie defektów, wprowadzaniu nowych funkcji lub migracji systemu. Kluczowe jest tu testowanie regresyjne, które sprawdza, czy modyfikacje nie wpłynęły na istniejącą funkcjonalność.

- *(Nikt nigdy tego tak nie nazywa, ewentualnie mówi się o Maintenance testów)*

3.1 Podstawy testowania regresyjnego

Testowanie regresyjne polega na ponownym uruchamianiu wcześniej przeprowadzonych testów po zmianach w kodzie, aby upewnić się, że nowe zmiany nie wprowadziły nowych defektów.

- *(Tyczy się głównie testów automatycznych uruchamianych z poziomu jakiegoś programu – poczytaj o Selenium, Cypress itd.)*
- **https://www.youtube.com/watch?v=dwVfZI-49bg&t=1s**

3.2 Informacje zwrotne i proces przeglądu

Przeglądy to systematyczne oceny artefaktów (np. kodu, dokumentacji) przez zespół, mające na celu wykrycie błędów na wczesnym etapie. Informacje zwrotne z przeglądów pomagają w doskonaleniu procesu wytwarzania i testowania.

- *(Czyli tzw. Review. Po każdym sprincie można przeanalizować, czy nasze testy i ich wyniki, oraz wymagania i user stories na których bazujemy są poprawne)*

4.1 Ogólna charakterystyka technik testowania

Techniki testowania pomagają w projektowaniu skutecznych przypadków testowych. Dzielą się na techniki oparte na specyfikacji! (czarnoskrzynkowe), na strukturze (białoskrzynkowe) oraz na doświadczeniu testera.

4.2 Czarnoskrzynkowe techniki testowania

Techniki te, jak klasy równoważności i analiza wartości brzegowych, opierają się na analizie specyfikacji funkcjonalnych bez znajomości kodu źródłowego.

- *Klasy równoważności:*
    - *Dane które podajemy systemowi*
    - *Dzielimy na poprawne i niepoprawne (absurdalne)*
    - *Przykład danych do testów:* https://www.youtube.com/watch?v=I0NGCKOeT1E
- **Wartości brzegowe**

    - **Wartości na granicach akceptowanego zbioru wartości, czyli jeśli system akceptuje wartości (-oo, 20), (21,100), (100, oo) to wartości brzegowe to 19,20,21,22,99,100,101 (zobacz sobie przyukłady z czatemGPT)**

4.3 Białoskrzynkowe techniki testowania

- ***Kod w zapisie algorytmicznym pokazuje, ile jest możliwych wariantów (ścieżek) którymi może podążyć logika programu***
- ***Często jej zapis wygląda jak odwrócone drzewko***
- ***Dlatego każda ścieżka, którą może podążyć logika kodu to tzw. „gałąź”:***
- **https://pl.wikipedia.org/wiki/Schemat_blokowy**
- ***W testach whitebox często mówi się o pokryciu gałęzi i instrukcji kodu***
    - ***Pokrycie gałęzi – czy każda logiczna ścieżka ma test***
    - ***Pokrycie instrukcji – czy każda linia kodu ma test***
- ***To głównie wiedza dla programistów!***

Techniki białoskrzynkowe, jak pokrycie kodu lub ścieżek, wykorzystują znajomość struktury wewnętrznej oprogramowania do tworzenia przypadków testowych.

4.4 Techniki testowania oparte na doświadczeniu

Opierają się na intuicji i doświadczeniu testera. Przykłady to eksploracyjne testowanie i ad-hoc. Tester samodzielnie projektuje i wykonuje testy bez formalnej specyfikacji.

- ***Nic ciekawego i oczywista oczywistość :)***

4.5 Podejścia do testowania oparte na współpracy

Polega na współpracy testerów z programistami, analitykami i klientami. Zespoły współdzielą wiedzę, co pozwala na szybsze wykrycie i naprawienie błędów.

- ***Nic ciekawego i oczywista oczywistość :)***

5.1 Planowanie testów

Planowanie testów obejmuje definiowanie strategii testowej, zasobów, harmonogramu i ryzyk. Dobry plan testów określa, co będzie testowane, w jaki sposób i przez kogo.

- ***Bierzemy wygmaganie lub user stories i na jej podstawie przygotowujemy Test Suites i Test Cases. Dodatkowo zespół szacuje, ile czasu zajmą takie testy i kto z zespołu będzie je robił (i kiedy)***

5.2 Zarządzanie ryzykiem

W testowaniu ryzyko dotyczy zarówno prawdopodobieństwa wystąpienia defektu, jak i jego potencjalnych konsekwencji. Zarządzanie ryzykiem polega na identyfikacji, analizie i minimalizowaniu ryzyk poprzez odpowiednie priorytetyzowanie testów.

- ***Ogólnie, w IT wszystko wiąże się z ryzykiem. Niektóre rzeczy są od nas zależne, niektóre nie. Np. jeśli mamy system który wymaga dobrego i szybkiego internetu, ryzykiem jest co kiedy go nie będzie i jak wtedy zachowa się system. Jako Tester lub QA Lider musimy te sytuacje wyłapać i podkreślać ich istnienie programistom i managerom***

5.3 Monitorowanie testów, nadzór nad testami i ukończenie testów

Procesy te obejmują śledzenie postępu testów, kontrolę jakości wykonania testów oraz formalne zakończenie testowania, gdy cele testowe zostały osiągnięte.

- ***Testy automatyczne są odpalane w Jenkinsie i widzimy ich stan i progres***
- ***Testy manualne – trzeba ich wyniki i stan gdzieś zapisywać, każda firma ma swój system***
- ***Gdy dopisujemy nowe testy, musimy wiedzieć ile nowych wymagań zostało pokrytych testami***

5.4 Zarządzanie konfiguracją

Zarządzanie konfiguracją to proces kontrolowania zmian w oprogramowaniu, aby zapewnić, że testy są wykonywane na poprawnych wersjach systemu oraz że artefakty testowe są odpowiednio zarządzane.

- ***Tester musi potrafić skonfigurować system do testów i móc przetestować go dla kilku konfiguracji i wersji***

5.5 Zarządzanie defektami

Proces zarządzania defektami obejmuje rejestrowanie, analizę, priorytetyzację i śledzenie błędów od momentu ich zgłoszenia aż do zamknięcia. Kluczowe jest efektywne zarządzanie defektami, aby minimalizować ich wpływ na jakość końcowego produktu.

- ***Czyli w bugtrackerze jak zapisujemy defekt (buga) w bugtrackerze (aplikacji do zapisywania i śledzenia błędów) to wypełniamy odpowiednio formularz pamiętając o poprawnym:***
    - ***Opisie***
    - ***Priorytecie***
    - ***Poważności (severity)***
    - ***Krokach do odtworzenia błędu***
    - ***Wersji gdzie błąd został znaleziony***
    - ***Powiązaniu z UserStory lub wymaganiem (czasami o tym się zapomina)***

6.1 Narzędzia wspomagające testowanie

Narzędzia te wspierają różne aspekty testowania, od zarządzania przypadkami testowymi, przez automatyzację testów, po analizę pokrycia kodu. Ułatwiają zarządzanie testami, ich wykonanie oraz raportowanie wyników.

- ***Czyli Postman, devtools, itd. Każdy tester w miarę jak pracuje powiększa swój zasób narzędzi których lubi i potrafi używać***
- ***Do automatyzacji i analizy pokrycia używa się narzędzi które konfiguruje dla nas DevOps lub administrator***

6.2 Korzyści i ryzyka związane z automatyzacją testowania

Automatyzacja testów pozwala na szybkie i wielokrotne uruchamianie testów, co zwiększa efektywność i pokrycie testowe. Jednakże wymaga początkowej inwestycji czasu i zasobów, a także odpowiedniej strategii, aby była opłacalna. Ryzykiem jest automatyzowanie testów, które są podatne na częste zmiany i mogą wymagać ciągłego dostosowywania.

- ***Korzyści są oczywiste – Tester manualny nie jest w stanie przeklikać codziennie paruset przypadków testowych. To co się da trzeba automatyzować, żeby nasz produkt miał jak najwięcej testów a tester manualny mógł skupić się na ważniejszych testach***

Dodatkowe INFO z dzisiaj:

- Quality Assurance – zbiór technik związanych z zarządzaniem i zapewnieniem jakości systemu

- Testowanie to tylko jeden z jego elementów

- Left Shift - testy, które wyłapują najwięcej błędów jak najszybciej powinny być wykonywane najpierw, żeby zaoszczędzić czas

- Najlepsza kolejność zazwyczaj:

    - Statyczna analiza kodu (czytamy kod)

    - Testy jednostkowe

    - Testy Integracyjne

    - Testy Systemowe

    - Testy wydajnościowe i akceptacyjne

- Testy potwierdzające – wykonuje tester, gdy programista twierdzi, że naprawił buga i trzeba to sprawedzić
- Śledzenie powiązań pomiędzy testami

    - musimy być w stanie powiedziećm które testy pokrywają (testują) które wymagania

    - Niektóre kluczowe testy należą do najważniejszycz czynników mówiązych o tym, że nasz system działa (tzw. Kejpiaje😊 czyli KPI(key performance indicator)

- Niezależnośc testowania – tester powinien być niezależny od nacisków zespołu i managerów gdy coś testuje. Powinien też (najlepiej) podlegać innemu managerowi albo być z innego działu, by nie następowała żadna zależność między programistami a testerami (w praktyce nierealne)

- Testowanie w kontekście modelu cyklu wytwarzania oprogramowania

    - Każda firma ma swój sposób organizowania testów i programowania

    - W zwinnych metodykach jak SCRUM testerzy też często pracują w sprintach, ale czasami są to inne sprinty niż zespołów programistycznych

- BDD (Behavioral Driven Design/Development) – Testy, w których główny nacisk kładzie się na testowaniu ZACHOWANIA się systemu lub danej funkcjonalności

    - Piszemy używając notacji GIVEN – WHEN – THEN

    - testy mają formę opisu zachowania się systemu

    - Np.

        (G) Użytkownik po zalogowaniu się,

        (W) Klika w przycisk „Drukuj”

        (T) wtedy pojawia się okienko z opcjami do wydruku

- ATDD (Acceptance Test Driven Design/Development) – Praca sterowana Testami akceptacyjnymi

    - Naszym głównym celem jest zawsze zadowolenie wymagań do testów akceptacyjnych

    - Zawsze staramy się widzieć "Duży obrazek” i skupić się na całym systemie i jaką funkcję ma spełniać

- TDD – Test Driven Development – Wytwarzania (kodu, specyfikacji) Sterowane Testami

    - Najpierw piszemy testy a dopiero później piszemy kod albo tworzymy produkt

    - Pomaga zminimalizować ilość rzeczy, które robi programista, bo patrzy on tylko na testy a nie na własną interpretację co, i jak miał zrobić

- DevOps – Osoby odpowiedzialne za konfigurację narzędzi do odpalania testów i mierzenia pokrycia oraz wielu innych przydatnych narzędi

    - Konfigurują i utrzymują Jenkinsa

    - Mogą konfigurować i przygotowywać środowiska testowe dla testerów

- Retrospektywa (Retro)

    - spotkanie, gdzie zespół omawia problemy związane z ich codzienną pracą

    - problemy z zespołem, narzędziami lub firmą

- Wymagania/Testy – funkcjonalne i niefunkcjonalne

    - Wymagania i testy funkcjonalne zajmują się opisaniem i przetestowaniem zachowań systemu i sprawdzenia, czy to co ma robić – robi dobrze

    - Niefunkcjonalne – służa do opisu i testowania rzeczy intuicyjnych, ciężkich do opisania jednoznacznie i matematycznie

- np. wydajność, „Look and Feel”, User Experience I tego typu nowoczesne określenia😊

- Ogólne wrażenie z korzystania i przydatności systemu

- Testowanie statyczne

- testujemy „na sucho” bez uruchamiania systemu

- analizujemy dokumentacje, TestCase’y, wymagania itd.

- możemy analizować nierealistyczne scenariusze i testować hipotetyczne sytuacje, które nie występują w rzeczywistości

- Testowanie dynamiczne

- Uruchamiamy system i testujemy jego realne zachowanie

- Testowanie przejść pomiędzy stanami

- Tworzymy Graf możliwych stanów systemu i testujemy czy system odpowiednio się po nim porusza

Testowanie przejść pomiędzy stanami *(ang.* ***State Transition Testing****) to technika testowania, która polega na sprawdzaniu, jak system reaguje na różne sekwencje zdarzeń, które prowadzą do zmian jego stanów. Jest szczególnie przydatna w przypadku systemów, które mają zdefiniowane różne stany (np. "zalogowany", "wylogowany", "czekający na weryfikację") i przechodzą między tymi stanami w odpowiedzi na działania użytkownika lub inne zdarzenia.*

Podstawowe pojęcia:

1. ***Stan****:*
    - *To konkretna sytuacja lub konfiguracja systemu w danym momencie. Na przykład, dla aplikacji bankowej stany mogą obejmować: "zalogowany użytkownik", "sesja wygasła", "oczekiwanie na autoryzację".*
2. ***Przejście****:*
    - *To zmiana z jednego stanu na inny w odpowiedzi na określone zdarzenie lub akcję. Na przykład, kliknięcie przycisku "Wyloguj" powoduje przejście ze stanu "zalogowany" do stanu "wylogowany".*
3. ***Zdarzenie****:*
    - *To akcja, która powoduje przejście z jednego stanu do innego. Może to być działanie użytkownika, takie jak kliknięcie przycisku, lub wydarzenie systemowe, jak upływ czasu.*
4. ***Warunek****:*
    - *To reguły lub kryteria, które muszą być spełnione, aby przejście mogło się odbyć. Na przykład, użytkownik musi być zalogowany, aby mógł wylogować się z aplikacji.*

Graf przejść stanów (State Transition Diagram)

Graf przejść stanów *(ang.* ***State Transition Diagram****), często nazywany* ***STG*** *(State Transition Graph), to graficzna reprezentacja stanów systemu i przejść między nimi.*

Jak to wygląda:

- ***Węzły (okręgi)*** *reprezentują różne stany, w których może znajdować się system.*
- ***Łączniki (strzałki)*** *między węzłami reprezentują przejścia, które mogą zachodzić w wyniku określonych zdarzeń.*
- ***Etykiety*** *na łącznikach często opisują, jakie zdarzenie powoduje przejście.*

Przykład:

Załóżmy, że mamy prostą aplikację do logowania. Może ona mieć następujące stany:

1. ***Stan początkowy****: "Ekran logowania".*
2. ***Stan środkowy****: "Zalogowany".*
3. ***Stan końcowy****: "Wylogowany".*

Przejścia *między tymi stanami mogą być następujące:*

- *Z "Ekranu logowania" do "Zalogowany" na podstawie poprawnych danych logowania.*
- *Z "Zalogowany" do "Wylogowany" po kliknięciu przycisku wylogowania.*

Graf przejść stanów może wyglądać tak:

- *Okrąg "Ekran logowania" połączony strzałką z okręgiem "Zalogowany" z etykietą "Poprawne dane logowania".*
- *Okrąg "Zalogowany" połączony strzałką z okręgiem "Wylogowany" z etykietą "Kliknięcie przycisku wylogowania".*

Zastosowanie testowania przejść pomiędzy stanami:

Testowanie przejść pomiędzy stanami polega na sprawdzeniu, czy system poprawnie przechodzi między stanami w odpowiedzi na różne zdarzenia. W praktyce testy obejmują sprawdzenie:

- *Czy system prawidłowo reaguje na zdarzenia prowadzące do zmiany stanu.*
- *Czy system pozostaje w odpowiednim stanie po wystąpieniu zdarzenia.*
- *Czy nieautoryzowane przejścia (takie, które nie powinny mieć miejsca) są blokowane.*

Przykład testu:

1. ***Test pozytywny****: Użytkownik na ekranie logowania wprowadza poprawne dane i przechodzi do stanu "Zalogowany".*
2. ***Test negatywny****: Użytkownik na ekranie logowania wprowadza niepoprawne dane, system nie przechodzi do stanu "Zalogowany", pozostając na "Ekranie logowania".*