Testowanie i walidacja oprogramowania

Informacje wstępne – zaliczenie

- Ogólne warunki zaliczenia

  - obecność na zajęciach (minimum 50%)

  - przygotowanie własnej prezentacji:

  a) związana z testowaniem i walidacją – własne doświadczenia

  b) wskazanie użyteczności wykorzystanych narzędzi (narzędzia)

  c) podsumowanie

  - utworzenie recenzji jednego z narzędzi (środowiska) do testowania oprogramowania

Informacje wstępne - zaliczenie

- test z pytaniami zamkniętymi i otwartymi sprawdzający wiedzę (< 50% obecności oznacza obowiązkowy test)

- tworzenie aplikacji i wskazanie użyteczności testowania i walidacji w swoim projekcie (projekt może być realizowany w ramach innych zajęć w dowolnym języku programowania/technologii i modelu życia aplikacji)

Informacje wstępne - zaliczenie

- Szczegółowe warunki zaliczenia:

  - obecność/test oraz przygotowanie prezentacji/recenzji narzędzia – ocena dostateczna bądź dobra

  - tworzenie własnego projektu z uwzględnieniem opisu testowania i walidacji – ocena bardzo dobra

- pod uwagę brana jest także aktywność (nawet wyższa ocena jednak trzeba spełnić punkt obecności)

Czym jest testowanie oprogramowania

- Testowanie to sprawdzanie produktu bądź elementów produktu pod kątem wykonania, spełniania norm, funkcjonalności, trwałości, zgodności z pozostałymi częściami zestawu, wykonania instrukcji obsługi itd.
- Testowanie jako proces występuje w wielu gałęziach przemysłu i biznesu
- Na ogół procesowi testowania poddawany jest element bądź szereg elementów składających się na końcowy produkt
- W procesie wytwarzania oprogramowania testowanie ma bardzo ważne znaczenie – pozwala określić wady i zalety zastosowanej funkcjonalności, prawdziwą użyteczność zaimplementowanych rozwiązań, błędne działanie rozwiązań w związku z różnymi sytuacjami problemowymi i użytkowymi itp..

Czym jest walidacja

- Walidacją (z ang. Validate) określa się proces sprawdzenia:

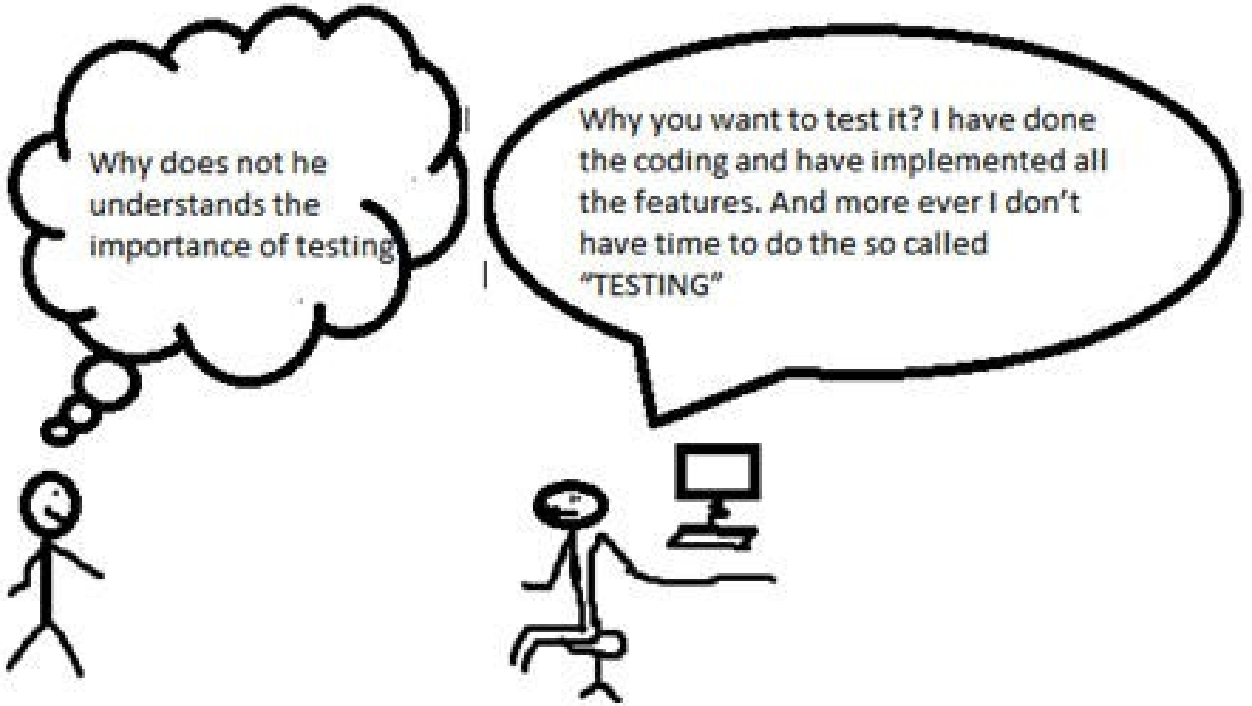
  - zgodności tworzonego oprogramowania z wytycznymi klienta

  - sposobu implementacji użytych algorytmów i fragmentów gotowych kodów w całokształcie aplikacji

  - zgodności z używanymi technologiami i oprogramowaniem współpracującym

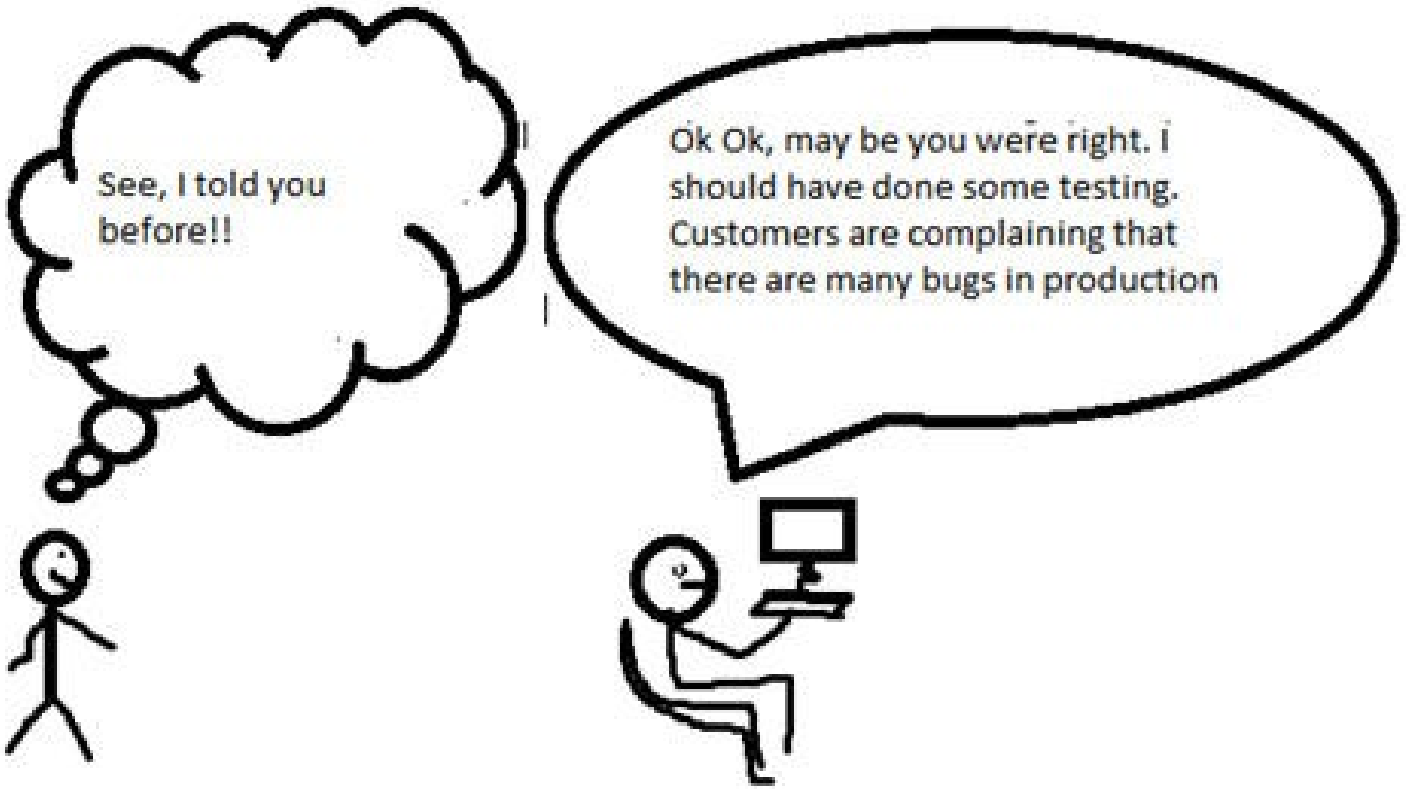
Krótka historia cyklu testowania oprogramowania

- Szczęśliwe lata 60-90 XX wieku



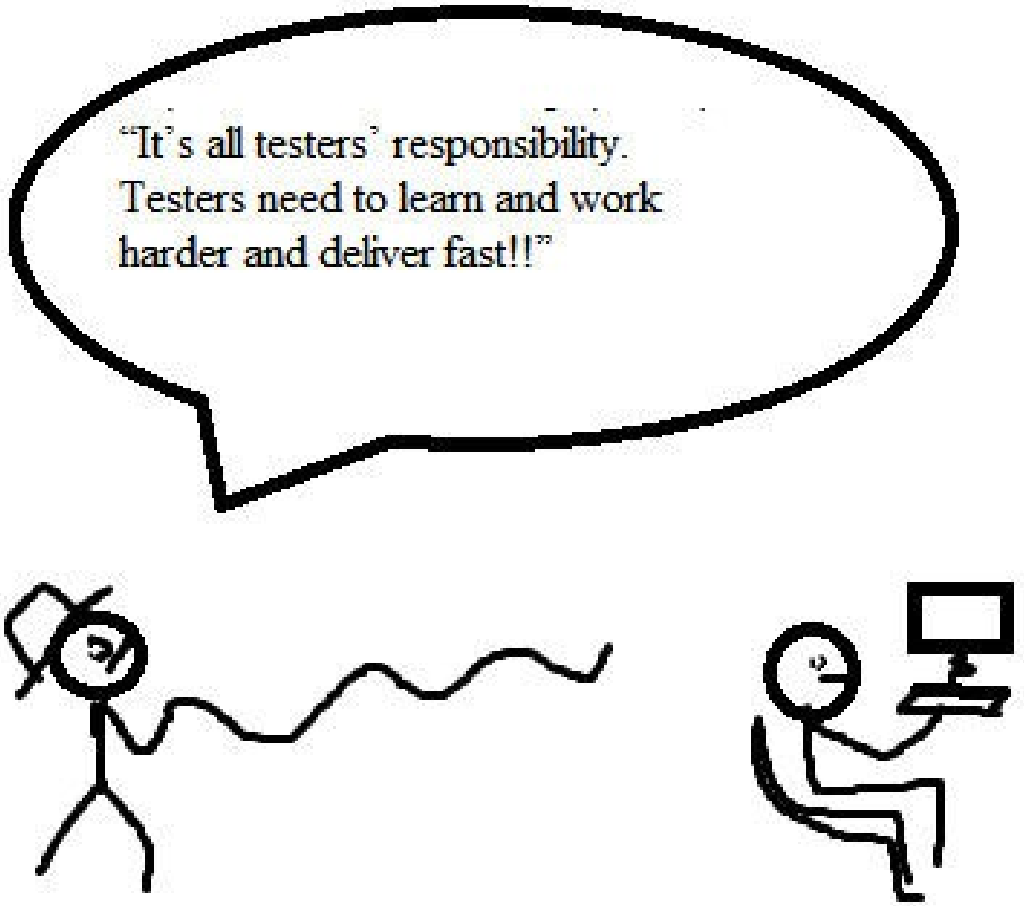
Krótka historia cyklu testowania oprogramowania

- Lata 90 XX wieku



Krótka historia cyklu testowania oprogramowania

- Obecnie (aby na pewno?)



Cykl życia oprogramowania

- Cyklem życia zwykło nazywać się zjawisko przemiany badanego elementu z jednej formy do kolejnej
- Przykładem elementu może być roślina. Jej cyklem życia będzie proces przeistoczenia się z nasiona do postaci rozwiniętej. Następnym etapem będzie produkcja nasion (kwitnięcie), a końcowym obumarcie.

Cykl życia oprogramowania

- Analogicznie elementem badanym może być tworzony przez nas projekt informatyczny/aplikacja.
- Szczególnym przypadkiem cyklu życia oprogramowania jest etap wykonywania i testowania poprawności działania wykonanego projektu. Etap ten, ze względu na unormowany i systematyczny charakter został ujęty w osobnym etapie – cyklu życia testowania i walidacji.

Cykl życia testowania oprogramowania

- Etap precyzowania wymagań
- Etap planowania
- Etap analizy
- Etap projektowania
- Etap wdrożenia
- Etap wykonywania (użytkowania)
- Etap podsumowania
- Etap zamknięcia

Precyzowanie wymagań

- Przeglądanie specyfikacji projektowej pod kątem wyszukiwania kluczowych wymagań
- Analiza wymagań pod kątem ich testowania (nadawania się do testowania)
- Ustalenie zakresu testowania
- Wskazana praca zespołowa

Planowanie

- Ustalanie aktywów i zasobów przydatnych do wykonania założeń procesu testowania
- Ustalenie metryk testowania, sposobów ich pozyskiwania oraz śledzenia
- Ustalenie strategii testowania
- Analiza ryzyka

Analiza

- Poziom i zagnieżdżenie procesu testowania
- Złożoność projektu
- Ryzykowność testowanego projektu
- Zaangażowanie w cykl życia rozwoju oprogramowania
- Zarządzanie testowaniem
- Określenie możliwości zespołu testerów
- Relacje z grupą docelową

Projektowanie

- Uszczegółowienie warunków testowania
- Zdobycie zbiorów testowych
- Wykonanie środowiska testowego
- Stworzenie wymagań dotyczących metryk śledczych
- Stworzenie metryk zakresu testu

Wdrożenie

- Tworzenie przypadków użycia (przypadków testów)
- Nadawanie priorytetów poszczególnym przypadkom
- Ustalenie testów regresyjnych (pojęcie regresji)
- Ustalenie możliwych testów skryptowych (automatycznych)
- Sprawdzenie czy przypadki użycia/testu są poprawne i realne

Wykonanie/Użytkowanie

- Właściwa faza testowania.
- Polega na użytkowaniu aplikacji i testowaniu jej pod kątem wyłonionych wcześniej kryteriów
- Należy odnotowywać każdy błąd i rozbieżność od oczekiwanego efektu
- Należy także na bieżąco sprawdzać postęp testów poprzez śledzenie utworzonych metryk

Podsumowanie

- Właściwie etap raportowania (podsumowanie testów)
- Ma charakter cykliczny (naprzemienny z testami użytkowania)
- Raportowanie działania rozwiązania może przybrać wariant dzienny, tygodniowy, miesięczny itp. (zależy od ustaleń wewnątrz zespołu oraz uzgodnień z ostatecznymi odbiorcami produktu)
- Przesyłane raporty powinny mieć treść dopasowaną do odbiorcy (różne raporty dla zespołu oraz odbiorcy)

Zamknięcie

- Sprawdzenie kompletności przypadków testów
- Sprawdzenie czy wszystkie przypadki zakończone błędami/uszkodzeniami zostały poprawione/zamknięte
- Sporządzenie odpowiedniej dokumentacji zawierającej informacje o wdrożonych poprawkach, usprawnieniach, wykrytych i usuniętych usterkach itp.

Metryki i miary w testowaniu oprogramowania

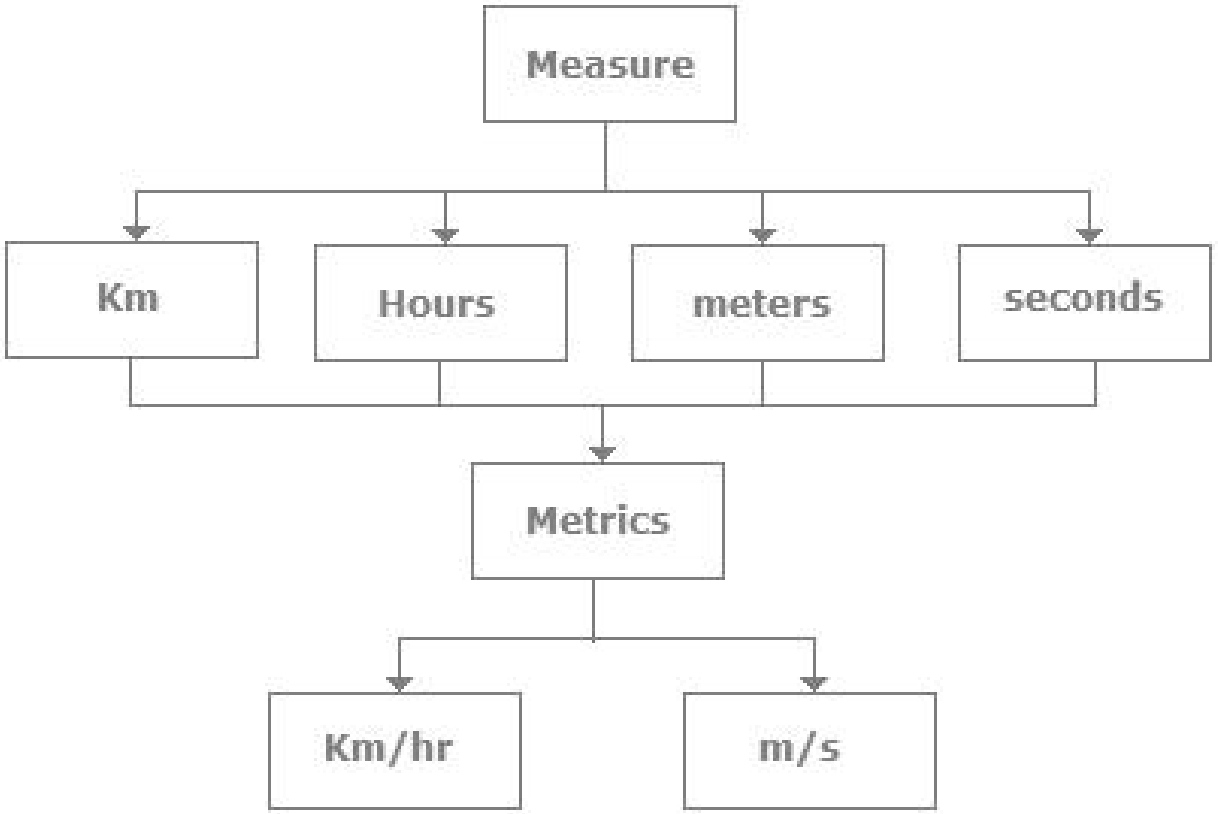
Po co mierzyć testy?

- Brak odnośnika jakości
- Nieokreślona efektywność projektu
- Brak wyceny kosztów
- Brak kontroli nad przebiegiem testów
- Brak wytworzonych standardów końcowych

Czym jest metryka

- Ilościowy pomiar stopnia wykonania wskazanych elementów systemu
- Metryka jest jednostką pomiarową!
- Przykładem metryki może być ilość błędnych linii kodu na wskazaną fragment pomiarowy (popularne jest branie wycinka 1000 linii – Defect per KLOC)
- Inne metryki związane są np. z węzłami (aplikacje sieciowe), wymiarami (bazy SQL), czasem (szybkość wykonywania względem zadania) itp.

Przykład miar i metryk



Cykl życia metryk

- Analiza → rozpoznanie zapotrzebowania na metryki, tworzenie metryk
- Rozpowszechnienie → przekazanie metryk dla testerów oraz klientów docelowych, objaśnienie każdej metryki (co sobą reprezentuje, jakie znaczenie ma w teście itp.)
- Rozwój → pozyskiwanie i weryfikacja danych, wyliczanie wartości metryk na podstawie danych
- Raport → sporządzenie raportu efektywności, udostępnienie wyników zainteresowanym, oczekiwanie na ustosunkowanie się do wyników/odpowiednia reakcja na informacje zwrotne

Przykład miar w testach

LP.	Nazwa miary	Otrzymane dane z pomiarów/ilość
1	Liczba wymagań	5
2	Średnia liczba sporządzonych testów na wymaganie	20
3	Liczba wszystkich testów na wszystkie wymagania	100
4	Wykonanych wszystkich przypadków	65
5	Testy zakończone pozytywnie	30
6	Testy zakończone negatywnie	26
7	Testy zablokowane przy wykonaniu	9
8	Testy niewykonywalne	35
9	Liczba znalezionych uszkodzeń	30
10	Uszkodzenia krytyczne	6
11	Uszkodzenia o stopniu wysokim	10
12	Uszkodzenia o stopniu średnim	6
13	Uszkodzenia o stopniu niskim	8

Metryki uzyskiwane z ustalonych miar

- Procent wykonanych testów – pozwala na ustalenie ile procent zaprojektowanych testów zostało uruchomionych (wykonanych)
- Procent nie wykonanych testów – określa ilość zaprojektowanych testów, których nie udało się uruchomić
- Procent testów wykonanych poprawnie – skuteczność w pełni wykonanych zaprojektowanych przypadków testowych, które nie spowodowały błędów
- Procent testów wykonanych zakończonych błędem
- Procent testów, których wykonanie zostało zablokowane

Metryki uzyskiwane z ustalonych miar

- Zagęszczenie błędów na wskazanym rozmiarze testowanego oprogramowania
- Efektywność usuwania błędów (DRE) – określa efektywność testów wewnętrznych z wynajdywaniem błędów przez użytkowników systemu.
- Wycieki błędów
- Klasyfikacja pojawiających się błędów podczas testów (procent błędów względem ich wagi)

Cel pomiarów i metryk

- Kontrola postępu
- Ocena obecnych błędów
- Opłacalność kolejnych testów
- Ryzyko z dopuszczenia oprogramowania w obecnym kształcie do fazy tzw. produkcyjnej
- Możliwość zmian wymagań w przyszłej procedurze testowej
- Możliwość zmian w środowisku testowym pod kątem przyszłych testów

Metryka pokrycia kodu testami

- Stosunek liczby testowanych części kodu do całkowitej liczny części kodu
- Częściami kodu mogą być: wyrażenia regularne, gałęzie kodu, linie kodu, funkcje, obiekty, moduły, stany
- Wskazuje na procentową ilość przetestowanego kodu
- Zastosowanie metryki – wady i zalety

Metody testowania

Priorytety testowania

- Nie da się przetestować oprogramowania w 100%
- Odpowiedni wybór metody testującej ma znaczący wpływ na jakoś i funkcjonalność testowanego systemu
- Nie wszystkie uszkodzenia kodu są sobie równoważne!
- Dokładnie testowanie wszystkich aspektów powoduje powiększanie liczby przypadków testowania, a tym samym wydłuża czas testów
- To może spowodować nieopłacalność przedsięwzięcia

Priorytety testowania

- Złożona funkcjonalność
- Regresja
- Optymalna liczba testerów
- Optymalizacja czasu testów do poprawy kodu/ponownych testów

Odpowiednie założenia przy projekcie testowania

- Który wariant cyklu życia oprogramowania będzie najkorzystniejszy do tworzonego programu/systemu?
- Jak wcześnie można rozpocząć testy niektórych aspektów projektu (wcześniej ↔ lepiej)
- Które z modułów będą potencjalnie zawierały najwięcej błędów?
- Które z funkcjonalności są krytyczne?
- Które wymagania funkcjonalne będą stwarzać najwięcej problemów?
- Która funkcjonalność może być krytyczna dla opłacalności projektu?
- Które części kodu były tworzone pod presją czasu (założenie po zakończeniu prac)?
- Które funkcjonalności są najważniejsze dla klienta?

Odpowiednie założenia przy projekcie testowania

- Które miary są najlepsze do zastosowania w tworzonym projekcie (pokryją największą część kodu)?
- Które przypadki wykonania testów dały największy współczynnik wykrywalności błędów (założenie po ukończeniu I fazy testów)?
- Które z założeń projektu mogą stwarzać największe problemy przy pielęgnacji aplikacji?
- Które aspekty tworzonego kodu w poprzednim projekcie powodowały najwięcej problemów?

Sposoby testowania kodu na poziomie kodu

Testy jednostkowe

- Testom podlegają pojedyncze jednostki oprogramowania
- Jednostką są najmniejsze, mogące podlegać testom części oprogramowania
- Cechą charakterystyczną jednostki jest posiadanie PRZYNAJMNIEJ jednego parametru wejściowego oraz jednego (rzadziej kilku) wartości wyjściowej

Testy jednostkowe

- Jednostką w programowaniu proceduralnym może być procedura, funkcja, podprogram lub cały program (odpowiednio mały)
- W OOP (Object-Oriented Programming) jednostką będzie prawdopodobnie metoda należąca do klasy, super klasy, interfejsu/klasy wirtualnej/klasy abstrakcyjnej, klasy pochodnej/dziedziczącej
- Błędem jest traktowanie pojedynczego modułu kodu obiektowego (zawierającego wiele metod będących bazą dla testów jednostkowych)

Testy jednostkowe – definicja ISTQB (International Software Qualifications Board)

- Testowanie odbywa się metodą białego pudełka/skrzynki (white box)
- Jednostkowe testowanie musi nastąpić przed testami integralności (całości oprogramowania)
- Testów dokonują przeważnie sami projektanci kodu
- Testy odbywają się w sferze mechaniki aplikacji, poniżej interfejsu użytkownika (zdarzają się wyjątki)

ISTQB – przebieg testów jednostkowych

- Planowanie
- Przypadki użycia/testu (ręczne bądź skryptowe)
- Przeprowadzenie testów

Właściwości testów jednostkowych

- Dobrze wykonane przypadki testów jednostkowych używane przy każdej zmianie kodu jednostkowego znacznie poprawia jego jakość
- Dobrze przetestowane jednostki mogą stanowić świetny fragment kodu do ponownego użycia (wstawki/moduły/funkcje/metody)
- Szybszy rozwój tworzonego oprogramowania
- Szybsza naprawa błędów niż w przypadku odnalezienia defektów w fazie późniejszych testów

Właściwości testów jednostkowych

- Niższe koszta odnalezienia i naprawienia błędów aplikacji
- Usprawnione odpluskwianie poprzez szybsze wynajdywanie błędów (najczęściej świeże błędy powstają poprzez ostatnie zmiany w kodzie – łatwo je prześledzić)
- Kod powstałej aplikacji jest mniej zawodny

Rady ISTQB dla testów jednostkowych

- Należy znaleźć odpowiednie skrypty/narzędzia/biblioteki testujące dla swojego języka/technologii
- Należy rozważnie dobierać testy by nie testować wszystkiego
- Dobrze jest oddzielić środowisko rozwojowe od testowanego
- Zestaw danych testujących powinien być podobny do produkcyjnego

Rady ISTQB dla testów jednostkowych

- W przypadku odnalezienia błędu dobrze jest napisać skrypt/przypadek testowania wyróżniający odnaleziony błąd
- Przypadki testowania powinny być od siebie niezależne
- Szczególną uwagą należy obdarzyć wszelkiego rodzaju fragmenty decyzyjne oraz pętle
- Należy używać kontroli wersji
- Prócz testów na zachowanie się jednostki należy też utworzyć przypadki testowe uwzględniające wydajność

Testy połączenia

- Kolejny etap po testach jednostkowych
- Testowane jest zachowanie połączonych w system dwóch lub więcej pojedynczych elementów tworzonej aplikacji
- Testy mają wykazać wszelkie nieprawidłowości podczas współdziałania ze sobą elementów, włączając w to ich komunikację, przesyłanie wiadomości/danych, reagowanie na niedyspozycję jednego z elementów (zakończenie z błędem)

Testy połączenia

- Metoda czarnego pudełka/skrzynki
- Metoda białego pudełka/skrzynki
- Metoda szarego pudełka/skrzynki
- Pozostałe metody stosowane niezwykle rzadko (jedynie w przypadku realnej potrzeby spowodowanej specyfiką problemu)

Testy połączeniowe - właściwości

- Testowanie Do dołu – polega na wykonywaniu operacji od elementów najwyższych (interfejs użytkownika/linia tekstowa wprowadzania poleceń) do elementów najniższych (wykonujących się w następstwie wywołania z wyższych warstw oprogramowania)
- Testowania Do góry – odwrotne do wyżej opisanego
- Testowania łączone – posiada cechy obu powyższych
- Wielki wybuch – testowanie wszystkich modułów naraz, z pominięciem podziału na jednostki interfejsu/mechaniki programu.

Testy połączeniowe - wykonanie

- Dobra dokumentacja wszelkich możliwych połączeń poszczególnych elementów jednostkowych
- Wystarczająca konfiguracja środowiska komputerowego
- Najlepiej by wszystkie testy jednostkowe były do tego etapu zakończone (jeszcze lepiej – pozytywnie)
- Dobrym rozwiązaniem jest automatyzacja tego etapu testów (regresja!)

Testy systemowe

- Wykonywane w odniesieniu do całokształtu aplikacji
- W przeciwieństwie do testów połączeń ich rola nie ogranicza się do sprawdzenia dopasowania elementów, lecz także do reakcji na dane wejściowe/generowane zdarzenia
- Testy przeprowadza się w oparciu o konkretne wymagania co do funkcjonalności, użyteczności i efektywności systemu

Testy systemowe

- Metodą testowania jest przeważnie czarna skrzynka/pudełko
- W przeciwieństwie do 2 poprzednich części systemu testowania, tę część przeprowadzają niezależni testerzy

Testy Odbiorcze

- Główny nacisk kładziony jest na dostosowanie testów do ostatecznych wymagań/codziennej pracy systemu u klienta
- Należy odpowiednio punktować każdą zgodność z ostatecznym zamówieniem
- Stosowanymi metodami jest test czarnej skrzynki/pudełka oraz metoda bezpośrednia/natychmiastowa
- Skrypty automatyzujące nie są w tej części mile widziane

Testy odbiorcze

- Faza Alpha – testy wewnątrz środowiska tworzącego oprogramowanie, jednak nie bezpośrednio związane z jego rozwojem
- Faza beta – testy zewnętrzne zamknięte oraz otwarte. Do zamkniętych zaliczamy zespół testerów należący/nie należący do firmy rozwijającej oprogramowanie. Do otwartych testów zaliczmy testy przeprowadzane przez chętnych użytkowników/użytkowników docelowych i/lub użytkowników, którzy są/będą klientami aktualnych użytkowników

Materiały:

- http://whatis.techtarget.com/definition/software-testing
- http://softwaretestingfundamentals.com/software-testing-life-cycle/
- https://www.guru99.com/software-testing-life-cycle.html
- http://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/
- http://www.softwaretestinghelp.com/software-test-metrics-and-measurements/

Materiały

- http://testerzy.pl/baza-wiedzy/monitorowanie-i-kontrola-w-testowaniu
- https://dariuszwozniak.net/2016/06/13/kurs-tdd-cz-22-pokrycie-kodu-testami-code-coverage/
- http://testerzy.pl/slownik/pokrycie-kodu
- http://www.testowanie.net/testowanie/priorytety-w-testowaniu-oprogramowania-analiza-ryzyka/
- http://www.istqb.org
- http://softwaretestingfundamentals.com/unit-testing/

Materiały