Testowanie oprogramowania

Robert Kaszubowski

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera

Pytanie?

- Po co testować software?

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Dlaczego testowanie jest ważne

- systemy oprogramowania coraz większą częścią życia społeczeństwa (praca, samochody, banki,…)

  oprogramowanie które nie działa poprawnie może doprowadzić do:

  - straty czasu, pieniędzy
  - utraty reputacji
  - uszkodzeń ciała a nawet śmierci

Dlaczego testowanie jest ważne

- systemy oprogramowania złudnie proste:
  - nie ma błędów produkcji
  - nie ma ograniczeń fizycznych
  - łatwy re-design
  - możliwość dużego przyrostu funkcjonalności
- systemy oprogramowania ze względu na specyfikę podatne na błędy ludzkie

większość osób miało do czynienia z oprogramowaniem które nie działało poprawnie

Przyczyny błędów oprogramowania

- błędy ludzkie:
  - natura ludzka
  - presja czasu
  - duże skomplikowanie kodu źródłowego
  - duże skomplikowanie infrastruktury
  - zmiany technologii
  - zmiany wymagań
  - interakcje z wieloma innymi systemami

Błędy ludzkie w oprogramowaniu

- człowiek może zrobić **pomyłkę** (error, mistake)
- … która powoduje **błąd, defekt** (defect, fault, bug) w kodzie, w oprogramowaniu, w systemie
- … który może doprowadzić do **awarii, upadku** (failure) jeżeli błędny kod zostanie wykonany
- defekt w oprogramowaniu (dokumentacji) może ale nie musi doprowadzić do awarii

Rola testowania oprogramowania

- zmniejsza ryzyko wystąpienia awarii w czasie użytkowania oprogramowania
- podnosi jakość oprogramowania
- konieczne do spełnienia wymagań kontraktu, wymagań prawnych, wymagań standardów przemysłowych

Testowanie a Jakość

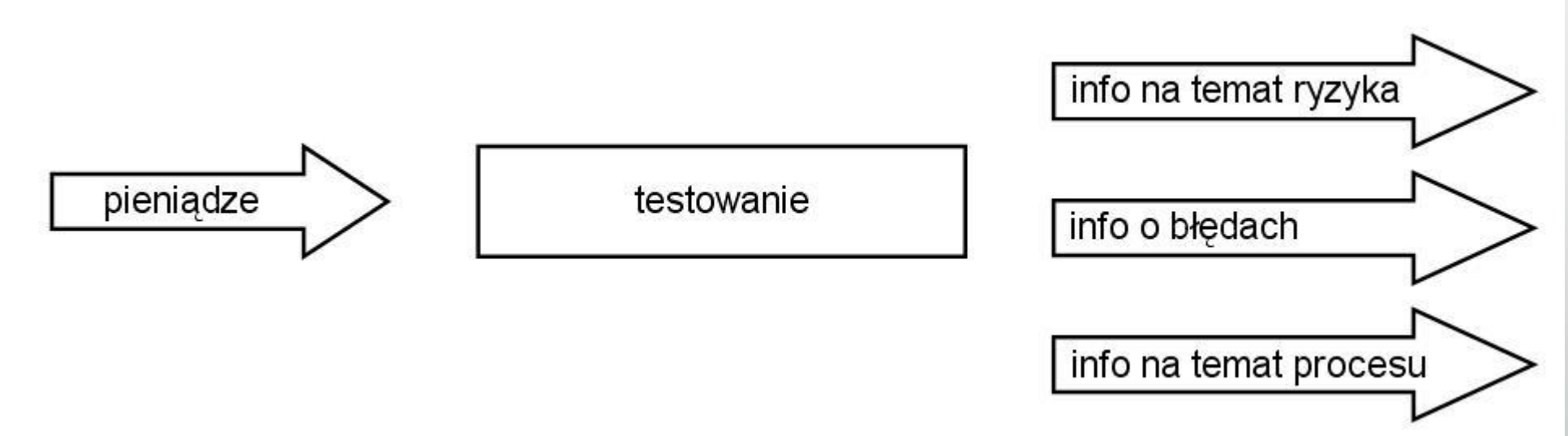
Testowanie:

- pozwala zmierzyć jakość oprogramowania
- pomaga budować zaufanie do oprogramowania
- zwiększa jakość jeżeli błędy są znajdowane i usuwane
- zmniejsza ryzyko upadku systemu
- pozwala na poprawę procesu produkcji oprogramowania

Testowanie wintegrowane w system zapewnienia jakości oprogramowania w organizacji (obok np. standardów kodowania, szkoleń, analizy incydentów)

Produkty testowania

Produkty testowania



Cele testów 1/3

- znalezienie defektów
- uzyskanie informacji o poziomie jakości oprogramowania (zbudowanie zaufania)
- monitorowanie ryzyk produktu
- zapobieganie defektom:
  - projektowanie testów jak najwcześniej,
  - przegląd dokumentacji (bazy testowej).

Cele testów 2/3

- szczegółowe cele zależne od punktu widzenia i momentu w cyklu życia oprogramowania:
  - testy developerskie – znalezienie i eliminacja jak największej liczby błędów
  - testy akceptacyjne – potwierdzenie że oprogramowanie spełnia wymagania
  - testy utrzymaniowe – sprawdzenie czy nowe defekty nie zostały wprowadzone
  - testy operacyjne – sprawdzenie charakterystyk niefunkcjonalnych oprogramowania

Cele testów 3/3

- Testy konfirmacyjne – potwierdzenie że poprawka błędu jest prawidłowa
- Test regresji – zapewnienie że nowe błędy w częściach nie zmienionych przez poprawki nie zostały wprowadzone/odkryte

7 zasad testowania

1. Testowanie udowadnia istnienie błędów

   Testowanie może pokazać że błędy istnieją ale nie może udowodnić że błędów nie ma

7 zasad testowania

2. Testy wyczerpujące są niemożliwe

Przetestowanie wszystkiego jest niemożliwe – w zamian za to analiza ryzyka i priorytetyzacja testów

7 zasad testowania

3. Testować jak najwcześniej

Aktywności testowe powinny rozpoczynać się w cyklu życia oprogramowania tak szybko jak to tylko możliwe

7 zasad testowania

4. Kumulowanie się błędów

Mała liczba modułów (funkcjonalności) zawiera większość błędów wychodzących w trakcie testów lub użytkowania oprogramowania

7 zasad testowania

5. Paradoks pestycydów – oprogramowanie uodparnia się na testy

Te same przypadki testowe powtarzane wielokrotnie nie znajdują nowych defektów. Przypadki testowe powinny być przeglądane, aktualizowane. Nowe przypadki testowe powinny być dodawane.

7 zasad testowania

6. Testowanie zależy od kontekstu

Testy powinny być dostosowane do kontekstu oprogramowania.

7 zasad testowania

7. Mylne przekonanie o braku błędów

Znalezienie i eliminacja błędów nie pomoże jeżeli system jest nieużyteczny i nie spełnia potrzeb i oczekiwań Klienta, użytkowników.

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Podstawowy proces testowy

1. Planowanie i kontrola
2. Analiza i projektowanie
3. Implementacja i wykonanie
4. Obliczanie kryteriów wyjścia i raportowanie
5. Zakończenie testów

Planowanie i kontrola testów

Planowanie testów:

- Weryfikacja misji testowania
- Zdefiniowanie strategii testowania
- Zdefiniowanie celów testowania (test objectives)
- Określenie aktywności testowych mających spełnić cele i misję testowania

Planowanie i kontrola testów

Kontrola testów:

- Porównywanie aktualnego postępu w stosunku do założonego planu
- Raportowanie statusu (szczególnie odchyłek od założonego planu)
- Podejmowanie kroków niezbędnych do spełnienia założonej misji i celów testów
- Aktywność ciągła w projekcie
- Kontrola możliwa tylko dzięki ciągłemu monitorowaniu testów

Analiza i projektowanie

Zamiana ogólnych celów testowania (zdefiniowanych na etapie Planowania) na rzeczywiste i namacalne **warunki testowe** (Analiza) i **przypadki testowe** (Projektowanie)

Analiza i projektowanie

- Przegląd bazy testów: wymagania, architektura, projekt systemu, projekt interfejsów
- Oszacowanie testowalności bazy testów i celów testów
- Identyfikacja i priorytetyzacja warunków testowych na podstawie analizy bazy testowej
- Projektowanie i priorytetyzacja przypadków testowych
- Identyfikacja koniecznych danych testowych
- Projekt środowiska testowego, identyfikacja wymaganej infrastruktury i narzędzi

Implementacja i wykonanie

Tworzenie **procedur testowych** i **skryptów testowych** na podstawie zdefiniowanych przypadków testowych – Implementacja

Wykonanie zaimplementowanych procedur testowych – Wykonanie

Implementacja i wykonanie

- Implementacja przypadków testowych poprzez projektowanie i priorytetyzację procedur testowych
- Wybór danych testowych
- Przygotowanie automatycznych skryptów testowych
- Stworzenie zestawów testów z procedur testowych dla wygodniejszego wykonywania testów
- Weryfikacja przygotowania Środowiska testowego

Implementacja i wykonanie

- Wykonanie testów i logowanie wyników testów
- Porównanie aktualnych wyników z wynikami spodziewanymi
- Raportowanie niezgodności jako incydentów
- Analiza przyczyn incydentów
- Testy konfirmacyjne poprawionych błędów
- Test regresyjne niezmienionego kodu

Obliczanie kryteriów wyjścia

Oszacowanie kryteriów zakończenia testów i porównanie ze zdefiniowanymi celami:

- Porównanie wyników testów z kryteriami wyjścia zdefiniowanymi na etapie planowania
- Oszacowanie czy konieczna jest kontynuacja testów
- Decyzja o dostarczeniu oprogramowania
- Przygotowanie Raportu Podsumowującego (Test Summary Report)

Zakończenie testów

Zebranie informacji z zakończonych testów w celu zgromadzenia doświadczeń, produktów testów, potrzebnych statystyk:

- Dostarczenie dokumentacji z testów
- Zamknięcie błędów
- zgłoszenie żądań zmiany
- Archiwizacja pełnej dokumentacji testowej
- Analiza wniosków w celu ulepszenia procesu testów w przyszłych dostawach

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

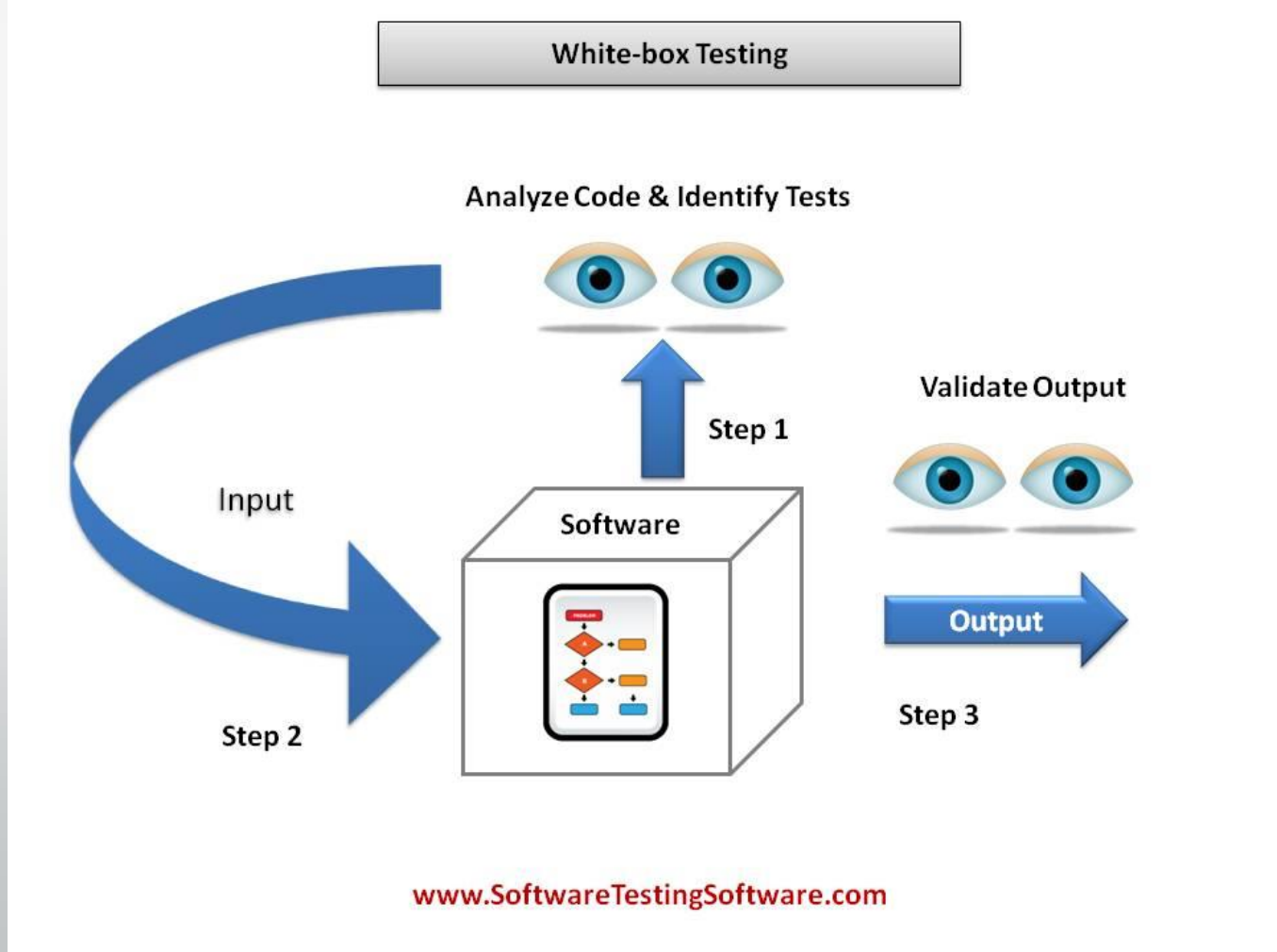
Podział testów

Ze względu na widzialność kodu

- Białoskrzynkowe
- Czarnoskrzynkowe

Testy białoskrzynkowe

- Unit Testing
- Static & dynamic Analysis
- Statement Coverage
- Branch Coverage
- Security Testing
- Mutation Testing



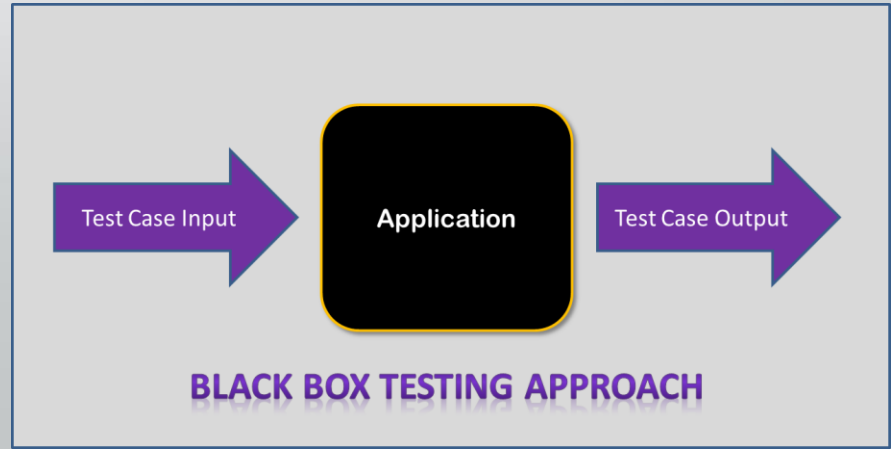
Testy białoskrzynkowe

- Zalety testowania metodą białej skrzynki:
  - ponieważ wymagana jest znajomość struktury kodu, łatwo jest określić jaki typ danych wejściowych/wyjściowych jest potrzebny, aby efektywnie przetestować aplikację
  - oprócz głównego zastosowania - testów, pomaga zoptymalizować kod aplikacji
  - pozwala dokładnie określić przyczynę i miejsce w którym znajduje się błąd
- Wady testowania metodą białej skrzynki:
  - ponieważ wymagana jest znajomość struktury kodu, do przeprowadzenia testów potrzebny jest tester ze znajomością programowania co podnosi koszty.
  - jest prawie niemożliwym przeglądniecie każdej linii kodu w poszukiwaniu ukrytych błędów co może powodować błędy po fazie testów.

Testy czarnoskrzynkowe

- Functional Testing
- Stress Testing
- Load Testing
- Ad-hoc Testing
- Exploratory Testing
- Usability Testing
- Performance Testing
- Smoke Testing
- Recovery Testing
- Volume Testing
- Domain Testing

- Scenario testing
- Regression Testing
- User Acceptance
- Alpha Testing
- Beta Testing



Testy czarnoskrzynkowe

- Zalety testowania metodą czarnej skrzynki:
  - testy są powtarzalne
  - testowane jest środowisko w którym przeprowadzane są testy
  - zainwestowany wysiłek może być użyty wielokrotnie
- Wady testowania metodą czarnej skrzynki:
  - Wyniki testów mogą szacowane nazbyt optymistycznie
  - Nie wszystkie właściwości systemu mogą zostać przetestowane
  - Przyczyna błędu nie jest znana

Poziomy testów

- Testy modułowe (unit/component testing)
- Testy integracyjne (integration testing)
- Testy systemowe (system testing)
- Testy akceptacyjne (acceptance testing)

Poziomy testów

1. Testy modułowe

- analiza ścieżek (path analysis)

- użycie klas równoważności (equivalence partition)

- testowanie wartości brzegowych

- testowanie składniowe

2a. Testy integracyjne pomiędzy modułami
- funkcjonalne
- wydajnościowe

2b. Testy integracyjne pomiędzy systemami
- funkcjonalne
- wydajnościowe
- regresywne

Poziomy testów

3. Testy systemowe
- instalacyjne
- funkcjonalne
- interfejsu (użyteczności)
- wydajnościowe
- regresywne
- bezpieczeństwa

4. Testy akceptacyjne
- funkcjonalne
- wydajnościowe
- bezpieczeństwa

Testy jednostkowe

- Testowanie na najniższym poziomie, podczas którego poszczególne metody (funkcje) testowane są pojedynczo, w oderwaniu od reszty aplikacji w celu sprawdzenia pod kątem zgodności ze zdefiniowanym typem/zakresem danych wejściowych.

JUnit

JUnit pozwala nam:

- Stworzyć test sprawdzający, czy wyniki działania metod spełniają wymagane zależności (AssertTrue, AssertEquals itd.)
- Organizować testy w spójne zestawy, testujące określoną część funkcjonalności projektu, lub wybrane klasy.
- Uruchamiać napisane testy zarówno przy użyciu interfejsu graficznego jak i z linii poleceń.

TestNG

- wykorzystanie adnotacji Java 5,
- elastyczną konfigurację testów,
- obsługę parametryzacji testów,
- testowanie w środowisku rozproszonym,
- rozbudowany mechanizm tworzenia zestawów testów,
- rozszerzalność za pomocą języka skryptowego BeanShell,

Test Driven Development

- Testy piszemy przed napisaniem fragmentu kodu, żeby określały wymagania, które kod ma spełniać.
- Testy piszemy w trakcie pisania kodu, żeby pokrywały wyjątkową sytuację, która akurat przyszła nam do głowy.
- Każdy fragment kodu, który może źle zadziałać musi być testowany przez jakiś test.
- Testy uruchamiamy w każdej chwili, gdy zużycie procesora spada poniżej 80% ;).
- Staramy się uruchomić każdy test przynajmniej raz dziennie.
- Małe testy sprawdzające aktualnie modyfikowany kod uruchamiamy jak najczęściej.

Testy integracyjne

- **Testowanie integracyjne** wykonywane jest w celu wykrycia błędów w interfejsach i interakcjach pomiędzy modułami.
- Podejście do testów integracyjnych:
  - Top-down
  - Bottom-up
  - Big bang
- Także projekty

Testy systemowe

Podczas testów systemowych cały system jest weryfikowany pod kątem zgodności z:

- wymaganiami funkcjonalnymi
- wymaganiami niefunkcjonalnymi (wydajność, użyteczność, niezawodność)

Innymi słowy system jest testowany **całościowo** z użyciem technik czarnej skrzynki. Wiedza o kodzie lub wewnętrznej strukturze aplikacji nie jest wymagana podczas testów systemowych.

Testy akceptacyjne

- Walidacja systemu pod kątem zgodności z wymaganiami klienta, który na swoim środowisku wykonuje przypadki testowe przy udziale przedstawicieli projektu.
- Produkcyjne testy akceptacyjne
- Testowanie akceptacyjne w środowisku użytkownika

Testy funkcjonalne

Testowanie funkcjonalne obejmuje:

- Testy jednostkowe
- Testy dymne / testy kondycji
- Testowanie integracyjne (testowanie zstępujące, testowanie wstępujące)
- Testy interfejsu i użyteczności
- Testy systemowe
- Testy regresji
- Testy Alfa i Beta
- Testy akceptacyjne użytkownika
- Testowanie metodami białej skrzynki oraz metodą czarnej skrzynki

Testy wydajnościowe

Performance testing (testy wydajnościowe)

- badanie czasu odpowiedzi krytycznych dla biznesu funkcji systemu
- jest sprawdzenie czy poszczególne akcje wykonywane są przez aplikację w akceptowalnym czasie

Stress testing (testy przeciążeniowe)

- założenie: zbyt wielu użytkowników, danych, czasu oraz malejące zasoby systemowe
- badanie czy system "zawiedzie" w oczekiwany sposób
- wyszukiwanie defektów w aplikacji działającej w trybie awaryjnym
- sprawdzanie konsekwencji utraty danych po awarii wywołanej nadmiernym obciążeniem

Load testing (testy obciążeniowe)

- duża liczba jednocześnie działających użytkowników / przeprowadzanych transakcji
- utrzymanie takiego stanu przez określony w scenariuszu czas
- jak wiele zapytań (requests) jest w stanie obsłużyć system w określonym przedziale czasu

Testy regresyjne

- Celem przeprowadzania testów regresywnych jest upewnienie się, że aplikacja działa po dokonaniu w niej modyfikacji, poprawieniu błędów lub po dodaniu nowej funkcjonalności.

- Cecha: powtarzalność

Testy regresyjne – co dają?

- Wyszukanie błędów powstałych w wyniku zmian kodu/środowiska.
- Ujawnienie wcześniej nie odkrytych błędów.
- Jest to dobry kandydat do automatyzacji ze względu na swoją powtarzalność.
- Iteracyjne metodologie oraz krótkie cykle w których dostarczane są kolejne funkcjonalności powodują, że testy regresywne pozwalają upewnić się czy nowe funkcjonalności nie wpłynęły negatywnie na istniejące już i działające części systemu.

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Kiedy zakończyć testy

- Deadlines, e.g. release deadlines, testing deadlines
- Test cases completed with certain percentage passed
- Test budget has been depleted
- Coverage of code, functionality, or requirements reaches a specified point
- Bug rate falls below a certain level
- The risk in the project is under acceptable limit

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Dokumentacja testowa – plan testów

Dokument IEEE 829 sugeruje rozdzialy:

- Test plan identifier
- Introduction
- Test items
- Features to be tested
- Features not to be tested
- Approach
- Item pass/fail criteria
- Suspension criteria and resumption requirements
- Test deliverables
- Testing tasks
- Environmental needs
- Responsibilities
- Staffing and training needs
- Schedule
- Risks and contingencies
- Approvals

Dokumentacja testowa – test script

A test script is a procedure, or programming code that replicates user actions. Initially the term was derived from the product of work created by automated regression test tools. Test Case will be a baseline to create test scripts using a tool or a program.

Dokumentacja testowa – test suite

The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Dokumentacja testowa – test fixture / data

In most cases, multiple sets of values or data are used to test the same functionality of a particular feature. All the test values and changeable environmental components are collected in separate files and stored as test data. It is also useful to provide this data to the client and with the product or a project.

Dokumentacja testowa – test case

- Test Script Title
- Test Design Title
- ID
- Version
- Creation Date
- Intended Tester
- Product
- Product version
- Intended Platform/OS
- Priority

- Summary
- Preconditions
- Test Steps
- Expected Results

Dokumentacja testowa – bug report

- Title
- ID
- Version
- Creation Date
- Intended Tester
- Product
- Product version
- Intended Platform/OS
- Priority
- Classification
- Can it be reproduced
- Attachments

- Summary
- Preconditions
- Test Steps
- Expected Results
- Results
- Comments

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Testy automatyczne

Zalety

- testy automatyczne świetnie uzupełniają testy manualne
- obliczenia, które człowiekowi mogą sprawić wiele trudności przez maszynę zostaną przetworzone w ułamku sekundy
- powtarzalność - np. regresja
- wraz z upływem czasu organizm człowieka się męczy, spada koncentracja i coraz trudniej jest mu ustrzec się błędów, automat się nie męczy
- mogą być uruchamiane cyklicznie (np. codziennie wraz z daily buildem oprogramowania)
- pozwalają oszczędzić czas testerów
- błędy związane z czynnikiem ludzkim praktycznie wyeliminowane

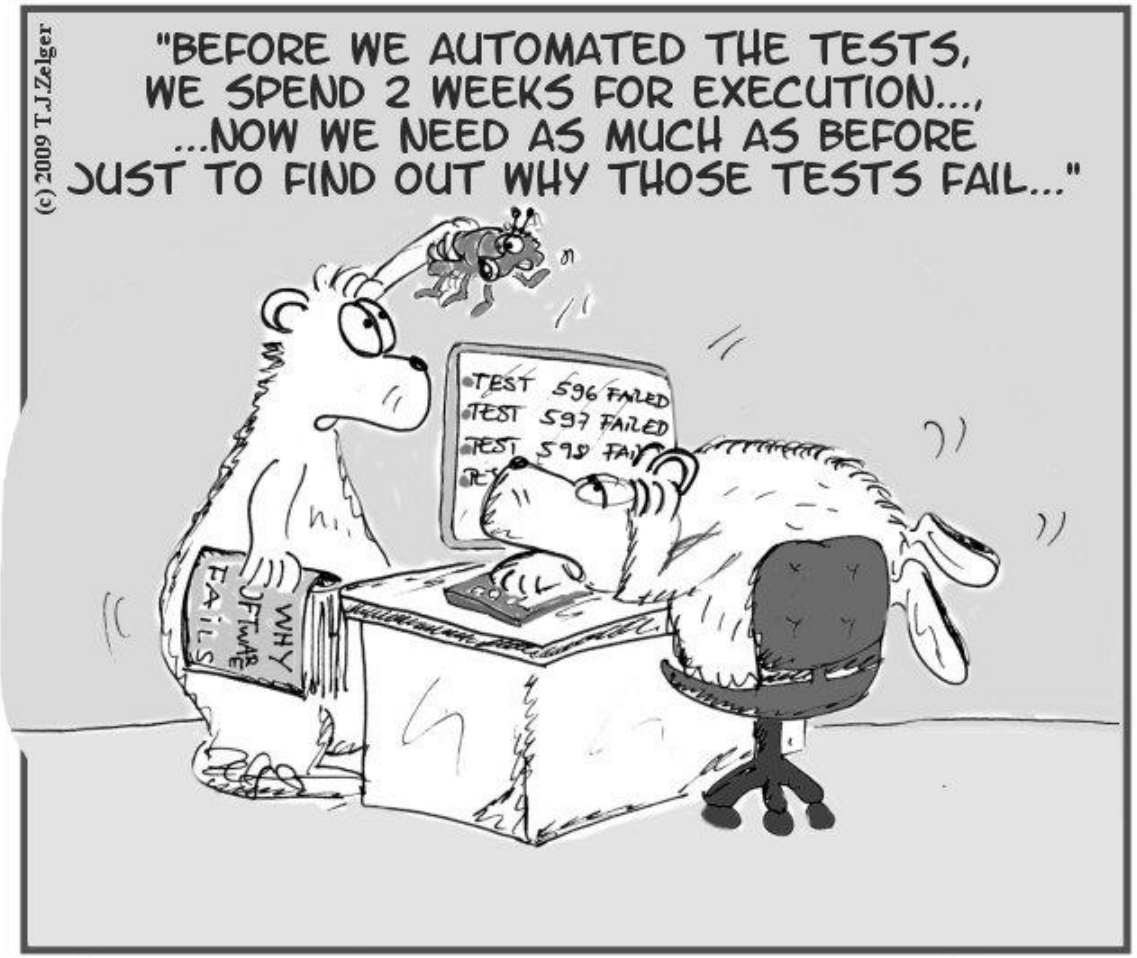
Testy automatyczne

Wady a zarazem zalety testów manualnych:

- nie mogą zastąpić testów manualnych
- komputery mają trudności z przetwarzaniem i interpretowaniem pewnych informacji, które są naturalne dla człowieka
- obsługa automatycznego środowiska testowego wymaga sporych nakładów
- testy manualne często nie wymagają wiedzy z programowania

Testy automatyczne a manualne

KOSZTA!



Test Automation side effect

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Selenium

to darmowa aplikacja do testowania aplikacji webowych. Umożliwia wykonywanie testów regresyjnych, testów automatycznych oraz testów wydajnościowych. Selenium umożliwia nagrywanie oraz uruchamianie skryptów bez wiedzy programistycznej. Współpracuje z przeglądarką Firefox 2+ oraz umożliwia nagrywanie aplikacji napisanych w językach programowania jak: C#, Java, Perl, PHP, Python oraz Ruby.

Squish

Squish is a commercial, functional, cross-platform GUI and regression testing tool that can test applications based on a variety of GUI technologies (see list below). It is developed and maintained by Froglogic.

TestAutomationFX

znana często pod skróconą nazwą: TAFX to darmowa aplikacja umożliwiająca programistom oraz testerom nagrywanie oraz zarządzanie utworzonymi przypadkami testowymi UI tworzonych aplikacji w języki .NET z poziomu aplikacji Visual Studio (2005 & 2008).

Test Automation FX

Badboy

- Badboy – to aplikacja typu opensource utworzona przez firmę Badboy Software.
- Narzędzie skierowane jest dla testerów oraz programistów, którzy spędzają dużo czasu na ciągłym i tym samym przeklikiwaniu testowanej lub tworzonej strony.
- Badboy wykorzystywany jest przy wykonywaniu testów regresyjnych, testów automatycznych oraz testów wydajnościowych aplikacji. Aplikacja jest darmowa natomiast w przypadku gdy w firmie jest wykorzytywanych więcej niż 5 stanowisk zalecany jest kontakt z firmą w celu opłacenia licencji.

AutoHotKey

- AutoHotkey – to darmowa aplikacja typu Open Source działająca pod systemem operacyjnym Windows. Znana także pod nazwą AHK z ang.: Auto Hot Key.
- Aplikacja umożliwia automatyzację każdej akcji wykonanej poprzez klawiaturę jak i mysz poprzez symulowanie dowolnej sekwencji klawiszy i dowolne ruchy myszy.
- Macro można napisać samemu lub skorzystać z funkcji rekord, która nagra całe macro.
- Konwertowanie utworzonego skryptu do pliku .exe, dzięki czemu można uruchomić skrypt.exe na komputerach nie posiadających zainstalowanej aplikacji AutoHotkey.

Agenda

- Dlaczego testy są potrzebne i czym są
- Podstawowy proces testowy
- Podział testów
- Kiedy zakończyć testy
- Dokumentacja testowa
- Testy automatyczne a manualne
- Narzędzia wykorzystywane w testach automatycznych
- Cechy dobrego testera i psychologia testów

Podejście do testowania

- Sposób myślenia podczas testowania (przeglądów) różny od nastawienia podczas tworzenia oprogramowania:
  - dociekliwość, powątpiewanie, dbałość o szczegóły
  - ciekawość, nastawienie „co jeśli?”
  - „profesjonalny pesymizm”, krytyczne spojrzenie
  - poszukiwanie, badanie, zagłębianie się
- Podejście Testera może być kontrowersyjne dla Programistów

Niezależność testowania

- Ryzyko związane z testowaniem własnego kodu
- Rozwiązanie – **niezależność testowania**
- Korzyści:
  - Wzrost efektywności znajdowania błędów
  - Spojrzenie na oprogramowanie pod innym kątem
  - Specjalistyczna wiedza o testowaniu
  - Brak powiązania z produktem
  - Weryfikacja założeń poczynionych na etapie specyfikacji wymagań i implementacji oprogramowania

Niezależność testowania

- Wady:
  - Izolacja w stosunku do zespołu programistów
  - Niezależni testerzy mogą stanowić wąskie gardło jako ostatni punkt sprawdzenia oprogramowania
  - Programiści mogą stracić poczucie odpowiedzialności za jakość oprogramowania

Poziomy niezależności

- Testy projektowane przez autora kodu (Programistę) – brak niezależności
- Testy projektowane przez inną osobę z zespołu Programistów
- Testy projektowane przez osoby z odrębnego zespołu w organizacji (np. zespołu testowego)
- Testy projektowane przez inną organizację lub firmę (outsourcing)
- Testy projektowanie przez testerów z organizacji klienta bądź użytkowników
- Niezależni specjaliści testowi do wybranych typów testów np. testów użyteczności, wydajności

Rozwiązania niezależności

- Dla dużych skomplikowanych projektów najlepiej kilka poziomów testowania realizowanych przez niezależnych testerów
- Programiści mogą uczestniczyć w testowaniu szczególnie na niższych poziomach testów
- Niezależni testerzy mogą mieć autorytet do definiowania procesu testowego ale tylko przy wyraźnej zgodzie kierownictwa

Nastawienie

- Znajdowane błędy - pomoc Programistom w podniesieniu ich umiejętności
- Testerzy nie mogą być traktowani jako posłańcy złych wiadomości
- Rywalizacja między zespołem Testerów a Programistami wpływa destrukcyjnie
- Jasno określone cele testów - wydajność Testerów
- W wypracowaniu odpowiedniego nastawienia kluczową rolę odgrywa **komunikacja**

Komunikacja

- Testerzy - zdolności interpersonalne i komunikacyjne
- Defekty komunikowane w konstruktywny sposób
- Rzeczowa informacja na temat defektów, postępu testów jak i występujących ryzyk

Komunikacja

Poprawa komunikacji między zespołem testowym a innymi zespołami:

- Nie rywalizacja a współpraca w celu osiągnięcia wspólnych celów
- Incydenty komunikowane w sposób neutralny, skupiony na faktach
- Nie krytykować a jedynie informować
- Postarać się zrozumieć uczucia innych osób
- Jasna i precyzyjna komunikacja w celu uniknięcia nieporozumień

Cechy dobrego testera wg. testerzy.pl

Steven Miller w artykule "The Seven Habits of Highly Effective Testers" opisuje róznice między testerem a dobrym testerem:

1) Badź proaktywny

2) Zaczynając myśl już o końcu

3) Najważniejsze rzeczy na początku

4) Myśl w kategoriach Win/Win (zwycięstwo/zwycięstwo)

5) Na początku zrozum, potem postaraj się być zrozumianym

6) Staraj się o synergię

7) Badź ostrzejszy niż brzytwa