Metodyka kształcenia testerów

Fundusze Europejskie Wiedza Edukacja Rozwój

Unia Europejska Europejski Fundusz Społeczny

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Spis treści

Wstęp - dlaczego testowanie oprogramowania jest tak ważne? 3
1. Definicja testowania 3
2. Jak wygląda proces testowy? 4
2.1. Planowanie i nadzór nad testami 4
2.2. Analiza i projektowanie testów 4
2.3. Implementacja i wykonywanie testów 5
2.4. Ocena kryteriów zakończenia i raportowanie 6
2.5. Czynności zamykające test 6
3. Jak poprawnie zgłosić znaleziony błąd? 6
3.1. Co zrobić, kiedy pojawi się błąd? 7
3.2. Jak prawidłowo zgłosić błąd? 7
4. Testowanie oprogramowania – rodzaje testów 8
4.1. Testy strukturalne i funkcjonalne 8
4.2. Testy manualne i automatyczne 9
4.3. Zakres testów 9
4.4. Podział testów z uwagi na ich przeznaczenie 10
5. Główne typy narzędzi ułatwiających testowanie 11
5.1. Narzędzia do zarządzania testami i raportami 11
5.2. Narzędzia do automatyzacji 11
5.3. Narzędzia do zarządzania incydentami 11
5.4. Narzędzia do wykonywania testów 11
5.5. Narzędzia do testów wydajnościowych 12
5.6. Narzędzia do przygotowywania danych testowych 12
6. Testy automatyczne 13
7. Kto powinien testować oprogramowanie? 14
8. Kiedy należy testować oprogramowanie? 14
9. Uwagi 15
10. Kiedy należy przestać testować? 15
11. Cechy testera i programisty 16
12. Komunikacja między testerem a programistą 18
13. Słownik pojęć podstawowych 18
Bibliografia 27

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Wstęp - dlaczego testowanie oprogramowania jest tak ważne?

Okazuje się, że pytanie to nie jest tak banalne, jak się może wydawać. Na samym początku warto zdać sobie sprawę z jednej bardzo ważnej rzeczy – testowanie samo w sobie nie podnosi jakości danego oprogramowania. Kiepsko działający system prowadzić może jednak do utraty sporej ilości pieniędzy, czasu, reputacji, a także może powodować niekiedy utratę życia czy zdrowia.

Testowanie powala na wykrycie ewentualnych usterek, a jakość oprogramowania podnosi się wraz z ich usunięciem. Testowanie umożliwia zmierzenie jakości oprogramowania, która wyrażona jest przez ilość wykrytych usterek dla funkcjonalnych i niefunkcjonalnych wymagań oraz atrybutów danego oprogramowania. Za sprawą testów możliwe jest budowanie zaufania do jakości tworzonego przez większy zespół ludzi oprogramowania.

Jakie są główne cele testowania?

- wyszukiwanie usterek,
- podnoszenie poziomu zaufania do jakości produktu,
- dostarczanie danych niezbędnych do podejmowania decyzji,
- zapobieganie defektom.
-

1. Definicja testowania

Testowanie oprogramowania jest wykonywaniem kodu dla kombinacji danych wyjściowych oraz stanów w celu wykrycia ewentualnych błędów. Testowanie jest szeregiem procesów, które zostały zaprojektowanie w celu zapewnienia, że skutek wykonania danego kodu zgadza

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

się w 100% z założeniami projektowymi, a wykonanie kodu nie powoduje skutków niezgodnych z założeniami początkowymi.

Należy jednak zdawać sobie sprawę z tego, że nie można przetestować wszystkich możliwych danych wejściowych i wyjściowych danego programu. Okazuje się bowiem, że liczba możliwych przypadków testowych może sięgać kilku milionów. Ich przetestowanie przekracza ludzkie możliwości.

Nie da się jednoznacznie określić, ile czasu powinno się przeznaczyć na testy. To sprawa w dużej mierze indywidualna, która zależy od czasu i przewidzianego na ten cel budżetu. Testowanie z założenia powinno dostarczać informacji, które są wystarczające do podejmowania świadomych decyzji odnośnie dopuszczenia danego oprogramowania czy systemu do kolejnej fazy rozwoju lub przekazania go klientowi.

2. Jak wygląda proces testowy?

2.1. Planowanie i nadzór nad testami

Planowanie testów obejmuje:

- weryfikację misji testowania,
- zdefiniowanie strategii testowania,
- zdefiniowanie celów testowania,
- określenie aktywności testowych, które mają za zadanie spełnić cele i misję testowania.

Kontrolowanie testów obejmuje:

- porównywanie aktualnego postępu prac z założonym na wstępie planem,
- raportowanie statusu prac (zwłaszcza wszelkich opóźnień),
- podejmowanie kroków potrzebnych do spełnienia danej misji i celów testów,
- kontynuowanie prac nad projektem,

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

- ciągłe monitorowanie testów.

2.2. Analiza i projektowanie testów

Do podstawowych zadań analizy i projektowania testów zalicza się:

- przeglądanie podstawy testów, a więc wymagań, raportów analizy ryzyka, architektury projektu, poziomu integralności oprogramowania itd.,
- ocenę testowalności podstawy testowania oraz przedmiotu testów,
- identyfikację warunków testowych na podstawie analizy elementów testowych, struktury oprogramowania i specyfikacji,
- projektowanie przypadków testowych wysokiego poziomu,
- ustalenie, które dane testowe są niezbędne dla warunków testowych i przypadków testowych,
- projektowanie środowiska testowego,
- identyfikowane niezbędnych narzędzi i infrastruktury,
- tworzenie dwukierunkowego śledzenia między podstawą testów i przypadkami testowymi.

2.3. Implementacja i wykonywanie testów

Do głównych zadań implementacji i wykonywania testów zalicza się:

- dokończenie, implementację oraz priorytetyzację przypadków testowych,
- przygotowanie oraz priorytetyzację procedur testowych, tworzenie danych testowych, pisanie automatycznych skryptów testowych,
- tworzenie zestawów testów na podstawie procedur testowych w celu szybszego i bardziej efektywnego wykonania testów,
- sprawdzenie czy dane środowisko testowe jest dobrze skonfigurowane,

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

- zweryfikowanie i uaktualnienie dwukierunkowego śledzenia między podstawą testów i przypadkami testowymi,
- wykonanie procedur testowych w określonej kolejności – ręcznie lub automatycznie,
- zapisywanie wyników testów,
- analizę porównawczą wyników rzeczywistych i wyników oczekiwanych,
- raportowanie wszelkich rozbieżności oraz ich analizowanie w celu ustalenia ich przyczyny (błąd w kodzie, danych testowych, dokumentacji).

2.4. Ocena kryteriów zakończenia i raportowanie

Do podstawowych zadań oceny spełnienia kryteriów zakończenia zalicza się:

- sprawdzanie w logach testów czy spełnione zostały kryteria zakończenia testów określone na etapie ich planowania,
- ocenienie czy istnieje potrzeba wykonania większej ilości testów i czy nie należy zmienić kryteriów zakończenia testów (stworzenie raportu podsumowującego testy).

2.5. Czynności zamykające test

Podstawowe zadania wykonywane w ramach czynności zamykających testy:

- sprawdzenie czy planowane produkty zostały dostarczone,
- zamknięcie raportów incydentów,
- udokumentowanie akceptacji systemu,
- dokończenie testaliów i ich zarchiwizowanie (z możliwością ich wykorzystania w przyszłości),
- przekazanie testaliów do zespołu serwisowego.

3. Jak poprawnie zgłosić znaleziony błąd?

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Testerzy, zarówno początkujący, jak i doświadczeni, popełniają często jeden, podstawowy błąd, a mianowicie zgłaszają błędy bez odpowiedniego opisu. Podczas zgłaszania znalezionego defektu należy tymczasem pamiętać o tym, że stworzony raport błędu jest ważnym nośnikiem informacji, w którym można wyjaśnić programiście, na czym polega problem. Podstawowym celem tworzenia raportu o błędzie jest umożliwienie programiście zobaczenie i zrozumienie danego problemu. Raport powinien zawierać również informację, jak dany błąd można wywołać ponownie. Trzeba go opisać w sposób jednoznaczny, precyzyjny i dokładny, tak aby czytający zgłoszenie programista od raz wiedział o co chodzi.

3.1. Co zrobić, kiedy pojawi się błąd?

Przede wszystkim nie można panikować. Tester oprogramowania ma obowiązek zgłoszenia błędu niezależnie od sytuacji, w jakiej go wykrył. Na początku należy przeczytać komunikat o błędzie i wykonać zrzut ekranu. Kiedy komunikat jest mało zrozumiały czy zawiera ciąg dziwnych znaków, nie trzeba wpadać w panikę. Dla programisty prawdopodobnie będą one miały duże znaczenie. Ponadto tester powinien sprawdzić również czy dany błąd jest powtarzalny. To dla programisty bardzo ważna informacja. Należy sobie również zadać pytanie, czy podanie innych danych wejściowych spowoduje pojawienie się tego samego błędu.

Tester oprogramowania na ogół nie zna faktycznego powodu pojawienia się błędu, a więc nie jest w stanie poinformować programisty, jak problem należy rozwiązać. Można jednak podzielić się z nim swoimi przemyśleniami. Wcale nie trzeba się tego wstydzić.

3.2. Jak prawidłowo zgłosić błąd?

Tytuł zgłoszenia powinien być zawsze zwarty i zwięzły, a jednocześnie oddawać istotę danego problemu. Należy unikać jednak byt ogólnych sformułowań np. menu nie działa.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Równie duże znaczenie ma miejsce występowania błędu. W tym przypadku należy podać adres URL bądź ciąg dostępu do miejsca, w którym pojawił się błąd oraz jego usytuowanie na danej stronie,

W przypadku aplikacji czy serwisów internetowych należy podać zawsze wersję przeglądarki oraz systemu operacyjnego. Kiedy błąd występuje na urządzeniu mobilnym konieczna jest także informacja o samym urządzeniu, wersji systemu operacyjnego oraz przeglądarce internetowej.

Programiści lubią konkrety, a niekiedy do naprawy błędu potrzebna jest również ścieżka jego odtworzenia. Jeśli deweloper nie wie, jak uzyskać dany błąd, prawdopodobnie zgłoszenie odrzuci. Sposób zapisu błędu można opisać w zasadzie dowolnie, ale im prościej, tym oczywiście lepiej. Warto unikać skomplikowanych i długich zdań.

W raporcie błędu zawsze jest też miejsce na opisanie swoich przypuszczeń czy spostrzeżeń. Warto sprawdzić czy dany błąd jest jednorazowy, a może powtarzalny. Na ogół programistom podaje się również swoje oczekiwania co do efektu. Podczas zgłaszania błędu nigdy nie wolno jednak krytykować programisty. Dobrze jest unikać konfliktów z analitykami, projektantami i programistami, komunikując defekty w sposób konstruktywny i obiektywny.

4. Testowanie oprogramowania – rodzaje testów

Oprogramowanie testować można na wiele sposobów, w zależności od tego jaki aspekt jego działania jest w danym momencie istotny. Kiedy oprogramowanie podlega szybkim zmianom i celem jest ich wprowadzenie bez naruszania istniejących funkcjonalności, stosuje się testy regresywne. Testy akceptacyjne z kolei są stosowane w kontakcie z klientem, aby pokazać mu, że dana aplikacja działa zgodnie z wytycznymi. Testy obciążeniowe wykonuje się w przypadku aplikacji, co do których trzeba mieć pewność w zakresie ich wydajności. Poza tym testy klasyfikuje się z uwagi na to co i w jaki sposób podlega procedurom testowym.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

4.1. Testy strukturalne i funkcjonalne

Do testowania oprogramowania można podejść na dwa różne sposoby. Pierwszy z nich to testy funkcjonalne, które polegają na tym że tester wciela się niejako w rolę użytkownika danego oprogramowania. Nie interesują go techniczne szczegóły działania programu. Testy te niekiedy nazywa się również testami czarnej skrzynki (z ang. black box testing). Drugim sposobem działania są testy strukturalne. W tym przypadku tester ma już dostęp do kodu źródłowego oprogramowania i może obserwować, w jaki sposób zachowują się różne części aplikacji oraz jakie moduły i biblioteki są wykorzystywane w czasie testu. Testy tego typu nazywa się testami białej skrzynki (z ang. white box testing). Przykładem testów strukturalnych są testy jednostkowe (z ang. unit tests), które polegają na tym, że tester lub programista tworzy kod, którego zadaniem jest sprawdzenie działania produkcyjnego kodu danej aplikacji.

4.2. Testy manualne i automatyczne

Testy mogą być wykonywane ręcznie przez samego testera, który przechodzi przez interfejs użytkownika zgodnie z daną sekwencją kroków bądź automatycznie, bez udziału testera. Na ogół automatycznie przeprowadzane są testy jednostkowe. W tym celu z pomocą przychodzą narzędzia takie jak Jakarta Ant, które posiadają wbudowaną funkcjonalność uruchamiania testów jednostkowych. O wiele trudniej jest zautomatyzować testy czarnej skrzynki. Do tego celu konieczne jest już specjalistyczne oprogramowanie, które pozwala na uruchomienie wcześniej napisanych lub nagranych przez testera skryptów.

4.3. Zakres testów

Powyższe klasyfikacje dzieliły testy z uwagi na sposób ich wykonywania. Duże znaczenie ma jednak również zakres aplikacji jaki dane testy obejmują. W tym przypadku wyróżnia się testy:

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

- Testy jednostkowe - testują oprogramowanie na podstawowym poziomie, a więc na poziomie działania pojedynczych funkcji (metod),
- Testy integracyjne - pozwalają na sprawdzenie, w jaki sposób współpracują ze sobą różne komponenty oprogramowania; obecnie monolityczne aplikacje należą do rzadkości, na ogół tworzy je je modułowo, a więc istnieje konieczność sprawdzenia, czy całość razem działa poprawnie,
- Testy systemowe - dotyczą działania aplikacji jako całości, na tym poziomie najczęściej testuje się różnego rodzaju wymagania niefunkcjonalne, a więc szybkość działania, bezpieczeństwo, niezawodność itd.

4.4. Podział testów z uwagi na ich przeznaczenie

Ten podział jest o tyle interesujący, że powala na wybranie rodzaju testów w zależności od tego, do czego ma być on wykorzystywany. W tym przypadku wyróżnia się testy:

- Testy akceptacyjne – wykonywane w celu sprawdzenia na ile dane oprogramowanie działa zgodnie z zaleceniami klienta,
- Testy funkcjonalne – testy sprawdzające działanie oprogramowania zgodnie z ich specyfikacją, warto przy tym pamiętać, że testy akceptacyjne są rodzajem testów funkcjonalnych, ale niekiedy klient do akceptacji produktu wymaga również wyników testów jednostkowych,
- Testy regresywne – ważne testy, których podstawową rolą jest sprawdzenie, czy dodając do oprogramowania nową funkcjonalność czy poprawiając błędy, nie naruszana jest jego ogólna funkcjonalność; testy tego typu należy wykonywać na poziomie kodu aplikacji, o ile jest to możliwe, oraz na poziomie działania całej aplikacji,
- Testy wydajnościowe i obciążeniowe,

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

- Testy instalacyjne (testy konfiguracji) – ich celem jest sprawdzenie jak dane oprogramowanie zachowuje się na różnych platformach sprzętowych, systemach operacyjnych, różnych wersjach systemów itd.,
- Testy wersji alfa i beta – celem tych testów jest zdobycie od użytkowników informacji zwrotnej: wybranej grupie przekazywana jest wstępna wersja produktu, a następnie zbiera się od jej członków opinie i uwagi odnośnie jego działania,
- Testy używalności (z ang. usability tests) – ich celem jest sprawdzenie, jak szybko potencjalni użytkownicy będą w stanie opanować działanie danej aplikacji,
- Testy post-awaryjne – służą do sprawdzenia czy dana aplikacja działa poprawnie po wystąpieniu sytuacji awaryjnej, to niekiedy bardzo ważny rodzaj testów.

5. Główne typy narzędzi ułatwiających testowanie

5.1. Narzędzia do zarządzania testami i raportami

Narzędzia tego typu dostarczają interfejsów do wykonywania zadań, śledzenia defektów oraz zarządzania wymaganiami. Ponadto wspierają analizę ilościową i raportowanie odnośnie obiektów testów. Wspierają również śledzenie powiązań między obiektami testów. Mogą posiadać własne mechanizmy zarządzania wersjami bądź interfejs do zewnętrznych narzędzi zarządzających testami.

5.2. Narzędzia do automatyzacji

Narzędzia te pozwalają na automatyzację czynności, które wymagają wielkich nakładów pracy. Dzięki nim można w znacznym stopniu zwiększyć efektywność czynności testowych poprzez automatyzację powtarzających się zadań i wsparcie dla czynności testowych wykonywanych w sposób ręczny takich jak planowanie testów, projektowanie testów czy

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

raportowanie testów. Automatyzacja jest dobrym rozwiązaniem w przypadku projektów długoterminowych.

5.3. Narzędzia do zarządzania incydentami

Narzędzia te znacznie ułatwiają rejestrację incydentów oraz śledzenie ich aktualnych statusów. Niekiedy oferują również funkcje śledzenia i kontrolowania przepływu pracy związanego z przydziałem i naprawą. Pozwalają także na raportowanie.

5.4. Narzędzia do wykonywania testów

Narzędzia tego rodzaju uruchamiają skrypty zaprojektowane do implementacji testów przechowywanych elektronicznie.

Nagrywanie akcji, które są wykonywane prze testy ręczne jest dość abstrakcyjne, ale nie skaluje się do dużej liczby zautomatyzowanych skryptów testowych. Nagrany skrypt to liniowa reprezentacja z określonymi danymi i akcjami, które są jego częścią. Tego typu skrypt niekiedy może okazać się niestabilny, gdy wystąpią zdarzenia nieprzewidziane.

Dobrze skonstruowany system zarządzania incydentami powinien zapewniać:

a) elastyczne narzędzia, które redukują natłok zdarzeń prezentowanych administratorowi,
b) skalowalny interfejs graficzny, który prezentuje selektywną informację określonym osobom,
c) metody powiadamiania (telefon, e-mail, pager) danych grup obsługi technicznej zarządzania oraz użytkowników o ewentualnych problemach,
d) narzędzia do korelowania zdarzeń, które skracają czas poszukiwania i izolowania uszkodzeń i nie wymagają angażowania ekspertów do pracy,
e) narzędzia wspomagające szybą ocenę wpływu uszkodzenia na biznes.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

5.5. Narzędzia do testów wydajnościowych

Narzędzia tego typu wspierają testowanie wydajnościowe i mają na ogół dwie funkcjonalności – generują obciążenia oraz mierzą transakcje. Generowanie obciążenia można symulować poprzez dużą liczbę użytkowników bądź dużą liczbę wprowadzanych danych. W czasie przeprowadzania testów pomiary są logowane jedynie z wybranych transakcji. Narzędzia do testów wydajnościowych najczęściej dostarczają raporty, które są oparte na logowanych transakcjach i wykresy pokazujące obciążenie w zależności od czasów odpowiedzi.

5.6. Narzędzia do przygotowywania danych testowych

To narzędzie pozwalające na wybranie danych z istniejącej już bazy danych bądź na ich stworzenie, wygenerowanie, przetworzenie oraz edycję.

6. Testy automatyczne

Wyróżnia się kilka rodzajów testów automatycznych:

- Testy modułowe (tworzone przez programistów dla programistów) – celem jest testowanie pojedynczych modułów danego programu,
- Testy obciążeniowe (tworzone i wykonywane przez testerów) – testowanie w celu określenia wydajności danego programu,
- Testy funkcjonalne (tworzone i wykonywane przez testerów) – testowanie przeprowadzane jest na podstawie analizy specyfikacji funkcjonalności danego systemu lub wybranego modułu.

Do automatyzacji nadaje się aplikacja, która jest stabilna. Wówczas koszty utrzymania testów automatycznych są stosunkowo niewielkie. Każda aplikacja, która jest przeznaczona do

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

automatyzacji, powinna posiadać odpowiednią dokumentację. Gdy dokumentacja jest błędna lub niekompletna, testy nie zostaną dobrze zaprojektowane.

Automatyzować należy przede wszystkim te obszary aplikacji, które są wykorzystywane najczęściej. Dzięki testom automatycznym skróceniu ulega bowiem czas testowania. Ponadto testy automatyczne wyręczają testera w wielokrotnym przechodzeniu tej samej ścieżki. Testy tego rodzaju są również przydatne z uwagi na ilość danych testowych potrzebnych do przeprowadzenia testów. W ten sposób można bowiem przygotować zestaw danych testowych niezbędnych do testów danej aplikacji.

Testy automatyczne bywają problemem, gdy automatyzacji ma zostać poddana aplikacja złożona. Wówczas ze względu na dużą ilość możliwych przypadków, ciężko jest przewidzieć i zautomatyzować wszystkie przypadki, tak aby test pozostał stabilny. Lepiej też wstrzymać się z automatyzacją aplikacji, które często są poddawane modyfikacjom. Ma to związek z kosztownym utrzymaniem takich testów, które wymagają ciągłej pracy.

7. Kto powinien testować oprogramowanie?

Testy manualne na ogół wykonują testerzy. Z kolei automatyczne testy regresyjne powinno uruchamiać również modyfikujący oprogramowanie programiści. Dzięki temu mogą oni na bieżąco kontrolować, czy nie naruszyli ważnej funkcjonalności. Poza tym mogą również wykryć przypadek nieodpowiedniej modyfikacji skryptu testowego, gdzie nie uwzględniono zmian wynikających ze specyfikacji itd. Programista powinien mieć również możliwość zgłoszenia wymaganych zmian skryptu.

Obecnie pozycja testera w zespole projektowym staje się coraz bardziej istotna. Systemy informatyczne są dość skomplikowane, a dobry tester zna oprogramowania, którym na co

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

dzień się zajmuje. Wie także jak działa ono w całości. Tymczasem programiści niekiedy koncentrują się tylko na wybranych komponentach, nad którymi aktualnie pracują. Okazuje się więc, że tester jako osoba, która zna całość, może być bardzo pomocna.

Tester jest również w pewnym sensie przedstawicielem klienta. To on pomaga programistom i architektom w zrozumieniu, jak działa zwykły użytkownik oprogramowania. Dlatego też warto dać testerom pewną swobodę i włączyć ich do aktywnej pracy nad oprogramowaniem. Dobrze jest także słuchać uwag testerów, które mogą być niekiedy bardzo pomocne dla innych członków zespołu.

8. Kiedy należy testować oprogramowanie?

Częstotliwość wykonywania testów zależy przede wszystkim od ich typu. Testy regresyjne wykonywać należy regularnie, najlepiej po każdej zmianie oprogramowania, tak aby w każdej chwili można było sprawdzić, czy wprowadzane zmiany nie naruszyły nieodpowiednich funkcjonalności programu. Klasyczne testy regresyjne wykonuje się raz na dobę. Z kolei testy adaptacyjne przeprowadzane są stopniowo, gdy określona przez klienta funkcjonalność jest uznawana za gotową. Nie powinno się ich pozostawiać na sam koniec projektu. Może się bowiem wówczas okazać, że przeoczono istotny element danej funkcjonalności, a prace nad projektem są ukończone. Ich wprowadzenie mogłoby zaburzyć stabilność danego oprogramowania. Na bieżąco dobrze jest wykonywać również testy wymagań niefunkcjonalnych.

9. Uwagi

Tworzenie testów jest swojego rodzaju sztuką. Utworzenie testu, który łatwo jest ukończyć z wynikiem pozytywnym to nic trudnego, ale wyzwaniem jest skonstruowanie trudnego, precyzyjnego testu dla danej funkcjonalności. Warto jednak pamiętać o tym, że fakt przejścia nawet przez najlepszy zestaw testów nie jest gwarancją, że dane oprogramowanie zawsze

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

będzie działać poprawnie. Testowanie oprogramowania pozwala bowiem na znalezienie błędów, ale nie na wykazanie ich braku.

Niekiedy u testerów pojawia się chęć testowania na bardzo szczegółowym poziomie i testowania wszystkiego. Tymczasem wiadomo, że każde oprogramowanie z czasem będzie podlegać modyfikacjom, nie warto więc tworzyć testów dla elementów, które prawdopodobnie wkrótce całkowicie się zmienią czy zostaną usunięte.

Czasami myli się ze sobą dwa pojęcia, a więc testowanie oprogramowania i zapewnienie jakości (z ang. quality assurance – QA). Testowanie to istotny element QA, ale zapewnienie jakości jest pojęciem znacznie szerszym i obejmuje również zbieranie wymagań, zarządzanie zmianami itd.

10. Kiedy należy przestać testować?

W zasadzie testować można bez końca. Nie można jednak kontynuować testów, aż do chwili znalezienia każdego błędu w danym systemie. W końcu zawsze przychodzi moment na dostarczenie produktu klientowi. Kiedy jednak można uznać oprogramowanie za wystarczająco dobrze przetestowane?

Według zasady Project Triangle testowanie mieści się między budżetem przeznaczonym na realizację projektu, czasem na jego wykonanie oraz jakością dostarczonego rozwiązania. Najbardziej pesymistyczny scenariusz zakłada przerwanie testowania w chwili, gdy jeden z tych zasobów zostanie wyczerpany. Ma to miejsce wówczas, gdy:

- wyczerpany zostanie budżet projektu,
- nadejdzie data wypuszczenia oprogramowania na rynek,
- upłynie czas przeznaczony na testy alfa czy beta,
- określony z góry procent wykonanych przypadków testowych uzyska status „passed”,
- pokrycie kodu/funkcjonalności/wymagań osiągnie określony z góry poziom,
- liczba wykrytych błędów spadnie poniżej danego progu.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

W optymistycznej wersji, zakończenie testowania kończy się, gdy:

- oprogramowanie spełni określone wymagania jakościowe,
- korzyści z kontynuowania testów nie będą rekompensować kosztów ponoszonych z tego tytułu.

Ma to miejsce wówczas, gdy:

- częstotliwość występowania błędów osiągnie akceptowalny (niski) poziom,
- pokrycie kodu/funkcjonalności/wymagań osiągnie określony procent,
- liczba wykonań przypadków testowych, zakończonych sukcesem osiągnie określony poziom.

11. Cechy testera i programisty

Proces produkcji oprogramowania wymaga zaangażowania ze strony całej grupy projektowej. Szczególnie istotna jest jednak rola testerów i programistów. Obie te grupy mają zupełnie inne spojrzenie na ten sam projekt. Dlatego dla powodzenia projektu tak ważna jest odpowiednia komunikacja między testerami i programistami.

W dobrze funkcjonującym zespole projektowym testerzy i programiści wzajemnie się uzupełniają, dopełniają się wiedzą, umiejętnościami oraz doświadczeniem. Podstawowym błędem jest postrzeganie testerów jako młodszych programistów, a przy tym zachęcanie ich do rozwijania umiejętności itd. Tymczasem w rzeczywistości dobry tester posiada te same cechy, co dobry programista. Warto, aby wiedział o tym manager, który tworzy zespół do pracy nad danym projektem.

Wciąż wielu programistów nie zdaje sobie sprawy z tego, że testowanie jest zadaniem trudnym i wymagającym. W tym przypadku trzeba wykazywać się cierpliwością, dokładnością, elastycznością, umiejętnością dostrzegania detali itd. Dlatego wielu testerów

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

frustruje współpraca z programistami, którzy testowanie uważają niekiedy za pracę drugiej kategorii lub za coś, czym zajmować się może każdy i to bez większego wysiłku.

W praktyce testerzy potrzebują tego samego rodzaju wiedzy, co końcowi użytkownicy danego systemu. W ten sposób mogą użytkować produkt w taki sam sposób, jak robi to klient, a nie tak jak oczekiwaliby tego programiści. Czy znajomość wewnętrznej architektury aplikacji pomaga więc, a może przeszkadza w przeprowadzaniu testów? Na ogół nie jest potrzebna do wykonywania testów.

Testerzy powinni więc przejawiać pewny rodzaj ignorancji, a nawet naiwności w stosunku do testowanego przez siebie systemu. Na to samo nie mogą sobie pozwolić programiści. Dlatego też programiści niekiedy uznają testerów za osoby, które nie posiadają podstawowej wiedzy i zadające głupie wręcz pytania. Dobry tester musi mieć natomiast wiedzę na temat wymagań stawianych przez klienta oraz odnośnie działania danego systemu od środka (w pewnym zakresie). W ten sposób może na niego spojrzeć od innej strony niż programista.

Jakie są cechy dobrego testera?

- nie boi się oznajmiać złych informacji,
- potrafi odróżnić błąd od osoby, która jest za niego odpowiedzialna,
- informacje o znalezionych błędach przedstawia w sposób neutralny, tak aby nikogo nie obrazić.

12. Komunikacja między testerem a programistą

Testerzy nie powinni obawiać się konfrontacji z programistami. Ich pracą jest w końcu poszukiwanie błędów i defektów, co jednak nie zawsze jest dobrze przyjmowane. Czasami

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

określenie przyczyny błędu nie jest łatwe. Być może stoi za nim pomyłka w kodzie lub dokumentacji. Czasami nawet błędu nie ma, a tester źle interpretuje wyniki działania danej aplikacji.

Zawsze podstawowym zadaniem testera jest zgłoszenie błędu. Jednak niektórzy testerzy zgłaszają błąd oraz przypisują od razu do niego programistę, który jest za niego odpowiedzialny. Nie należy tego robić. Spekulowanie i szukanie winnego nie pomaga nikomu, a na pewno nie ułatwia skupieniu się na wykonywanej pracy. Powoduje również konflikty w zespole projektowym.

13. Słownik pojęć podstawowych

Poniższe pojęcia zostały zaczerpnięte z dokumentu „Słownik wyrażeń związanych z testowaniem", który został wydany przez Stowarzyszenie Jakości Systemów Informatycznych w roku 2013 (wersja 2.2.2.). Znaleźć w nim można znacznie większą ilość pojęć związanych z testowaniem oprogramowania.

Analiza pokrycia – pomiar pokrycia, który uzyskiwany jest w czasie wykonywania testów według określonych na wstępie kryteriów, przeprowadzany jest w celu określenia, czy potrzebne są jeszcze dodatkowe testy, jeśli tak, to należy podjąć decyzję, które przypadki testowe powinno się wykonać.

Analiza ryzyka – proces oceny zidentyfikowanych ryzyk, który ma na celu oszacowanie ich wpływu i prawdopodobieństwo realizacji.

Atak na oprogramowanie – działanie ukierunkowane, które ma na celu ocenę jakości, a przede wszystkim niezawodności obiektu testów, poprzez wymuszenie wystąpienia danej awarii.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Bezpieczeństwo – zdolność oprogramowania do osiągania akceptowalnych poziomów ryzyka wystąpienia szkody w stosunku do ludzi, oprogramowania, majątku, biznesu bądź środowiska w danym kontekście użycia.

Błąd – ludzkie działanie, które powoduje powstanie nieprawidłowego rezultatu.

Cykl testowy – przeprowadzenie procesu testowego w stosunku do pojedynczego, możliwego do określenia wydania testowanego produktu.

Cykl życia oprogramowania – okres, który rozpoczyna się w momencie pojawienia się pomysłu na oprogramowanie, w kończy w chwili, gdy oprogramowanie wychodzi an rynek; na ogół cykl ten zawiera fazę koncepcji, fazę wymagań, fazę projektowania, fazę implementacji, fazę testów, fazę instalacji i zastępowania, fazę wykorzystywania produkcyjnego i pielęgnowania, a niekiedy i fazę wycofywania.

Dane testowe – dane istniejące w bazie danych przed wykonaniem testu, które mają wpływ na poddawany testom moduł lub system bądź na które wywierany jest wpływ przez testowany system lub moduł.

Debugowanie – wyszukiwanie, analizowanie i usuwanie przyczyn awarii oprogramowania; to czynność wykonywana przez programistów.

Defekt - wada modułu bądź systemu, która powoduje, że dany moduł lub system nie wykona określonej czynności; do defektów zalicza się m.in. niepoprawną definicję danych; defekt pojawiający się podczas uruchamiania programu może prowadzić do awarii modułu lub całego systemu.

Faza testów – zbiór aktywności testowych zebrany w podlegającą zarządzaniu fazę projektu.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Gęstość usterek – liczba usterek, które znaleziono w module czy systemie przypadająca na jednostkę wielkości modułu lub systemu (wyrażana w liniach kodu, liczbie klas lub w punktach funkcyjnych).

Harmonogram testów – lista aktywności, zadań i zdarzeń z procesu testowego, która określa planowaną datę rozpoczęcia i zakończenia testów.

Incydent – zdarzenie, które wymaga zbadania.

Infrastruktura testu – artefakty organizacyjne, które są niezbędne do przeprowadzenia testów; składają się ze środowisk testowych, narzędzi testowych, procedur i odpowiedniego wyposażenia biurowego.

Integracja – proces łączenia modułów i systemów w większe zespoły.

Jakość oprogramowania – funkcjonalności i cechy oprogramowania, które wpływają na jego zdolność do zaspokajania stwierdzonych i przewidywanych potrzeb.

Kryterium wejścia – zbiór ogólnych i szczegółowych warunków, które muszą zostać spełnione, aby kontynuować proces dla danego zadania (np. fazy testów); kryteria wejścia ustalane są w celu ochrony przed rozpoczęciem zadania, gdy wiąże się to z większym nakładem pracy niż ten, który potrzebny jest do osiągnięcia stanu spełnienia danego kryterium wejścia.

Kryterium wyjścia – zbiór ogólnych i szczegółowych warunków, które zostały uzgodnione z udziałowcami, a których spełnienie jest konieczne do oficjalnego zakończenia procesu; kryteria wyjścia ustala się w celu ochrony przed uznaniem danego zadania za zakończone, w sytuacji, gdy jego niektóre elementy nie są w pełni wykonane; kryteria wyjścia są na ogół stosowane jako argument przeciwko zakończeniu testów.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Małpie testowanie – metoda testowania, która polega na losowym wybieraniu z szerokiego zakresu wejść oraz na losowym naciskaniu przycisków; ignorowany jest sposób, w jaki dany produkt powinien być używany.

Maskowanie defektów – sytuacja, w której występowanie jednego defektu nie pozwala nw wykrycie drugiego.

Podejście do testu – implementacja strategii testów dla danego projektu; na ogół zawiera decyzje podjęte na podstawie celów i analizy ryzyka projektu, punkty startowe procesu testowego, techniki projektowania testów, kryteria wyjścia oraz typy testów do wykonania.

Podstawowy zestaw testów – zestaw przypadków testowych, który powstał na podstawie wewnętrznej struktury modułu lub specyfikacji, a który zapewnia osiągnięcie w 100% założonego kryterium pokrycia.

Pokrycie – proces przypisania liczby lub kategorii do danego obiektu, a który ma na celu opisanie danej właściwości obiektu.

Pokrycie kodu – analityczna metoda, która określa jakie części programu zostały pokryte (wykonane) przez zestaw testowy, a jakie części nie zostały pokryte.

Pokrycie ścieżek – procent ścieżek w danym module, które zostały wykonane przez zestaw testowy; realizacja 100% pokrycia ścieżek to 100% pokrycia LSKiS.

Proces testowy – podstawowy proces testowy, który składa się z kilku faz: fazy planowania testów i kontroli, fazy analizy i projektowania testów, fazy implementacji i wykonania testów, oceny kryteriów wyjścia i raportowania oraz czynności związanych z zakończeniem testów.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Przypadek testowy – zbiór danych wejściowych, podstawowych warunków wykonania, oczekiwanych skutków oraz końcowych warunków wykonania, który opracowany jest w danym celu lub dla warunku testowego.

Przypadek użycia – ciąg transakcji w dialogu między użytkownikiem i systemem z namacalnym wynikiem.

Retestowanie – testowanie, które polega na uruchomieniu przypadków testowych, które w czasie ostatniego włączenia wykryły defekty; celem jest sprawdzenie poprawności naprawy błędu.

Rezultat fałszywie niezaliczony – test, w którym defekt został zgłoszony, mimo że faktycznie go nie ma.

Rezultat fałszywie zaliczony – test, w którym nie stwierdzono obecności defektu, który w danym obiekcie istnieje.

Ryzyko – czynnik, który prowadzić może w przyszłości do negatywnych konsekwencji; na ogół opisywany jest jako wpływ i prawdopodobieństwo.

Rzeczywisty rezultat – wytworzone i zaobserwowane zachowanie się danego modułu lub systemu w czasie jego testowania.

Skrypt testowy – nazwa specyfikacji procedury testowej, w szczególności procedury automatycznej.

Specyfikacja testów – dokument, który zawiera specyfikację projektu testów, specyfikacje przypadków testowych oraz specyfikację procedury testowej.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Spotkanie retrospektywne – spotkanie na zakończenie projektu, w czasie którego członkowie zespołu projektowego oceniają dany produkt i wyciągają wnioski, które można wykorzystać przy okazji realizacji kolejnego projektu.

Stabilność – zdolność danego produktu oprogramowania do unikania dziwnych, niespodziewanych zachowań z modyfikacji w oprogramowaniu.

Test dymny – podzbiór wszystkich zdefiniowanych i zaplanowanych przypadków testowych pokrywających podstawowe funkcjonalności modułu bądź systemu, który ma na celu potwierdzenie, że najważniejsze funkcjonalności programu działają; podczas tworzenia oprogramowania powinno się codziennie budować testy dymne.

Test wstępny – specyficzny typ testu dymnego, którego celem jest podjęcie decyzji czy dany moduł bądź system są gotowe do dalszego testowania; na ogół wykonuje się go na początku fazy wykonywania testów.

Testalia – wszystkie dokumenty i narzędzia wytworzone i wykorzystywane w czasie procesu testowania, a które są niezbędne do planowania, projektowania i wykonywania testów; zalicza się do nich dokumentację, skrypty, oczekiwanie rezultaty, wejścia, pliki, bazy danych, procedury, środowiska oraz wszystkie narzędzia wykorzystywane w czasie testowania.

Testowanie dokumentacji – kontrolowanie jakości dokumentacji np. opisu instalacji czy podręcznika użytkowania.

Testowanie dopasowania – proces testowania, którego celem jest zapewnienie dopasowania oprogramowania do potrzeb.

Testowanie interfejsu – testowanie, które wykonywane jest w celu wykrywania błędów w interfejsach między modułami.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Testowanie kombinatoryjne – sposób na identyfikowanie odpowiedniego podzbioru kombinacji testów dla osiągnięcia uprzednio zdefiniowanego poziomu pokrycia; kiedy testowany jest obiekt wieloparametrowy, a każdy z tych parametrów ma kilka wartości, pojawia się więcej kombinacji niż można w danym czasie przetestować.

Testowanie konsultacyjne – testowanie, które prowadzone jest przy współpracy i pod nadzorem ekspertów biznesowych, którzy nie wchodzą w skład zespołu testowego.

Testowanie konwersji – testowanie programów wykorzystywanych do przenoszenia danych z systemów istniejących do nowych.

Testowanie negatywne – testowanie, którego celem jest wykazanie, że dane oprogramowanie nie działa prawidłowo; testowanie to ma związek przede wszystkim z postawą testerów i techniką projektowania testów (np. testowanie z błędnymi wartościami wejściowymi).

Testowanie parami – testowanie, w którym dwie osoby (dwóch testerów, programista i tester, klient i tester) pracują razem w celu wykrycia błędów; na ogół praca odbywa się na jednym komputerze.

Testy ad-hoc (testy eksploracyjne) – testy wykonywane bez formalnego planu testów, dokumentacji czy przypadków użycia; to najmniej formalna metoda testowania; przeprowadzanie tego typu testów pomaga testerom nauczyć się danej aplikacji przed rozpoczęciem planowanych rodzajów testów.

Testy end-to-end – testy sprawdzające czy cały flow danej aplikacji działa zgodnie z założeniami początkowymi; testy te mają za zadanie potwierdzić czy aplikacja spełni oczekiwania użytkownika końcowego.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Użyteczność – zdolność oprogramowania do bycia używanym zrozumiałym, łatwym w obsłudze czy atrakcyjnym dla użytkownika końcowego.

Walidacja – sprawdzenie poprawności i dostarczenie obiektywnych dowodów, że dany produkt spełnia potrzeby i wymagania jego użytkownika.

Współczynnik awarii – stosunek liczby awarii w danej kategorii do określonej jednostki miary (awarie na jednostkę czasu, awarie na liczbę uruchomień komputera czy awarie na liczbę transakcji).

Wstrzykiwania defektów – zamierzone dodawanie do systemu defektów w celu wykrycia, czy można je wykryć i pracować mimo ich istnienia; wstrzykiwanie błędów imituje błędy, które mogą się pojawić w procesie produkcji.

Wydajność – stopień, w którym moduł lub system realizują swoje, przypisane im funkcje w danych ramach czasu przetwarzania i przepustowości.

Zablokowany przypadek testowy – przypadek testowy, którego nie da się wykonać, gdyż warunki wstępne dla niego nie mogą zostać osiągnięte.

Zarządzanie defektami – proces, który składa się z rozpoznania, analizy, prowadzenia działań oraz likwidacji usterek; to proces rejestracji usterek, ich klasyfikowania oraz określania ich wpływów.

Zarządzanie incydentami – proces, który składa się z rozpoznania, analizy, prowadzenia działań oraz likwidacji incydentów; to proces rejestracji incydentów, ich klasyfikowania oraz określania ich wpływów.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zarządzanie jakością – całość skoordynowanych działań, których celem jest kierowanie organizacją oraz jej kontrolowanie pod kątem jakości; pojęcie to obejmuje na ogół definiowanie polityki jakościowej i celów jakościowych, planowanie jakości, kontrolowanie jakości, poprawę jakości i zapewnienie jakości.

Zarządzanie ryzykiem – regularne wdrażanie procedur i praktyk dla identyfikowania, analizowania, ustalania priorytetów oraz kontrolowania ryzyka.

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Bibliografia

1. http://www.wstt.edu.pl/pliki/materialy/mta/metodologia_testowania_aplikacji_c2.pdf
2. http://autentika.pl/blog/automatyzacja-testow-serwisu-internetowego
3. http://itcraftsman.pl/testy-automatyczne-interfejsu-uzytkownika-przy-uzyciu-selenium-ide/
4. http://rst.com.pl/rst-blog/automatyzacja-testow-selenium/
5. http://qa-24.pl/poczatki-selenium-ide/
6. http://docs.seleniumhq.org/projects/webdriver/
7. Meyers Glenford J., Sandler Cory, Badgett Tom, Thomas Todd M. „Sztuka testowania oprogramowania", Wydawnictwo Helion, 2005.
8. Słownik testerski dla Poziomu Podstawowego i Zaawansowanego + tester zwinny (v.2.3) http://sjsi.org/wp-content/uploads/2014/10/slownik-termin%C3%B3w-testowych-ver-2.3PL.pdf
9. Sylabus ISTQB, poziom podstawowy http://sjsi.org/wp-content/uploads/2013/08/sylabus-poziomupodstawowego-wersja-2011.1.1_20120925.pdf

Fundusze Europejskie Wiedza Edukacja Rozwój
Unia Europejska Europejski Fundusz Społeczny
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.