Jak usprawnić proces testowania oprogramowania: procedury, metodologia i narzędzia.

Streszczenie

W artykule są przedstawione sposoby usprawnienia i automatyzacji procedur testowania oprogramowania. Omówione są podstawowe rodzaje testów, możliwości wykonywania ich automatycznie.

Ponadto zobaczymy jak wygląda proces wdrożenia systemu testowania zarówno od strony merytorycznej, czyli tworzenia samych testów, jak i od strony odpowiednich narzędzi, które w znaczący sposób pozwalają usprawnić proces testowania w kontekście całego projektu informatycznego

Autor

Autorem artykułu jest *Piotr Kochański*, konsultant firmy *Erudis*.

Kontakt:

e-mail: p.kochanski@erudis.pl

tel: (22) 855-34-55

1. Wstęp

Zacznijmy od dwóch historii.

4 czerwca 1996 roku bezzałogowa rakieta kosmiczna Ariane 5 eksplodowała po czterdziestu sekundach lotu, zamieniając w popiół 7 miliardów dolarów wydane przez 10 lat na jej budowę oraz ładunek, który był wart 500 milionów dolarów. Po dwóch tygodniach zespół badający przyczyny katastrofy przedstawił raport: okazało się, że źródłem katastrofy był prosty błąd w oprogramowaniu, związany z niepoprawną konwersją liczby zmiennoprzecinkowej na liczbę całkowitą.

W 1997 roku firma Mercedes-Benz, znana z produkcji dużych, luksusowych samochodów wypuściła na rynek swój pierwszy „mały" model – Mercedesa klasy A. Samochód przyciągnął uwagę mediów, zwłaszcza, gdy okazało się, że przeprowadzone testy bezpieczeństwa jazdy nowego produktu Mercedesa wypadły bardzo niezadowalająco. Okazało się, że samochód przewraca się podczas „testu łosia", polegającego na wykonaniu gwałtownego skrętu najpierw jedną a potem w drugą stronę. Firma od razu zabrała się za usuwanie usterek – jedynym elementem, który wymagał poprawienia było oprogramowanie komputera sterującego systemem amortyzatorów.

Przez długi czas problemy związane ze złym działaniem różnego rodzaju urządzeń były związane przede wszystkim z kwestiami mechanicznymi bądź elektrotechnicznymi. Dzisiaj coraz częściej okazuje się, że najbardziej zawodnym elementem jest oprogramowanie. Niewątpliwie jakość oprogramowania ma coraz bardziej znaczący wpływ na nasze bezpieczeństwo i możliwość normalnego funkcjonowania. Bazy danych z informacjami o ocenach ze studiów, chorobach, które przeszliśmy, systemy informatyczne obsługujące banki, obliczające wysokość składki ubezpieczeniowej muszą działać niezawodnie, w innym przypadku powodują duże, czasem nieodwracalne szkody.

Poniższy artykuł jest poświęcony jednemu z najważniejszych aspektów zapewniania jakości oprogramowania (ang. *quality assurance*): testowaniu. Omówimy najważniejsze metody testowania oprogramowania ze szczególnym naciskiem na to, jak sprawnie wpisać procedury testowania w cały proces tworzenia aplikacji.

Zajmiemy się między innymi następującymi zagadnieniami:

- Co, jak, kiedy i przez kogo powinno być testowane
- W jaki sposób zorganizować i zarządzać procesem testowania: plan testów, repozytorium testów, dokumentacja.
- Jak można zautomatyzować testowanie
- Jakich narzędzi warto użyć do testowania i zarządzania testami w kontekście całego projektu informatycznego

Obecnie systemy informatyczne są na tyle złożone, że sprawne zarządzanie projektem przez cały cykl życia aplikacji staje się bardzo pracochłonne, o ile nie wspomagamy się odpowiednimi narzędziami. Testowania oprogramowania powinno być integralną częścią tego cyklu, zatem i narzędzia użyte do testowania muszą mieć odpowiednią funkcjonalność.

2. Testowanie oprogramowania – rodzaje testów

Oprogramowanie testujemy na bardzo różne sposoby, zależnie od tego jaki aspekt jego działania jest w danym momencie dla nas ważny. Jeżeli oprogramowanie podlega szybkim zmianom i zależy nam na tym by ich wprowadzanie nie naruszało istniejącej funkcjonalności stosujemy testy regresyjne. Z kolei jeśli chcemy pokazać klientowi, że aplikacja działa zgodnie z jego wymaganiami przygotowujemy testy akceptacyjne. Testy obciążeniowe będziemy stosować w przypadku aplikacji, co do których musimy mieć pewność, że działają wydajnie.

Testy można klasyfikować na różne sposoby, ze względu na to, co i w jaki sposób podlega procedurą testowym. Poniżej krótko przyjrzymy się wybranym taksonomią i rodzajom testów, poznając przy tym ich zastosowanie.

2.1 Testy strukturalne i funkcjonalne.

Do testowania oprogramowanie można podchodzić na dwa odmienne sposoby.

Pierwszy sposób to testy ***testy funkcjonalne***. Polegają one na tym, że wcielamy się w rolę użytkownika, traktując oprogramowanie jak „czarną skrzynkę", która wykonuje określone zadania. Nie wnikamy wogóle w techniczne szczegóły działania programu. Testy te często są nazywane testami ***czarnej skrzynki*** (***black box testing***).

Drugi sposób postępowania to ***testy strukturalne***. Tym razem tester ma dostęp do kodu źródłowego oprogramowania, może obserwować jak zachowują się różne części aplikacji, jakie moduły i biblioteki są wykorzystywane w trakcie testu. Te testy czasami są nazywane testami ***białej skrzynki*** (***white box testing***).

Typowym przykładem testów strukturalnych są ***testy jednostkowe*** (***unit tests***), które polegają na tym, że tester lub sam programista tworzy kod, którego jedynym zadaniem jest sprawdzenie działania produkcyjnego kodu aplikacji.

2.2 Testy manualne i automatyczne

Ten podział jest oczywisty. Testy mogą być wykonywane ręcznie przez testera, który przechodzi przez interfejs użytkownika zgodnie z określoną sekwencją kroków lub automatyczne, których wykonanie nie wymaga udziału testera.

Zazwyczaj zautomatyzowane jest przeprowadzanie testów jednostkowych. Zrobienie tego jest dość proste, gdyż istnieją takie narzędzia jak ***Jakarta Ant***, które mają wbudowaną funkcjonalność uruchamiania testów jednostkowych.

Znacznie bardziej skomplikowaną sprawą jest automatyzacja testów w schemacie czarnej skrzynki. Potrzebne do tego jest specjalistyczne oprogramowanie, które pozwala uruchamiać uprzednio napisane lub nagrane przez testera skrypty. Przykładem takiego oprogramowania jest oferowany przez nas QA Wizard.

2.3 Zakres testów

Powyższe dwie klasyfikacje dzieliły testy ze względu na sposób ich przeprowadzania, teraz będziemy chcieli określić zakres aplikacji jaki obejmują testy.

- ***Testy jednostkowe*** **testują oprogramowanie na najbardziej podstawowym poziomie – na poziomie działania pojedynczych funkcji (metod).**
- ***Testy integracyjne*** **pozwalają sprawdzić jak współpracują ze sobą różna komponenty oprogramowania, obecnie rzadko mamy do czynienia z monolitycznymi aplikacjami, raczej są one tworzone modułowo, więc trzeba sprawdzić, czy wszystko razem działa poprawnie, nie ma niezgodności interfejsów itp.**
- ***Testy systemowe*** **dotyczą działania aplikacji jako całości, zazwyczaj na tym poziomie testujemy różnego rodzaju wymagania niefunkcjonalne: szybkość działania, bezpieczeństwo, niezawodność, dobrą współpracę z innymi aplikacjami i sprzętem.**

2.4 Podział testów ze względu na ich przeznaczenie

Ten podział jest niewątpliwie najbardziej interesujący, gdyż pozwala nam wybrać odpowiednio rodzaj testów zależnie od tego, do czego mają nam one służyć.

- ***Testy akceptacyjne*** **– testy wykonywane w celu sprawdzenia na ile oprogramowanie działa zgodnie z wymaganiami klienta**
- ***Testy funkcjonalne*** **– testy sprawdzające działanie oprogramowania zgodnie ze specyfikacją, oczywiście testy akceptacyjne są siłą rzeczy rodzajem testów funkcjonalnych. Aczkolwiek zdarza się, że klient wymaga do akceptacji produktu także wyników testów jednostkowych. Dlatego też odróżniamy kategorię testów funkcjonalnych i akceptacyjnych.**
- ***Testy regresyjne*** **– jest to bardzo ważny rodzaj testów, pełniących zasadniczą rolę jeśli traktujemy poważnie kwestię jakości oprogramowania. Celem testów regresyjnych jest sprawdzenie, czy dodając nową funkcjonalność, poprawiając błędy nie naruszyliśmy niespodziewanie innej funkcjonalności oprogramowania.**

  Testy regresyjne powinny być wykonywanie zarówno na poziomie kodu aplikacji (jeśli to możliwe) – zazwyczaj są to testy jednostkowe – jak i na wyższym poziomie, działania całej aplikacji. Aplikacja jest testowana w ten sposób, że przechodzimy przez wybrane ścieżki działania oprogramowania tak, jak by to robił jego użytkownik.
- ***Testy wydajnościowe i obciążeniowe***

Poza tymi, podstawowymi typami często używamy następujących testów:

- ***Testy instalacyjne/testy konfiguracji*** **– służą do tego, żeby sprawdzić, jak oprogramowanie zachowuje się na różnych platformach sprzętowych, systemach operacyjnych, różnych wersjach tych systemów, przy różnym zestawie oprogramowania, jakie może mieć odbiorca.**
- ***Testy wersji alfa i beta*** **– testy te służą głównie zdobyciu informacji zwrotnej od użytkowników – wybranej grupie przekazujemy wstępne wersje produktu, następnie zbieramy ich opinie i komentarze dotyczące działania produktu.**
- ***Testy używalności*** **– tzw. *usability tests*, służą temu by sprawdzić jak szybko potencjalni użytkownicy mogą opanować działanie aplikacji, na ile użyteczna i jasna jest dokumentacja, itp.**
- ***Testy post-awaryjne*** **– testy służące sprawdzeniu, czy aplikacja zachowuje się poprawnie po wystąpieniu sytuacji awaryjnej. W pewnych przypadkach jest to bardzo ważny rodzaj testów, np. przykład producent bazy danych powinien sprawdzić na ile awaria wpłynie na integralność przechowywanych danych.**

Proszę zwrócić uwagę na to, że różne wymienione kategorie testów mogą się pokrywać, na przykład do testów regresyjnych można użyć zarówno testów strukturalnych jak i funkcjonalnych.

3. Co, jak, kiedy i przez kogo ma być testowane

Wiemy już co mamy do dyspozycji, jakiego rodzaju testy są najczęściej używane. Musimy teraz zadecydować co wybrać. Musimy także określić co, jak kiedy i przez kogo będzie testowane. Odpowiedź na te pytania jest pierwszym krokiem jaki doprowadzi nas do utworzenia ***planu testów***. Więcej na ten temat dowiemy się później, gdy zajmiemy się procesem wdrażania testów w projekcie informatycznym.

3.1 Podjęcie decyzji co będziemy testować

Nie zawsze jesteśmy w komfortowej sytuacji, kiedy możemy sobie pozwolić od razu na testowanie wszystkiego, co tylko uznamy za warte sprawdzenia – często mamy ograniczenia czasowe i budżetowe. Przeważnie musimy się zdecydować na ograniczony zakres testów i ewentualnie później go rozszerzać. Z tego względu automatyzacja możliwie wielu czynności testowych jest bardzo ważna: dzięki użyciu odpowiednich narzędzi tymi samymi środkami i w tym samym czasie możemy przetestować więcej elementów systemu.

Najprostszą metodą wyboru testów, które powinny być utworzone w pierwszej kolejności jest uwzględnienie najczęściej występujących lub najważniejszych dla działania systemu przypadków użycia. Mogą nam w tym pomóc priorytety wymagań i przypadków użycia (z tego względu należy pamiętać, żeby zbierając wymagania zapisywać tę informację). Bardziej zformalizowane podejście tego typu jest znane jako *Metoda Najważniejszego Testu - The Most Important Test Method* albo *MIT Method* [Marnie L. Hutcheson, Software Testing Fundamentals: Methods and Metrics].

MIT, oprócz priorytetów wymagań, wskazuje analizę ryzyka jako jeden z dodatkowych elementów branych pod uwagę przy określaniu zakresu testowania. Wiedząc, że pewne elementy systemu są kluczowe dla działania całości, powinniśmy je testować w pierwszej kolejności.

Jeżeli jakaś wersja produktu jest już używana w środowisku produkcyjnym, to nieocenionym źródłem testów będą błędy znalezione przez użytkowników oraz informacje o tych elementach systemu, które są dla nich kluczowe. Warto w takim przypadku rozważyć przeprowadzenie wywiadów z użytkownikami bądź przygotowanie dla nich ankiet.

Podobnie jeżeli nie mamy gotowego produktu, ale istnieje jego prototyp, to jego można użyć jako pomoc w definiowaniu procedur testowania.

W dużej mierze zakres tego, co będziemy testować wynika z ograniczeń, jakie mamy, potrzeb oraz naszej wiedzy, doświadczenia i intuicji, które pozwalają przewidzieć jakie elementy systemu mogą sprawiać kłopoty.

3.2 Jak testować: wybór technik tworzenia testów i danych testowych

Podstawową techniką tworzenia testów jest ***analiza funkcjonalna***, to znaczy definiowanie testów w oparciu o istniejącą specyfikację: przypadki użycia i wymagania. Zakładamy, że taka specyfikacja istnieje i jest na tyle dobrze napisana, że można na jej podstawie opracować testy. Trzeba pamiętać także o wymaganiach niefunkcjonalnych, których testowanie jest często zaniedbywane.

Oprócz tego często posługujemy się ***analizą ścieżek***, czyli dostępnych dla użytkownika sposobów przejścia przez aplikację. Nie muszą one koniecznie odpowiadać oczekiwanym sposobom jej użycia. Pozwoli to zbadać te sytuacje, w których użytkownik będzie działał w sposób „twórczy”, inaczej niż to przewidujemy.

W wyborze danych do testowania pomoże nam analiza ***wartości brzegowych*** i ***podział danych wejściowych na klasy równoważności***. Do klasy równoważności trafiają wszystkie te rodzaje danych, które powinny wywołać to samo działanie systemu. Na przykład jeśli hasło

dostępu do naszego systemu nie może być krótsze niż 5 znaków, to wszystkie hasła krótsze niż 5 znaków trafiają do jednej klasy i są traktowane w ten sam sposób. Wystarczy wtedy testować tylko po jednym przedstawicielu każdej klasy równoważności, aby pokryć w ten sposób cały zakres wartości, które może podać użytkownik.

Analiza wartości brzegowych pozwala sprawdzić jak zachowuje się aplikacja, gdy są jej przekazywane dane z poza dozwolonego zakresu. Wyobraźmy sobie, że nasza aplikacja nie zezwala na przyjęcie danych liczbowych mniejszych niż 10. Sprawdzamy wtedy co się dzieje

- w dozwolonym zakresie (np. dla 15)
- w zabronionym zakresie (np. dla 3)
- na granicy stosowalności, czyli dla wartości 10

Ponadto sprawdzamy co się stanie, gdy przekażemy aplikacji zamiast liczby litery, znaki specjalne itp.

Kolejną często wykorzystywaną metodą jest metoda ***tablic decyzyjnych***, pozwala ona zredukować liczbę kombinacji parametrów testowych tak, żeby pominąć testy, które są redundantne.

Nie da się ukryć, że popularną techniką tworzenia testów jest ***testowanie ad-hoc***. W tym przypadku polegamy nie na formalnych wytycznych co do sposobu testowania, a bardziej liczymy na intuicję i doświadczenie testerów czy programistów, którzy będą potrafili „wyczuć" co tak na prawdę należy przetestować. Mimo, że ta metoda jest niezbyt „naukowa" to często przy jej pomocy można opracować testy, których potrzebę trudno by było wykryć jakąś formalną metodą.

Kolejną metodą testowania, także opierającą się na umiejętnościach i doświadczeniu testerów jest ***metoda eksploracyjna***. W tym przypadku tester jednocześnie poznaje aplikację, tworzy testy i je wykonuje. Decydując się na takie podejście nie tworzymy uprzednio pełnego, formalnego planu testów, tylko powstaje on dynamicznie, w czasie bezpośredniej pracy z aplikacją.

3.3 Kto powinien testować oprogramowanie

Testy manualne wykonują najczęściej testerzy, z kolei możliwość uruchomienia wykonywania automatycznych testów regresyjnych powinni mieć także programiści modyfikujący oprogramowanie. Dzięki temu na bieżąco mogą kontrolować, czy nie naruszyli istniejącej funkcjonalności. Będą mogli także wykryć sytuację, w której skrypt testowy nie został odpowiednio zmodyfikowany, tak, żeby uwzględnić zmiany wynikające ze specyfikacji lub jej zmian. Programista będzie mógł także zgłosić wymaganie zmiany skryptu.

Obecnie pozycja testera w zespole projektowym staje się coraz ważniejsza – systemy informatyczne są skomplikowane, a dobry tester zna oprogramowanie, którym się zajmuje, bardzo dokładnie. Orientuje się jak ono działa jako całość – programiści bardzo często koncentrują się tylko i wyłącznie na komponentach, nad którymi pracują. Osoba, która ma ogólne spojrzenie na wszystkie elementy razem może być bardzo pomocna. Tester jest także w pewnym sensie „przedstawicielem klienta", pomaga programistom, architektom zrozumieć jak rozumuje zwykły użytkownik.

Z tych powodów warto dać pewną swobodę testerowi, włączyć go do aktywnej pracy przy tworzeniu oprogramowania i słuchać jego uwag, bo może on wydatnie pomóc innym członkom zespołu.

3.4 Kiedy testować oprogramowanie

Częstotliwość wykonywania testów zależy od ich typu. Testy regresyjne powinny być wykonywane regularnie, po każdej zmianie oprogramowania, tak aby w każdej chwili budowy aplikacji można było sprawdzić, czy wprowadzane zmiany nie naruszyły tej funkcjonalności, która nie powinna ulec modyfikacją. Typowo testy regresyjne wykonywane są raz na dobę.

Z kolei testy akceptacyjne będą wykonywane stopniowo, gdy określona przez klienta funkcjonalność będzie uznana za gotową. Nie jest dobrą praktyką pozostawianie tych testów na sam koniec projektu. Może okazać się, że przeoczono jakiś niekoniecznie znaczący element potrzebnej klientowi funkcjonalności, ale jego dodanie w momencie gdy prace nad produktem są ukończone i działa on stabilnie prowadzą do poważnych perturbacji.

Podobnie warto na bieżąco testować wymagania niefunkcjonalne.

3.5 Uwagi

Tworzenie testów jest sztuką, nie jest niczym nadzwyczajnym utworzyć taki test, który będzie łatwo przejść z wynikiem pozytywnym. Prawdziwym wyzwaniem jest utworzenie trudnego testu, precyzyjnie testującego wybraną funkcjonalność – każde jej naruszenie będzie się kończyło negatywnym rezultatem.

Warto pamiętać, że fakt przejścia z sukcesem przez nawet najpełniejszy zestaw testów nie gwarantuje, że oprogramowanie na pewno będzie działać zawsze poprawnie, jak to powiedział Dijkstra: t*estowanie oprogramowania pozwala na znalezienie błędów, nigdy zaś na wykazanie ich braku*.

Kolejną rzeczą jest pokusa testowania na zbyt szczegółowym poziomie, próba testowania „wszystkiego". Wiadomo, że oprogramowanie będzie podlegać modyfikacją, nie warto więc tworzyć testów dla tych elementów, które z dużym prawdopodobieństwem mogą się całkowicie zmienić, albo wogóle zniknąć.

Często mylone są ze sobą dwa pojęcia: testowanie oprogramowania i zapewnianie jakości (quality assurance - QA). Testowanie jest na pewno ważnym elementem QA, ale zapewnianie jakości obejmuje znacznie większy zakres problemów, między innymi zbieranie wymagań, zarządzanie zmianami, wdrożenie, itp.

4. Testowanie a metodologie Agile

W ostatnich latach bardzo popularne są metodologie prowadzenia projektów informatycznych typu Agile (zwinne). Najbardziej znaną jest programowanie ekstremalne (XP – eXtreme Programming). Pomijając kontrowersyjną kwestię użyteczności tego typu metodologii zobaczmy, jak podchodzą one do testowania oprogramowania.

Metodologie tego typu charakteryzują się tym, że zamiast tworzyć na początku pełną specyfikację produktu i potem dopiero przystępować do pisania kodu staramy się jak najszybciej doprowadzić do zbudowania działającej wersji aplikacji, którą można pokazać odbiorcy. Powstaje ona na bazie wymagań dość ogólnie sformułowanych przez użytkowników. Wstępna wersja jest następnie modyfikowana, tak by uwzględnić ich kolejne wymagania co do produktu. Zakładamy tutaj, że użytkownik bierze intensywny udział w procesie rozwoju aplikacji.

Przy takim podejściu nie mamy do dyspozycji precyzyjnej specyfikacji wymagań, także nie może być ona bazą dla tworzenia testów. Testowanie oprogramowania przebiega w tym przypadku dwutorowo;

- programista tworzy testy regresyjne (zazwyczaj testy jednostkowe) – jest to jedna z podstawowych zasad XP
- użytkownik ma do dyspozycji aktualną wersję aplikacji, pracuje na niej, tym samym testując ją.

Na marginesie warto zwrócić uwagę na to, że przeciwstawianie sobie tradycyjnych metodologii takich jak *RUP* metodologią typu *Agile* jest często nadużyciem. Tradycyjne metodologie również zakładają iteracyjne rozwijanie oprogramowania, modyfikację wymagań użytkownika i testy regresyjne. Faktem jest natomiast, że bardzo często użycie tych metodologii sprowadzało się do bezrefleksyjnego produkowania ton zbytecznej dokumentacji.

W każdym razie niezależnie od wyboru metodologii prowadzenia projektu testowanie

oprogramowania pełni istotną rolę.

5. Miary / metryki

Istnieje wiele miar pozwalających ocenić jakość, efektywność i kompletność testów, poniżej zajmiemy się najczęściej stosowanymi.

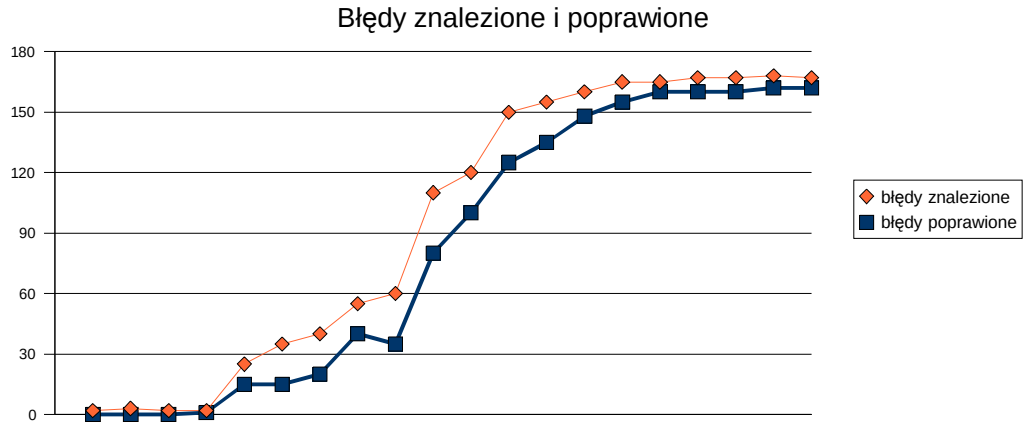
Pokrycie testami jest jedną z najistotniejszych miar pozwalających ocenić na ile oprogramowanie działa niezawodnie. Pokrycie testami możemy określić jako stosunek liczby testów, które są faktycznie przeprowadzane do liczby testów, które należałoby przeprowadzić. Diabeł, jak zwykle, tkwi w szczegółach – nie zawsze jest łatwo policzyć liczbę testów, które należy przeprowadzić.

W przypadku testów jednostkowych można pokusić się o zliczanie linii kodu, które są przetestowane i porównywać tę wielkość do wszystkich linii kodu dla określonego fragmentu oprogramowania. Istnieją tego typu narzędzia np. dla języka programowania Java: dla każdej klasy można policzyć pokrycie testami jednostkowymi. Oczywiście warto pamiętać o tym, że być może istnieją elementy kodu, które nie wymagają bardzo szczegółowego testowania.

W przypadku testowania strukturalnego (testy czarnej skrzynki), gdy nie wnikamy w szczegóły kodu, liczbę testów, które powinny być przeprowadzone można oceniać na podstawie znajomości przypadków użycia, zakładając, że każdemu z nich powinien odpowiadać przynajmniej jeden test.

Warto rejestrować różnego rodzaju miary związane z testowaniem oprogramowania:

- Liczba wszystkich wykrytych błędów.
- Liczba błędów należących do określonej kategorii, zdefiniowanej w ramach używanej terminologii (np. liczba błędów krytycznych, liczba błędów interfejsu użytkownika, itp.).
- Liczba wykrytych błędów przez dany test. Obliczając stosunek tej wielkości do liczby wszystkich wykrytych błędów możemy określić efektywność testu.
- Liczba błędów danego komponentu oprogramowania, co pozwoli nam wykryć te elementy aplikacji, które są najbardziej zawodne, albo, być może, najbardziej złożone i trzeba im poświęcić więcej uwagi.
- Czas potrzebny na przeprowadzenie pojedynczych testów lub serii testów. Miara ta ma duże znaczenie, jeśli testy wykonywane są manualnie, gdyż jest składnikiem oceny kosztowności przeprowadzenia danego testu. Jeśli testy są przeprowadzane automatycznie, to czas ich wykonania ma drugorzędne znaczenie, o ile, oczywiście, nie testujemy wydajności aplikacji, a testy służą do wywołania jej obciążenia.
- Liczba poprawionych błędów, stosunek tej wielkości do liczby ogólnie znalezionych błędów daje nam informację o tym, w jakim stopniu nasz produkt jest stabilny i gotów do wdrożenia. Stanowi to nieocenioną pomoc dla kierownika projektu, gdy chce sprawdzić postępy prac z harmonogramem.



Na wykresie widzimy zaznaczoną liczbę błędów znalezionych i poprawionych w czasie trwania projektu. Gdy produkt staje się stabilny liczba nowych błędów spada i wykres zaczyna przebiegać równolegle do osi poziomej.

Jeśli testy są przeprowadzane w sposób niezaplanowany i brakuje określonej procedury raportowania wystąpienia błędów, to opisane wyżej miary są dla nas niedostępne. Z kolei ręczne spisywanie (przy pomocy arkusza kalkulacyjnego bądź dokumentu tekstowego) występujących błędów jest pracochłonne i dość zawodne, dlatego dużą rolę odgrywają narzędzia które porządkują zbieranie występujących błędów.

Dużym ułatwieniem jest w związku z tym integracja narzędzia do testowania i narzędzia do śledzenia błędów.

6. Narzędzia

Obecnie coraz trudniej wyobrazić sobie pracę testera bez wsparcia różnego rodzaju narzędzi wspomagających proces testowania. Dotyczy to szczególnie testów regresji, które powinny być wykonywane za każdym razem, kiedy wprowadzamy zmiany do oprogramowania, czyli, być może nawet codziennie. Gdy proces ten nie jest zautomatyzowany, to wykonywanie testów jest bardzo czasochłonne.

W przypadku testów jednostkowych zazwyczaj używamy połączenia narzędzia do tworzenia tych testów, takich jak na przykład *Junit,* czy znacznie lepszy TestNG, dla języka *Java*, *DUnit* dla *Delphi*, *PHPUnit* dla *PHP* czy *CPPUnit* dla *C++* z narzędziem do ich automatycznego uruchamiania. Może być to *Jakarta Ant*, *make* lub inne narzędzie tego typu. Aby testy były wykonywane w regularnych odstępach czasu można posłużyć się albo Windows Task Schedulerem albo, w systemach unixowych, cronem.

Znacznie trudniej jest testować aplikację od strony użytkownika, a nie od strony kodu. Na rynku pojawiło się trochę narzędzi pozwalających zautomatyzować tego rodzaju testy. Proces tworzenia testów regresyjnych, czy testów interfejsu użytkownika (GUI) wygląda tak, że tworzymy skrypt testowy przechodząc przez aplikację w sposób określony w dokumencie testowym – skryptu nie trzeba tworzyć ręcznie, jest on nagrywany automatycznie w czasie pracy z programem. Możemy także ustawiać punkty sprawdzające (*checkpoints*), które pozwalają sprawdzić, czy faktycznie test przebiega tak jak powinien. Bardziej zaawansowane narzędzia pozwalają na samodzielne pisanie skryptów testujących, co w praktyce jest znacznie bardziej wygodne.

Po utworzeniu testu pozostaje tylko jego uruchamianie, co można robić ręcznie lub automatycznie przy pomocy *crona* lub *Windows Task Schedulera*.

Warto wspomnieć, że w momencie, gdy mamy wiele testów różnego rodzaju konieczne jest użycie narzędzi, które pomogą nimi zarządzać: wersjonować, synchronizować ze zmianami wymagań, kodu źródłowego aplikacji. Często w tym przypadku używa się popularnego CVS-a

lub Subversion, które pozwalają śledzić wersje różnego rodzaju dokumentów.

7. Wdrażanie procedury testowania

W tym rozdziale zobaczymy przykładowy scenariusz wdrożenia procesu testowania w projekcie. Nie jest to oczywiście jedyne słuszne podejście, może się okazać, że w prostym projekcie nie jest potrzebne aż takie rozbudowywanie procedur testowania i tworzenia dokumentacji, z drugiej strony w trudnym projekcie, który jest obarczony z różnych przyczyn dużym ryzykiem może się okazać, że proces weryfikacji poprawności działania oprogramowania musi być jeszcze dokładniej zdefiniowany i udokumentowany.

7.1 Działania wstępne

- ***Poznanie architektury systemu***. Na tym etapie można będzie wstępnie określić na ile skomplikowany będzie proces testowania
- ***Określenie jakiego typu testy są potrzebne*** - może być to związane zarówno z charakterystyką testowanego systemu jak i z wymaganiami klienta, standardami zapewniania jakości przyjętymi w firmie itp.
- ***Weryfikacja testowalności systemu*** - jest to istotne przy dużych, wielowarstwowych aplikacjach. Tester, aby móc skutecznie testować system, musi mieć dostęp do wszelkich potrzebnych mu danych. Jeżeli na przykład określone działanie użytkownika wywołuje uruchomienie jakiegoś procesu na zdalnym serwerze tester powinien mieć możliwość sprawdzenia, czy faktycznie ten proces został uruchomiony i działa prawidłowo.
- ***Poznanie dodatkowych źródeł informacji*** o działaniu aplikacji, na przykład możliwości dostępu do logu aplikacji, możliwości uruchomienia aplikacji w konfiguracji debugowania, która dostarcza więcej informacji o wewnętrznym działaniu oprogramowania.

7.2 Definicja przypadków testowych.

- Przegląd zebranych wymagań oraz przypadków użycia (jeśli są spisane).
- Dla każdego z wybranych wymagań powinien zostać określony jeden lub więcej przypadek użycia (o ile nie jest to już zrobione albo nie jest oczywiste), następnie należy utworzyć scenariusze testowe dla wymagań/przypadków użycia.
- Podjęcie decyzji co będziemy testować, możemy tu użyć metod opisanych w rozdziale 3.1.
- Wybór technik tworzenia testów, wyboru danych testowych (rozdział 3.2).
- Wybór sposobu wykonywania testów (manualne, automatyczne)

Uwaga: nie należy pomijać testowania wymagań niefunkcjonalnych czyli, na przykład, wymagań co do bezpieczeństwa, wydajności, używalności itp.

Warto pamiętać, że często przypadki użycia i wymagania nie będą dostatecznie precyzyjne, aby móc opracować testy samodzielnie, bez udziału osób, które zajmowały się zbieraniem wymagań. Wynika to z tego, że „idealne" przypadki użycia, czyli testowalne, kompletne i szczegółowe rzadko występują w naturze i to często z dobrze uzasadnionych przyczyn: nie chcemy, żeby przypadki użycia były bardzo szczegółowe, bo stają się wtedy mało czytelne.

7.3 Określenie terminologii

Warto także określić terminologię stosowaną do klasyfikowania występujących problemów, od tego może zależeć sposób traktowania zgłaszanych błędów: inaczej będziemy reagować przy literówce w nazwie przycisku, a inaczej w przypadku problemów z bazą danych.

Trzeba także skategoryzować błędy ze względu na ich pochodzenie. Możemy mieć błędy pojawiające się jako wynik automatycznych funkcjonalnych testów regresji, zgłoszonych przez odpowiednie oprogramowanie. Mogą być także błędy wykryte w czasie testów jednostkowych, zgłoszone przez użytkowników systemu, czy przez testerów, którzy testowali oprogramowanie manualnie.

Jeśli nasz system będzie działał w jakiś specyficznych warunkach, na przykład dla określonych systemów operacyjnych, określonych przeglądarek stron internetowych to być może jest potrzebna dodatkowa klasyfikacja związana z charakterystyką środowiska pracy aplikacji.

7.4 Określenie standardów dokumentowania testów

Dokumentacja testów musi odzwierciedlać działania, które podejmujemy w czasie testowania aplikacji. Opis każdego testu powinien zawierać poniższe elementy

- Identyfikator, zgodny z wybraną przez nas notacją.
- Opis testu: krótka informacja co dany test weryfikuje.
- Data wykonania testu.
- Informacje o testerze, jeśli test jest wykonywany manualnie.
- Autor testu (skryptu testowego).
- Cel testu.
- Związane z danym testem wymagania, przypadki użycia: podajemy identyfikatory odpowiednich przypadków użycia bądź wymagań, które są weryfikowane przez test.
- Warunki początkowe, założenia wstępne, zależności. Określamy w tym punkcie warunki w jakich należy testować aplikację (np. musimy być podłączeni do Internetu). Może się też zdarzyć, że dany test musi być poprzedzony innym lub dwa testy nie mogą być wykonane w tym samym czasie – warto zaznaczyć, że takich sytuacji należy unikać, testy powinny być możliwie autonomiczne.
- Sposób weryfikacji: określamy jak będziemy weryfikować wynik testu, w przypadku testów niefunkcjonalnych może być to dość złożone i polegać na przykład na analizie logu aplikacji, zebraniu opinii użytkowników na temat używalności.
- Działania użytkownika: podajemy kroki jakie trzeba wykonać w ramach testu, nazwę skryptu testowego (lub jego zawartość). Jeśli test nie polega na automatycznym wykonaniu skryptu, a jest manualny, to opisujemy dokładnie kroki jakie powinien wykonać tester.
- Oczekiwany wynik testu: określamy stan aplikacji, w jakim powinna się znaleźć aplikacja po przeprowadzeniu testu; inny stan będzie oznaczać niepowodzenie testu.
- Zawartość logu aplikacji
- Faktyczny wynik testu. Jeśli test się powiódł to po prostu wpisujemy „tak jak oczekiwany”, w przeciwnym razie podajemy informację jaki był rezultat testu, załączmy zrzut ekranu, numer błędu itp.
- Opis danych, jakich należy użyć do testowania

Oczywiście nie zawsze musimy uwzględniać wszystkie elementy, wszystko zależy od tego jak złożona jest procedura testowa.

Pełna dokumentacja testów może być bardzo rozległa, więcej szczegółów można znaleźć w standardzie IEEE: IEEE Standard for Software Test Documentation (IEEE829-98).

Dokumenty zawierające informacje muszą być uwzględnione w ramach systemu zarządzania zmianami, jest to ważne z tego względu, że testy będą ewoluowały wraz z oprogramowaniem, więc trzeba utrzymać synchronizację pomiędzy tymi dwoma elementami. Ponadto powinniśmy móc łatwo stwierdzić jakie zmiany w kodzie są związane z wystąpieniem określonych błędów – ma to wielkie znaczenie, gdy musimy stworzyć wersję oprogramowania, która zawiera poprawki błędów.

7.5 Metryki i raporty

Musimy zadecydować jakiego rodzaju informacje będziemy zbierać i zachowywać, więcej szczegółów na ten temat możemy znaleźć w rozdziale 5.

7.6 Plan testów

W tym momencie jesteśmy gotowi do utworzenia planu testów i budowy bazy testów (ang. *tests inventory*).

Plan testów jest dokumentem, który zawiera informację o celach, zakresie i przyjętych technikach testowania.

Przygotowywanie planu testów jest dobrą okazją do tego, żeby określić warunki pozwalające uznać oprogramowanie za wystarczająco sprawne i gotowe do wdrożenia. W przypadku projektu zewnętrznego zazwyczaj musimy uzgodnić testy akceptacyjne z klientem.

Celem utworzenia takiego dokumentu jest umożliwienie osobom z poza naszego zespołu projektowego zrozumienie jak i dlaczego oprogramowanie jest sprawdzane, kiedy można bezpiecznie przyjąć, że produkt jest gotowy.

Plan testów określa jakie elementy oprogramowanie będą testowane, na jakim poziomie, w jakiej kolejności, jakie techniki testowania zostaną zastosowane i jakie założenia zostały przyjęte.

Plan testów powinien stać się częścią dokumentacji projektowej, podlegać kontroli zmian.

8. Zakończenie. Jak możemy pomóc?

Oczywiście nie sposób w tak krótkim artykule omówić wszystkich aspektów testowania oprogramowania, starałem się przedstawić najważniejsze elementy tego ważnego zagadnienia, w sposób możliwie praktyczny.

Nasi konsultanci chętnie pomogą w doborze dostępnych na rynku narzędzi do testowania oprogramowania. Przygotowujemy także szkolenia, które będą obejmowały tę tematykę.