O czym tester powinien pamiętać nawet po północy

Autor: Hans Schaefer Hans Schaefer, **Software Test Consulting**
hans.schaefer@ieee.org.
Hans Schaefer to jeden z najbardziej znanych na świecie specjalistów w dziedzinie testowania oprogramowania komputerowego oraz znakomity prelegent i wykładowca. Przewodniczący *Norwegian Software Testing Qualifications Board*.
http://home.c2i.net/schaefer/testing.html

Tłumaczenie - na podstawie teksu Hansa Schaefera– wykonał Bogdan Bereza
(bogdan.bereza@victo.eu)

Tester może po prostu wykonywać swoją pracę, co niekiedy wystarcza. Może także w nią włożyć nieco więcej zaangażowania, czyli zrobić lepszą robotę. W tym artykule próbuję wyjaśnić, na czy polega to nieco więcej zaangażowania.

Nie wystarcza testowanie w zwykły sposób.

Systematycznego testowania systemów i oprogramowania można się nauczyć tak samo, jak każdej innej inżynierskiej umiejętności. Dostępne są systemy certyfikacji wiedzy dla testerów (ISTQB, ISEB, GTB), dostępne są książki (Myers 79, Beizer 95, Kaner 99, Copeland 2004) i normy (terminologia ISTQB, BS 7925, IEEE 829, 1008, 1012). Książki i standardy dostępne są już od dawna, a wiele technik jest powszechnie uznawanych, co oznacza, że testowania można się uczyć, a następnie wykonywać je w jakiś usystematyzowany sposób.

Nie oznacza to wprawdzie, że opisane w tych materiałach, typowe metody testowe są powszechnie stosowane (Schaefer 2004), ale przynajmniej istnieje możliwość wykonywania testowania wg zasad opisanych w podręcznikach.

Tester, albo inżynier testowy, wykonuje dwa podstawowe zadania: **projektowanie** przypadków testowych oraz ich **wykonywanie** połączone z rejestrowaniem i analizowaniem wyników. Jeśli uzyskane wyniki są różne od spodziewanych, zgłasza się niezgodność i nadzoruje jej rozwiązywanie. Ponadto nowoczesne metody, takie jak testowanie eksploracyjne (zob. witryna Jamesa Bacha), zalecają także stosowanie automatyzacji testów oraz zarządzanie przez testerów czasem swojej pracy.

Testowanie w zwykły sposób polega na tym, by opanować kilka technik i następnie realizować je, wykonując testy i raportując ich wyniki. Zwykle stosowana formuła zaleca, aby „przetestować system" i nie określa żadnych dodatkowych szczegółów. W niektórych książkach testowanie definiuje się jako „uzyskiwanie informacji o jakości" lub „pomiar jakości" (Hetzel). Ponieważ testowanie jest jedną z ostatnich wykonywanych w projekcie czynności, znajduje się zwykle pod silną presją czasową, co powoduje, że testerzy mają – jawną bądź ukrytą – motywację do tego, aby zrezygnować z dokładnego i starannego wykonywania swej pracy. Na przykład, ponieważ znajdowanie błędów opóźnia zarówno samo testowanie, jak i dostawę czy odbiór produktu, pojawia się ukryty nacisk na to, by błędów nie znajdować, a przynajmniej nie te poważne. Testowanie po prostu ma być „zrobione". W takiej sytuacji praca przestaje być motywująca, nie

inwestuje się w kształcenie, ludzie zaczynają szukać innej pracy, a jakość testowania nie poprawia się.

Czy można testować lepiej?

Glenford Myers już w 1979 roku zdefiniował[1] cel testowania odmiennie: celem testowania jest **znajdowanie błędów**. Negatywne podejście, próby złamania produktu pozwalają na znajdowanie liczniejszych i poważniejszych błędów, a nie tylko sprawdzanie, że oprogramowanie „działa".

Ludzie zwykle robią to, co im się każe. Jeśli polecić im znajdowanie jak największej liczby błędów, będą próbować. Jeśli natomiast kazać im szybko skończyć testowanie, przy okazji – wprost lub pośrednio – komunikując, że znajdowanie błędów opóźnia projekt, wówczas będą starać się nie znajdować błędów, albo wiele z nich przeoczą.

Pierwszą zasadą powinno więc być jasno określić cel testowania, i upewnić się, że jest on dla wszystkich w pełni zrozumiały. Omówimy o poniżej.

Świat się zmienia, zwłaszcza w dziedzinie wytwarzania oprogramowania, i trzeba to wziąć pod uwagę przy każdym zajęciu, nie tylko przy testowaniu. Pojawiają się nowe techniki, metody i narzędzia, nowe sposoby projektowania. Oprogramowanie staje się coraz bardziej skomplikowane, a produkty programistyczne wciąż bardziej zależne jedne od drugich. Zmieniają się wymagania, na przykład coraz większy nacisk kładzie się na bezpieczeństwo, użyteczność i możność wykonania na wielu platformach. Wszystko to powoduje, że zmieniają się warunki wykonywania pracy testerów, którzy muszą się wobec tego wciąż rozwijać pod względem zawodowym. Ten temat także zostanie obszerniej omówiony poniżej.

Problemem jest także nastawienie wielu ludzi. Niektórzy chętnie uznają podawane oficjalnie informacje lub obowiązujące zasady. Inni są bardziej krytyczni – badają i zadają pytania. Skoro jednym z celów testowania jest znajdowanie problemów i wad, można testować skuteczniej, gdy nie przyjmuje się rzeczy na wiarę, sprawdza szczegóły, poszukuje dodatkowych informacji i samodzielnie myśli.

Do zadań testera należy zgłaszanie błędów. Nie jest to łatwe. Literatura na temat testowania zwykle opisuje zasady zarządzania zgłoszeniami błędów, czyli cykl życiowy zgłoszeń na który składa się ich raportowanie, określanie wagi oraz rozwiązywanie. To są jednak tylko podstawy – w rzeczywistości zgłaszanie błędów wymaga czegoś więcej. Można je porównać do zadania, jakie stoi przed sfrustrowanym użytkownikiem, szukającym pomocy w dziale wsparcia dostawcy: trzeba problem opisać w taki sposób, aby druga strona uznała go za na tyle ważny, aby mu zaradzić. Oznacza to zgromadzenie dodatkowych informacji na temat błędu, a także umiejętność „sprzedania" zgłoszenia błędu osobie odpowiedzialnej.

Wreszcie, tester ma także pewne prawa. Nie powinniśmy testować wszystkiego, co tylko zostanie nam zwalone na biurko. Jeśli brak nam niezbędnych informacji, lub gdy przekazany do testowania produkt roi się od błędów, testowanie go jest zwykłym marnotrawstwem. Powinniśmy stosować kryteria dopuszczenia do testowania jako swego rodzaju „Kartę Praw Testera" (Gilb 2005). Będzie o tym także mowa poniżej.

[1] W książce „The Art of Software Testing"; jej kolejne, poszerzone wydanie – pod niezmienionym tytułem - ukazało się w polskim tłumaczeniu w 2005 roku (wyd. Helion).

Dotykamy tutaj zagadnień z zakresu filozofii testowania. Istnieją jednak pewne najzupełniej podstawowe reguły dotyczące testowania. Choć temat ten budzi wiele kontrowersji, pozwolę sobie przedstawić niektóre z nich z mojej konferencyjnej prezentacji (Schaefer 2004).

Na tym sprawy się nie kończą. Testerzy powinni nieustannie poszukiwać nowych wyzwań. Niniejszy artykuł jest tylko początkiem.

Cel testowania

Istnieje wiele różnych celów testowania. Jeden z głównych, choć być może dość nudny, to pomiar jakości testowanego produktu. Testowanie traktujemy wtedy jak najzwyklejsze urządzenie pomiarowe. Wykonywanie pomiarów nie jest szczególnie zabawne, ale konieczne i musi być zrobione starannie. Aby jednak móc mierzyć skutecznie i sprawnie, tester powinien uzyskać odpowiedzi na kilka pytań. Po pierwsze, jakie właściwości (atrybuty) jakości są najważniejsze i jak dokładnie powinno się je mierzyć?

Według innej definicji, celem testowania jest uzyskanie zaufania do produktu. Ale zaufanie najlepiej się osiąga, kiedy próbuje się – bez powodzenia – je podważyć. Przypomina to pracę naukową: ktoś przedstawia nową teorię, a inni specjaliści usiłują ją odrzucić. Po jakiś czasie, jeśli teoria się obroni, zaczyna być zwolna uznawana. To podejście jest zgodne z definicją Myersa: znajdowanie błędów! Jest to podejście pesymistyczne. Pesymista przypuszcza, że coś nie działa i usiłuje znaleźć potwierdzenie. Każda znaleziona wada czy usterka to sukces.

Ludzkim działaniem kieruje motywacja. Definicja testowania jako aktywnego poszukiwania wad motywuje, a także pozwala znajdować więcej błędów. Są tego dwie przyczyny. Po pierwsze, projektuje się bardziej destruktywne lub po prostu liczniejsze przypadki testowe. Po drugie, uważniej analizuje się uzyskane wyniki, wnikając w szczegóły pomijane przez słabiej zmotywowanego testera. Może to np. oznaczać analizowanie wyników niedostrzegalnych wprost na monitorze, lecz ukrytych w głębi plików, baz danych, buforów pamięci lub w sieci.

Tester powinien usiłować znajdować błędy! Wady mogą być ukryte tam, gdzie niełatwo je dostrzec, to znaczy niekoniecznie na ekranie monitora!

Na tym jednak sprawy się nie kończą. Wady są towarzyskimi stworzeniami: lubią występować w gromadzie. To tak jak z komarami: jeśli zobaczy się i zabije jednego, czy sądzi się, że więcej ich już nie ma w okolicy? Dlatego właśnie należy przyjrzeć się uważnie rejonom, gdzie znalazło się błędy.

Testerzy powinni domyślać się, gdzie szukać błędów. Powinni zapoznać się z różnymi wskaźnikami jakości, czytać raporty, w których można je odnaleźć. Jako wskaźniki mogą służyć dane na temat wcześniej znakowanych błędów, braki w komunikacji w projekcie, złożoność programu, częste zmiany, nowe technologie, nowe narzędzia, nowe języki programowania, zmiany w zespole, konflikty między uczestnikami projektu itd. Kłopot w tym, że te wskaźniki czasem wskazują w niewłaściwym kierunku. Wady mogły zostać wykryte w obszarze nieomal bezbłędnym dlatego, że akurat na tam skupiły się przypadki testowe. Komunikacja w projekcie może wydawać się rozpaczliwa, jeśli jego uczestników dzieli duża odległość. Zdarza się jednak, że mimo tego ludzie sprawnie się ze sobą porozumiewają korzystając z nieformalnych kanałów, albo też interfejs między komponentami okazał się zaprojektowany nadzwyczaj dobrze, nieomal „idiotoodpornie". Z jednej strony wiele danych pokazuje na korelacje między wskaźnikami a rozkładem błędów; z drugiej strony, zawsze zdarzają się wyjątki. Na przykład, najbardziej złożone

komponenty konstruowane są przez najbardziej doświadczone osoby. Częste zmiany na przykład zwykle powodują błędy albo wskazują słabo zrozumiane i źle zaprojektowane obszary, ale zdarzają się też zmiany dobrze przemyślane i poddane wnikliwym inspekcjom.

W niektórych projektach zdarzają się „psy pogrzebane" w centralnych miejscach, gdzie lepiej niczego nie ruszać. Takie obszary mogą być słabo zrozumiane, źle zdefiniowane i rojące się od błędów. Ponieważ każdy wie, że lepiej ich nie ruszać, nikt nie domaga się ich modyfikowania.

Nowe technologie także powodują zagrożenia, częściowo dlatego, że mogą one wynikać z samych technologii, częściowo zaś dlatego, że ludzie nie są jeszcze z nimi oswojeni, nie znają kryjących się w ich użytkowaniu trudności. To samo dotyczy testerów. Z drugiej strony, może wystąpić efekt odwrotny: nowe technologie mogą uwolnić nas od wielu zagrożeń, po prostu je uniemożliwiając.

Na zakończenie, przypatrzmy się czynnikowi ludzkiemu. To ludzie mylą się i popełniają błędy. Badania wykazały, że „dobrzy" i „źli" programiści różnią się od siebie zarówno produktywnością jak i częstotliwością błędów. Błędy nie wynikają tylko z programowania, trudniej jednak precyzyjnie określić odpowiedzialność za kłopoty z projektowaniem i specyfikacjami. Jeden czynnik ma niezmiennie negatywny wpływ na jakość: fluktuacja kadr. Kiedy nowe osoby przejmują czyjąś pracę, zwykle nie uzyskują o niej pełnej wiedzy, gdyż doświadczenie oraz intuicja z trudem tylko są przekazywane. Dotyczy to zwłaszcza sytuacji, gdzie pewne obszary były znane tylko jednej osobie.

Podsumowując: istnieje wiele wskaźników, na podstawie których można wnioskować o błędach, ale należy się nimi posługiwać ostrożnie.

Błędy występują gromadnie: są towarzyskie!
Wiele rozmaitych błędów może mieć wspólną przyczynę. Warto ją zidentyfikować, a później szukać jej kolejnych skutków!
Gdzie znało się błędy, tam warto szukać dalej!

Kolejna definicja testowania to „pomiar ryzyka". Oznacza to, że testowanie jest najwyraźniej formą zapobiegania zagrożeniom, częścią zarządzania ryzykiem. W najgorszym razie, to testerzy powinni zadawać pytania o zagrożenia związane z produktem, zwłaszcza jeśli nie zadaje ich nikt inny. Podstawowy sposób identyfikacji ryzyka to wzięcie pod uwagę profilu użytkowania i skutków ewentualnych awarii. Tester powinien spytać się o to, którzy użytkownicy jak często będą wykonywali jakie czynności? Jakie będą różnice między użytkownikami? Czy ten rozkład będzie zmieniał się w czasie?

Koniecznie musi się uwzględnić możliwe błędy wprowadzania danych i inne pomyłki. Użytkownicy miewają niezdarne palce, a programy miewają błędy! Należy więc zastosować nie tylko poprawne dane wejściowe, ale zapewne także bardzo wiele niepoprawnych.

Kolejny czynnik to możliwe koszty awarii. Może być trudno je określić. Na początku warto sobie przynajmniej zadać pytanie: co najgorszego może się wydarzyć w razie awarii funkcji, funkcjonalności lub zadania użytkownika?

Testowanie jest formą zapobiegania zagrożeniom.
Od czego zależy ryzyko?

Jakie są skutki niepoprawnych danych wejściowych?

Podsumowując: najlepiej, gdy tester jest pesymistą (pesymista to optymista mający już doświadczenie). Jeśli coś nie działa, to znakomicie, gdyż nikt nie będzie musiał doznać tej awarii w przyszłości. Pozytywne skutki okażą się dopiero na dłuższą metę. Lepsze testowanie zmusza także konstruktorów do lepsze pracy, informuje kierownictwo o istniejących zagrożeniach, a ponadto obniża koszty naprawy błędów. Testerzy przynoszą złe wieści, ale na tym polega ich praca. Nikt nie lubi kontroli prędkości na szosie, ale dzięki nim szosy są bezpieczniejsze i wszyscy z tego korzystamy.

Pesymista jest lepszym testerem!

Ciągłe uczenie się

W niemal każdym zawodzie trzeba wciąż się uczyć, ale dla testerów jest to zupełnie niezbędne. W większości przypadków testowanie wykonuje się w pewnym stopniu systematycznie, przy pomocy jakiegoś podejścia czarnoskrzynkowego. Ponadto projektowanie testów może wykorzystywać pewne heurystyki. Ale podejścia czarnoskrzynkowe nie gwarantują pełnego pokrycia testowego. Zasady heurystyczne nigdy nie gwarantują pełnego pokrycia, gdyż zależą od osobistego doświadczenia, względnie od doświadczenia uzyskanego za pośrednictwem innych osób. Z kolei podejście białoskrzynkowe nie jest w stanie ujawnić np. pominiętych wymagań. Wszystko to sprowadza się o kwestii: to, o czym tester nie wie, nie może być przetestowane! A więc tester powinien wiedzieć jak najwięcej - tylko jak?

Testerowi przydaje się doświadczenie w programowaniu. Nawet po testach modułowych przeprowadzonych przez programistów, w kodzie pozostaje wiele błędów – a ponadto testy modułowe rzadko kiedy są starannie przeprowadzane. Tester powinien orientować się w trudnościach, jakie stwarza dany język programowania. Na przykład pętle i liczniki z reguły sprawiają kłopoty, powodując błędy o jedność. Jeśli tester nie ma o tym pojęcia, nie przetestuje pętli z liczbą iteracji zerową, maksymalną, większą o jeden od maksymalnej ani też z jedną iteracją. Wówczas jedynie przypadek pozwoli na znalezienie błędu o jedność.

Testerowi przydaje się doświadczenie w projektowaniu. Projektowanie dotyczy w znacznym stopniu określania zakresu i komunikacji: który moduł, w jakim zakresie, przy użyciu jakich zależności, powinien wykonywać które zadanie? Gdzie znajdują się te moduły, jak się komunikują, jak rozwiązują konflikty? Jeśli tester nie orientuje się w sprawach architektury i związanych z nią problemów, nie będzie mu łatwo zaplanować testy integracyjne.

Tester potrzebuje też wiedzy dziedzinowej. Testowanie systemowe polega na sprawdzaniu, czy system realizuje właściwe rozwiązanie, czy spełnia wymagania dziedzinowe. Kto potrafi przetestować system sygnalizacji kolejowej? (eee… na czym polega sygnalizacja kolejowa…?) Choć TROCHĘ znajomości dziedziny jest niezbędne, albo porozumiewanie się ze znającymi ją innymi udziałowcami.

Niestety, to nadal nie wystarcza. Ważna w testowaniu jest umiejętność uzyskiwania właściwych informacji od właściwych ludzi. Warto nauczyć się czegoś o technikach przeprowadzania wywiadów. Posiadłszy tę umiejętność, uzyskuje się wiele nowych informacji! Systemy są zależne od innych systemów w coraz bardziej zawiły sposób, pojawia się wiele nieoczekiwanych interfejsów. Dla przykładu, ktoś może integrować TWÓJ serwis internetowy ze SWOIM portalem,

i jednocześnie z innymi portalami. Może twój serwis działa pod jakimś względem inaczej niż wszystkie inne, i dlatego nie jest już atrakcyjny- lub jest o wiele atrakcyjniejszy, niż można się było spodziewać. Co oznacza, że testowanie pod kątem obecnych udziałowców systemu nie zawsze wystarcza. Mogą istnieć najzupełniej nowe sposoby wykorzystania lub interfejsu do twojego systemu, nowe nieprzewidziane podejścia do niego i należy przynajmniej spróbować część z nich przewidzieć!

Tester powinien stale usiłować znaleźć nowe punkty widzenia testowanego systemu, nowe sposoby podejścia i zastosowania. Wreszcie, chcemy, żeby testerzy korzystali z najnowszych metodyk i technologii. Trzeba się ich nauczyć. Należy czytać książki poświęcone testowaniu, wyszukiwać i poznawać narzędzia, czytać czasopisma, udzielać się w grupach dyskusyjnych, rozmawiać z innymi profesjonalistami i brać udział w konferencjach testowych!

Uczmy się więcej i o wszystkim!
Uczmy się programowania, architektur, nowych dziedzin, narzędzi – wszystkiego!
„Trzy rzeczy ciągną mój sprzęt: psy, psy i psy." (Roald Amundsen).
Dla testera, te trzy rzeczy to: uczyć się, uczyć się i uczyć się.

Krytyczne nastawienie

Nie wierzcie w byle co! Mój kolega powiedział kiedyś, że: „wierzyć to należy w niedzielę w kościele. Poza tym wszystko należy sprawdzać."

Niestety – łatwiej bywa wierzyć, nie wymaga to żadnego wysiłku. Pomyślcie, co jest napisane w gazecie. Czy to prawa? Gdzie była broń masowego rażenia? Czy naprawdę za wszystko co złe odpowiadają Żydzi? Czy rzeczywiście oglądanie telewizji jest niebezpieczne dla dzieci? Czy naprawdę pewien sok jest zdrowy? Odpowiedź: łatwiej o to nie pytać. Jeśli wszystko poddawać w wątpliwość, niczego nie można osiągnąć, więc w codziennym życiu przyzwyczailiśmy się nie pytać, lecz wiele rzeczy przyjmować na wiarę. Wierzyć, zakładać i NIE ZADAWAĆ PYTAŃ!

Testerowi nie wolno niczego zakładać – to może być nieprawdą! Projektanci, autorzy specyfikacji, programiści dokonują wielu założeń. Czasem trudno nam zadać pytanie, bo obawiamy się wypaść głupio. Ten, kto zna odpowiedź, może przebywać daleko albo dostęp do niego może być trudny. Może nam nawet nie przyjść do głowy, że możliwa jest inna interpretacja od powszechnie przyjmowanej. Ktoś inny nie zna odpowiedzi, jeszcze ktoś odpowiada sarkastycznie…

Będąc pesymistą, można równie dobrze przyjąć założenie, że każde przypuszczenie jest błędne.

Niczego nie zakładaj! Pytaj!

Są sposoby na to, aby nie musieć czuć się głupio pytając. Trzeba nauczyć się postępować z ludźmi, przeprowadzać wywiady, zdobyć pewność siebie (jak się uczyć zostało napisane wcześniej). Można spytać kogoś innego, przeczytać, dokonać przeglądu, przespać sprawę, spróbować znaleźć inne wytłumaczenie. Byle tylko niczego nie przyjmować na wiarę! Niczego nie traktować jako oczywistości! A nade wszystko, nie wierzyć, że „system zapewne działa poprawnie". Co najmniej jeden system informatyczny w banku nieprawidłowo obliczał procent – w końcu niełatwo to skontrolować… Był sobie system informacji geograficznej, który wybierał

trasę dookoła świata zamiast najkrótszej. Są systemy linii lotniczych do rezerwacji miejsc, które nie pokazują wszystkich dostępnych połączeń. Istnieje wiele więcej podobnych przykładów.

Jeśli nikt nie zadaje właściwego pytania, może właśnie Ty mógłbyś to zrobić! Pomyśl o nowych możliwościach, nieznanych problemach i o wszystkim, czego się nauczysz! Myśl niestandardowo!

Wady

Nikt nie kocha tego, co przynosi złe wieści. Testerzy zwykle przynoszą złe wieści (czasem nie ma złych wiadomości, wszystko wydaję się działać poprawnie, ale to inna historia).

Złe wieści to błędy, albo „zdarzenia" czy „kwestie", aby nazwać je jakoś neutralnie. Dużo na ten temat napisano w podręcznikach. Zdarzenia są raportowane, rejestrowane, zarządzane, usuwane, poddawane testom ponownym i testom regresji, i wszyscy dobrze o tym wiemy. Ale pewne sprawy książki przemilczają:
1 – zdarzenie jest dla testera interesujące tylko wtedy, kiedy zostanie zaakceptowane jako wada i naprawione;
2 – zdarzają się awarie, które występują tylko wtedy, kiedy wykonuje się szereg przypadków testowych po kolei w określonej kolejności.

Pierwsza sprawa to umiejętność sprzedaży i dyscyplina. Nikt nie ma ochoty wydawać pieniędzy na naprawę błędów – naprawia się je tylko wówczas, gdy są dostatecznie ważne. Wobec tego tester powinien zgłosić błąd w taki sposób, aby konstruktor zrozumiał konieczność jego naprawienia. Konsekwencje awarii musze wydawać się znaczne, jej prawdopodobieństwo – wysokie, z zdarzenie musi być łatwo odtworzyć. Tester nie może wobec tego poprzestać na zapisaniu zdarzenia w raporcie. Tester powinien pomyśleć w następujący sposób: czy nie ma czasem scenariuszy wydarzeń, w których ta sama wada mogłaby spowodować jeszcze gorszą awarię? Może dla niektórych udziałowców sprawa jest poważniejsza, niż się wydaje? Czy sprawa ogranicza się do zaobserwowanego symptomu, czy kryje się za nią coś więcej? Raz jeszcze warto myśleć niestandardowo. Czasem trzeba wymyślić i wykonać dodatkowe testy. Sporo ciekawego materiału na ten temat można znaleźć u Cema Kanera.

Nie można lekceważyć czynnika ludzkiego: dyplomacji, grzeczności i tak dalej. Tester powinien zadbać o to, aby nikogo nie urazić osobiście swoim zgłoszeniem błędu. Ktoś powiedział, że „dyplomacja to umieć kazać komuś pójść w diabły tak, by tamten z radością wybrał się w drogę."

Każde zdarzenie albo błąd należy zbadać dokładniej!
Zgłoszenie błędu musi podkreślać związane z nim zagrożenie!
Raportowanie błędów wymaga umiejętności skutecznej sprzedaży!
Zgłaszając błędy nie wolno zapominać o dyplomacji!

Zdarzają się jednak gorsze sytuacje: awarie, których nie da się odtworzyć. Raz problem się pojawia, za drugim razem go nie ma. Takie wydarzenia nazywa się „przejściowymi błędami". Są z nimi szczególnie duże kłopoty, jeśli pociągają za sobą awarię całego systemu. Ponowne uruchomienie systemu usuwa stare dane z pamięci, co zaciera ślady. Często przejściowe błędy spowodowane są trwającą przez dłuższy czas, stopniową degradacją pamięci lub innych zasobów, na przykład tzw. przeciekami pamięci. Kiedy jakaś funkcja programu nie zwraca zużytej

pamięci, system może mimo to działać tak długo, aż pamięć się nie wyczerpie. Jeszcze gorzej, gdy zjawisko takie nie występuje każdorazowo, tylko w szczególnych okolicznościach.

Także inne zasoby, nie tylko pamięć, mogą ulec wyczerpaniu. Na przykład sonda Mars Explorer przestała działać po 18 dniach, gdy otwartych zostało zbyt wiele plików – na szczęście NASA zdołała załadować nowe oprogramowanie. Wiele wbudowanych systemów czasu rzeczywistego uruchamia procesy ponownie co pewien czas, aby zapobiegać zagrożeniu stopniowej degradacji zasobów. Niestety, WSZYSTKIE wspólne zasoby mogą ulec awarii, dlatego należy sprawdzać wszystkie dane wyjściowe, nie tylko te łatwe do zobaczenia na ekranie. Pliki, wskaźniki do pamięci, bufory, bazy danych, meldunki, rejestry – cokolwiek. Może powstać sytuacja wyścigu, zależnie od precyzyjnego położenia w czasie równoległych wątków lub procesów. Nietrudno spostrzec dane pojawiające się na ekranie – wszystko inne wymaga narzędzi lub dodatkowych czynności ze strony testera, które to czynności mogą okazać się zbyt czasochłonne. Błędy przejściowe wymagają – aby je ujawnić – całego szeregu przypadków testowych, nie tylko pojedynczej pary we-wy. Wreszcie błąd może tkwić gdzieś w systemie operacyjnym, w bibliotekach, w komponentach nie będących częścią produktu.

Kiedy pojawi się przejściowy błąd, warto móc ponownie wykonać tę samą co poprzednio sekwencję przypadków testowych, może nawet z identycznymi ustawieniami czasowymi, i wykonać dodatkowe kontrole. James Bach (Bach 2005) proponuje szereg rad, jak badać takie problemy:

Należy analizować nawet przejściowe błędy!
Należy w trakcie testów rejestrować wszystkie wykonywane czynności i obserwowane zdarzenia!
Musi istnieć możliwość powtórzenia wykonanych testów z uruchomianymi dodatkowymi urządzeniami rejestrującymi i analizującymi!

Tester ma także prawa

Będąc testerem masz swoje prawa.

Testowanie wykorzystywane jest często przez innych do czyszczenia Stajni Augiasza. Zamiast porządnie pracować od początku, wbudowuje się błędy w system i zleca testerom ich wyszukiwanie. To marnotrawstwo czasu i pracy. Koszt usunięcia błędu znalezionego przez testerów przekracza wielokrotnie koszt zapobieżenia temu błędowi. Powoduje to także opóźnienia dostaw. Zadaniem testerów nie powinno być sprzątanie po innych, lecz wykonywanie pomiarów jakości i raportowanie poziomu ryzyka. Sprzątanie jest po prostu pracą nie dla nich.

Tester nie jest odkurzaczem!

Rozwiązaniem tego absurdu jest stosowanie kryteriów wejścia, co wymusza na innych dostawy o rozsądnym poziomie jakości. Karta Praw Testera pojawiła się w co najmniej dwóch źródłach.

W *Extreme Programming Projects* Lisa Crispin pisze o testerach: najważniejsze trzy prawa testerów są następujące.
* Prawo do wykonywania i zmiany swoich oszacowań (…).
* Prawo do korzystania z niezbędnych do pracy narzędzi (…).

* Prawo do tego, aby również pozostali uczestnicy zespołu projektowego, nie tylko testerzy, poczuwali się do odpowiedzialności za jakość.

Tom Gilb (Gilb 2003) stworzył następującą listę praw testera (przytoczoną tutaj za zgodą autora). Karta Praw Testera.
1. Testerzy mają prawo do wyrywkowych kontroli otrzymywanego produktu i odrzucania go, gdy nie spełnia kryteriów jakości.
2. Testerzy mają prawo do jednoznacznych i zrozumiałych wymagań.
3. Testerzy mają prawo rozpoczynać testowanie jak najwcześniej w kolejnych fazach rozwoju systemu.
4. Testerzy mają prawo do tego, aby specyfikacje testów traktowane były na równi z pozostałymi technicznymi specyfikacjami.
5. Testerzy mają prawo do współdecydowania o przyjętych kryteriach jakości.
6. Testerzy mają prawo domagać się zasobów niezbędnych o wykonywania swej pracy.
7. Testerzy mają prawo do równomiernego obciążenia pracą i do prywatnego życia.
8. Testerzy mają prawo określić skutki braku czasu na wykonanie dostatecznych testów.
9. Testerzy mają prawo dokonywać przeglądów specyfikacji, które dotyczą ich pracy.
10. Testerzy mają prawo wykonywać testy produktów spełniających kryteria jakości i odrzucać produkty zbyt wadliwe.

Ostatnie sformułowanie w tej liście jest szczególnie ważne: testerzy powinni odrzucać produkty nie spełniające kryteriów jakości!

Skrzynka z narzędziami dla testera – do użytku nawet w środku nocy

Jak tester powinien pracować? O czym powinien zawsze pamiętać wykonując swoją pracę?

Pierwsza kwestia to testowanie „wszystkiego”. To o wiele za dużo, tego się nie daje osiągnąć. Tester powinien jednak orientować się, co jest przetestowane, co nie jest, a co częściowo. Nazywamy to pokryciem testowym.

W skrócie, istnieją trzy podstawowe pojęcia pokrycia, które można zastosować do modeli działania systemów zapisanych w postaci grafów; na przykład do grafu przepływu sterowania (schematu blokowego), grafu przepływu danych, grafu przejść stanów, drzewa wywołań, diagramu architektury systemu lub przypadku użycia.

Podstawowe pokrycie to pokrycie każdego „klocka” (węzła grafu). Kolejny poziom to przetestowanie każdego łącza (krawędzi grafu). Osiągnięcie obu tych poziomów należy traktować jako minimum zalecane przy testowaniu. Jeśli ma się do dyspozycji więcej czasu, na kolejnym poziomie można testować kombinacje, na przykład pary krawędzi.

Tester powinien umieć określić osiągnięte pokrycie testowe!

Dalej, testerzy powinni postępować zgodnie z profilem użytkowania systemu, co nie zawsze jest łatwe w testowaniu modułów i podsystemów. Jednak będąc testerem, należy przynajmniej spróbować mieć choć częściowe rozeznanie, jak stosowany będzie przedmiot aktualnych testów. Jeśli nie mamy na ten temat żadnej wiedzy, należy równomiernie testować różne tryby użycia pod kątem wytrzymałości aplikacji. W tej sytuacji interesujące są zwłaszcza niepoprawne dane wejściowe.

Jeśli możliwe, testujmy godnie z profilem użycia!
Gdy to niemożliwe, testujmy odporność na błędne dane wejściowe.

Jedna z technik jest podstawowa dla testów czarnoskrzynkowych: podział na klasy równoważności. Pozwala ona ograniczyć wysiłek testowy, uzupełnia także inne techniki testowe. Testerzy powinni ją znać, ale uświadamiać sobie jej ograniczenia. Testy czarnoskrzynkowe grożą przeoczeniem istotnych aspektów działania systemu. Należy zdawać sobie sprawę z pożytków testowania kombinacji. Lee Copeland (Copeland 2004) opublikował dobre wprowadzenie do tej techniki.

Podział na klasy równoważności to dobra technika podstawowa!
Nie zapominajmy o testowaniu kombinacji!

Najgorszy scenariusz to konieczność przyznania się, że testu nie dało się wykonać lub jego wynik był niepoprawny. Dużym problemem bywa też środowisko testowe – musi być wcześnie przygotowane i przetestowane. Opóźnienie się środowiska testowego zabije każdy projekt testowy – co inni będą nam wytykać palcami! Wreszcie, wykryta awaria nie musi być spowodowana błędem testowanego obiektu, lecz testowych danych wejściowych lub analizy danych wyjściowych. Bądźmy samokrytyczni!

Trzeba sprawdzać środowisko testowe!
Trzeba sprawdzić dane testowe!

Na zakończenie, kwestia automatyzacji testów. Oprogramowanie powinno być programowalne, to znaczy łatwe do zmiany. Ale zmiana oznacza ryzyko, co pociąga za sobą konieczność testowania po każdej zmianie, wykonywania testów ponownych i testów regresji. Wykonywanie testów przy pomocy robotów ułatwia testy regresji. Jednak automatyzacja testów to coś więcej: istnieją narzędzia, które na podstawie specyfikacji automatycznie generują przypadki testowe. Są też narzędzia służące do automatycznego tworzenia środowisk testowych, a także wspomagające zarządzanie testowaniem i testaliami.

Automatyzujcie testowanie!
Pamiętajcie, że testowanie to nie tylko wykorzystanie robotów testowych!

Wybrane referencje:

1. Bach 2005: James Bach. Notatki na temat przyczyn błędów przejściowych: http://blackbox.cs.fit.edu/blog/james/
2. Beizer 95: Boris Beizer, Black Box Testing, John Wiley, 1995
3. Better Software Magazine, www.bettersoftware.com. www.stickyminds.com Bardzo praktyczne!
4. BS7925: British Standard: www.testingstandards.co.uk/bs_7925-1.htm
5. Copeland 2004: Lee Copeland, A Practitioner's Guide to Software Test Design, Artech House, 2004.
6. Crispin: Lisa Crispin, Tip House, Testing Extreme Programming, Addison-Wesley, 2002, także http://home.att.net/~lisa.crispin/XPTesterBOR.htm
7. Gilb 2003: "Testers Rights: What Test should demand from others, and why?"., Prezentacja plenarna na EuroSTAR 2003
8. GTB: German Testing Board: www.german-testing-board.info German

Testing Board stworzył wcześniejszą wersję obecnego certyfikatu ISTQB.
9. IEEE Standards: zobacz www.ieee.org
10. ISEB: Information Systems Examinations Board of British Computer Society. http://www.bcs.org/BCS/Products/Qualifications/ISEB/ prowadzi system certyfikacji dla testerów od 1999.
11. ISTQB: www.istqb.org International Software Testing Qualifications Board.
12. ISTQB Glossary: www.istqb.org/fileadmin/media/glossary-current.pdf
13. Kaner 99: C. Kaner, J. Falk, H. Q. Nguyen, “Testing Computer Software (3rd ed.), John Wiley, 1999.
14. Kaner bugadvoc: Prezentacja na temat sposobów raportowania błędów. http://www.kaner.com/pdfs/BugAdvocacy.pdf
15. Myers 79: Glenford Myers: The Art of Software Testing, John Wiley, 1979.
16. Schaefer 2004; Hans Schaefer, “What Software People should have known about Software Testing 10 years ago - What they definitely should know today. Why they still don't know it and don't use it”, EuroSTAR 2004

W języku polskim:

1. Glenford Myers "Sztuka testowania oprogramowania" (translated to Polish 2005 and pubished by Helion publ. house)
2. Ron Patton "Testowanie oprogramowania", tłumaczenie polskie 2002, MIKOM.
3. Bogdan Wiszniewski i Bogdan Bereza-Jarociński "Praktyka i teoria testowania programów", MIKOM 2006.