

Testujemy więcej, częściej i lepiej! Prawda, czy fałsz? Prawda! Rola testów jako czynnika mierzącego jakość, niejednokrotnie jedyne w projektach zaczyna być zwolna doceniana z należąca powagą. Abstrahuję od tego, czy więcej i częściej znaczy lepiej – z tym bywa różnie. Obserwuję jednak, że znaczenie testów chociażby jako elementu przewagi konkurencyjnej przedsiębiorstwa zaczyna nabierać wymiernego znaczenia. W sytuacji dużej konkurencji, gdy na rynku jest wiele podobnych do siebie produktów, o sukcesie handlowym decydują dwa sprzężone ze sobą czynniki, cena i właśnie jakość.

Jednym z celów Stowarzyszenia jest krzewienie wiedzy i „potencjału konkurencyjnego”, jaki tkwi w testach i szeroko pojętym QA. Kiedy ponad rok temu rozpoczynaliśmy działalność, mieliśmy pewne obawy jak rozwinię się nasza inicjatywa. Wiele z przedsięwzięć, zwłaszcza społecznych po pewnym okresie „zachłyśnięcia” się ideą, upada.

Możemy się cieszyć, że z SJSI jest inaczej. Wydajemy kolejny numer kwartalnika, za niecały miesiąc organizujemy ogólnopolską konferencję -o czym w dalszej części numeru, uruchomiliśmy kilka dni temu forum dyskusyjne na naszej stronie, kilku z członków naszego Stowarzyszenia aktywnie uczestniczy w pracach International Software Testing Quality Board, co miesiąc - w czwartki - spotykamy się równoległe na ciekawych wykładach na Politechnice Warszawskiej i w Akademii Ekonomicznej w Krakowie. Generalnie – działamy. Wszystkich tych, którzy chcą wnieść coś dodatkowego do „testerskiego i jakościowego światka” zapraszamy do współpracy w ramach SJSI.

Dziękuję tym wszystkim, którzy przyczyniają się do rozwoju Stowarzyszenia, a w tym miejscu w szczególności autorom artykułów oraz Redaktorowi.

Oby tak dalej.

Z pozdrowieniami
W imieniu Zarządu
Wojciech Jaszcz
Prezes SJSI

Od redaktora

Oddajemy do Waszych rąk, Drodzy Czytelnicy, drugi numer kwartalnika **TESTER.PL**.

Chcemy by stał się on – ten kwartalnik - miejscem wymiany informacji, nurtujących problemów, rozwiązywanych i rozwiązanych (lub nierozwiązanych) problemów. Chcemy też poznać Wasze uwagi tym co chcielibyście przeczytać, co Was gnębi i nie daje Wam spać po nocach?

W tym numerze cztery pozycje:

1. Piotr Kałuski o swoich „idealnym” środowisku testowym – zgodnie z doświadczeniami autora
2. Anita Pankowska o raportowaniu błędów
3. Bogdan Bereza Jarociński o swej niewierze w nowość paradygmatu testowania eksploracyjnego
4. Mariusz Janczewski o roli testera w całym procesie produkcyjnym – tester jako łącznik pomiędzy całym zespołem

Niestety, nie udało się w tym numerze wstawić kolejnego artykułu z serii „automaty testowe” Wina jest głównie po stronie Redakcji- były wakacje i nie dość energicznie popędzaliśmy autorów. W kolejnym numerze na pewno się poprawimy.

Numer otwiera zaproszenie na Pierwszą Krajową konferencję naszego Stowarzyszenia. Będzie to na pewno miejsce wymiany interesujących nas poglądów, zauważenia nowej tematyki dotyczącej naszych zainteresowań zawodowych. Gorąco wszystkich zapraszamy.

Równocześnie chciałbym gorąco powitać nowych autorów w naszym gronie, a zwłaszcza pierwszą przedstawicielkę płci pięknej. Chciałbym też gorąco zachęcić wszystkich czytelników tego periodyku do aktywnej współpracy. Czekamy na Wasze uwagi, artykuły, felietony – mieszczące się w spektrum naszych zainteresowań.

Redaktor

I Konferencja Stowarzyszenia Jakości Systemów Informatycznych

**18-19 listopada 2004 r.
Hotel 500, Zegrze k/Warszawy**

Szanowni Państwo,

w imieniu Stowarzyszenia Jakości Systemów Informatycznych chcielibyśmy zaprosić Państwa do udziału w I Konferencji Stowarzyszenia Jakości Systemów Informatycznych, która odbędzie się w dniach 18-19 listopada w Hotelu 500, w Zegrzu k/Warszawy.

Celem konferencji jest:

- szerzenie wiedzy z zakresu zapewnienia jakości systemów informatycznych i testowania oprogramowania poprzez cykl wykładów, w których położono nacisk na rozwiązania praktyczne, nowoczesne metody testowania oraz zastosowanie narzędzi wspomagających pracę zespołów testujących,
- prezentacja nowoczesnych narzędzi związanych z testowaniem oprogramowania i zapewnieniem jakości,
- zaprezentowanie specyfiki działalności i dotychczasowych osiągnięć Stowarzyszenia,
- integracja członków i potencjalnych członków Stowarzyszenia.

Konferencja skierowana jest przede wszystkim do osób zajmujących się na co dzień testowaniem oprogramowania lub uczestniczących w procesie zapewnienia jakości, zorientowanych na praktyczne rozwiązywanie problemów, wykorzystywanie nowoczesnych metod testowania oraz efektywną automatyzacją działań związanych z jakością w przedsiębiorstwie. Tym samym organizowana przez Stowarzyszenie Jakości Systemów Informatycznych konferencja skierowana jest przede wszystkim do:

- specjalistów z dziedziny zarządzania jakością i testowania oprogramowania,
- menedżerów testów,
- członków zarządów odpowiedzialnych za zarządzanie jakością w firmach,
- dyrektorów i menedżerów działów jakości,
- kierowników projektów,
- osób odpowiedzialnych w firmach za wprowadzanie systemów zarządzania jakością.

Cena udziału w konferencji (1 osoby) wynosi:

- dla członków SJSI – 400 zł
- dla osób spoza SJSI – 500 zł

W cenie zawarty jest koszt udziału w wykładach i warsztatach konferencyjnych, w imprezie organizowanej wieczorem pierwszego dnia konferencji oraz koszt wyżywienia i noclegu w pokoju 2-osobowym.

Zgłosić uczestnictwo w konferencji można wypełniając formularz rejestracyjny dostępny na stronie SJSI (<http://www.sjsi.org>).

Serdecznie zapraszamy!

Zarząd SJSI

Przewodniczący Komitetu Organizacyjnego Konferencji
Joanna Nowakowska

Agenda konferencji

Dzień I

Godzina	Temat	Prelegent	Firma/Organizacja
09:00	Rejestracja uczestników, poranna kawa	-	-
09:15	Otwarcie konferencji	-	-
09:20	Jakość – Piękna choroba?	Bogdan Bereza- Jarociński	bbj Test
10:00	Dyskusja panelowa nt. "good enough quality"	Lilianna Wierzchoń - Prowadząca	ComputerLand S.A.
		Lucjan Stapp	Politechnika Warszawska
		Krzysztof Jakubowski	Infovide S.A.
		Piotr Wąsikowski	Sollers
		Tomasz Byzia	Premium Technology
		Wojciech Jaszcz	ComputerLand S.A.
10:45	Najnowsza norma ISO/IEC 90003:2004 wytyczne do stosowania ISO 9001:2000 dla oprogramowania komputerowego	Bolesław Szomański	Politechnika Warszawska
11:30	Przerwa kawowa	-	-
11:45	Wdrożenie standardów jakości w organizacji	Maciej Bodych	Premium Technology
12:45	Analiza biznesowa w UML a projektowanie scenariuszy testowych	Lucjan Stapp	Politechnika Warszawska
13:30	Przerwa obiadowa	-	-
14:15	Goal-Question-Metric metodą na wybór miar	Joanna Nowakowska	Rodan Systems S.A
15:15	Przerwa kawowa	-	-
15:30	Warsztaty z organizacji procesu zarządzania zespołem QA	Marcin Stachura	K2C
18:00	Bankiet/Ognisko	-	-

Dzień II

Godzina	Temat	Prelegent	Firma/Organizacja
08:00	Śniadanie	-	-
09:00	Rozpoczęcie drugiego dnia konferencji	-	-
09:05	Prezentacja narzędzi automatyzujących testowanie firmy Compuware	Marek Drózdź Sławomir Michalik	Compuware, PB Polsoft
10:35	Przerwa kawowa	-	-
10:50	Prezentacja rozwiązań firmy ComputerLand S.A.	Michał Mazur	ComputerLand S.A.
11:50	Continuous Integration	Jan Topiński	Infovide S.A.
12:45	Narzędzia do automatycznego tworzenia dokumentacji	Michał Rowicki, Piotr Krachel	Polkomtel
13:15	Przerwa obiadowa	-	-
14:00	Proste sposoby generowania danych testowych	Janek Sabak	Infovide S.A.
14:45	Testy wydajności aplikacji, „sztuka dla sztuki” a może wierne odwzorowanie życia	Tomasz Wodziński, Wawrzyniec Żurowski	Polkomtel
15:30	LReport – narzędzie do dokumentowania zapytań i porównywania ich z oczekiwaniami	Piotr Kałuski	CGI
16:15	Zakończenie konferencji	-	-



Idealne środowisko testowe

Piotr Kałuski

CGI

Piotr Kałuski

Jest absolwentem Wydziału Elektroniki i Technik Informatycznych Politechniki Warszawskiej, kierunek informatyka. Od 1995 uczestniczył w wielu projektach informatycznych jako programista analityk, projektant i kierownik zespołu. Obecnie jest pracownikiem firmy CGI i jako konsultant jest członkiem zespołu System Test w firmie Polkomtel. Dziedzina jego zainteresowań to sposoby usprawnienia procesu testowania i wykorzystanie narzędzi Open Source w procesie wytwarzania i testowania oprogramowania. Szczegóły można znaleźć pod adresem www.piotrkaluski.com.



Idealne środowisko testowe

Piotr Kałuski

CGI

W poniższym artykule chciałbym przedstawić cechy, jakie powinno mieć idealne środowisko testowe. Idealne z punktu widzenia testera. Z góry zaznaczam, że środowisko idealne z punktu widzenia testera może być w wielu sytuacjach niemożliwe do stworzenia ze względu na koszty sprzętu, licencji, na nakład pracy potrzebnej do utworzenia tego środowiska, zarządzania nim i jego utrzymania. Jednak czasami warto zastanowić się nad tym, co czyni ideał ideałem. Wiedza ta, nawet przy pełnej świadomości nierealności tegoż ideału, pozwala nam łatwiej wyznaczać cele i planować prace nad produktem, który ma być możliwie najlepszym ucieleśnieniem teoretycznej doskonałości.

Jest jeszcze jeden cel, który chcę osiągnąć w tym artykule. Koszty implementacji danych cech środowiska są widoczne prawie że gołym okiem. Obliczamy ile będzie potrzeba sprzętu, ilu ludzi do utworzenia i utrzymania środowiska i mamy zgrubne oszacowania. Tymczasem ocena strat spowodowanych brakiem pewnych udogodnień w środowisku testowym wymaga głębszej wiedzy na temat procesu testowania, czasami wymaga doświadczeń z „pola bitwy”. Na takie właśnie ukryte konsekwencje oszczędzania na ułatwianiu życia testerom, postaram się zwrócić uwagę. Skupię się na następujących cechach ideału:

- Niezależność i samowystarczalność środowisk poszczególnych testerów
- Łatwość w odtwarzaniu stanów środowiska
- Łatwość w debugowaniu środowiska
- Automatyzacja testów powtarzających się

Niezależność i samowystarczalność środowisk

W idealnym świecie każdy tester powinien mieć w pełni niezależne i samowystarczalne środowisko. Myślę, że określenie „niezależne” jest zrozumiałe. Mogą się zrodzić wątpliwości, co mam na myśli pisząc „samowystarczalne”. Wszystko wyjaśnię. Generalnie chodzi o to, aby każdy tester mógł przeprowadzić swoje testy bez potrzeby koordynacji z kimkolwiek i bez potrzeby proszenia kogokolwiek o wykonanie jakiejś operacji, bez której rezultatów testy nie mogą być doprowadzone do końca.

W firmie, gdzie środowisko produkcyjne jest złożone a testerów jest 15, oznaczałoby to konieczność konfiguracji 15 pełnych środowisk. W większości firm jest to niewykonalne a jeżeli nawet wykonalne fizycznie, to na pewno niewykonalne finansowo. Jednak poważne zaniedbanie tematu niezależności może mieć bardzo kosztowne konsekwencje.

Jeżeli mamy w zespole 5 testerów, i wszyscy testują tę samą aplikację typu klient-serwer to istnieją różne możliwe konfiguracje. Rozważmy sytuacje skrajne - Wszyscy testerzy są podłączeni do jednego serwera, bądź też każdy ma swoje środowisko i swój serwer. Pierwsze rozwiązanie ma nad drugim tę przewagę, że jest prostsze w konfiguracji. Drugie ma

jednak pewną zaletę, która czasami ma kapitalne znaczenie - środowiska testerów są **niezależne**. Każdy może przeprowadzić swoje testy bez potrzeby koordynacji z innymi. Każdy może prowadzić testy bez ryzyka, że niechcący popsuje coś innemu testerowi i bez strachu, że ktoś zniszczy mu jego dane, które przygotowywał przez ostatni tydzień. Możliwe straty wynikające z takich pomyłek są różne - od żadnej po zniweczenie tygodnia pracy kilku osób. Jeżeli wykonujemy tylko proste testy typu nasza akcja - reakcja systemu, to ryzyko kolizji i ewentualne straty są niewielkie. Jednak przy testowaniu aplikacji biznesowych takie proste testy stanowią tylko część wszystkich zaplanowanych scenariuszy. Aby przetestować np. nowy sposób naliczania należności za usługę, gdzie w grę wchodzi zniżki i staż klienta, wykonanie testu wymaga wykonania kilku akcji, często w określonej kolejności ze względu na testowane zależności czasowe. Inny przykład to testowanie funkcjonalności odpowiedzialnej za raporty. Jeżeli chcemy sprawdzić czy odpowiednie pola w raporcie odpowiednio się sumują, to musimy wprowadzić odpowiednie dane szczegółowe będące bazą dla raportu.

Jak może dochodzić do „*tragicznych*” pomyłek? Na przykład system pozwala poszczególnym testerom wykonywać testy prawie niezależnie w jednym środowisku. Każdy może wykonać większość kroków bez obawy, że coś komuś popsuje. Jednak wszyscy muszą uważać, żeby nie wykonać ostatniego kroku - procesu, który np. bierze dane wygenerowane przez wszystkich testerów, po czym zmienia zawartość bazy danych w taki sposób, że poprzednie dane testowe stają się nieaktualne, tzn. nie można ich już użyć do powtórzenia testów. Oczywiście zmiany w bazie danych są odwracalne, ale jeżeli tym ostatnim krokiem był proces uaktualniający, globalnie, dane związane z należnościami za cały miesiąc, to złożoność tych zmian może czynić je nieodwracalnymi z praktycznego punktu widzenia. Jeżeli coś takiego się wydarzyło i został zniweczony tydzień pracy 5 ludzi, to kierownik zespołu testerów, któremu się to komunikuje, powinien w pierwszej kolejności dziękować Bogu, że się wogóle o tym dowiaduje. Jeżeli szkoda powstała w wyniku akcji niedoświadczonego testera, który bardzo boi się utraty pracy, to może on spróbować ukryć ten fakt przed innymi i próbować zamazać ślady. Kierownikowi zespołu zaraportuje, że wszystko jest w najlepszym porządku i system pójdzie na produkcję nieprzetestowany. A o tym, jak kosztowne jest wprowadzenie wadliwego oprogramowania na produkcję, wie każdy doświadczony informatyk.

Oczywiście, niektórzy czytelnicy mogą mieć wrażenie, że wyolbrzymiam problem, lub też, że problem ma proste rozwiązania:

1. *Skoro ten nieodwracalny krok jest tak niebezpieczny to najlepiej go nie wykonywać.*

To nie jest takie proste. Jeżeli jest to proces standardowo wykonywany na produkcji, to koniec końców, musi on być też wykonany w testach

2. *Testerzy mogą koordynować między sobą swe poczynania*

Tak, to prawda. Co więcej, testerzy tak właśnie robią. Ale w pośpiechu łatwo o pomyłkę. A nawet, jeżeli nie ma pośpiechu, to przy wykonywaniu skomplikowanych scenariuszy ze złożonymi danymi każdy może się pomylić.

Sprawa się pogarsza, jeżeli takich nieodwracalnych procesów jest więcej i to one są akurat testowane. Jeżeli nawet nie dochodzi do pomyłek i nawet jeżeli testerzy sobie nawzajem nic nie popsuje to pojawia się następny problem - testy jednych wstrzymują testy drugich. Jedna grupa testerów nie chce uruchamiać nieodwracalnych procesów w swoich testach. Druga musi to zrobić. Jeżeli chce się to pogodzić, to grupa druga musi poczekać, aż pierwsza grupa zakończy swoje testy. A w informatyce czekanie kosztuje. Kosztuje tym

więcej im więcej jest osób w dziale testów i im wyższe są kwalifikacje testerów (czyli ich pensje).

Rozwiązaniem tego problemu jest danie każdemu testerowi niezależnego środowiska. W zależności od sprzętu, na którym testy są wykonywane, cena zasobów potrzebnych na powielenie środowiska (procesory, pamięć RAM i dyskowa) może się różnić. Jednak ceny zasobów komputerowych systematycznie tanieją (licencje na oprogramowanie niestety nie). To jednak niestety nie wszystko. Ktoś musi środowiskami testowymi zarządzać. Jeżeli 10 testerów pracowało na jednym środowisku i teraz chcemy każdemu z nich dać osobne środowisko, to osoba odpowiedzialna za konfigurację tych środowisk, będzie miała 10 razy więcej pracy. Jej czas musi być uwzględniony w naszych kalkulacjach. Podejmując decyzje w kwestii środowisk testowych, musimy więc uwzględnić następujące elementy:

- jak bardzo poszczególni testerzy są zależni od drugich, jak długie mogą być zastoje niektórych grup,
- ile czasu zostanie zmarnowane przez ewentualne pomyłki,
- koszt osobodnia testera,
- koszt zasobów sprzętowych i licencji potrzebnych do dania każdemu testerowi osobnego środowiska,
- koszt konfiguracji i utrzymania tych środowisk,

Te parametry na pewno różnią się w zależności od projektu, i może się zdarzyć, że operowanie testerów na jednym środowisku, nawet po uwzględnieniu kolizji, jest finansowo bardziej opłacalne niż dawanie testerom osobnych środowisk.

A teraz coś o samowystarczalności. Najłatwiej będzie mi to zilustrować przykładem.

Wyobraźmy sobie, że w banku testuje się system windykacyjny. Raz na jakiś czas, system łączy się z systemem obsługi kart kredytowych, aby np. zablokować kartę dłużnika. Jeżeli środowisko jest samowystarczalne to tester ma do dyspozycji obydwa systemy – windykacyjny i do obsługi kart. Kiedy testuje scenariusz, w którym karta kredytowa dłużnika jest blokowana, może natychmiast sprawdzić czy system zablokował kartę czy też nie. Jeżeli tak się nie stało, to może łatwo powtórzyć test i zobaczyć, co nie działa.

A teraz wyobraźmy sobie inną sytuację. Każdy tester ma swoją kopię systemu windykacyjnego. Ale system obsługi kart kredytowych jest tylko 1 – ten używany na produkcji. Dzieje się tak - na przykład - dlatego, że pojedyncza licencja jest bardzo droga. Testujemy znów scenariusz z dezaktywacją karty dłużnika. Od obsługi technicznej systemu obsługi kart dostajemy pulę numerów kart, które na pewno nie będą użyte na produkcji, więc mogą być wykorzystane do testów.

Jest 9 rano. Tworzymy w systemie windykacyjnym scenariusz, w którym saldo i historia klienta powodują wysłanie żądania dezaktywacji karty. Żądanie jest wysyłane do systemu obsługi kart. Ponieważ, jest to system produkcyjny, zawierający informacje poufne, nie możemy ot, tak sobie sprawdzić czy karta jest dezaktywowana. Musimy zadzwonić do obsługi systemu kart i zapytać się jaki jest stan karty o numerze taki a takim.

Jest 9:30. Dzwonimy. Nikt nie odbiera.

O 10 udaje nam się kogoś złapać:

Tester czyli my: Słuchaj, jaki jest stan karty XXX? ”.

Ktoś z obsługi systemu kart: Wiesz co, musisz z tym poczekać. Coś się u nas sypie i mamy urwanie głowy. Zadzwoń o 11.

Cóż robić. Zabieramy się za coś innego (jeżeli mamy co innego do roboty) a przynajmniej udajemy, że coś robimy.

O 10:45 mamy zebranie zespołu.

Wracamy do biurka o **11:30**. Dzwonimy:

Tcm: No i jak?

Kzosc: Podaj mi jeszcze raz numer karty. Achaaaa, tiaaa... Jest aktywna.

Do diabła! A miała być dezaktywowana.

Tcm: A czemu? Wysłałem żądanie dezaktywacji

Kzosc: Tak? To poczekaj sprawdzę logi. Zadzwonię do Ciebie.

O 11:45 dzwoni:

Kzosc: No nie wiem co jest nie tak. Muszę poszperać głębiej. Mam teraz coś pilnego do zrobienia. Zajmę się tym po lunchu, OK?

Tcm: Dobra.

Jest 12:00, idziemy na lunch, bo nie mamy co robić, musimy czekać.

O 13 dzwonimy:

Kzosc: Dopiero wróciłem. Już patrzę.

Dzwoni o **13:20**:

Kzosc: No nie wiem, wygląda jakby żadne żądanie nie przyszło.

Sprawdzamy połączenie – powinno działać. Próbujemy jeszcze raz – dostajemy komunikat „Nie można dezaktywować karty nieaktywnej”. Czyli już gdzieś w systemie jest zapamiętane, że takie żądanie poszło. Dzwonimy. Słyszymy:

Kzosc: Teraz nie mogę, jestem na spotkaniu. Zadzwoń o 14:30.

Dzwonimy, mówimy co się dzieje.

Kzosc: To może żądania nie przechodzą przez interfejsy. Zadzwoń do interfejsów.

Dzwonimy. Ktoś odbiera i ma dla nas czas (a to dopiero zbieg okoliczności!). Okazuje się, że interfejs łączący środowisko testowe z systemem kart jest wyłączony. Włączają go. Żądanie przechodzi do systemu kart.

O 15 dzwoniemy i dowiadujemy się, że żądanie zostało odrzucone z błędem „Zła suma kontrolna karty”. Sprawdzamy – rzeczywiście. Pomyliliśmy się przy jednej cyferce.

Jest 15:30. Tworzymy nowy scenariusz.

O 16:00 idzie nowe żądanie. Dzwoniemy, nikt nie odbiera. No tak, tam pracują od 8 do 16. Musimy poczekać z tym do jutra.

Po 2 dniach okazuje się, że żądania są wysyłane w niepoprawnym formacie. A gdybyśmy mieli system obsługi kart do własnej dyspozycji, wykrylibyśmy to w 2 godziny. I niech mi nikt nie mówi, że przesadzam. Byłem świadkiem takich historii nie raz i nie dwa razy.

Jak to wdrożyć

No cóż, najprościej jest kupić odpowiednią ilość sprzętu i licencji. Jeżeli nie da się każdemu dać pełnego środowiska, to przynajmniej spróbujmy dać tym środowiskom tyle niezależności ile tylko jest możliwe.

Warto zastanowić się nad automatyzacją procesu tworzenia środowiska. To może pozwolić osobom utrzymującym system na spędzanie mniej czasu na ustawianiu pojedynczego środowiska a co za tym idzie będą mogli ich tworzyć więcej.

Przy negocjacjach zakupu oprogramowania specjalistycznego, warto mieć tę kwestię na uwadze. Jeżeli jesteśmy dużym klientem i płacimy ciężkie pieniądze, spróbujmy wymusić na dostawcy oprogramowania darmowe licencje do testów. Jeżeli damy się „wpuścić” w jedną licencję, na dodatek z kluczem sprzętowym, tak że produkcja będzie musiała dzielić klucz z testerami, bo nie będzie można korzystać jednocześnie, to od razu możemy czas testów wykorzystujących to oprogramowanie pomnożyć przez 5. Będzie to klasyczny brak samowystarczalności, o którym pisałem powyżej.

Łatwość w odtwarzaniu różnych stanów środowiska

Jeżeli test przebiega pomyślnie to po wykonaniu scenariusza temat się właściwie kończy. Jeżeli jednak test nie przejdzie to możemy mieć do czynienia z następującymi sytuacjami:

- 1. Błąd jest ewidentny, nie ma wątpliwości co poszło źle.** Dzwoniemy do programisty i mówimy co się stało. On patrzy w kod i widzi to, co jest nie tak albo powtarza nasz scenariusz u siebie i dostaje ten sam błąd. Ma wszystkie informacje potrzebne do naprawienia błędu

Powyższy scenariusz był najbardziej pozytywny z możliwych. Może jednak nie być tak łatwo

- 2. Błąd nie jest ewidentny.** Rezultaty nieznacznie różnią się od oczekiwań, ale jeżeli wykonujemy akurat skomplikowany scenariusz to nie jesteśmy pewni czy może nie popełniliśmy jakiejś pomyłki w scenariuszu. Najlepiej by było powtórzyć test jeszcze raz
- 3. Błąd jest ewidentny. Scenariusz skomplikowany i trudno odwracalny.** Dzwoniemy do programisty. Mówimy co jest nie tak. Ten próbuje u siebie. „Słuchaj” – mówi – nie udało mi się tego odtworzyć. Mógłbyś powtórzyć ten test jeszcze raz?”

4. **Jak punkt 3, z tym że programista może debugować w naszym środowisku.** Po wykonaniu scenariusza mówi – Już chyba wiem o co chodzi. Możemy ten test wykonać jeszcze raz?

We wszystkich tych sytuacjach możliwość szybkiego zapamiętania i odtworzenia danego stanu środowiska może nam oszczędzić godziny na każdym powtórzeniu scenariusza. To w sumie potrafi dać kilka dni oszczędności.

Jak to wdrożyć

Najprostszym sposobem zapamiętywania stanu środowiska jest robienia jego kopii. Może to być długotrwałe i miejsce-żerne dla dużych środowisk.

Inną drogą jest utworzenie skryptów, które szybko odtworzą stan środowiska z przed wykonania danego scenariusza.

Możliwość debugowania¹ w środowisku testowym

Ta cecha środowiska jest z obopólną korzyścią dla testera i programisty a także dla całego projektu.

Środowisko, na którym pracuje tester powinno być zorganizowane w taki sposób, aby na wypadek błędu, który jest trudny do odtworzenia w środowisku deweloperskim, programista mógł uruchomić debugger w środowisku testera.

Podobnie jak w innych przypadkach, dostępność tego ułatwienia może zaoszczędzić kilka dni pracy testera i programisty. Niejednokrotnie, środowisko aplikacji ma tyle stopni swobody, że ręczne odtworzenie środowiska testera w środowisku programisty może być albo praktycznie niemożliwe albo bardzo pracochłonne. Jeżeli w środowisku testowym jest debugger, to nie ma tematu odtwarzania środowisk. Tester odtwarza błąd, woła programistę a ten może prześledzić działanie programu w środowisku w którym ten błąd występuje.

Jak to wdrożyć

Trzeba w środowisku testowym zainstalować większość niezbędnych narzędzi, z których korzystają programiści przy szukaniu błędów. Czy jest to proste czy trudne – zależy od projektu. Ale w zasadzie, nie powinno być bardzo trudne ani bardzo kosztowne.

Automatyzacja testów powtarzających się

Automatyzacja to wśród testerów ostatnimi czasy bardzo popularne słowo. I na pewno warto się do automatyzacji przyczepić. Jest kolosalnym błędem próbować zautomatyzować wszystko (gorszym od nieautomatyzowania niczego). Ale automatyzacja rozważnie i mądrze wybranych zadań, może być źródłem wielotygodniowych (sic!) oszczędności.

¹ Debugowanie – Proces uruchamiania programu krok po kroku, z możliwością śledzenia zmian będących konsekwencją każdego kroku.

Jak to wdrożyć

Po prostu zainteresować się tematem.

Przepraszam, że ten temat traktuję tak skrótowo. To nie przez lenistwo. Automatyzacja testów jest procesem złożonym i nawet jej pobieżne potraktowanie wymaga osobnego artykułu. Dość ciekawą i wyważoną dyskusję na temat automatyzacji testów można znaleźć na przykład na www.qaforums.com. Ale nie tylko. Temat automatyzacji jest dość popularny w środowisku testerów.

Podsumowanie

Próbowałem pokazać jakie mogą być konsekwencje pewnych braków w środowiskach testowych. Czy to znaczy, że uważam narzędzia i ułatwienia opisane powyżej za rzecz niezbędną i uważam managerów, którzy z nich rezygnują za ignorantów? Absolutnie nie. Nie jest celem tego artykułu zdiagnozować, co jest najważniejsze i wskazać jedynie słuszną drogę. Chciałem tylko zasygnalizować istnienie zjawisk, które mogą umknąć uwadze osób na stanowiskach kierowniczych. Koszty generowane przez te zjawiska warto wziąć pod uwagę obok cen sprzętu i licencji. Każdy manager na ich podstawie będzie musiał sam podjąć decyzję, gdzie, na tej wadze szalkowej - jaką jest jego projekt, przesunąć odważnik tak aby waga pozostała w równowadze. Czy bliżej bieguna oszczędności, czy bliżej bieguna usprawniania procesów.

COMPUWARE®



THE POWER TO Create, Confirm, Control

Software and services from Compuware help create applications that move into action faster. Solutions for development, quality assurance and operations let you confirm efficient deployment and ongoing availability with confidence. Achieve control over application performance and exceed the quality your end users demand—now.

The power is right here.

The leader in IT value.

COMPUWARE
www.compuware.com



Jak raportować błędy?

Anita Pankowska

Computerland

Anita Pankowska jest absolwentką Wydziału Elektroniki, Elektrotechniki i Automatyki Akademii Górniczo-Hutniczej w Krakowie, kierunek matematyka stosowana oraz Podyplomowego Studium - Bankowość w Gospodarce Rynkowej w Akademii Ekonomicznej w Krakowie. Uczestniczyła w wielu projektach informatycznych jako analityk, programista, tester w Polsce oraz przez kilka lat w Izraelu. Obecnie pracuje jako konsultant do spraw testów w firmie ComputerLand. Zwolenniczka automatyzacji procesów testowania.



Jak raportować błędy?

Anita Pankowska

Computerland

Nawet najlepiej napisany program nie jest wolny od błędów. Rolą testerów jest ich zdiagnozowanie i poinformowanie programistów o ich zaistnieniu. Jednak nie wystarczy po prostu zasygnalizować wystąpienie błędu, trzeba to jeszcze zrobić efektywnie. W tym artykule postaram się, zarówno na podstawie własnego doświadczenia, jak i lektury artykułów innych osób działających w naszej branży, sprecyzować, co składa się na dobre zgłoszenie.

Zanim prześlemy informację programiście, zastanówmy się: czy raportujemy wystąpienie błędu, czy też postulujemy rozszerzenie funkcjonalności, a więc dodanie nowej cechy, która poprawiłaby jakość systemu. Od tego zależy priorytet zgłoszenia. Oczywiście jest, że usterki usuwane są w pierwszej kolejności, a potem może znaleźć się czas na realizację ulepszeń. W tym miejscu, niejako mimo woli, wyłania się problem kategoryzacji błędów. Na rynku istnieje wiele aplikacji wspomagających zarządzanie bazą „bugów”, jednak nawet w przypadku ich stosowania istotne jest wypracowanie własnych standardów pomocnych przy kategoryzacji błędów. Podczas standaryzacji niezbędna jest współpraca testerów z programistami, a niejednokrotnie również z analitykami. Ważne jest np., aby określenie „błąd krytyczny” dla wszystkich zainteresowanych stron oznaczało to samo. Każdy zespół wypracowuje własną skalę oceny błędów, jednak ogólnie można przyjąć, że najwyższą kategorię uzyskują błędy krytyczne blokujące działanie systemu, a następnie krytyczne nieblokujące, przy których dalsza praca jest niemożliwa. Następne w kolejności to błędy o wysokiej kategorii, potem średniej, a w końcu niskiej, tzw. błędy komunikacji. Błędy zgłaszamy od momentu zatwierdzenia tzw. „deadline’u”, tzn. zbudowania wersji oprogramowania przeznaczonej do testów, wyposażonej w unikalny numer. W tym czasie programiści nie dokonują żadnych zmian w środowisku testowym. Wyjątkowo, w przypadku wystąpienia błędów krytycznych blokujących, kierownictwo projektu może podjąć decyzję o przerwaniu testów, budowie nowej wersji i rozpoczęciu całego procesu od początku. Testowanie podzielone jest na etapy, a długość każdego etapu zależna jest od terminu realizacji projektu.

Poruszone powyżej kwestie dotyczą „szufladkowania” problemów. Niemniej ważny jest opis ich powstawania, krok po kroku. Zgłaszamy błąd, ponieważ chcemy, aby został naprawiony. Nasz problem powstał - być może - z winy programisty i być może mamy rację złoścąc się na niego, jednakże błąd zostanie naprawiony szybciej, gdy dostarczymy wszystkich informacji, których programista potrzebuje. Nie możemy ograniczyć się do lakonicznego stwierdzenia: „*To nie działa.*” Na pewno problem wróci do nas nierozwiązany z dopiskiem „*proszę doprecyzować*”. Jedną z najlepszych metod zgłaszania błędów jest pokazanie ich programiście, jednak często jest to niemożliwe ze względu na rozproszenie zespołów. Wówczas nie pozostaje nic innego, jak opisanie czynności prowadzących do błędnego zachowania aplikacji, krok po kroku. Rozpocząć należy od ogólnego opisu problemu. Następnie przystępujemy do odtwarzania sytuacji. Niezbędne jest podanie pełnej ścieżki wywołania funkcji, w której występuje błąd, potem następuje drobiazgowy opis wykonanych czynności. Jest to ważne, ponieważ często nawet drobna zmiana w scenariuszu, prowadzi do różnych rezultatów. Dla programisty niezwykle istotny jest nasz precyzyjny opis,

ponieważ dostarcza mu więcej danych, niż czasem nam się wydaje. Dokładny opis błędu służy również samym testerom. Często zdarza się nam wracać do własnych zgłoszeń po upływie czasu i niejednokrotnie trudno jest zrozumieć sens wypowiedzi, która w momencie tworzenia wydawała się tak oczywista. A pomyślmy o kolegach, którzy z jakiś przyczyn muszą przejść po nas naszą „działkę” i tracą czas na rozszyfrowanie tego typu informacji. Jeśli testujemy systemy wielodostępne, w których poszczególni użytkownicy posiadają różne uprawnienia, istotne jest również zaznaczenie, dla którego z nich wystąpił błąd. W przypadku systemów Web -owych, ważny jest system operacyjny, na którym przeprowadzamy testy, jak również przeglądarka internetowa oraz jej wersja.

Niezwykle pomocne bywa dokonanie zrzutów ekranów. Często w przypadku niemożności odtworzenia błędu jest to jedyny dowód na to, że miał on jednak miejsce, a przyczyna, że nie jesteśmy go w stanie powtórzyć tkwi w specyficznych okolicznościach jego zaistnienia. Bywa, że programista jest w stanie ze zrzutu odczytać te okoliczności. Nie sugerujemy programiście, co powinien naprawić. Wyobraźmy sobie wizytę u lekarza, w trakcie której sami sobie stawiamy diagnozę, zamiast przedstawić mu objawy choroby. Tak samo jest z programistami. Dostarczenie własnej diagnozy czasem może być przydatne, ale zawsze właściwsze – jest opisanie symptomów.

Podsumowując:

Zgłaszając błędy starajmy się być precyzyjni, wyrażajmy się jasno, tak by interpretacja opisu była jednoznaczna. Jeśli pojawiają się komunikaty o błędach, notujmy je, a najlepiej dokonujmy zrzutów ekranów. Bądźmy gotowi zawsze dostarczyć programiście dodatkowych informacji. Pamiętajmy o numerze wersji, w której wystąpił błąd. Jest to zawsze jedna z najważniejszych informacji.

Ekstremalne zręczne badawcze oparte na ryzyku piramidy (Extremely Agile Exploratory Risk – based Pyramids)

Bogdan Bereza-Jarociński

bbjTest

Bogdan Bereza-Jarociński

Psycholog z Uniwersytetu Warszawskiego i Londyńskiego, informatyk z uniwersytetu w Lund. Testował, zarządzał, automatyzował, koordynował lub ulepszał procesy testowe zarówno w zastosowaniach wbudowanych (telekomunikacja – Ericsson, sygnalizacja kolejowa – Bombardier) jak i otwartych (bazodanowe, Java). Lubi udzielać się na międzynarodowych konferencjach i prowadzić szkolenia z dziedziny testowania. Od 2002 roku prowadzi w Polsce szkolenia m. in. na certyfikat ISEB.



Ekstremalne zręczne badawcze oparte na ryzyku piramidy²

(Extremely Agile Exploratory Risk – based Pyramids)

Daję zawsze jedną radę uczestnikom moich kursów ISEB: *jeżeli mają problem z odpowiedzią na niejasne pytanie egzaminacyjne, powinni wybrać odpowiedź ze słowem „ryzyko”*.

Słowa mogą połamać nam kości: są bardzo mocne. Nazywając rzeczy możemy zmienić ich naturę, staną się one inne!

Dawno temu testowanie wcale nie oznaczało myślenia z wyprzedzeniem, ale prostą zabawę z systemem, w nadziei na znalezienia jakiś usterek. Jak testowanie dojrzało, niektórzy praktycy poważyli się stwierdzić, że myślenie z wyprzedzeniem i zapisywanie tego co testujemy może gwarantować lepsze testowanie, mierzalną jakość oraz bardziej realistyczną ocenę ryzyka. Jakiś czas później myślenie i spisywanie przypadków testowych, by mogli je poznać inni testerzy i można je było powtórzyć nazwane zostało „testowaniem skryptowym” (script testing); dobre testowanie nagle zaczęło oznaczać nie myślenie z wyprzedzeniem ale po prostu ... zabawę z systemem – opatrzone to mianem „badanie” (exploration) !!!

Podobnie, setki lat temu ludzie myśleli, że władza jest dana od Boga i to daje prawo do wywłaszczania rezultatów pracy innych ludzi. Następnie odkryto wolny rynek i prawo do indywidualnej własności, co spowodowało bezprzykładowy wzrost bogactwa. Tym niemniej średniowieczne idee powróciły pod maską socjalizmu, zgodnie z którym „prawa historii” lub „sprawiedliwość społeczna” dała pewnym ludziom prawo do łupienia innych ludzi. Ten ciemny okres skończył się – raczej niespodziewanie – około 1989 w większości krajów. Jak długo będzie trwała dyktatura ekstremalnego zręcznego badawczego opartego na ryzyku testowania?

Instrukcja obsługi sprzedawcy narzędzi rejestrująco - odtwarzających rzadko zawiera sformułowania typu:

„UWAGA: REJESTROWANIE MOŻLIWE TYLKO GDY TESTOWANE OPROGRAMOWANIE JEST PRAWIE GOTOWE”

lub

„SZKODLIWE DLA STRUKTURALNEGO TESTOWANIA”

Podobnie, w prezentacjach neofitów testowania eksploracyjnego unika się stawiania tego samego, jakże prostego przesłania. Dlaczego?

Słowa mogą ponoć połamać kości – lub przynajmniej nagiąć prawdę. Wiele idei w testowaniu eksploracyjnym jest słuszne i użyteczne: używanie intuicji do generowania przypadków testowych, aktywne szukanie informacji, gdy brak wymagań, wyprowadzanie podstaw nowych przypadków testowych z już istniejących. Tym niemniej, naprawdę nie należy „w tym samym czasie uczyć się aplikacji, wykonywać przypadki testowe i projektować nowe”, jak określa to motto testów eksploracyjnych. Można zastosować te heurystyczne metody równie dobrze przed rozpoczęciem i – co za okropna myśl – przelać je na papier jako pogardzany „skrypt”.

Podczas wykonywania testów, projektowanie nowych testów oparte się na wynikach i doświadczeniach z już przeprowadzonych testów jest oczywiście dobrym pomysłem. Nic nie powstrzymuje cię od użycia tego w celu rozbudowy - na piśmie – już istniejących skryptów testowych. To nie jest żadna podstawowa zmiana paradygmatu – jak ostrzegano!!

² Autor dostarczył artykuł w wersji anglojęzycznej, tłumaczenia dokonała redakcja i na nią też spadają wszystkie błędy w tekście i wypaczenia oryginalnych myśli autora

Główni wrogowie testów eksploracyjnych – testy „skryptowe” – nie stanowią większej przeszkody w elastyczności i kreatywności w myśleniu niż testy eksploracyjne w użyciu formalnej, strukturalnej techniki testowej. Poza wszystkim innym, nawet najbardziej lojalny tester eksploracyjny może gwałtownie chcieć użyć metody podziału na klasy równoważności lub zastosować testowanie syntaktyczne przy projektowaniu nowych przypadków testowych. Dostatecznie eksploracyjne, w moim pojęciu.

Co więcej, próbując zrobić „*materiały lub procedury, które umożliwiają powtórne użycie testów [...] lub przejrzeć wykonaną pracę*” można krępować wolnego ducha eksploracji („fiat ubi vult”). Tym niemniej, ten rodzaj argumentów jest bardzo podobny do argumentacji, jaką się czasami słyszy od programistów, uchylających się od umieszczania w kodzie komentarzy i pisania dokumentacji. Najwyraźniej takie niehumanitarne traktowanie zabija kreatywność prawdziwego programisty.

Testy eksploatacyjne zawierają naprawdę wielkie idee, jak projektowanie testów z wyprzedzeniem może – i powinno – być wzmacniane przez dodawanie nowych przypadków testowych, opartych na pomysłach powstałych przy testowaniu. To generuje sensowne i czasami nowe pomysły, co należy robić w sytuacji, jeżeli musimy testować w pośpiechu, bez wystarczającej specyfikacji. Dobre pomysły? Tak. Radykalne? Ledwo, ledwo. Nowe? Raczej nie. Jak na razie nikt nie sformułował wymagania by zaprzestać myśleć po rozpoczęciu testów!!!

Kiedy nie można wyspecyfikować testów z wyprzedzeniem, testowanie zgodne z zasadami ET oznacza po prostu stosowanie dobrych, elastycznych metod do generowania testów „w locie”.

W pewnym względzie, czytanie o ET jest podobne do czytania politycznego programu lub ideologicznego manifestu. Jest pełne dobrze brzmiących fraz, ale w tym samym czasie są nijakie i chyba nie unikalne dla tej konkretnej partii ... lub metody testowej. Przyjrzyjmy się kilku przykładom.

W rozdziale 13.2 książki „Practising exploratory testing” podany jest schemat procesu testowanie eksploracyjnego dla produktu. Zaczyna się od „*zbadań i analizy elementów produktu [...], generuje schemat pokrycia testowego*”. Pamiętamy zawartość planu testów z IEEE 829? Pozycje testów, cechy podległe testowaniu...Dość podobnie, nieprawdaż? Kontynuujemy. „*Zidentyfikować i przetestować wszystkie polecenia [...] z instrukcji obsługi*”. Testowanie oparte na wymaganiach, nieprawdaż? Oczywiście, instrukcja obsługi to nie specyfikacja, ale czy „eksploracja” polega na użyciu instrukcji, zamiast specyfikacji?. Każdy dobry podręcznik inżynierii oprogramowania wymienia listę nietypowych źródeł, gdy nie ma specyfikacji. Nie ma potrzeby używać ET dla tak oczywistych spraw.

„*Zdefiniuj przepływy [...] i wypróbuj każdy*”. Czyż nie jest to testowanie ścieżek lub użycie testowanie wg przypadków? Dobra metoda, ale czy naprawdę taka nowa i tak różna od istniejących jak nam sugerują?

„*Przetestuj wszystkie pola, do których można wprowadzać dane*”; „*sprawdź interfejs użytkownika zamiast standardowego*”; „*sprawdź integrację z zewnętrznymi aplikacjami*”. Dobra lista kontrolna, ale – na Boga – co to ma wspólnego z „eksploracją”? Podział na klasy równoważności, testowanie użyteczności, testy integracyjne – to doskonale znane sposoby.

„*Pamiętaj, w ET używamy maksymalnie umiejętności, zamiast kusić się na przedstawianie każdej akcji na piśmie*”. Gdybyśmy chcieli sformułować manifest „anty – eksploracyjny”, można by sformułować postulat: „my w testach skryptowych używamy maksymalnie umiejętności, zamiast kusić się na sprawdzanie każdej nowej ekscytującej idei, która się właśnie objawiła”. Nie byłoby to rzetelnie w stosunku do ET. Tym niemniej, zakładając, że w istniejących anty – eksploracyjnych testach chodzi głównie o „przedstawianie każdej akcji na piśmie” to założenie, że nie używa się w nich maksymalnych umiejętności także jest nierzetelne.

„Tester skryptowy powinien tylko obserwować, co skrypt każe mu obserwować”. Nieprawda. Jedyne sposoby, które umożliwiają interpretować „skryptowy” w ten sposób to stwierdzenie ”postępować na ślepo wg bardzo szczegółowego skryptu”, co jest poważnym nieporozumieniem. Pasywny, słabo motywowany „tester skryptowy” może tak pracować, ale pasywny, słabo motywowany „tester ET” może – na przykład – zawieść obserwując i badając pewien obiecujący (w błędy) zakres. Co więcej, w „testach skryptowych” mamy szansę obmyślenia i planowania co obserwować dwukrotnie: gdy przypadki testowe są projektowane z wyprzedzeniem, potem gdy testy są wykonywane.. Powinno to skutkować lepszym pokryciem testowym.

Uwielbiam – naprawdę!! – sposób w jaki ET przedstawia pewne wielkie idee. „*Jest to podstawowa zasada ET – to co dzieje się w mózgu testera*”; „*tester ET jest po pierwsze głównym projektantem testów*”. Dokładnie tak!. Zgadza się w pełni. Każdy tester powinien być po pierwsze głównym projektantem testów. Czasy, gdy testerzy pasywnie wykonywali przypadki testowe zaprojektowane przez innych (kto to są ci inni?) – minęły dawno temu. Te idee są jasno wyjaśnione w tak podstawowych materiałach jak sylabus Fundacji ISEB, organizacji raczej nie znanej z rewolucyjnego podejścia do testowania.

Przedstawię teraz przepis. Zbierzmy kilka dobrych i raczej mętnych haseł, np. „**błędy pojawiają się, bo nie jesteśmy doskonali**”, lub „**pojedynczy błąd może nic nie kosztować, ale może kosztować majątek**”. Obudujmy to regułami („**tester powinien dążyć do perfekcji**”, „**szacujmy koszt błędów przez rozmowy z akcjonariuszami**). Znajdźmy do tego nośną nazwę, np. Perfekcyjne Testowanie Makiaweliczne PTM i zwróćmy na to uwagę wszystkich zainteresowanych. Właśnie urodził się nowy paradygmat testowania! Któż wątpi, że te same reguły mogą być stosowane do testowania i w ogólności do zapewnienia jakości, bez konieczności nadawania im specjalnych nazw lub stwierdzenia, że proponują coś radykalnie nowego i różnego?

Niech mi będzie wolno stwierdzić kilka rzeczy. Tak, myślę, że testowanie eksploracyjne zawiera wiele rozsądnych idei. Tak, zdaję sobie sprawę, że pewne wyolbrzymienie i przejawianie, prawie religijna gorliwość mogą być przypisane do hierarchicznych, biurokratycznych i sztywnych idei o tym, jak powinno przebiegać testowanie, rozpowszechnionych w pewnych, na ogół dużych, organizacjach. Nie, nie wierzę, że testowanie eksploracyjne jest nowym paradygmatem i wnosi wiele nowych idei. Tak, nikt nie sugeruje, że ET może być zawsze używane. Powinno być stosowane w sytuacjach, gdy wyniki muszą być osiągnięte szybko i nie ma dostatecznej informacji dla bardziej strukturalnego podejścia. To, co ET proponuje, nie jest bardzo nowatorskie – oznacza po prostu szukania wymagań w sposób niestandardowy, projektowania i wykonywania testów równocześnie, wykorzystywania techniki orientowania się na zgadywanie błędów w sposób maksymalny. Jest to rozsądne, inteligentne podejście, zastosowanie dobrych praktyk nawet w niestandardowych sytuacjach. Ale to **NIE** jest „eksploracja”, to **NIE** jest alternatywa – i nie ma powodu, żeby tak to traktować. Wspaniałe nazwy nie ranią bezpośrednio, ale mogą kreować fałszywe cele i fałszywych bogów i będziemy tracić na nie czas – zamiast testować.

Wszystkie cytaty pochodzą z „Exploratory testing” Jamesa Bacha, w „The Testing Practitioner” Erika van Veenendaala.

Personel testujący - jaki powinien być?

Mariusz Janczewski

Mariusz Janczewski

Jest absolwentem matematyki Uniwersytetu Gdańskiego. Ukończył Podyplomowe Studium Inżynierii Oprogramowania na Politechnice Gdańskiej oraz Studium Bankowości i Finansów w Gdańskiej Akademii Bankowej. Pracował jako programista i szkoleniowiec. Zajmował się testowaniem, prowadzeniem wdrożeń i wsparciem użytkowników. Brał udział w licznych projektach w sektorze bankowym i ubezpieczeniowym. Interesuje się inżynierią oprogramowania, szczególnie zaś zagadnieniami zapewnienia jakości.



Personel testujący - jaki powinien być?

Mariusz Janczewski

Do czego służą testerzy?

Gdyby nie błędy powstające podczas wytwarzania oprogramowania, gdyby programy powstawały idealne, a działanie ich było zawsze poprawne, to testerzy nie byli by potrzebni. W pewnym sensie my testerzy żerujemy na porażkach innych osób. To ich błędy dają nam pracę. Czynią, że jesteśmy potrzebni. Jak wiemy ludzie nie są doskonali, a „Errare humanum est” – „błądzenie jest rzeczą ludzką”. Warunki, w jakich żyjemy i pracujemy, wywierają znaczący wpływ na przebieg projektów informatycznych, w których uczestniczymy. Konkurencja, prędkość zachodzenia zmian, wewnętrzna i zewnętrzna presja na szybkie dostarczenie rozwiązań wywołują warunki pracy obciążone dużym stresem. Realizacja w zbyt krótkim czasie zbyt wielu czynności i zadań, szybkie zmiany w technologii, to środowisko, w którym działamy, na co dzień. Otoczenie to wpływa nieustannie na nas wykonawców oraz na cały proces wytwarzania oprogramowania, stanowi ono doskonałą pożywkę dla powstawania błędów małych i mało istotnych, dużych i ważnych, wreszcie krytycznych i bardzo niebezpiecznych.

Oczywiście trudne warunki pracy w sektorze informatyki nie są w żadnym wypadku czymś wyjątkowym w dzisiejszym świecie. Inne zawody funkcjonują w porównywalnie trudnych warunkach. Jednak należy pamiętać, że inżynieria oprogramowania ma na tle innych rzemiosł bardzo krótką historię. Możemy poszczycić się dopiero skromną kilkudziesięcioletnią historią, co na tle innych bardziej „historycznych” zawodów wygląda nadal tak sobie. Można powiedzieć, że w pewnym sensie ciągle raczkujemy.

Te jak i inne powody, a wymieniać je można by jeszcze przez długi czas, dają nadzieję, że testerzy długo jeszcze będą potrzebni. Jeden z moich profesorów określił, to następująco „błędy były, są i będą”. Możemy podejrzewać, że testerzy będą potrzebni zawsze. Ich obecność wprawdzie nie likwiduje zagrożenia wystąpienia błędów, ale potrafi to zagrożenie w znaczący sposób zminimalizować.

Reprezentanci wielu firm powiadają, że najważniejsze aktywa tych firm, to ich pracownicy. Teza ta brzmi pięknie i miło ją słyszeć będąc pracownikiem, czy jednak na pewno jest ona prawdziwa? Autor jednej z moich ostatnich lektur odrzuca ten pogląd, twierdząc, że owszem pracownicy są najważniejszymi aktywami firmy, ale tylko dobrzy pracownicy. Trudno nie zgodzić się z tym stwierdzeniem. Lepiej mieć dobrych pracowników niż kiepskich. Najlepsi są zwykle tymi, którzy napędzają firmy i ciągną za sobą pozostałych. Sukces czeka na firmy, które potrafią wyłowić z rynku najlepszych ludzi, ściągnąć ich do siebie i właściwie wykorzystać.

Dobry pracownik, dobry tester

Sprawa może wydawać się prosta: zatrudniamy najlepszych i już. Jak to jednak zrobić? Jak do nich dotrzeć? Jak rozpoznać? Co to wreszcie znaczy dobry, najlepszy tester? Proponuję od tego zacząć. Zastanówmy się, jakie są atrybuty dobrego testera. W ten sposób

w pole naszych zainteresowań trafia pojedynczy pracownik, ten który nas najbardziej w tym momencie interesuje - pojedynczy tester.

W wielu firmach istnieje tendencja do przydzielania roli testerów produktu pracownikom z małym doświadczeniem, np.: studentom, praktykantom, osobom nowozatrudnionym. Inne osoby przydzielane do działu testów, to często osoby niebędące bezpośrednio związane z tworzeniem produktu lub takie, które nie pasują do innych zespołów. Taka praktyka jest wyjątkowo krótkowzroczna. Umiejętności potrzebne by efektywnie testować oprogramowanie nie są mniejsze niż te, które są wymagane przy jego tworzeniu. W rzeczywistości personel powinien czerpać z bardzo szerokich doświadczeń. Napotyka on bowiem sytuacje i problemy, których deweloperzy raczej nie spotykają.

Czy istnieje krótki i prosty algorytm wskazujący na cechy dobrego testera? Myślę, że tak. Dobry tester, to po prostu „człowiek dobrej roboty”. Znając wielu bardzo dobrych testerów jestem przekonany o tym, że postawieni przed całkiem odmiennymi zadaniami, po krótkim czasie wykonywaliby je zapewne z równym zaangażowaniem i z równie świetnymi wynikami.

Cóż można i trzeba dodać? Ponieważ testowanie to zajęcie szczególne, polegające na polowaniu na błędy, stanowiące ostatnią linię obrony przed przekazaniem oprogramowania zawierającego błędy do klienta, myślę, że warto powyższy algorytm rozwinąć i doprecyzować.

Instykt testera

Według mnie najlepszymi testerami są osoby będące pewnego rodzaju pasjonatami zniszczenia. Osobom tym szczególną zabawę sprawia znęcanie się nad przekazanym do testów oprogramowaniem. Radość daje im wykrywanie błędów. Do tego trzeba mieć nosa i naprawdę to lubić. Zgoda nie jest to atrybut precyzyjny, nie wiadomo jak rozpoznać w trakcie rozmowy kwalifikacyjnej takiego pracownika. Ale część kandydatów zdradza się przejawiając tkwiącą w nich ciekawość i zaangażowanie. Podczas pracy osoby te prezentują instykt, który wraz z nabieranym przez nie doświadczeniem czyni z nich prawdziwych łowców błędów.

Z takimi właśnie pragniemy współpracować i są oni najbardziej cenieni przez najlepszych programistów. Tak, doświadczenie pokazuje, że najlepsi programiści rozpoznają z łatwością najlepszych testerów i czynią wszystko by z nimi współpracować. Programiści „tacy sobie” lekceważą weryfikację własnego kodu i zawsze są przekonani o tym, że błędów nie ma, release jest od nich wolny i można go od razu wysłać do klienta.

Ron Patton w swej książce pisze, że dobry tester jest odkrywcą i łowcą problemów. Osoba taka jest nieustępliwa i twórcza. Dodatkowo dobry tester powinien być perfekcjonistą, wykazywać rozsądek, być taktownym i dyplomatycznym i umieć przekonywać. Zastanówmy się przez chwilę nad tymi atrybutami.

Wytrwałość

O chęci odkrywania i łowieniu problemów wspomniałem już wyżej. Ponieważ błędy potrafią pojawiać się i znikać, często trudno jest je zlokalizować i właściwie opisać. W tej sytuacji testowanie polega na wielokrotnym powtarzaniu tych samych czynności. Praca taka wymaga często bardzo wiele czasu i wysiłku. Do jej wykonywania potrzebne są osoby niezwykle wytrwałe i cierpliwe. Nie jest to praca dla kogoś, kto szybko się poddaje.

Komunikacja

Jedną z najważniejszych cech są zdolności komunikacyjne. Można powiedzieć, że są ważne w przypadku każdego pracownika i będzie to prawda. Jednak znam wielu doskonałych programistów, z którymi trudno się dogadać, a tester musi to zrobić. Dobry tester musi kontaktować się ze znaczącą grupą osób. Musi umieć rozmawiać zarówno z użytkownikami jak i twórcami programu. A obie grupy najczęściej posługują się innym żargonem. Dyskusje z użytkownikiem skupiają się na tym, co system robi lub nie robi dobrze. Informacje te muszą być następnie przetłumaczone i przekazane członkom zespołu tworzącego system. Zadanie to spada na testerów.

Empatia

Do komunikacji trzeba dodać empatię, robiąca ostatnimi czasy karierę w literaturze dotyczącej psychologii i komunikacji. Jest to spowodowane tym, że każdy kto związany jest z wytwarzaniem oprogramowania ma swoje obawy i troski. Użytkownicy boją się, że system jaki otrzymają nie będzie spełniał ich potrzeb. Programiści obawiają się, że nie powiedziano im czego tak naprawdę chcą użytkownicy i że będą musieli przebudowywać cały system. Kierownik projektu obawia się, że cały projekt się „zawali” i wszystkie pretensje będą kierowane do niego. Testerzy kontaktują się z każdą z tych grup i muszą rozumieć i współodczuwać z każdą z nich. Poprzez okazywanie empatii testerzy mogą minimalizować ryzyko powstania atmosfery wrogości i konfrontacji. Dobry tester wszystko zrozumie i będzie w stanie zapobiec rodzącym się konfliktom.

Dyplomacja

Dlatego dobry tester, to także dyplomata. Przekazanie innym, że są w błędzie wymaga sporych umiejętności dyplomatycznych. Testerzy muszą umieć dyplomatycznie wytłumaczyć programistom, że oprogramowanie zawiera błędy. Jeśli jest to osiągnięte na drodze konfrontacji, tester może „wygrać bitwę, ale przegrać wojnę” w rozumieniu przyszłej współpracy z daną osobą. Ważny jest cel jakim jest dostarczenie klientowi oprogramowania wolnego od błędów. Dlatego to właśnie z błędami powinniśmy walczyć, a nie z innymi osobami w zespole projektowym.

Poczucie humoru

Praca w większości projektów staje się z czasem obciążona znacznym stresem, a zaangażowani w nią pracownicy narażeni są na liczne konflikty, które dodatkowo utrudniają pracę i oczywiście jeszcze bardziej zwiększają stres. Poza umiejętnościami komunikacyjnymi i dyplomacją, w tej sytuacji może pomóc zwykłe poczucie humoru. Celny żart lub zabawna uwaga nie jest w stanie zlikwidować problemów technicznych, ale jest w stanie rozładować niebezpieczne i utrudniające pracę napięcia.

Pewność siebie

W sytuacjach, jakich jest wiele podczas projektów związanych z wytwarzaniem oprogramowania, kiedy zarzuca się testerom iż są w błędzie – „to nie błąd, to cecha”, testerzy muszą okazać zdecydowanie i pewność siebie, by móc zmierzyć się z zarzutami. Jeśli

pozwołą by łatwo było ich zbić z tropu lub zmienić ich zdanie, to nie osiągną wiele, a ucierpieć może jakość produktu końcowego.

Umiejętności techniczne i znajomość dziedziny

Pewność siebie nie powstaje w próżni i musi być zbudowana na wiedzy i kompetencjach. Dobry tester to profesjonalista. Profesjonalizm ten oparty jest o dwa filary, pierwszy z nich to umiejętności techniczne, drugi (czasami nawet ważniejszych od tego pierwszego) to znajomość dziedziny testowanego oprogramowania.

Umiejętności techniczne poza tym, że ułatwiają realizację wielu zadań związanych z testowaniem (np. przygotowanie środowiska testowego, kontrola nad nim, swoboda przeprowadzania instalacji nowych wersji, praca z bazami danych, wykorzystanie dostępnych narzędzi), pozwalają także łatwiej znaleźć wspólny język z projektantami i programistami. Programiści zwykle mają mało zrozumienia dla osób, które oceniają jako technicznych analfabetów. Tester, który będzie domagać się czegoś co źle nazwał lub nie do końca wiedział jak nazwać, może szybko utracić wiarygodność. Również obecny rozwój narzędzi wspomagających testowanie wpływa znacznie na zwiększenie wymagań tego typu. Tester musi mieć doświadczenie w pracy z narzędziami wykorzystywanymi w trakcie projektu. Może się zdażyć, że kilka lat praktyki programistycznej staje się ważnym atutem. Z mojego doświadczenia wynika, że „nawróceni” programiści są doskonałymi kandydatami na testerów. Jeśli programista, który z jakichś powodów porzuca programowanie, trafia do zespołu testowego, to wnosi ze sobą doświadczenie i wiedzę techniczną. Jeśli osoba taka dodatkowo odkryje w sobie duszę testera i okaże się, że posiada przynajmniej kilka z wymienionych tu cech, może stać się naprawdę bardzo dobrym testerem.

Drugim filarem jest znajomość dziedziny. Program, który wytwarzamy ma pracować na rzecz zamawiającej go instytucji gdzie poza firma informatyczną. Osoby dobrze znające biznes tej instytucji, jej procesy, metody działania, wreszcie specyficzne wymagania i potrzeby, a do tego chcą testować, to idealni kandydaci do zespołów testowych. Tester rozumiejący dobrze użytkownika i potrafiący korzystać z programu w sposób, w jaki robi to użytkownik, najlepiej odpowiada idei wczesnej weryfikacji i walidacji oprogramowania. Gdy zadaniem naszym jest wytworzenie i przetestowanie na przykład systemu kredytowego, to wartość testera pracownika firmy informatycznej, który bez zaangażowania i bez wycucia jedynie „klika” w aplikację, jest zdecydowanie mniejsza niż osoby z doświadczeniem bankowym. Dlatego też adaptacyjne metody wytwarzania kładą tak silny nacisk na zaangażowanie użytkowników w proces twórczy i angażują ich już we wczesne fazy testów. Tester ze znajomością dziedziny, stanowi namiastkę użytkownika końcowego, do którego trafi w końcu wytwarzane rozwiązanie. Oczywiście poznawanie dziedziny problemowej może stać się elementem dodatkowych analiz, jak ma to miejsce w przypadku testowania kontekstowego, w którym właśnie poznanie i zrozumienie kontekstu rozwiązania stanowi podstawowe zadanie.

Dobra pamięć

W pracy testera bardzo ważna jest dobra pamięć. Ile to razy mieliśmy wrażenie, że już gdzieś spotkaliśmy podobny problem? Gdzie i kiedy to było? Dobry tester będzie potrafił wydobyć te fakty z pamięci. Umiejętność ta jest bardzo potrzebna podczas testów, ponieważ bardzo wiele problemów okazuje się być nieco zmodyfikowaną wersją wcześniej napotkanych błędów.

Sceptycyzm

Programiści zwykle próbują wyjaśniać błędy na różne sposoby. Należy tego oczekiwać. Tester musi słuchać każdego, ale jednocześnie pozostawać sceptycznym zanim samemu nie sprawdzi jak jest naprawdę.

Właściwa motywacja

Testowanie jest trudnym zajęciem, potrafi być monotonne i nudne. Testerzy pracują zwykle w trudnych warunkach i narażeni są na stres i liczne naciski. W tych warunkach na tego rodzaju stanowisku łatwo jest się zniechęcić. Tylko taka osoba, która potrafi sama siebie motywować, będzie zdolna do tego by codziennie rzucać się w wir pracy i wytrwać w niej przez długie lata. Trzeba to po prostu lubić.

Zrób to lepiej

Ponieważ natura testowania jest związana z powtarzaniem szeregu czynności, dobry tester szuka sposobów na ułatwienie sobie pracy. Dlatego jest on otwarty na nowości techniczne i korzysta z dostępnym narzędzi.

Podczas opracowywania tego materiału autor korzystał z książki „Testowanie oprogramowania” Rona Pattona oraz z „Testing Client/Server Systems” Kellye’go C. Bourne i materiałów dostępnych w Internecie.