Rafał Pawlak

TESTOWANIE OPROGRAMOWANIA

Podręcznik dla początkujących

Testuj programy i śpij spokojnie!

- Ogólna teoria testowania,
  czyli po co nam testy i jak sobie z nimi radzić
- Projekt a proces testowania,
  czyli kiedy zacząć testować i jak to robić z głową
- Automatyzacja i dokumentacja,
  czyli jak ułatwić sobie pracę podczas testowania

Helion

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Ewelina Burska
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/szteop
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-9308-5

Copyright © Helion 2014

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

Przedmowa ... **5**

Wstęp ... **7**

Rozdział 1. Ogólna teoria testowania ... **11**

1.1. Techniki testowania ... 13
1.2. Miara jakości oprogramowania ... 17
1.3. Środowisko testowe i produkcyjne ... 23
1.4. Replikacja błędów ... 28
1.5. U mnie błąd nie występuje ... 30
1.6. Symulatory aplikacji oraz generatory danych ... 31
1.7. Dokumentowanie testów ... 34
1.8. Kontrola wersji oprogramowania ... 35
1.9. Obsługa zgłoszeń ... 39
1.10. Testowanie obsługi wyjątków w kodzie ... 43
1.11. Narzędzia wsparcia pracy testera ... 51
1.12. Presja czasu ... 52
1.13. Profil profesjonalnego testera ... 54
1.14. Testowanie w oknie czasu ... 58
1.15. Jak wygląda realizacja projektu w praktyce? ... 60
1.16. Testowanie w cyklu życia oprogramowania ... 62

Rozdział 2. Poziomy wykonywania testów ... **65**

2.1. Testy modułowe ... 66
2.2. Testy integracyjne ... 67
2.3. Testy systemowe ... 71
2.4. Testy akceptacyjne ... 72

Rozdział 3. Typy testów ... **73**

3.1. Testy funkcjonalne ... 73
3.2. Testy niefunkcjonalne ... 74
3.2.1. Testy wydajności ... 74
3.2.2. Testy bezpieczeństwa aplikacji ... 91
3.2.3. Testy przenośności kodu — testy instalacji ... 117
3.2.4. Testy ergonomii systemu informatycznego ... 118
3.3. Testy regresywne ... 125

Kup ksi k Pole ksi k

Rozdział 4. Wprowadzenie do projektowania testów 129

4.1. Projektowanie testu w oparciu o technikę czarnej skrzynki 131
4.1.1. Wartości brzegowe 131
4.1.2. Przejścia pomiędzy stanami 134
4.1.3. Projektowanie testu w oparciu o przypadki użycia 135
4.2. Projektowanie testu w oparciu o technikę białej skrzynki 136
4.3. Projektowanie testu w oparciu o doświadczenie testera 140
4.4. Przypadki testowe w ujęciu praktycznym 140

Rozdział 5. Psychologiczne aspekty procesu testowania 149

Rozdział 6. Syndrom zniechęcenia testami 153

Rozdział 7. Testowanie usług sieciowych 165

7.1. Narzędzie SoapUI — klient usługi sieciowej 165
7.2. Symulator serwera usług sieciowych — SoapUI Mock Services 171
7.3. Monitor TCP — Apache TCPMon 177

Rozdział 8. Wprowadzenie do automatyzacji testów 183

Dodatek A Generowanie sumy kontrolnej 187

Dodatek B Membrane SOAP Monitor 189

Dodatek C Wireshark — analizator ruchu sieciowego 195

Dodatek D Generowanie danych testowych 197

O autorze 207

Skorowidz 209

Kup ksi k Pole ksi k

Rozdział 2.

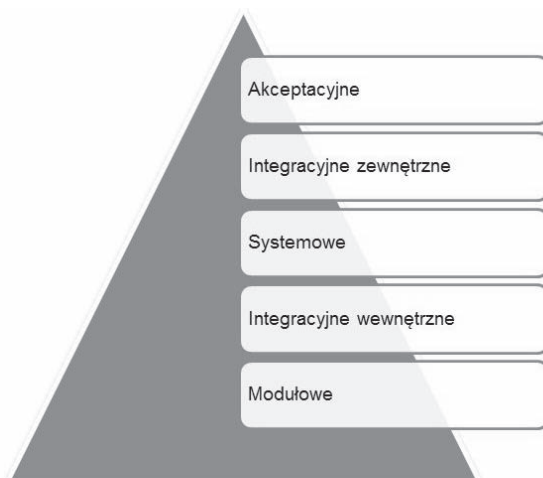
Poziomy wykonywania testów

Proces wytwarzania oprogramowania podzielony jest na fazy, w których wykonuje się specyficzne dla każdego z etapów testy.

Testy dzieli się na poziomy:

- testy modułowe (jednostkowe),
- testy integracyjne wewnętrzne,
- testy systemowe,
- testy integracyjne zewnętrzne,
- testy akceptacyjne (odbiorcze).

Rysunek 2.1 przedstawia piramidę poziomu testów. Testy wykonywane są zgodnie z oddolną interpretacją infografiki, tj. od testów modułowych aż po testy akceptacyjne.



Rysunek 2.1. *Piramida poziomu testów*

2.1. Testy modułowe

Testy modułowe wykonuje się na etapie wytwarzania kodu. Uczestnikami tego rodzaju testów są najczęściej programiści, którzy w fazie implementacji poddają weryfikacji własny kod. Wspomniane testy muszą odnosić się do niewielkich i ściśle wyizolowanych fragmentów kodu. Polegają one na wykonywaniu wybranego fragmentu instrukcji w celu zweryfikowania, czy realizuje ona swoją funkcję zgodnie z założeniami. Programista przygotował metodę, która konwertuje numer NRB (*Numer Rachunku Bankowego*) na IBAN (ang. *International Bank Account Number* — pol. *Międzynarodowy Numer Rachunku Bankowego*). Wspomniana metoda będzie wielokrotnie używana w różnych miejscach systemu. Test modułowy polegać będzie na wywoływaniu owej metody z podaniem jako parametru wejściowego numeru NRB i weryfikacji otrzymanego wyniku. Wywołanie metody może odbywać się z poziomu środowiska deweloperskiego bez zastosowania GUI. Na tym koniec. Testy modułowe powinny odnosić się do małych podmiotów, a ich wynik powinien być zależny od innych elementów, które potencjalnie mają znaleźć się w gotowej aplikacji. Testy modułowe nie powinny wnikać w szczegóły procesu biznesowego. Analizie i ocenie podlega jedynie mały i wyizolowany fragment kodu. W przypadku kiedy nabierzemy zaufania do testowanych fragmentów kodu, możemy rozpocząć składanie ich do postaci gotowego produktu lub większego komponentu.

Testy modułowe wykonuje się zwykle w środowisku deweloperskim z dostępem do kodu źródłowego. Wymaga to od testera — o ile nie jest programistą — umiejętności czytania i wykonywania kodu oraz pisania własnych skryptów (fragmentów kodu). Bywa, iż przetestowanie pewnego modułu wymaga zaangażowania elementów pobocznych, np. symulatora usługi sieciowej.

Dlaczego należy wykonywać testy modułowe?

Błędy wykryte we wczesnej fazie produkcji oprogramowania kosztują znacznie mniej aniżeli poprawa oraz usuwanie ich skutków w kolejnych fazach wytwarzania lub użytkowania produkcyjnego aplikacji. Wykryte problemy usuwane są natychmiast, przez co minimalizuje się ryzyko propagacji negatywnego wpływu wadliwego kodu na pozostałe moduły np. poprzez odziedziczenie wady. Błędy wykryte w tej fazie zwykle nie znajdują odzwierciedlenia w postaci formalnego zgłoszenia, co przyspiesza udoskonalanie kodu i nie niesie ze sobą ryzyka krytycznych uwag osób trzecich. Jest to bardzo bezpieczna i efektywna forma testów własnego kodu. Kompilator nie weryfikuje intencji programisty. Zatem pomimo iż kod został skompilowany, nie można go uznać za poprawny bez przeprowadzenia testu jednostkowego. Odłożenie procesu testowania do momentu złożenia w całość wszystkich elementów niesie ze sobą znaczne ryzyko, iż nie uda się uruchomić aplikacji lub pewnych funkcjonalności za pierwszym razem. Błędy, które zostaną ujawnione, mogą mieć przyczynę głęboko ukrytą w kodzie. Przeszukiwanie „rozdmuchanych" źródeł w celu wyizolowania przyczyny problemu będzie czasochłonne i zapewne irytujące. Testy modułowe pozwalają na wyeliminowanie typowych problemów w podstawowej ścieżce obsługi systemu. Im mniej problemów zostanie przeniesionych z fazy implementacji do fazy formalnych testów, tym szybciej gotowy produkt zostanie przekazany do odbiorcy, a jego jakość wzrośnie.

Kup ksi k Pole ksi k

2.2. Testy integracyjne

U podstaw testów integracyjnych leży opieranie jednego procesu biznesowego na wielu różnych systemach, podsystemach i aplikacjach. Heterogeniczność ostatecznie oferowanego użytkownikowi końcowemu rozwiązania wymusza przeprowadzenie pełnego testu biznesu w oparciu o scalone środowisko. Aby było to możliwe, uprzednio należy zweryfikować interakcję pomiędzy poszczególnymi modułami. Zwieńczenie sukcesem owych prac stanowi podstawę do traktowania wszystkich modułów jako jeden system. Niezachwiane przekonanie o spójności systemu otwiera drogę do pełnych testów funkcjonalnych w całościowym ujęciu obsługi biznesu.

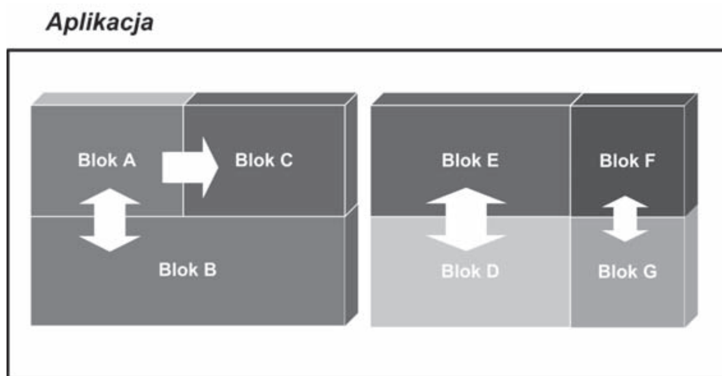
Testy integracyjne to szczególny rodzaj działań podejmowanych w celu zbadania kooperacji oraz wzajemnego oddziaływania dwóch lub więcej modułów systemu. Przez desygnat „moduł" należy rozumieć zupełnie autonomiczny produkt lub fragment kodu, który ostatecznie jest ściśle spojony z główną architekturą systemu.

Testy integracyjne mają wykazać:

- czy moduły poprawnie współpracują, tj. czy nie wystąpiły przeszkody natury technologicznej oraz czy wzajemnie świadczone usługi spełniają oczekiwania (logika);
- jak zachowują się poszczególne elementy w sytuacji awarii, błędów lub niestabilnej pracy w przypadku dysfunkcji jednego z nich (wzajemne oddziaływanie);
- czy kojarzone wzajemnie elementy realizują założony proces biznesowy (logika biznesu);
- czy infrastruktura techniczna zapewnia optymalne warunki pracy dla skomplikowanego systemu (wielomodułowego);
- czy nie ma luk w logice biznesu, tj. czy nie ujawniły się problemy i/lub potrzeby, które nie zostały przewidziane na etapie projektowania rozwiązania.

Prace integracyjne rozpoczynają się już w momencie łączenia kodu dwóch lub więcej modułów tej samej aplikacji (integracja wewnętrzna). Owe komponenty mogą być przygotowane przez zupełnie niezależnych programistów, a finalnie podlegają integracji. Nadmienione elementy mogą w mniejszym lub większym stopniu ulegać integracji. Rolą testera jest zweryfikowanie relacji pomiędzy nimi. Zdarza się, iż programista tkwi w przekonaniu, że integrowane elementy mają śladowy wpływ na pracę pozostałych fragmentów kodu. Niemniej jednak w ujęciu całościowym błąd w jednym z nich poważnie rzutuje na pracę całego systemu. Iluzoryczne przekonanie o nikłej zależności lub wręcz przezroczystości kodu integrowanego z dotychczasowymi funkcjonalnościami aplikacji bywa zgubne w skutkach. Takowe prace należy podejmować jak najbliżej kodu, tj. na etapie prac programistycznych i testów wewnętrznych. Rysunek 2.2 przedstawia szkolny schemat blokowy modułów jednej aplikacji. Programista X przygotował blok B. Chcąc wykonać jego testy, musi zintegrować go z blokiem A lub zasymulować jego działanie. Programista Y wykonał blok A, który z założenia ma pracować z blokami B i C. Modułowe testy integracyjne polegają na łączeniu ze sobą fragmentów kodu, które wchodzą w bezpośrednią interakcję. Programista lub tester

Kup ksi k Pole ksi k



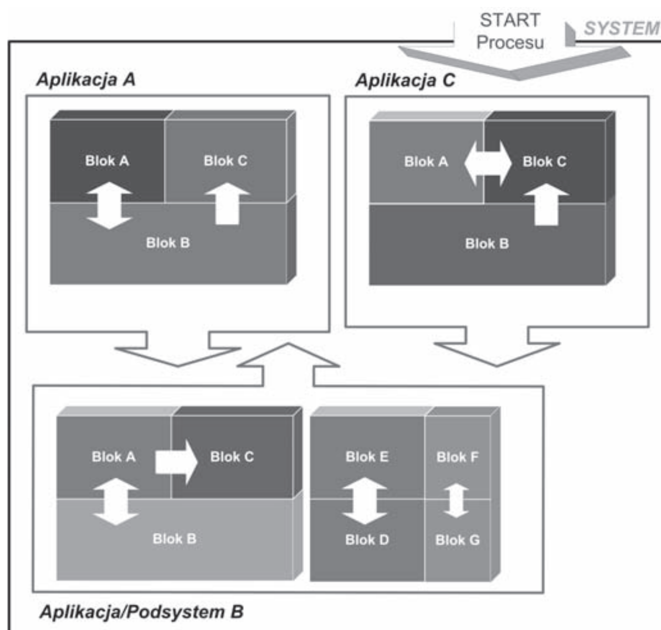
Rysunek 2.2. *Schemat wzajemnego oddziaływania bloków jednej aplikacji*

uruchomi blok A i sprawdzi, czy to, co „wysyła" do elementu C, jest poprawne. Integrując bloki A, B i C, weryfikujemy ich wzajemną kooperację, mimo iż nie realizują one pełnej funkcjonalności z punktu widzenia użytkownika końcowego.

Kolejnym momentem w procesie produkcji oprogramowania, w którym wykonywane są testy integracyjne, jest zestawianie odrębnych modułów na etapie weryfikacji funkcjonalnej. Jest to arcyciekawa sytuacja z uwagi na to, iż kod odpowiadający za odrębne fazy obsługi biznesu może być zrealizowany przez zupełnie niezależne podmioty. Zwykle jako podstawa do przygotowania „wtyczki" dla obcego systemu musi wystarczyć dokumentacja dostarczona przez potencjalnego integratora, tj. klienta, który będzie „spinał wszystko w całość". Nadmieniony materiał może zawierać definicję plików wymiany danych (np. TXT, XML, CSV itp.), opis usługi sieciowej, np. WSDL, schemat udostępnionych obiektów na bazie danych w postaci perspektyw, przykłady wywołania upublicznionych procedur etc. Niezależnie od wybranego rozwiązania dostęp do niego bywa znacznie utrudniony lub wręcz niemożliwy. Oczywiście owa sytuacja już na etapie kodowania stanowi poważne perturbacje, lecz prawdziwe trudności ujawniają się dopiero w momencie testów jakościowych.

Rozważmy hipotetyczną sytuację w oparciu o rysunek 2.3. Pełny proces biznesowy wymaga zaangażowania trzech aplikacji. Testy integracyjne będą odbywały się na styku produktu C i B oraz B i A. Biznes (funkcjonalność) natomiast będzie weryfikowany przy użyciu wszystkich bloków A, B oraz C. Aplikacja C stanowi graficzny interfejs użytkownika, posiadający nikłą logikę biznesową, a zarazem bardzo rozbudowane możliwości prezentacji i obsługi danych. Główny mechanizm realizacji zadanego biznesu spoczywa na programie B. Tym niemniej mogą wystąpić sytuacje, w których podsystem B będzie musiał posiłkować się wsparciem programu A. Reasumując, w realizację logiki mogą być zaangażowane bloki B oraz A. Natomiast aplikacja C jest inicjatorem przetwarzania, a zarazem konsumentem wyniku, tj. prezentacji rezultatu. Systemy o złożonej architekturze najczęściej poddawane są realnej próbie dopiero w środowisku testowym zamawiającego. Odbiorca oprogramowania ma przewagę nad wykonawcami poszczególnych komponentów w postaci nieskrępowanego dostępu do wszystkich elementów. Drugą kwestią jest posługiwanie się danymi bardzo zbliżonymi do rzeczywistych np. poprzez włączenie w proces testowania kopii bazy z produkcji. Powyższe zabiegi urealniają testy funkcjonalne (pełnego biznesu) poprzez posługiwanie się niemalże faktycznymi danymi zgromadzonymi w dużym wolumenie. Obsługa pełnego

Kup ksi k Pole ksi k



Rysunek 2.3.
Schemat interakcji trzech niezależnych aplikacji

procesu biznesowego w fazie prac integracyjnych lub ostatecznych testów odbiorczych może ujawnić luki w opracowanej koncepcji. Zdarza się, iż w wyniku testów zidentyfikowane są problemy, które nie były przewidziane w projekcie. Owe problemy mogą przełożyć się na zmianę rozwiązania. Dokładniej rzecz ujmując: na jego dopracowanie, na przykład poprzez dopisanie drobnych funkcjonalności, modyfikację założeń wstępnych etc. Testy integracyjne stanowią pierwszy etap weryfikacji, czy przyjęte rozwiązanie oraz jego implementacja zaspakajają potrzeby biznesowe (w całościowym ujęciu procesu).

Pierwszym, a zarazem najmniej kłopotliwym scenariuszem jest integrowanie modułów (całych programów), które wykonane są przez jednego producenta. W teorii wskazane testy powinny być wykonane bez większych problemów, a ewentualne przeszkody w postaci błędów w implementacji lub niejasności w dokumentacji rozwiązane wewnątrz organizacji. Praktyka dowidzi, iż niestety również ten scenariusz naznaczony jest pewnymi zakłóceniami w trakcie realizacji projektu testowego. Wynika to głównie z organizacji pracy oraz nie do końca jasnej i zdrowej rywalizacji pomiędzy zespołami wewnątrz korporacji. Wskazałem korporację, gdyż pojęcie klienta zewnętrznego oraz wewnętrznego jest szeroko stosowane w dużych przedsiębiorstwach. Pojęcie samo w sobie nie kryje niczego złego, aczkolwiek zauważalne są problemy w relacjach pomiędzy zespołami wynikające z zawiłej, a bywa, że i naciąganej interpretacji tegoż terminu. Generalnie idealizując, można przyjąć, iż funkcjonujące procedury formalne w pełni zabezpieczają potrzebę kooperacji różnych zespołów w celu osiągnięcia ostatecznego rezultatu.

Nieco mniej przyjemną sytuacją jest nieposiadanie jednego lub więcej komponentów „na własność", kiedy są one udostępnione grzecznościowo przez klienta. Jest to stosunkowo dobre rozwiązanie, aczkolwiek uzależnione od dobrej woli współpracy. Pewną

trudnością dla procesu integracji mogą być ewentualne niedociągnięcia w kodzie po drugiej stronie, ale właśnie po to się wykonuje testy integracji, aby owe problemy wyeliminować. Generalnie „my" czy partner po przeciwnej stronie musimy czekać na ewentualne modyfikacje kodu w celu kontynuowania procesu integracji. Niemniej jednak jest to dobra droga. Wymaga ona zaangażowania zasobów klienta jako pośrednika, lecz uzyskujemy wartość dodaną w postaci wczesnego rozminowania pola. Bujna wyobraźnia podpowiada, co może się stać, jeżeli podejmiemy próbę zintegrowania dwóch programów bez wcześniejszego „docierania" ich razem.

Najtrudniejszą sytuacją jest konieczność wyprodukowania modułu klienckiego do obcego systemu bez dostępu do tegoż zasobu. Zaiste gros trudności ujawni się w momencie uruchamiania kodu i testów funkcjonalnych. Dużo zależy od roli, jaką dla testowanej funkcjonalności pełni zewnętrzny system. Bywa, iż brak dostępu do drugiego programu w minimalnym stopniu ograniczy testy. Bywa również, iż zupełnie je uniemożliwi. Załóżmy, iż testowana funkcjonalność realizuje jakiś biznes zdefiniowany w 10 logicznych krokach, gdzie kroki 2., 5. oraz 8. wymagają połączenia z usługą sieciową w celu pobrania/przeliczenia jakichś danych. W takich okolicznościach przetestujemy tylko krok nr 1. Drugi już wymaga interakcji z web service (WS). W tym momencie wyklarowała się potrzeba zaślepienia wywoływanej metody WS przez naszą aplikację, tak aby mogła ona przejść do kroku nr 3 itd. Trzeba wyprodukować symulator systemu zewnętrznego. Temat zaślepiania żądań do zewnętrznych aplikacji będzie omówiony szerzej w dalszej części książki.

Do typowych problemów (błędów) wykrywanych w procesie integracji należy zaliczyć:

- Niespójność wysyłanego schematu XML z oczekiwaniami klienta lub serwera np. w wyniku przygotowania klienta WS w oparciu o nieaktualny opis usługi (WSDL) lub błąd w kodzie.
- Treść danych przekazywana w pliku wymiany danych zawiera znaki specjalne, na które negatywnie reaguje druga aplikacja. Przykładowa nazwa jakiejś organizacji, fundacji może zawierać cudzysłów, który eksportowany jest do treści pola. Dokumentacja opisująca pliki wymiany może nie opisywać szczegółów lub wyjątków.
- Zderzenie dwóch technologii lub nawet różnych wersji tych samych serwerów może powodować problemy natury integracyjnej.
- Bywają sytuacje, iż w jakichś okolicznościach jedna ze stron będzie przekraczała czas odpowiedzi na żądanie (ang. *time out*).
- Wyobrazić można sobie sytuację, że obrót danymi odbywa się za pomocą pliku wymiany danych, w którym koniec linii ustalony jest na znak LF, a w wyniku pośredniego działania jakiegoś programiku, który kopiuje plik z miejsca A do B, zajdzie niejawna konwersja znaku końca linii na CRLF. Próba odczytania takiego zbioru zakończy się niepowodzeniem. Czy taki scenariusz można przewidzieć w testach funkcjonalnych? W praktyce jedna aplikacja wygeneruje poprawnie LF, a druga poprawnie odrzuci znak CRLF. Gdzie jest błąd?! Może w jakimś „starym" programiku, którego nie braliśmy pod uwagę na etapie testów?

Kup ksi k Pole ksi k

- Testy integracyjne stanowią dobrą okazję do masowego wczytywania plików, wymiany danych przez funkcje WS itp. Bywa, iż błędy, np. przepełnienie bufora, przekroczenie zakresu zmiennej itp., ujawniają się dopiero przy „ogromnej” ilości danych.
- Integracja systemu ujawnia wszelkie różnice w rozumieniu funkcji, jakie ma pełnić każdy z elementów, w tym po której stronie mają być obsługiwane określone wyjątki.
- Luki w opracowanej koncepcji rozwiązania, na podstawie której była wykonana implementacja.
- Wiele innych problemów…

2.3. Testy systemowe

Przedmiotem testów systemowych jest cała aplikacja lub jej samodzielny fragment, który znajduje odwzorowanie w projekcie, tj. wchodzi w zakres projektu. Najodpowiedniejszą formą testów funkcjonalnych jest zastosowanie techniki czarnoskrzynkowej. Niemniej jednak technika białej skrzynki z powodzeniem może być wykorzystywana jako uzupełnienie głównego wątku testów. Kluczem do sukcesu jest prowadzenie testów w środowisku testowym jak najbardziej zbliżonym do produkcyjnych warunków funkcjonowania aplikacji. Weryfikacja aplikacji w warunkach możliwie wiernie odwzorowujących docelowe środowisko pracy zmniejsza ryzyko przeoczenia błędów i problemów, które mogą wynikać z różnic w specyfice obu środowisk. Więcej informacji na temat środowiska testów znajduje się w rozdziale poświęconym owej tematyce w niniejszej książce.

Równie istotne jest to, kto wykonuje testy systemowe. Najlepiej byłoby, aby czynił to niezależny zespół testerów, tzn. taki, który zachowuje dużą autonomiczność wobec zespołu programistycznego w aspekcie środowiska testów oraz zasobów ludzkich.

Testy systemowe (ang. *system testing*) winny ujmować wymagania zarówno niefunkcjonalne, jak i funkcjonalne. Jest to faza, w której ocenia się globalnie produkt bez nadmiernego nacisku na zgłębianie wewnętrznej architektury i instrukcji kodu aplikacji.

Testy systemowe odnoszą się do aplikacji w ujęciu całościowym. Oznacza to, iż zobowiązani jesteśmy do weryfikacji wszystkich funkcji wraz z analizą korelacji pomiędzy nimi. Załóżmy, iż otrzymaliśmy do testów aplikację lub nowy moduł, który administruje fakturami VAT. Testy systemowe wymagają zweryfikowania wszystkich ścieżek obsługi owego dokumentu, m.in. wystawienia FV, korekty, wydruku, filtrowania, generowania do pliku PDF, akceptacji etc.

Tego rodzaju testy mogą ujawnić potrzebę zastosowania zaślepek lub symulatorów zewnętrznych systemów, z którymi nasza aplikacja będzie wchodzić we współzależność lub występować jedynie jako klient.

Specyfika i zakres testów systemowych stawia ten rodzaj weryfikacji w roli bardzo dobrego kandydata do automatyzacji czynności (testów).

Kup ksi k Pole ksi k

Testy systemowe to:

- testy funkcjonalne,
- testy wydajnościowe,
- testy regresywne,
- testy ergonomii,
- testy instalacji,
- testy bezpieczeństwa.

2.4. Testy akceptacyjne

Testy odbiorcze wykonywane są bezpośrednio przez zamawiającego w oparciu o własne zasoby lub poprzez zlecenie prac niezależnemu zespołowi testerskiemu. Testy te mają potwierdzić zgodność weryfikowanego produktu z zapisami w umowie i obowiązującymi przepisami prawa. Celem testów akceptacyjnych jest weryfikacja i potwierdzenie, czy wszystkie zapisy w kontrakcie zostały zrealizowane w sposób zaspokajający oczekiwania. Pozytywne zamknięcie fazy testów odbiorczych stanowi podstawę do finansowego rozliczenia kontraktu. Testy akceptacyjne winny odnosić się do formalnie spisanych kryteriów odbioru oprogramowania. Wspomniane kryteria zwykle uzgadniane są na etapie negocjacji kontraktu. Nadmieniona powyżej sytuacja odnosi się do oprogramowania pisanego na zamówienie.

Nieco inaczej kreuje się sytuacja dla aplikacji „pudełkowych”. Oprogramowanie z półki może być poddane dwóm typom testów akceptacyjnych: *alfa* i *beta*. Testy alfa wykonywane są wewnątrz organizacji, która wyprodukowała oprogramowanie, aczkolwiek weryfikacji dokonuje niezależny zespół, tj. zespół, który nie brał udziału w procesie wytwarzania. Betatesty realizowane są poza organizacją wykonującą kod przez grupę użytkowników docelowych. Firmy produkujące oprogramowanie pudełkowe są żywo zainteresowane uzyskaniem informacji zwrotnej (potwierdzeniem) o wysokiej jakości własnego produktu przed oficjalnym wprowadzeniem go na rynek.

Niezależnie od typu oprogramowania (pudełkowe, na zamówienie) winno ono również być poddane testom akceptacyjnym w aspekcie obowiązujących przepisów prawa.

Kup ksi k Pole ksi k

Skorowidz

A

administracja środowiskiem testów, 27
adres URL, 109
adresowanie IP, 180
algorytm SHA, 188
algorytmy walidacyjne, 33
analizator ruchu sieciowego, 195
API, Application Programming Interface, 76
architektura wielowarstwowa, 36
arkusze kalkulacyjne, 52
atak
  Blind SQL-Injection, 102, 103
  SQL-Injection, 96, 99, 104
  typu XSS, 93
autodiagnoza, 161
automatyzacja testów, 183

B

baza danych, 76
bezpieczeństwo, 92
Blind SQL-Injection, 102, 103
blokowanie spamu, 92
błąd, 11
  etap symulacji, 29
  potencjalne przyczyny, 30
  replikacja, 29
błędna obsługa wyjątku, 44
błędy
  blokujące, 12
  krytyczne, 12
  projektowe, 145
  sterownika JDBC, 77
  wewnętrzne, 42
  w koncepcji, 145
  w operacji mnożenia, 141
  w procesie integracji, 70
  zewnętrzne, 40

C

certyfikat ISTQB, 58
cykl życia oprogramowania, 62
czas wykonania zapytania, 83, 85

D

debugowanie, 12
dokumentacja, 13
dokumentowanie testów, 34
doświadczenie testera, 140
drzewo
  MockService, 174
  wyników, 83

E

edytory tekstu, 52
emulatory, 52
ergonomia
  komunikatów walidacyjnych, 121
  przycisków akcji, 122
  systemów informatycznych, 122

F

filtrowanie danych, 97
formularz doładowania telefonu, 94, 107
funkcja zwracająca wersję, 36

G

generator
  danych, 32
  danych testowych, 197, 200
    CSV, 201, 203
    SQL, 201
    XML, 201

Kup ksi k   Pole ksi k

generator
certyfikatów SSL, 51
MockService, 173
sumy kontrolnej, 51, 187
GIT, 35
graf przepływu sterowania, 138, 139
GUI, 15
GUI dgMaster, 205

I

ICT, Information and Communication Technologies, 92
IEEE, 13
implementacja monitora TCP, 178
informacja o znaku końca linii, 187
informacje
o nazwie serwera, 113
o systemie, 111
o wersji serwera, 113
zwrotne, 18
instrukcja
IF, 15
instrukcja INSERT INTO, 80
intensywność testów, 53
interfejs użytkownika, 15
interpretacja projektu, 149

J

jakość oprogramowania, 17, 20, 63, 163
JDBC, 80
język
WADL, 166
WSDL, 166

K

klauzula UNION, 99, 100
klient
MockService, 176
usługi sieciowej, 165
kod JavaScript, 95
kodowanie znaków, 52
konfiguracja
nasłuchu, 178
nowej instrukcji, 85
połączenia JDBC, 79
programu Fiddler, 114
TCPMon, 180
testu dla aplikacji, 87
zapytania SQL, 80
żądania, 88
kontrola
jakości, 46, 52
wersji oprogramowania, 35
krok pośredni, 109
kryterium ilościowe, 17

L

logowanie błędów, 45, 50

M

Manifest Zwinnego Wytwarzania Oprogramowania, 64
menedżer plików, 51
metoda
białej skrzynki, 14
czarnej skrzynki, 13
metodyki zwinne, 64
metryka pakietu Oracle, 37
miara jakości oprogramowania, 17
migracja danych, 48
model
kaskadowy, 63
V, 64
modyfikacja
kodu, 115
kodu formularza, 117
treści strony, 112
monitor TCP, 177
monitorowanie jakości, 21

N

nagłówek HTTP, 113
narzędzia automatyzacji testów, 52, 186
narzędzie
Apache JMeter, 51, 52, 199
Apache TCPMon, 177, 178, 180, 181
dgMaster, 203
Eclipse, 51
Fiddler, 105–108, 111–117
Fiddler2, 51
Firebug, 51, 106
JMeter, 76–90
KeyStore Explorer, 51
Membrane Monitor, 51
Membrane SOAP Monitor, 189–93
NetBeans IDE, 51
Notepad++, 51
Oracle OpenScript, 52
PL/SQL Developer, 51
PuTTY, 51
Selenium, 52

SoapUI, 51, 165–176, 180
Spawner Data Generator, 203
TCPMon, 51
Total Commander, 51
VNC, 51
WebScarab, 105
WinSCP, 51
Wireshark, 51, 195, 196

O

obsługa
błędów wewnętrznych, 42
błędów zewnętrznych, 40
wyjątków, 43
zgłoszeń, 39
ogólna teoria testowania, 11
opis kodu źródłowego, 38
oprogramowanie
Bugzilla, 39
JIRA, 39
Mantis Bug Tracker, 39
SVN, 35

P

plik wynikowy CSV, 203
podgląd odpowiedzi, 114
podmienianie wartości, 105
polecenie DESC, 96
porty, 180
poziomy wykonywania testów, 65
presja czasu, 151
procentowy rozkład zgłoszeń, 21
proces
integracji, 70
obsługi błędów, 40, 42
testowania, 149
projekt REST, 170
projektowanie testu, 129
doświadczenie testera, 140
technika białej skrzynki, 136
technika czarnej skrzynki, 131
protokół REST, 168
przebieg
realizacji projektu, 9, 53
rejestracji błędów, 22, 23
przechwycenie sesji, 108
przejścia pomiędzy stanami, 134
przekazywanie zmian w kodzie, 39
przyczyny
błędów, 30
zniechęcenia, 156
przypadek testowy, 12
przypadki użycia, 135

R

realizacja projektu, 60
reguły odpowiedzi, 175
rejestracja błędów, 22, 23
rekonesans, 111
replikacja błędów, 28
REST, Representational State Transfer, 165
retest, 11
równomierne obciążenie pracą, 54

S

scenariusz testów, 13
serwer poczty, 52
SHA, Secure Hash Algorithm, 188
skrypt JS, 95
skutki zniechęcenia, 159
SOAP, Simple Object Access Protocol, 165
specyfikacja, 13
spójność nazewnictwa, 121
SQL-Injection, 96, 99, 104
SSL, Secure Socket Layer, 166
standardy, 13
Subversion, SVN, 35
suma kontrolna, checksum, 187
sygnatura wersji, 103
symulacja błędu, 29
symulator
aplikacji, 31
serwera usług sieciowych, 171
syndrom zniechęcenia, 153
system kontroli wersji, 35
GIT, 35
Subversion, 35

Ś

środowisko
deweloperskie, 30
produkcyjne, 25
testów, 23, 27, 28

T

tabela, 81
technika
białej skrzynki, 136
czarnej skrzynki
przejścia pomiędzy stanami, 134
przypadki użycia, 135
wartości brzegowe, 131
techniki testowania, 13

Kup ksi k Pole ksi k

teoria testowania, 11
tester oprogramowania, 54
  aspekty psychologiczne, 149
  syndrom zniechęcenia, 153
testowanie
  aplikacji internetowej, 86
  bazy danych, 76
  bezpieczeństwa aplikacji, 91
  ergonomii systemu informatycznego, 118
  instalacji, 117
  instrukcji, 136
  migracji danych, 47
  obsługi wyjątków, 43
  przejść pomiędzy stanami, 134
  przenośności kodu, 117
  usług sieciowych, 89, 165
testy
  akceptacyjne, 72
  decyzyjne, 138
  funkcjonalne, 73
  ilościowe, 197
  integracyjne, 67
  jakościowe, 8
  modułowe, 66
  niefunkcjonalne, 74
  obciążeniowe, 75
  przeciążeniowe, 75
  regresywne, 125
  systemowe, 71
    bezpieczeństwa, 72
    ergonomii, 72
    funkcjonalne, 71
    instalacji, 72
    regresywne, 72
    wydajnościowe, 71
  w cyklu życia oprogramowania, 62
  w oknie czasu, 58
  wewnętrzne, 8
  wydajności, 74, 76
trywialność wykrycia błędu, 12
tworzenie symulatora WS, 172
typy testów, 73

U

usługi sieciowe, 165

W

WADL, Web Application Description Language, 166
walidacja
  dopuszczalnych znaków, 144
  pól dla kwot, 147
walidatory, 52
wartości brzegowe, 131
wersje oprogramowania, 35
wrażliwe informacje, 109
WSDL, Web Services Description Language, 166
wstrzyknięcie kodu JS, 95
wyjątki, 43
wykres testu aplikacji, 89
wyłączenie części kodu, 104
wymaganie, 13
wytwarzanie oprogramowania, 64
wywołanie monitora TCP, 179
wywoływanie
  funkcji MockService, 176
  procedury, 84

X

XSS, Cross Site Scripting, 93

Z

zasada kumulowania się błędów, 9
zastosowanie
  opcji disabled, 117
  ORDER BY, 102
  SELECT UNION, 99, 101
złamanie zasad ergonomii, 122
zniechęcenie, 153
  przyczyny, 156
  skutki, 159

Ź

źródła pliku WSDL, 166
źródło zmodyfikowanej strony, 116

Ż

żądanie REST, 171

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

TESTOWANIE OPROGRAMOWANIA

Podręcznik dla początkujących

Testowanie oprogramowania jest niesłychanie ważnym czynnikiem, wręcz decydującym o sukcesie lub porażce systemu, aplikacji czy sieci. Bezlitośni użytkownicy wykryją najdrobniejszy błąd, skutkujący choćby niewielkim spowolnieniem działania programu... i nie omieszkają wyrazić swojej opinii w internecie. Jeśli chcesz uniknąć takich niespodzianek, a ponadto zależy Ci na jak najszybszym ukończeniu realizowanego projektu i wypuszczeniu perfekcyjnego produktu, musisz natychmiast zacząć go testować!

Ta książka pomoże Ci zorientować się w metodach i technikach testowania. Jej autor, praktyk z wieloletnim doświadczeniem, zawarł w niej informacje o narzędziach i procesach, opisał również własne doświadczenia związane z konkretnymi projektami. Dowiesz się między innymi, jak radzić sobie na kolejnych etapach weryfikacji jakości oprogramowania — wybrać odpowiedni typ testu i przejść przez proces jego projektowania — a także jak uporać się ze znużeniem nieustannym testowaniem. Odkryjesz też, do czego służy automatyzacja i jak przejrzyście dokumentować całe przedsięwzięcie. Usuń błędy, zanim zaczną sprawiać Ci kłopoty!

- **Poziomy wykonywania testów**
- **Typy testów**
- **Wprowadzenie do projektowania testów**
- **Psychologiczne aspekty procesu testowania**
- **Syndrom zniechęcenia testami**
- **Testowanie usług sieciowych (WebServices)**
- **Wprowadzenie do automatyzacji testów**
- **Generowanie sumy kontrolnej i danych testowych**
- **Membrane HTTP/SOAP Monitor**
- **SoapUI**

Postaw na jakość! To się opłaca!

helion.pl
księgarnia internetowa

Nr katalogowy: 20646

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:

0 801 339900

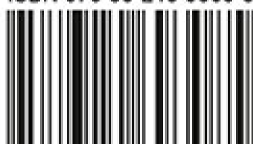
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-9308-5



Cena: 44,90 zł

Informatyka w najlepszym wydaniu