Prezentacja z SO: Testy

Andrzej Olejniczak i Maciej Fijałkowski

12 stycznia 2005

1 Wstęp

1.1 Definicja testowania

Testowanie to wykrywanie błędów w oprogramowaniu, analiza powstającego systemu pozwalająca na ocenę na ile dany system jest obciążony ryzykiem wystąpienia błędów.

Testowanie ma na celu wykrycie jak największej liczby wcześniej nie wykrytych błędów.

Testowanie to nie ulepszanie interfejsu użytkownika, ujawnianie wymagań, ani analiza systemu.

Testowanie polega na wykonywaniu fragmentów kodu oprogramowania przy różnych kombinacjach danych wejściowych.

Testowanie nigdy nie daje pewności wykrycia wszystkich błędów w oprogramowaniu, ale pozwala minimalizować szansę pojawienia się błędu już w czasie wykorzystywania oprogramowania przez użytkownika.

1.2 Czym jest błąd?

Błąd to dowolne niepożądane działanie systemu (jakiekolwiek działanie niezgodne z oczekiwaniami użytkownika), a zatem:

- Nieprawidłowy wynik wykonania programu
- Wynik osiągany w niedogodnym z punktu widzenia użytkownika czasie

1.3 Po co testy?

1.3.1 Testowanie przyda się przy wykrywaniu

- Skutków ubocznych, nieprzewidzeń, niepożądanych interakcji właściwości pisanego oprogramowania
- Nieodpowiedniej wydajności, zakleszczeń (niewłaściwy cel, algorytm, niewydajne programowanie, niewykonalne zadanie)
- Niepoprawnych wyników
- Awaryjnych zatrzymań programy (zaniechanie zadania, załamanie aplikacji lub systemu np. błękitny ekran Windows)

Po co się przeprowadza testy wydajności? Na ogół musimy odpowiedzieć na pytania typu:

1. Który ze sterowników urządzenia xxx jest najwydajniejszy
2. Jaki sprzęt będzie najlepszy do robienia yyy
3. Co osiągniemy zmieniając zzz na aaa

1.4 Zakresy testów

Testy w zależności od tego na jakim etapie projektu są wykonywane możemy podzielić na trzy poziomy:

- Testy jednostek - test niewielkiego fragmentu kodu (klasy lub kilku klas) w celu sprawdzenia zgodności jej implementacji ze specyfikacją
- Testy zintegrowane - testy podsystemu lub systemu, współpracujących grup jednostek w świetle spełnienia jakiegoś wymagania
- Testy systemu - testy całkowicie scalonej aplikacji, skupiające się na aspektach całościowych, wymaganiach prezentowanych przez całość systemu

1.5 Analiza statyczna i dynamiczna

Analizę oprogramowania możemy podzielić na dwa potoki:

Analizę statyczną - która nie wymaga uruchamiania programu i sprowadza się do wyszukiwania błędów w kodzie źródłowym

Analizę dynamiczną - utożsamianą z uruchamianiem oprogramowania i badaniem pod kątem ścieżki przebiegu i czasu wykonywania

1.6 Białe i czarne skrzynki

Analiza dynamiczna obejmuje testy:

- Białej skrzynki - czyli tzw. testy strukturalne opracowywane na podstawie kodu źródłowego oprogramowania, zakładające jego znajomość
- Czarnej skrzynki - czyli testy funkcjonalne, które nie zakładają znajomości realizacji funkcji systemu, ale jedynie jasno sprecyzowanych wymagań wobec systemu. System traktowany jest jako wspomniana czarna skrzynka, która pobiera dane i produkuje wyniki w tylko sobie znany sposób

1.7 Białe skrzynki

Ponieważ przy tego rodzaju testach dużą rolę pełni znajomość kodu źródłowego oprogramowania, ważne jest by konstruować testy tak aby pokrycie procentowe instrukcji testami było możliwie duże (najlepiej gdyby każda instrukcja wykonywała się choć raz). Ponadto należy zadbać aby każda gałąź programu była testowana, więc dane wejściowe należy dobierać w taki sposób, aby warunki każdej z instrukcji warunkowych były choć raz spełnione i choć raz niespełnione.

1.8 Czarne skrzynki

Konstruowanie testów czarnej skrzynki wymaga dokładnego opisu wymagań dla danej funkcji! Zazwyczaj nie można przetestować oprogramowania dla wszystkich możliwych danych, należy więc wyodrębnić klasy danych wejściowych i oprzeć się na założeniu, że system działający poprawnie dla reprezentanta danej klasy (konkretnych danych) zadziała poprawnie dla dowolnych danych z tej klasy. Ważne jest nie zapominanie o testowaniu zachowań oprogramowania dla wartości granicznych.

2 Zanim zaczniemy testować

Musimy sobie odpowiedzieć na kilka pytań. Po pierwsze musimy dokładnie znać zmienną, którą chcemy testować. Porównywanie maszyn zupełnie różniących się pod względem konfiguracji sprzętowej i programowej nic nie obrazuje. Na przykład nie można porównywać wydajności kompilatorów na różnych systemach operacyjnych, ponieważ wprowadzamy zbyt dużo zmiennych.

Kolejną rzeczą, którą musimy wiedzieć, nawet zanim zaczniemy szukać narzędzi do przeprowadzania konkretnych testów, jest przyjęcie odpowiadającej nam metodologii. Czy potrzebny nam jest pomiar wydajności typowych aplikacji (i wybieranych z jakiego klucza), czy interesuje nas czysty wynik np. wzrostu wydajności konkretnego urządzenia. Czy obchodzi nas wpływ innych czynników na obserwowany w normalnych (nie testowych) warunkach na wydajność urządzenia?

2.1 Automatyzacja testów

Automatyka testu to oprogramowanie pozwalające automatyzować wszystkie operacje wykonywane przy testowaniu systemu czy aplikacji, począwszy od

generowania danych wejściowych i oczekiwanych wyników, poprzez wykonywanie zestawów testów bez ręcznej ingerencji testera, a skończywszy weryfikacji wyników i ocenie czy test "przeszedł".

2.1.1 Co daje?

- Skuteczność testów
- Powtarzalność testów (możliwość ich odtworzenia)
- Ominięcie omyłek powstających przy ręcznym wprowadzaniu danych
- Szybkość i wydajność weryfikacji poprawek błędów (także łatwość redukcji źłych poprawek")
- Przyśpieszenie przeprowadzania testów
- Ominięcie niespójności i częściowej informacji o systemie i ujednolicenie jej prezentacji (pełne sprawozdania, standardy, kompleksowa analiza wyników)
- Umożliwienie wprowadzania testów długich i złożonych (także obejmujących ogromne ilości danych)
- Szybkość zwrotu kosztów automatyzacji testowania (często 2-3 projekty)

2.1.2 Ograniczenia

- Kosztowności automatyzacji testowania
- Niektóre funkcje mają zdolności tzw. znieczulania i przypadkowej poprawności (czyli ukrywania wad)

2.2 Koszt błędów

Koszty naprawienia błędu mogą być różne w zależności od etapu na jakim projekt się znajduje. Naprawienie błędu na etapie analizy lub projektowania wymaga kilku do kilkunastu minut i szacuje się na koszt kilku dolarów, podczas gdy próby korygowania ich już po wdrożeniu to setki dolarów i dziesiątki godzin pracy programistów. Często oszczędza na testach licząc, że ponieważ wcześniej jakoś się udawało to i tym razem uda się stworzyć produkt dobrej jakości, który ewentualnie na koniec się przetestuje, co prowadzi do nagromadzenia błędów i niepotrzebnego zwiększania kosztów ich naprawy.

3 Przygotowanie do testów

W celu przeprowadzenia testów, bardzo istotne jest wyizolowanie testowanego czynnika. Najlepiej jest przygotować kilka maszyn, o różnej konfiguracji (w tej części, która mamy nadzieje że nie wpływa znacząco na to co robimy).

Następnym krokiem jest zapewnienie powtarzalnych warunków testowanie. Efektem takich zabiegów powinna być powtarzalność testów, a środkiem jest zamknięcie wszystkich aplikacji korzystających z testowanych rzeczy.

4 Przeprowadzanie testów

4.1 Analiza statyczna

Analiza statyczna może być bardzo istotnym narzędziem przydatnym przy poszukiwaniu prostych błędów (prostych, a nie trywialnych), takich jak przepełnienia buforów, odwołania do zwolnionych wskaźników, czy przepełnianie liczb. Wykorzystuje się ją między innymi do poszukiwania możliwych metod włamania się, do sprawdzania co tak naprawdę robią binarne patche (np. Servive Pack do Windows), czy też do testowania zgodności konkretnych implementacji z założeniami np protokołu sieciowego.

W tej chwili jest to szeroko rozwijana gałąź, jednak większość prowadzonych badań nie udostępnia własnych rozwiązań (za darmo), nie znalazłem żadnych rozwiązań gotowych do zastosowania.

Analiza statyczna jest trudna do zastosowania, często wymaga analizy grafu przepływu, który jest ciężko uzyskać, można to zrobić pisząc własny analizator, lub modyfikując kompilator np gcc.

4.2 Narzędzia

4.2.1 Prymitywne

Wiele rzeczy da się sprawdzić korzystając z naprawdę prostych narzędzi małym kosztem. Na przykład ping może służyć do pomiaru jakości łącza w zależności od obciążenia (opcje -f -s), hdparm pomoże nam sprawdzić wydajność operacji I/O na dysku, a wavemon, czy iwconfig pomoże nam testować jakość łącza radiowego.

4.2.2 Zaawansowane

- lmbench - jest to narzędzie do przeprowadzania globalnych benchmarków linuxa. Jest (według autorów) dosyć powszechnie używane i pozwala

coś powiedzieć o ogólnej wydajności sprzętu i oprogramowania.

- JUnit - JUNIT jest przedstawicielem rodziny programów do automatyzacji testów jednostkowych na klasach (obok niego istnieje szereg programów "XUNIT", m.in. CppUNIT, DUNIT,SQLUNIT). Podstawowymi zaletami JUNIT są przystępność, łatwość wprowadzania testów oraz to, że jest on darmowy. JUNIT jest dostępny pod adresem: `www.junit.org`

- JMeter - O ile testy kodu można przeprowadzać ręcznie, o tyle test wydajności czy obciążenia w większości przypadków wymagają automatyzacji testów. Jednym z automatycznych narzędzi do testowania wydajności jest JMETER.

  JMETER oryginalnie został stworzony do testowania APACHE, jednak dziś wykorzystywany jest do testowania wydajności usług sieciowych, baz danych czy obsługi żądań FTP. JMETER daje możliwości symulowania zachowania użytkowników korzystających z witryny internetowej. Automatyzacja pozwala na niemal dowolne przeciążanie aplikacji celem przekonania się o jej wydajności. Symulowani użytkownicy podzieleni są na trzy grupy:

  - Controller - w sposób bezpośredni odpowiada za testowanie (jego zachowanie może być konfigurowane - CONFIG)
  - Timer - poprzez zarządzanie opóźnieniami działania upodabnia wirtualnego użytkownika do rzeczywistego
  - Listener - zbiera informacje o przebiegających testach celem wyprodukowania raportów

  Przedstawiciele każdej z opisanych grup powinien pojawić się w projektowanym planie testów. Nadawszy wirtualnym użytkownikom strony i podstrony, które powinni odwiedzać pozostaje nam już tylko stworzenie łistenera"przedstawiającego wyniki - najlepiej w postaci diagramu (wykresu).

4.3 Case study - testy Access Pointów na potrzeby hotspot.mimuw

4.3.1 system

Przeprowadziliśmy w zespole Sebastian Zagrodzki, Paweł Puterla i Maciek Fijałkowski w listopadzie bieżącego roku testy kilku Access Pointów na potrzeby hotspot.mimuw.

Dysponowaliśmy trzema urządzeniami D-Link 1000 AP+, LinkSys WAP11, Cisco Aironet.

4.3.2 przygotowanie systemu

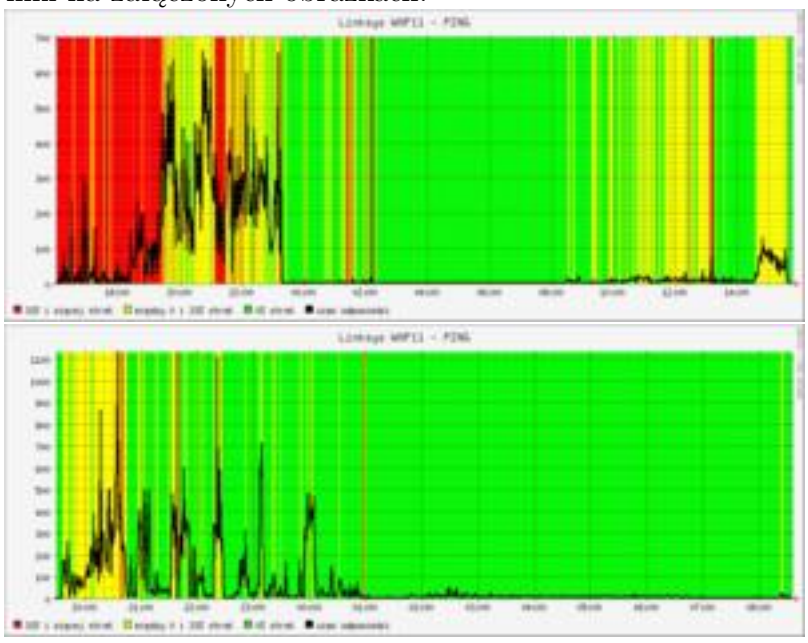
Wszystkie trzy AP testowaliśmy w warunkach pełnego obciążenia. Czyli ustawiliśmy je w stronę akademików DS1 i DS2, skąd ludzie łączą sie do hotspota. Wszystkie 3 były skonfigurowane na publiczny dostęp.

4.3.3 przeprowadzanie testów

Mierzyliśmy kilka rzeczy: siłę sygnału od czasu, czas odpowiedzi na pinga i straty pakietów podczas pinga. Testy zostały przeprowadzone w dniach 17-20.XI bieżącego roku. Celem testów było sprawdzenie niezawodności AP bez względu na obciążenie.

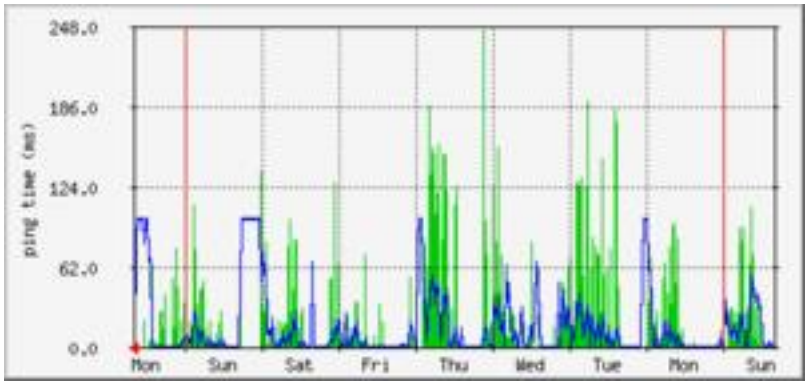
4.3.4 wyniki

Wyniki na załączonych obrazkach.



Dwa powyższe wykresy obrazują ping, ping packet loss i siłę sygnału od czasu, zostały zebrane w ciągu około doby, zobrazowane przy pomocy rrdgraph.

Pierwszy dotyczy AP linksys WAP11, drugi Cisco Aironet 350.



Wykres uzyskany przy pomocy mrtg i pokazuje ping oraz ping packet loss od czasu na przestrzeni tygodnia.

4.3.5 wnioski

4.3.6 co poszło nie tak?

Lista tego co powoduje, że wyniki tych testów były niemiarodajne:

1. Wyniki były prowadzone w różnych dniach tygodnia, co powoduje nierównomierność obciążenia AP.
2. Dane były zbierane przez zbyt krótki okres czasu.
3. Czyli: danych jest za mało

5 Obróbka wyników

Jest to szeroki temat, dlatego omówię pokrótce niektóre istotne fragmenty:

5.1 Prosta matematyka

5.1.1 Pomiar pojedynczej wielkości

Jest to najprostszy przypadek. Podstawowa zasada brzmi tak: Mierzymy daną wielkość więcej niż raz. Odpowiedzią jest średnia wielkości ± odchylenie standardowe. Jeżeli $\overline{x}$ jest średnią pomiarów x_i, taką że

$$\overline{x} = \frac{\sum_i x_i}{n}$$

to odchylenie standartowe wynosi:

$$\sigma = \sqrt{\frac{\sum_i (x_i - \overline{x})^2}{n}}$$

Oczywiście wszystko powyższe ma sens tylko wtedy, kiedy mierzona wielkość jest zgodna z rozkładem Gaussa. Można to bardzo prosto sprawdzić rysując sobie wykres (niezbyt dokładny) wyników i patrzenie czy otrzymane wyniki układają się w krzywą dzwonową.

Pomiary, jeżeli chcemy być naprawdę pewni, należy powtórzyć w nieco innych warunkach (takich, które wydają nam sie nie mieć wpływu) i porównać wyniki.

5.1.2 Pomiar niebezpośredni

Jeżeli wartość mierzona nie jest mierzalna bezpośrednio, lecz jest uwikłana jako współczynnik f-cji f. Jeżeli X i Y to wartości mierzone i mamy nadzieje, że $Y = f(X)$, to wtedy dążymy do tego, aby średni błąd kwadratowy był jak najmniejszy.

Odpowiednie wzory są w książkach, można też użyć gotowego programu, na przykład gnuplot'a.

5.1.3 Błędy

Rozróżniamy (w naszym przypadku) 2 rodzaje błędów: błąd statystyczny i błąd pomiaru. Błąd pomiaru jest związany z dokładnością użytych przedmiotów i jest znany przed pomiarem - na przykład mierzenie czasu wykonania programu przy pomocy time jest obarczone błędem pomiaru około 40ms.

Błąd statystyczny jest związany z metodą pomiarów wielokrotnych i można go policzyć ze wzorów.

5.2 Narzędzia

- gnuplot - doskonałe narzędzie do przedstawiania graficznego wyników oraz (ale to już nie jest temat referatu) obróbki bardziej skomplikowanych danych (np. dopasowywanie funkcji).
- rrdgraph - narzędzie do składania wykresów zależności, z ograniczonymi możliwościami przeliczania
- mrtg - narzędzie przeznaczone do zbierania statystyk sieciowych, może być dostosowane do prowadzenia ciągłych wykresów czegokolwiek.

5.3 Przedstawianie wyników

Od nas zależy w jaki sposób przedstawimy wyniki. Po pierwsze powinny być czytelne. Jeżeli naszym celem jest przedstawienie jakiejś korelacji, korelacja

powinna być widoczna gołym okiem.

6 Wyciąganie wniosków

Przede wszystkim nie należy przesadzać. Należy dokładnie udokumentować, co mierzyliśmy, jakie warunki mogły wpłynąć na niewiarygodność testów, ewentualnie jakby je można poprawić.

Typowy przykład: po przeprowadzeniu syntetycznych testów nie można powiedzieć, że wydajność jednej konfiguracji jest większa niż drugiej, ponieważ nie wiemy jaki mają wpływ rzeczy dziejące się w tle (a możliwości interakcji jest dużo).