Marcin Apostoluk
Wydział Informatyki I Zarządzania
Politechnika Wrocławska

Wrocław, 07.05.2006

Automatyzacja procesu testowania oprogramowania

Abstract

Testowanie jest jednym z najważniejszych etapów cyklu życia oprogramowania. Finalna jakość aplikacji w bardzo dużym stopniu zależy od sposobów oraz jakości wykonywanych testów. Proces testowania jest bardzo czasochłonny oraz monotonny, łatwo tutaj o przeoczenia i pomyłki. Te same testy trzeba powtarzać wielokrotnie wraz ze zmieniającymi się wersjami aplikacji. Dlatego też elementem niezwykle istotnym jest automatyzacja jak największej ilości testów. Istnieje wiele narzędzi wspierających i ułatwiających automatyczne testowanie oprogramowania. Zostaną one pokrótce opisane w niniejszej pracy.

Wprowadzenie

Testowanie oprogramowania jest nieodłącznym procesem wytwarzania aplikacji. Istnieje wiele rodzajów testów: testy jednostkowe, akceptacyjne, obciążeniowe itd. Niewątpliwie jednymi z najważniejszych są testy jednostkowe, gdyż to one pojawiają się w największej części procesu wytwarzania oprogramowania. Testy jednostkowe sprawdzają na bieżąco poprawność tworzonego kodu i pomagają w unikaniu wielu pomyłek. Testowanie może odbywać się na różnych poziomach abstrakcji, przykładowo na poziomie kodu lub interfejsu. Wręcz niewyobrażalne jest, aby testowanie wykonywane było manualnie i powtarzane za każdym razem, gdy zajdzie najdrobniejsza zmiana w aplikacji. Istnieje wiele narzędzi wspierających automatyzację procesu testowania, których wykorzystanie zapewnia znaczną oszczędność czasu i pozwala uniknąć wielu niepotrzebnych błędów.

1. Testy JUnit

JUnit to prosty framework służący do tworzenia testów jednostkowych. Pisząc testy jednostkowe na bieżąco do tworzonych funkcji nie tylko polepsza się jakość kodu, ale i przyspiesza się cały proces eliminując błędy i zmniejszając okres czasu potrzebny na debuggowanie aplikacji.

Idea JUnit polega na stworzeniu testu do każdej potencjalnie niebezpiecznej funkcji. Testy powinny sprawdzić zarówno prawidłowe przebiegi wykonania metod jak i sytuacje, w których metoda np. rzuca wyjątek.

Przykładowo do klasy:

```
public class Performer
{
     public int getValue(int x) throws Exception
     {
          if( x == 0 )
          {
               return 0;
          }

          if( x > 0 )
          {
               return 2 * x;
          }

          throw new Exception("wrong x");

     }
}
```

należy stworzyć test sprawdzający poprawność działania metody `getValue`. Może on wyglądać następująco:

```
public class PerformerTest extends TestCase
{

     public void testGetValue()
     {

          Performer p = new Performer();
          try
          {
               assertEquals( p.getValue( 0 ), 0 );
               assertEquals( p.getValue(1), 2);
```

```
                assertEquals( p.getValue(3), 6);
            }
            catch ( Exception e )
            {
                e.printStackTrace();
                assertTrue( false );
            }

            try
            {
                p.getValue(-1);
                assertTrue(false);
            }
            catch(Exception e )
            {
                assertEquals( e.getMessage(),
                    "wrong x");
            }
        }
}
```

Jak widać w powyższym przekładzie, testom podlega jak największa liczba różnych przebiegów funkcji, włączając w to wywołania zakończone niepowodzeniem. Asercje sprawdzają, czy otrzymane wyniki są zgodne z oczekiwaniami. W przypadku niepowodzenia, test danej metody zostaje zatrzymany.

Klasa testowa JUnit dziedziczy po klasie `junit.framework.TestCase`. Metody testowe muszą zaczynać się od słowa test, przykładowo `testGetValue()`. Dodatkowo klasa testowa może przeciążać metody `setUp()` i `tearDown()`, aby wykonywać pewne wspólne czynności dla wszystkich testów uruchamianych w klasie.

Klasy testowe mogą być grupowane w hierarchię. Służy do tego klasa `junit.framework.TestSuite`, wewnątrz której wywoływane zostają testy klas testowych lub inne zestawy testów.

Pisane testy powinny być jak najprostsze. Należy z pewnością unikać sytuacji, gdy to nie same klasy, lecz klasy testowe muszą być debuggowane.

Niemal wszystkie środowiska programistyczne Javy wspierają tworzenie testów jednostkowych JUnit. JUnit jest frameworkiem bardzo popularnym i docenionym przez wielu programistów.

2. Testy Cactus

Cactus jest frameworkiem dostarczonym przez fundację Jakarta przeznaczonym do testów jednostkowych kodu działającego po stronie serwera. Cactus korzysta z JUnit i rozszerza go. Można stwierdzić, że testy Cactus są odpowiednikiem testów JUnit dla aplikacji działających po stronie serwera aplikacji.

Cactus łatwo integruje się z serwerami aplikacji takimi jak Tomcat, czy Jetty, stworzone są także pluginy do popularnych IDE tkj. Eclipse.

Cactus pozwala na zdefiniowanie żądania http wysyłanego do serwera aplikacji, określenie czynności, które wykonuje serwer aplikacji oraz sprawdzenie zwróconej odpowiedzi. Możliwe jest także ustalanie oraz weryfikowanie zmiennych sesji i aplikacji.

Struktura testów jest niemal identyczna, każda klasa testu musi dziedziczyć z klasy `org.apache.cactus.ServletTestCase`. Przykładowa klasa testowa może wyglądać następująco:

```
public class AuthorArticleListTest
      extends ServletTestCase
{

      public void beginGotoAuthorArticleList(
            WebRequest theRequest )
      {

            theRequest.addParameter( "action",
             “gotoArticleList” );
      }
```

```
    public void testGotoAuthorArticleList()
          throws ServletException
    {
          MainServlet     servlet = new MainServlet();
          this.servlet.init( this.config );
          this.servlet.doGet(
                this.request, this.response );

          ArrayList< Article > articles = (ArrayList)
                this.request.getAttribute( "articles" );
          assertNotNull( articles );
          assertEquals( articles.size(), 3 );
    }

    public void endGotoAuthorArticleList(
          WebResponse theResponse )
    {

          assertTrue( theResponse.getText().contains(
                "<title>article list</title> ) );

    }
}
```

Klasa ta posiada 3 metody:

- `beginGotoAuthorArticleList` - wywoływana na początku, przygotowuje żądanie, które ma być wysłane do servletu,
- `testGotoAuthorArticleList` - inicjalizująca serwer aplikacji, odbierająca żądanie po stronie serwera aplikacji i sprawdzająca parametry przekazane dalej w żądaniu do wynikowej strony jsp,
- `endGotoAuthorArticleList` - sprawdzająca otrzymany wynik.

Również do testów Cactusa dostępne są metody `setUp()` oraz `tearDown()` przygotowujące i kończące testy na danej klasie. Testy można, tak jak w przypadku JUnit, łączyć w zbiory testów, które mogą być wywoływane jako spójna całość.

3. Testy interfejsu oraz transmisji sieciowych na przykładzie narzędzia Rational Robot

Narzędzie Rational Robot jest częścią pakietu Rational Suite Enterprise dostarczanego przez firmę Rational Software(IBM). Rational Robot wspomaga proces automatycznego testowania oprogramowania. Umożliwia nagrywanie odpowiednich skryptów podczas pracy użytkownika z aplikacją oraz ich późniejsze odtwarzanie w celu sprawdzenia, czy aplikacja wciąż zachowuje się poprawnie. Narzędzie to jest bardzo rozbudowane, posiada liczne funkcje, których brak prostym programom testującym. Robot jest także zintegrowany z innymi narzędziami z pakietu Rational Suite Enterprise, co znacznie zwiększa jego możliwości.

Skrypty – praca z programem

Praca z narzędziem Rational Robot opiera się na tworzeniu skryptów, a następnie ich odtwarzaniu. Skrypty składają się z dwóch części:

1) Plik z treścią skryptu, który może być uruchamiany przez Robota. Treści skryptów GUI lub skryptów zapamiętujących informacje przesyłane przez sieć są generowane automatycznie, gdy użytkownik pracuje z aplikacją. Robot tłumaczy akcje wykonywane przez użytkownika na odpowiednie wpisy w treści skryptów.

2) Zbiór właściwości skryptu – przykładowo: typ i cel skryptu.

Rational Robot obsługuje dwa typy skryptów:

1) Skrypty interfejsu użytkownika(ang. GUI scripts)
2) Skrypty używane do zapamiętywania i odtwarzania zapytań i odpowiedzi odpowiednio klienta i serwera w trakcie sesji(ang. VU scripts)

GUI Scripts

Skryptu GUI służą do rejestrowania akcji, które wykonuje użytkownik pracując z aplikacją. Rational Robot zapisuje wszystkie naciśnięcia klawiszy (ang. keystrokes), czy kliknięcia myszą. Akcje te są następnie umieszczane w pliku skryptu napisanym w języku SQABasic. Testy wykonywane za pomocą tych skryptów mogą posłużyć zarówno jako testy funkcjonalne, jak i testy wydajnościowe.

Tworzenie skryptów GUI za pomocą programu Rational Robot powinno składać się z etapów:

a) **Oszacowanie prawdopodobnego punktu rozpoczęcia skryptu oraz punktu jego zakończenia.** Jeśli ustali się początek i koniec wszystkich skryptów w jednym miejscu(przykładowo na pulpicie lub w głównym oknie aplikacji) możliwe będzie odtwarzanie wszystkich napisanych skryptów jeden po drugim w dowolnej kolejności bez konieczności każdorazowego dostosowania środowiska testowego.
b) **Przygotowanie środowiska testowego.** Wszystkie elementy środowiska testowego obecne w momencie nagrywania skryptu powinny być odtworzone w momencie jego późniejszego uruchamiania. Ważnym elementem jest więc minimalizacja ilości działających aplikacji, tak aby późniejsze odtwarzanie skryptu nie wymagało wielu dostosowań.
c) **Załadowanie rozszerzeń środowiska, w którym wykonana jest aplikacja.** Umożliwia to testowanie punktów weryfikacyjnych dla komponentów z różnych środowisk – funkcjonalność ta zostanie omówiona w dalszej części pracy.
d) **Ustalenie opcji zapisywania skryptów interfejsu użytkownika** (przykładowo zapisywanie czasów pracy aplikacji, sposobów rozpoznawania kontrolek, ustalanie skrótów klawiszowych, czy zapisywanie dokładnych pozycji testowanych okien).
e) **Rozpoczęcie nagrywania skryptu.** Okno programu Rational Robot jest automatycznie minimalizowane, od tego momentu wszystkie akcje wykonane przez użytkownika są nagrywane.
f) **Zakończenie nagrywania skryptu i zapisanie utworzonego skryptu.**

Możliwe jest tworzenie skryptów na dwóch poziomach abstrakcji:

a) **Nagrywanie zorientowane na obiekty(Object-Oriented Recording)** – Rational Robot używa zaawansowanych metod(jak identyfikacja wewnętrznych nazw obiektów), często zależnych od środowiska, w jakim wykonano aplikację, do wykonywania akcji na konkretnych obiektach. Robot zapisuje tylko wykonywane akcje, niezależnie od pozycji obiektów na ekranie. Tak więc sytuacja, w której twórca programu przesunie przycisk z jednego miejsca w drugie nie zmieni działania testu – dalej będzie on zachowywał się poprawnie.

   Poniżej przedstawiono przykładowy skrypt na poziomie zorientowania na obiekty. Definiuje on akcję przechwycenia okna programu *Gadu-gadu* oraz przesuwania okna w różne miejsca ekranu.

```
Sub Main
    Dim Result As Integer

    'Initially Recorded: 2006-04-02  17:48:40
    'Script Name: nowy

    Window SetContext, "Class=Shell_TrayWnd", ""
    Toolbar Click, "Text=Obszar powiadomień;\;ItemText=Głośność",
"Coords=9,15"

    Window SetContext, "Caption=Ja (5924416)", ""
    Window MoveTo, "", "Coords=135,140"
    Window MoveTo, "", "Coords=344,253"
    Window MoveTo, "", "Coords=482,178"
    Window MoveTo, "", "Coords=526,121"
    Window MoveTo, "", "Coords=592,181"
    Window MoveTo, "", "Coords=432,209"
    Window CloseWin, "", ""

End Sub
```

b) **Nagrywanie niskopoziomowe(Low-level Recording)** – zapisuje wszystkie ruchy myszy oraz wciśnięcia klawiszy i na ich podstawie buduje skrypt. Funkcjonalność ta jest przydatna w testowaniu aplikacji graficznych, kiedy ważne jest dokładne odtwarzanie ruchów wykonywanych przez użytkownika. Nagrywanie niskopoziomowe jest bardzo wrażliwe na zmiany położenia obiektów w aplikacji.

Poniżej przedstawiono fragment skryptu o funkcjonalności identycznej jak skrypt z przykładu a), wykonanego jednak z użyciem nagrywania niskopoziomowego.

Rational Robot-Windows Lowlevel Script File
Printed at 17:52:22 on 2006-04-02

Started Recording at 17:52:05 on 2006-04-02
Finished Recording at 17:52:11 on 2006-04-02

Message	*Message Parameters*		*Delta Time*
MOUSEMOVE	*x: 262*	*y: 256*	*0*
MOUSEMOVE	*x: 264*	*y: 261*	*15*
MOUSEMOVE	*x: 266*	*y: 264*	*15*
MOUSEMOVE	*x: 269*	*y: 267*	*31*
MOUSEMOVE	*x: 271*	*y: 272*	*31*
MOUSEMOVE	*x: 274*	*y: 275*	*47*
MOUSEMOVE	*x: 278*	*y: 280*	*47*
MOUSEMOVE	*x: 280*	*y: 285*	*62*
MOUSEMOVE	*x: 286*	*y: 293*	*78*
MOUSEMOVE	*x: 293*	*y: 304*	*78*
MOUSEMOVE	*x: 299*	*y: 316*	*93*
LBUTTONDOWN	*x: 948*	*y: 746*	*1781*
MOUSEMOVE	*x: 948*	*y: 746*	*1781*
MOUSEMOVE	*x: 948*	*y: 746*	*1781*
LBUTTONUP	*x: 948*	*y: 746*	*1859*
MOUSEMOVE	*x: 948*	*y: 746*	*1859*
MOUSEMOVE	*x: 945*	*y: 741*	*1922*
MOUSEMOVE	*x: 488*	*y: 215*	*3140*
MOUSEMOVE	*x: 492*	*y: 215*	*3140*

MOUSEMOVE	*x: 494*	*y: 215*	*3156*
MOUSEMOVE	*x: 495*	*y: 215*	*3218*
LBUTTONUP	*x: 495*	*y: 215*	*3343*
MOUSEMOVE	*x: 495*	*y: 215*	*3343*
LBUTTONDOWN	*x: 495*	*y: 215*	*3531*
MOUSEMOVE	*x: 497*	*y: 215*	*3547*
MOUSEMOVE	*x: 500*	*y: 218*	*3547*
MOUSEMOVE	*x: 500*	*y: 218*	*3547*

Punkty weryfikacyjne

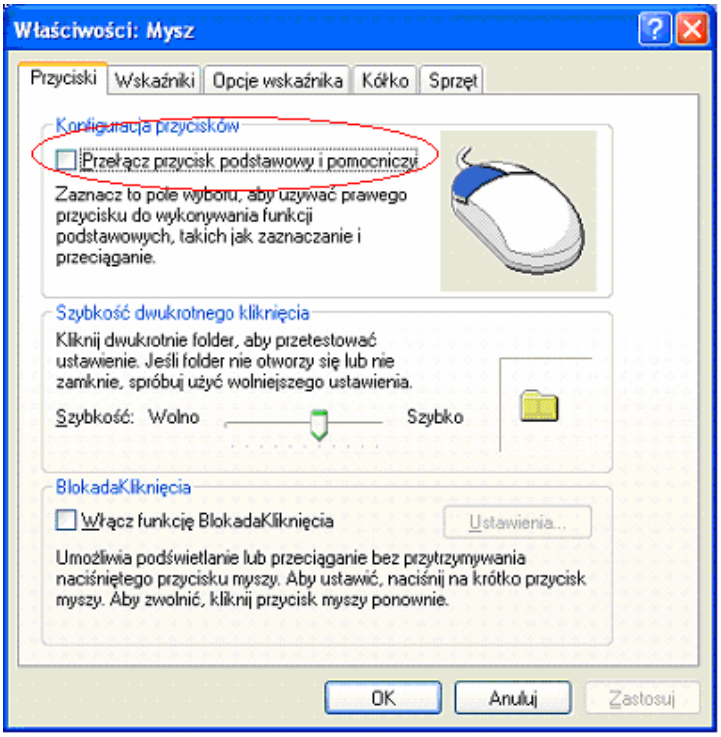
Do utworzonych przez Rational Robot skryptów GUI można manualnie dodawać elementy użytkownika. Dostępne są proste funkcje służące do uruchamiania innych aplikacji, operowanie na *timerach*, czy też sztuczne wprowadzanie opóźnień w wykonywaniu skryptu. Najważniejszymi elementami są jednak punkty weryfikujące poprawność stanu aplikacji w danym momencie wykonywania skryptu. To one faktycznie sprawdzają poprawność aplikacji względem wykonywanego testu.

Dostępna jest spora gama predefiniowanych punktów weryfikacyjnych gotowych do wykorzystania w skryptach:

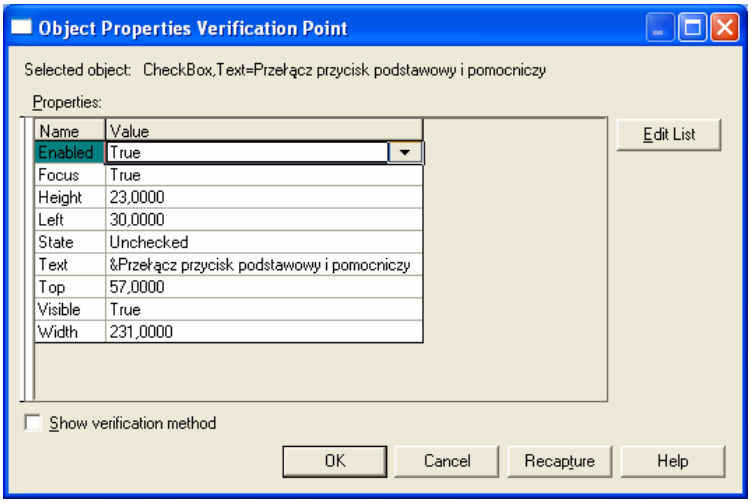
1) **porównania parametrów obiektów aplikacji** – najważniejszy z punktów weryfikacyjnych narzędzia Rational Robot. Umożliwia odczytanie parametrów praktycznie dowolnej kontrolki z dowolnej działającej aplikacji. Wykorzystuje wywołania API do przechwytywania zdarzeń zachodzących w obiektach oraz odczytuje ich atrybuty. Przykładowo po wybraniu okna użytkownik może przejrzeć wszystkie elementy okna takie jak przyciski, menu, pola tekstowe itd. – dla każdego elementu dostępne są jego podstawowe atrybuty wraz z możliwością sprawdzania ich wartości. Narzędzie Rational Robot w prosty sposób umożliwia tworzenie sprawdzeń, czy dany atrybut wybranego elementu spełnia podane kryteria. Wydawałoby się, że można w ten sposób przetestować dowolną z kontrolek widocznych na ekranie. Niestety niektóre kontrolki nie są standardowymi kontrolkami Windows i ich parametry nie mogą być odczytane w taki sposób. Podobnie dzieje się np. dla aplikacji pisanych w Javie, czy też w HTML. Dlatego też, aby przetestować te komponenty należy załadować dodatkowe rozszerzenia

dla aplikacji napisanych w konkretnym języku programowania, a także biblioteki dla komponentów tworzonych przez użytkownika. Umożliwi to w efekcie przetestowanie wszystkich obiektów.

2) **sprawdzanie stanu schowka systemowego** – porównywanie z innym ciągiem znaków,
3) **porównywanie wartości alfanumerycznych** znajdujących się w aktywnej kontrolce,
4) **porównywanie zawartości plików** – przykładowo do sprawdzania poprawności danych zapisywanych przez aplikację,
5) **porównywanie wyświetlanych obrazów** – narzędzie daje możliwość zaznaczenia wybranego fragmentu okna testowanej aplikacji – wycinek okna jest w trakcie testu porównywany z odpowiednim wycinkiem okna testowanego. Idealnie nadaje się to do testowania aplikacji graficznych.
6) **sprawdzanie istnienia pliku, okna lub modułu aplikacji wykonywalnej,**
7) **skanowanie strony internetowej** – Rational Robot wyszukuje wszystkie elementy dostępne pod podanym adresem internetowym, rozpoznaje strukturę katalogów i istnienie plików. Umożliwia następnie na porównanie stanu, który odczyta ze stanem podczas testowania. Porównanie zawartości jest możliwe także dla pojedynczych stron internetowych, nie jest wtedy wymagane skanowanie całego portalu.



Rysunek 1. Rozpoznawanie komponentów wybranego okna przez Rational Robot (Widok okna)



Rysunek 2. Rozpoznawanie komponentów wybranego okna przez Rational Robot, (Widok Rational Robot)

Manualne tworzenie skryptów

Możliwe jest także tworzenie skryptów manualnie. Rational Robot dostarcza edytor skryptów wspierający kolorowanie składni, brak jednak automatycznych podpowiedzi dostarczanych przez narzędzie. Dostępny jest kompilator skryptów wykrywający wszystkie błędy, a także debugger umożliwiający przechodzenie przez skrypt krok po kroku oraz zakładanie pułapek na linie kodu, które użytkownik chce testować.

Manualne tworzenie skryptów można łatwo integrować z automatycznym ich generowaniem podczas wykonywania akcji przez użytkownika. Skrypty takie można edytować manualnie, a do manualnie stworzonych skryptów można w dowolnym miejscu „dograć" dowolny fragment na podstawie akcji użytkownika.

Tworzenie logów dla wykonywanych skryptów

W kod wykonywanego skryptu można wplatać linie kodu odpowiedzialne za logowanie informacji. Funkcjonalność logowania znacznie ułatwia późniejszą weryfikacje przyczyn niepowodzeń w działaniu skryptów. Rational Robot pozwala także na proste wyłączenie zaimplementowanej funkcjonalności logowania.

Uruchamianie testów

Podczas uruchamiania testów Rational Robot odtwarza wszystkie akcje zapisane w skrypcie. Dokładne sprawdzanie i analizowanie wyników można realizować przy pomocy programu Rational TestManager.

Skrypty powłoki interfejsu użytkownika

Rational Robot pozwala na tworzenie skryptów powłoki GUI(w prosty sposób pozwalają na wykonywanie sekwencji wcześniej utworzonych skryptów GUI). Kreator w prosty sposób udostępnia możliwość wybrania skryptów, które mają być wykonywane oraz określenia ich kolejności. W treści skryptu umieszcza wywołania do skryptów wybranych we wcześniejszym kroku.

VU scripts

Skrypty te służą do zapisywania żądań klienta pracującego przez sieć z serwerem(zapis sesji). Dużą zaletą tych skryptów jest fakt, że wiele takich skryptów może być wykonywanych na komputerze w jednym czasie. W skryptach VU zapisane są informacje żądań, jakie klient wysyła do serwera. Testy z użyciem tych skryptów wykorzystywane są głównie do mierzenia wydajności serwera w momentach różnego obciążenia. W testach tych nie można umieszczać punktów weryfikacyjnych.

Rational Robot udostępnia 3 rodzaje zapisywania sesji:

a) Metoda wykorzystująca API(ang. API Recording Method) – Robot zapisuje wywołania API pomiędzy aplikacją klienta i serwera. Nagrywanie odbywa się po stronie klienta, nie zaś w trakcie wysyłania danych, jak dzieje się to w dwóch pozostałych metodach. Metoda ta bardzo dobrze sprawdza się w przypadku szyfrowanych połączeń – Robot przechwytuje dane zanim zostaną one zaszyfrowane. Dużym plusem tej metody jest również fakt, że nie trzeba definiować adresów sieciowych klienta i serwera(potrzeba taka zachodzi w pozostałych metodach). Istnieje możliwość nagrywania wywołań API z wielu klientów jednocześnie.

b) Metoda sieciowa(ang. Network Recording Method) – Robot zapisuje ruch na poziomie pakietów korzystając z interfejsu sieciowego systemu operacyjnego. Wykorzystuje tryb *promiscuous* karty sieciowej. Nagrywanie jest niezależne od rodzaju danych. Można powiedzieć, że proces zapisywania zdarzeń odbywa się „na kablu”(w trakcie wysyłania danych), a nie po stronie klienta. Plusem tej metody jest to, że może przechwytywać żądania klienta z innego komputera – stosowana jest w przypadku użycia nie wspieranych przez Rational Robot platform lub gdy brak fizycznego dostępu do klienta. Ponadto możliwe jest nagrywanie zdarzeń od wielu klientów rezydujących na różnych komputerach.

c) **Metoda proxy(ang. Proxy Recording Method)** – przy zapisywaniu informacji tą metodą, ruch danych pomiędzy klientem i serwerem odbywa się za pośrednictwem komputera proxy. Mechanizm nagrywania podobnie jak w poprzedniej metodzie korzysta z interfejsu sieciowego systemu operacyjnego i polega na odbieraniu pakietów z jednego komputera i wysyłaniu ich do

drugiego przy użyciu gniazd systemowych. Możliwe jest nagrywanie wielu połączeń klientów z różnymi serwerami jednocześnie – żadna inna metoda na to nie pozwala. Dużym plusem jest także to, że Rational Robot nie musi być zainstalowany na żadnym kliencie, ani na żadnym serwerze – wystarczy jedynie instalacja na komputerze proxy.

Poniżej podano przykładowy fragment skryptu VU dla metody API:

```
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK |
HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response
time */
push Think_def = "LR";
Min_tmout = 120000;      /* Set minimum Timeout_val to 2 minutes          */
push Timeout_val = Min_tmout;
push Think_avg = 0;

www_google_pl = http_request ["xxx001"] "www.google.pl:80",
  HTTP_CONN_DIRECT,
  "GET / HTTP/1.1\r\n"
  "Accept: */*\r\n"
  "Accept-Language: pl\r\n"
  "Accept-Encoding: gzip, deflate\r\n"
  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1)\r\n"
  "Host: www.google.pl\r\n"
  "Connection: Keep-Alive\r\n"
  "Cookie: AdSenseLocale=pl;
PREF=ID=cebc936ffb98356b:TM=1141077711:LM=114"
  "1077711:S=xqc_z7p0GJKXp-kv\r\n"
  "\r\n";

  { string SgenURI_001; }
  SgenURI_001 = _reference_URI; /* Save "Referer:" string */

  set Server_connection = www_google_pl;
```

Do wygenerowanych plików sesji można dodawać podobne elementy, jak przy skryptach typu GUI, za wyjątkiem punktów weryfikacyjnych. Przykładowo można dodać *Timer*, który sprawi, że dane będą wysyłane do serwera co 5 sekund. Wygenerowane przez Rational Robot skrypty, także w przypadku skryptów VU można edytować ręcznie. Także tutaj użytkownik ma do dyspozycji kompilator i debugger, które znacznie ułatwiają i przyspieszają pracę. Skrypty VU mogą być ponadto konwertowane do skryptów Visual Basic.

Przy ponownym odtwarzaniu, skrypty VU są uruchamiane przy pomocy programu Rational TestManager, który w znacznym stopniu ułatwia śledzenie wyników, tworzenie raportów oraz znajdowanie błędnych sytuacji.

Pule danych

Gdy przykładowa aplikacja pracuje z serwerem i wysyła informacje o dodaniu nowych danych do bazy, żaden inny tester nie może dodawać tych samych danych, gdyż operacja taka zakończy się niepowodzeniem(duplikacja danych w bazie). Potrzebne są więc zbiory danych, które będą rozdysponowane pomiędzy różne programy testujące w celu popranej obsługi pewnych operacji. Rational Robot wprowadza pule danych, które są używane w przypadku wielokrotnych sesji przy użyciu skryptów VU. Pula danych jest niczym innym jak zbiorem danych testowych wprowadzonych przez użytkownika. Może być to przykładowo plik w formacie .csv zawierający informacje, z których korzystać będzie robot przy wykonywaniu operacji.

Podsumowanie narzędzia Rational Robot

Rational Robot zapewnia kompleksową funkcjonalność testowania automatycznego opartą o testy interfejsu użytkownika oraz walidację stanów aplikacji podczas wykonywania testów. W odróżnieniu od prostych programów testujących, Rational Robot udostępnia generyczny mechanizm rozpoznawania obiektów z różnych języków programowania i środowisk programistycznych, co daje bardzo duże możliwości i znacznie ułatwia automatyzację procesu testowania. Za pomocą tego narzędzia możliwe jest także testowanie interfejsów sieciowych aplikacji, testowanie wydajności

serwera. Również ten mechanizm jest bardzo rozbudowany i powinien zaspokoić potrzeby każdego użytkownika.

Niemniej jednak Rational Robot posiada także wady. Niewątpliwie należy do nich mało rozwinięty moduł pomocy, przez który dość trudno jest poznać funkcjonalność aplikacji. Rational Robot, choć jest prawie w pełni autonomicznym programem, nie może być zakupiony i użyty jako osobna aplikacja, nie jest więc produktem „dla każdego".

4. Podsumowanie

W niniejszej pracy przedstawiono sposoby oraz narzędzia służące do automatyzacji procesu testowania oprogramowania. Temat ten jest niezmiernie istotny, a co za tym idzie – popularny, tak więc ilość narzędzi wspierająca testowania jest bardzo duża. Powyżej przedstawiono tylko przykładowe frameworki i aplikacje. Proces testowania w bardzo dużym stopniu może podlegać automatyzacji. Istnieją nawet próby automatyzacji testów wydajności i użyteczności interfejsu. Nieprędko jednak komputer będzie mógł oceniać prawdziwą użyteczność i jakość pracy użytkownika z programem, tak więc ludzie są nadal potrzebni i niezbędni w procesie testowania aplikacji.