Symulacja w Badaniach i Rozwoju
Vol. 1, No. 3/2010

Kazimierz WORWA
Wojskowa Akademia Techniczna, ul. Kaliskiego 2, 00-908 Warszawa
E-mail: kworwa@wat.edu.pl

Analiza porównawcza wybranych strategii losowego testowania oprogramowania

1 Wprowadzenie

Tradycyjna organizacja procesu produkcji oprogramowania obejmuje 4 główne etapy prac: określenie wymagań, opracowanie projektu, implementację oraz testowanie.

Treścią pierwszego etapu jest analiza potrzeb użytkownika i określenie szczegółowych wymagań na oprogramowanie. W tym etapie prac bardzo pożądany jest udział przyszłego użytkownika oprogramowania, który powinien współtworzyć i zatwierdzić listę wymagań. Specyfikacja wymagań programowych stanowi podstawę dalszych prac, mających na celu wytworzenie oprogramowania.

Etap projektowania oprogramowania obejmuje projektowanie wstępne i projektowanie szczegółowe. Treścią projektowania wstępnego jest określenie szeroko rozumianej współpracy z użytkownikiem oraz określenie architektury oprogramowania, identyfikującej wszystkie komponenty oprogramowania oraz definiującej pośrednictwa i połączenia między nimi. Projektowanie szczegółowe obejmuje opracowanie tzw. logiki wewnętrznej wszystkich wyodrębnionych komponentów czyli szczegółowe opracowanie algorytmów ich działania oraz określenie struktur wykorzystywanych danych.

Etap implementacji obejmuje wyrażenie specyfikacji projektowych, będących wynikiem poprzedniego etapu prac, w wykorzystywanym środowisku programowym.

Treścią etapu testowania jest ocena zgodności wytworzonego oprogramowania ze specyfikacją wymagań. Podstawowym celem tego etapu prac jest wykrycie i usunięcie maksymalnej liczby błędów, popełnionych w trakcie realizacji wcześniejszych etapów. Wymienione etapy składają się na cykl rozwoju oprogramowania (*software development cycle*). Praktyczne metody realizacji wymienionych etapów, w tym wykorzystywane techniki i narzędzia projektowo-implementacyjne, jak również metody organizacji pracy zespołów wykonawczych, są przedmiotem zainteresowania inżynierii oprogramowania. Dokonujący się, bardzo znaczący, wzrost zastosowań systemów komputerowych i towarzyszący mu wzrost wymagań jakościowych na oprogramowanie tych systemów, wymusza stały rozwój i doskonalenie metod wytwarzania oprogramowania. Wykrycie i usunięcie określonej liczby błędów na ogół przyczynia się do zwiększenia niezawodności programu. Liczba wykrytych błędów silnie zależy od zakresu, dokładności i organizacji prac związanych z testowaniem. Praktyka produkcji oprogramowania pokazuje, że prace te są bardzo czasochłonne i wymagają wysokich nakładów finansowych. Powoduje to m.in., że etap testowania charakteryzuje się bardzo znacznym udziałem w ogólnym koszcie wytworzenia programu. W przypadku dużego i złożonego systemu programowego koszt testowania może stanowić 40-70% łącznego kosztu jego produkcji. Fakt ten jest bezpośrednią konsekwencją dużej czasochłonności prac prowadzonych na tym etapie,

konieczności zaangażowania doświadczonych i wykwalifikowanych programistów oraz dużego zużycia czasu pracy wykorzystywanego zestawu komputerowego.

Podstawowym problemem, jaki należy rozwiązać na etapie planowania i przygotowania procesu testowania oprogramowania jest określenie takiego podzbioru zbioru wszystkich możliwych zestawów danych wejściowych, który maksymalizowałby prawdopodobieństwo wykrycia wszystkich błędów, popełnionych we wcześniejszych etapach procesu wytwarzania oprogramowania. Problem ten pojawia się z uwagi na fakt, że na ogół - ze względu na czas trwania i koszt procesu testowania - niemożliwe jest przetestowanie oprogramowania w oparciu o cały zbiór wszystkich możliwych zestawów danych wejściowych. Problem określenia wspomnianego podzbioru jest problemem projektowania zbioru danych testowych (testów). Każdy test stanowi dopuszczalną kombinację wartości, które mogą przyjmować zmienne wejściowe testowanego produktu programowego, wymaganą dla jego pojedynczego uruchomienia, przy czym zmienne wejściowe są to takie zmienne, wartości których są ustalane bezpośrednio na podstawie danych wejściowych, np. w wyniku wykonania instrukcji wczytywania danych.

Sposób tworzenia zbioru danych testowych, w oparciu o który realizowany będzie proces testowania oprogramowania, zależny jest od przyjętej metody testowania, każda z których opiera się na określonym kryterium selekcji zestawów danych testowych.

Obok sposobu doboru poszczególnych zestawów danych testowych, decydujące znaczenie dla wyników testowania oraz wielkości poniesionych w jego trakcie nakładów czasowo-finansowych, mają liczności zbiorów testów, wykorzystywanych w poszczególnych etapach procesu testowania.

Projektowanie zbioru danych testowych, wykorzystywanych w procesie testowania złożonych systemów programowych, jest procesem złożonym, czasochłonnym i wymagającym bardzo dużej systematyczności w pracy. W większości przypadków testowania takich systemów zadawalające rezultaty można osiągnąć tylko przy wykorzystaniu narzędzi komputerowego wspomagania procesu generowania danych testowych. Liczność wykorzystywanego zbioru danych testowych pozostaje w ścisłym związku z przyjętymi kryteriami zakończenia procesu testowania oprogramowania.

W zależności od sposobu podejścia do problemu projektowania zbioru danych testowych wykorzystywane w praktyce metody testowania można podzielić na dwie klasy:

- metody deterministyczne,
- metody losowe.

Deterministyczne metody testowania zakładają tworzenie zbioru danych testowych w sposób wyłącznie deterministyczny, tj. bez udziału czynnika losowego. Poszczególne zestawy danych wejściowych, będące elementami projektowanego zbioru testów, określane są w oparciu o analizę specyfikacji wymagań lub analizę kodu źródłowego testowanego programu.

W odróżnieniu od metod deterministycznych w metodach losowych część zestawów danych testowych (lub nawet wszystkie), wykorzystywanych w procesie testowania oprogramowania, tworzona jest z wykorzystaniem czynnika losowego, tj. poprzez losowanie, przy czym najczęściej wykorzystywaną w praktyce projektowania zbiorów danych testowych metodą losowania jest losowanie bez zwracania.

Metoda tworzenia zbioru danych testowych w oparciu o losowanie jest bardzo powszechnie stosowana w praktyce testowania. Wynika to przede wszystkim z jej prostoty (nie wymaga np. bardzo pracochłonnej analizy struktury logicznej testowanego programu), bezpośrednią konsekwencją której jest niski koszt realizacji procesu testowania oraz duża podatność na jego automatyzację. Drugą, przesądzającą o szerokim praktycznym wykorzystaniu, cechą metody losowej jest jej duża skuteczność, mierzona liczbą błędów, wykrytych w procesie testowania. Badania prowadzone w celu porównania efektywności najczęściej wykorzystywanych w praktyce metod testowania wykazały, że dla szerokiej klasy programów testowanie oparte na losowej generacji danych testowych jest bardziej efektywne od pozostałych metod, przy czym szczególnie dobre wyniki daje losowe generowanie danych testowych w oparciu o rozkład prawdopodobieństwa opisany przez profil operacyjny (*operational profile*) testowanego programu, określający prawdopodobieństwa występowania poszczególnych zestawów danych wejściowych rozpatrywanego programu w warunkach jego rzeczywistej eksploatacji użytkowej.

Analiza literatury testowania losowego pozwala na dokonanie podziału metod jego realizacji na metody, istotą których jest:

- testowanie w pełni losowe (*random testing*),
- strukturalne testowanie losowe (*partition testing*), tj. testowanie losowe w ramach wyróżnionych podzbiorów zbioru danych wejściowych.

Testowanie w pełni losowe polega na wykorzystywaniu w procesie testowania programu zbioru zestawów danych testowych, utworzonego drogą losowania (najczęściej bez zwracania) kolejnych jego elementów z całego zbioru zestawów danych wejściowych testowanego programu. Podstawowym problemem jest tutaj określenie rozkładu prawdopodobieństwa, z jakim losowane są poszczególne zestawy danych, składające się na tworzony w ten sposób zbiór danych testowych. W przypadku braku stosownych przesłanek co do natury tego rozkładu przyjmuje się często rozkład równomierny na całym zbiorze danych wejściowych testowanego programu.

Bardzo często stosowaną w praktyce odmianą testowania w pełni losowego jest testowanie, w którym wykorzystywany zbiór danych testowych tworzony jest także na podstawie całego zbioru danych wejściowych testowanego programu, ale losowanie kolejnych zestawów danych testowych odbywa się w oparciu o rozkład prawdopodobieństwa określony przez profil operacyjny programu.

Różnice występujące w podejściu do określenia rozkładu prawdopodobieństwa, zgodnie z którym losowane są poszczególne zestawy danych testowych, występujące w opisanych odmianach testowania w pełni losowego, dają w konsekwencji różnice w ich skuteczności. W szczególności, wykorzystanie profilu operacyjnego testowanego programu pozwala ukierunkować proces testowania na wykrycie tych wszystkich błędów, których wystąpienie w trakcie użytkowania programu w warunkach rzeczywistych jest najbardziej prawdopodobne. Z kolei, wykorzystywanie zbioru danych testowych, określonego metodami losowymi bez uwzględnienia profilu operacyjnego testowanego programu, ukierunkowuje proces testowania na wykrycie wszystkich błędów programowych, niezależnie od prawdopodobieństw ich ujawnienia się w trakcie użytkowej pracy programu.

Drugą, z wcześniej wymienionych, grup metod testowania losowego są metody, w których kolejne elementy tworzonego zbioru danych testowych są losowane z pewnych podzbiorów całego zbioru danych wejściowych programu, nazywanych partycjami (*partitions*). Opisywane podejście wymaga zatem dokonania podziału zbioru wszystkich możliwych zestawów danych wejściowych programu na partycje, przed rozpoczęciem właściwego procesu losowania poszczególnych zestawów danych testowych. Ten sposób tworzenia wykorzystywanego w procesie testowania zbioru zestawów danych testowych wykorzystywany jest najczęściej w ramach metod testowania strukturalnego. Partycje, na które dzielony jest cały zbiór wszystkich możliwych zestawów danych wejściowych testowanego programu, wyodrębniane są wówczas na ogół w taki sposób, aby tworzące je zbiory danych wejściowych uaktywniały określone drogi logiczne (lub ich klasy). W konsekwencji, proces testowania realizowany jest w oparciu o pewną liczbę zbiorów danych testowych (odpowiadających liczbie wyodrębnionych partycji), przy czym losowanie poszczególnych zestawów danych w ramach wyodrębnionych partycji odbywa się najczęściej w oparciu o rozkład równomierny.

Duran i Ntafos [l] oraz Hamlet i Taylor [2] przeprowadzili serię eksperymentów i badań symulacyjnych w celu porównania efektywności testowania w pełni losowego i strukturalnego testowania losowego. Analiza uzyskanych rezultatów pokazała, że w szeregu przypadkach tańsze testowanie w pełni losowe okazało się być techniką efektywniejszą (w sensie zdolności wykrywania błędów) od droższego strukturalnego testowania losowego.

Weyuker i Jeng [3] przeprowadzili formalną analizę porównawczą efektywności obu ww. technik testowania losowego. Wyniki tej analizy wskazują na to, że strukturalne testowanie losowe może być techniką lepszą lub gorszą od testowania w pełni losowego, w zależności od sposobu rozłożenia się zestawów danych wejściowych powodujących wystąpienie błędu, pomiędzy poszczególne partycje.

Niniejsza praca, zainspirowana wynikami uzyskanymi przez Weyuker i Jeng [3], zawiera pewne uogólnienia tych wyników , jak również pewne nowe rezultaty w zakresie określenia warunków, w których jedna technika testowania losowego jest lepsza od drugiej.

2 Porównanie efektywności technik testowania losowego

Niech dla rozpatrywanego programu *P*, zbiór jego wszystkich możliwych danych wejściowych będzie oznaczony przez *D*, a jego rozmiar, tj. liczba tworzących go zestawów danych wejściowych, przez *d, d>0.* Elementy zbioru *D*, odpowiadające niepoprawnemu działaniu programu będą nazywane błędnymi zestawami danych wejściowych (w skrócie: błędnymi wejściami), a ich liczba będzie oznaczona przez *m*, $0 \le m \le d$. Pozostałe zestawy danych wejściowych będą nazywane poprawnymi zestawami danych wejściowych (w skrócie: poprawnymi wejściami), a ich liczba będzie oznaczona przez *c*.

Niech θ oznacza tzw. współczynnik błędu, określony jako $\theta = m/d$. Dla potrzeb porównania efektywności obu rozpatrywanych technik testowania losowego zakłada się, że w każdej z tych technik testowania łączna liczba wylosowanych zestawów danych wejściowych (testów) wynosi *n*. W przypadku strukturalnego testowania losowego zbiór zestawów danych wejściowych *D* jest dzielony na *k* podzbiorów, $k > 1$, oznaczanych

przez D_i, gdzie $i=1, 2, ..., k$. Podzbiór D_i ma rozmiar d_i i zawiera m_i błędnych wejść, $0 \le m_i \le d_i$, oraz $c_i = d_i - m_i$ wejść poprawnych oraz współczynnik błędu $\theta_i = m_i / d_i$. Liczba testów losowanych z i-tego podzbioru wynosi n_i, $n_i \ge 1$. Oczywiście zachodzi

$$\sum_{i=1}^{k} n_i = n .$$

Zgodnie z wcześniejszym założeniem porównanie efektywności obu technik testowania losowego przeprowadzone zostanie dla tej samej łącznej liczby losowanych testów. Przyjmując, że podzbiory D_i można napisać:

$$\sum_{i=1}^{k} d_i = d , \quad \sum_{i=1}^{k} m_i = m , \quad \sum_{i=1}^{k} c_i = c .$$

W dalszych rozważaniach przyjmuje się, że losowanie testów odbywa się bez zwracania, zgodnie z rozkładem równomiernym, tzn. że wszystkie testy losowane są z takim samym prawdopodobieństwem. Oznacza to w szczególności, że kiedy test jest losowany ze zbioru D lub D_i, prawdopodobieństwo, że wylosowany test jest błędnym wejściem wynosi odpowiednio θ lub θ_i.

Dla potrzeb porównania efektywności obu rozpatrywanych technik testowania losowego rozpatrywane będzie prawdopodobieństwo zdarzenia, polegającego na tym, wśród wylosowanych przypadków testowych znajduje się co najmniej jeden błędny zestaw danych wejściowych, co oznacza, że w oparciu o tak wygenerowany zbiór zestawów danych wejściowych w procesie testowania rozpatrywanego programu wykryty zostanie co najmniej jeden błąd. Prawdopodobieństwo to oznaczane będzie odpowiednio przez P_r dla testowania w pełni losowego i P_p dla strukturalnego testowania losowego. Prawdopodobieństwa te są określone następująco [3]:

$$P_r(n) = 1-(1-\theta)^n \text{ oraz } P_p(\boldsymbol{n}) = 1-\prod_{i=1}^{k}(1-\theta_i)^{n_i} ,$$

gdzie $\boldsymbol{n} = (n_1, n_2, ..., n_k)$, $k > 1$, jest wektorem liczby testów n_i wylosowanych z i-tego podzbioru zestawów danych wejściowych (partycji) programu.

Dla przypadku tylko jednej partycji, tj. dla $k=1$, zachodzi $P_p=P_r$.

Podobnie, jeśli $\theta_1 = \theta_2 = ... = \theta_k = \theta$ otrzymuje się

$$P_p(\boldsymbol{n}) = 1-\prod_{i=1}^{k}(1-\theta_i)^{n_i} = 1-\prod_{i=1}^{k}(1-\theta)^{n_i} = 1-(1-\theta)^{n_1+n_2+\cdots+n_k} = 1-(1-\theta)^n = P_r(n)$$

co oznacza, że prawdopodobieństwa P_r oraz P_p są równe.

Mogłoby się wydawać, że z punktu widzenia liczby wykrytych błędów strukturalne testowanie losowe powinno być bardziej efektywne od testowania w pełni losowego. Jednakże z badań Durana i Ntafosa [1] jak również Hamleta i Taylora [2] wynika, że różnice w efektywności obu metod są bardzo niewielkie, a bardzo często testowanie w pełni losowe może być nawet bardziej efektywne od strukturalnego testowania losowego.

Łatwo zauważyć, że - zgodnie z intuicją - strukturalne testowanie losowe będzie lepsze od testowania w pełni losowego, jeśli większość testów będzie pochodziła z partycji mających większe wartości współczynników błędu. Spełnienie tego warunku nie gwarantuje jednak zachodzenia tego efektu, co ilustruje poniższy przykład 1. W przykładzie tym rozpatrywane są 3 przypadki strategii testowania, każdy z których opiera się na 3. partycjach ($k=3$), przy czym w każdym przypadku ogólna liczba testów oraz łączna liczba testów niepoprawnych jest taka sama. Dla pierwszego przypadku strukturalne testowanie losowe jest lepsze od testowania w pełni losowego, tzn. $P_r<P_p$, dla drugiego przypadku jest odwrotnie, tzn. $P_r>P_p$, podczas gdy w trzecim przypadku obydwa prawdopodobieństwa są jednakowe, tzn. $P_r=P_p$.

Oczywiste jest, że w praktyce nie można mieć a priori dokładnych informacji o rozkładzie niepoprawnych zestawów danych wejściowych testowanego programu, które mogłyby być wykorzystane w procesie strukturalnego testowania losowego. Pomimo braku takich informacji Weyuker i Jeng [3] wskazali taki sposób podziału zbioru wszystkich możliwych danych wejściowych programu na podzbiory (partycje), który gwarantuje, że strukturalne testowanie losowe nie będzie gorsze od testowania w pełni losowego. Autorzy cytowanej pracy pokazali, że jeśli $d_1 = d_2 = \ldots = d_k$ oraz $n_1 = n_2 = \ldots = n_k$, to $P_p \geq P_r$. Jeśli ponadto niepoprawne zestawy danych są równo rozłożone pomiędzy partycje, tj. $m_1 = m_2 = \ldots = m_k$, to wówczas $P_p = P_r$. Strukturalne testowanie losowe zatem nigdy nie będzie gorsze od testowania w pełni losowego jeśli partycje będą miały jednakowe liczności, tj. $d_1 = d_2 = \ldots = d_k$ i z każdej z nich wylosowane zostaną takie same liczby przypadków testowych, tj. $n_1 = n_2 = \ldots = n_k$. Efekt ten ilustruje przykład 2. Warunek ten jest warunkiem dostatecznym, aby strukturalne testowanie losowe było bardziej efektywne od testowania w pełni losowego. Spełnienie tego warunku w praktyce testowania jest jednakże bardzo trudne (jeśli w ogóle możliwe), ponieważ w oparciu o współcześnie stosowane techniki strukturalnego projektowania zbiorów danych testowych bardzo trudno jest uzyskać jednakowo liczne partycje.

Przykład 1.

	k=3										
Przypadek	d_i		m_i		n_i		θ_i		θ	P_r	P_p
1	d_1	*1200*	m_1	*10*	n_1	*10*	θ_1	*0,0083*	*0,01*	*0,1821*	*0,1912*
	d_2	*500*	m_2	*7*	n_2	*7*	θ_2	*0,014*			
	d_3	*300*	m_3	*3*	n_3	*3*	θ_3	*0,01*			
Razem		*2000*		*20*		*20*					
2	d_1	*1200*	m_1	*11*	n_1	*13*	θ_1	*0,0092*	*0,01*	*0,1821*	*0,1725*
	d_2	*500*	m_2	*7*	n_2	*3*	θ_2	*0,014*			
	d_3	*300*	m_3	*2*	n_3	*4*	θ_3	*0,0067*			
Razem		*2000*		*20*		*20*					
3	d_1	*1200*	m_1	*12*	n_1	*13*	θ_1	*0,01*	*0,01*	*0,1821*	*0,1821*
	d_2	*500*	m_2	*5*	n_2	*3*	θ_2	*0,01*			
	d_3	*300*	m_3	*3*	n_3	*4*	θ_3	*0,01*			
Razem		*2000*		*20*		*20*					

Przykład 2.

	$k=3$										
Przypadek	d_i		m_i		n_i		θ_i		θ	P_r	P_p
1	d_1	*700*	m_1	*10*	n_1	*50*	θ_1	*0,0143*	*0,0100*	*0,7785*	*0,7788*
	d_2	*700*	m_2	*7*	n_2	*50*	θ_2	*0,0100*			
	d_3	*700*	m_3	*4*	n_3	*50*	θ_3	*0,0057*			
Razem		*2100*		*21*		*150*					
2	d_1	*700*	m_1	*7*	n_1	*50*	θ_1	*0,0100*	*0,0100*	*0,7785*	*0,7785*
	d_2	*700*	m_2	*7*	n_2	*50*	θ_2	*0,0100*			
	d_3	*700*	m_3	*7*	n_3	*50*	θ_3	*0,0100*			
Razem		*2100*		*21*		*150*					

W przykładzie 2. rozpatrywane są 2 przypadki, każdy z których zawiera 3 partycje. W pierwszym przypadku $d_1=d_2=d_3$ oraz $n_1=n_2=n_3$. Wówczas zachodzi $P_p \geq P_r$. W drugim przypadku dodatkowo spełniony jest warunek $m_1=m_2=m_3$ i wówczas wartości prawdopodobieństw P_p i P_r są jednakowe.

Chen i Yu [4-5] dokonali następującego uogólnienia wcześniej omawianego warunku wskazanego przez Weyuker i Jeng [3]: jeśli partycje są rozłączne to strukturalne testowanie losowe będzie bardziej efektywne od testowania w pełni losowego, jeśli liczba przypadków testowych losowanych z poszczególnych partycji jest proporcjonalna do liczności tych partycji. Formalnie oznacza to, że jeśli $k \geq 2$ i $n_1/d_1=n_2/d_2=\ldots=n_k/d_k$, to $P_p \geq P_r$. Efekt ten ilustruje przykład 3, w którym $k=3$ oraz $n_1/d_1=n_2/d_2=n_1/d_1=0,06$. Dla każdego trzech przypadków łączna liczba testów pozostaje taka sama $n=2100$, łączna liczba testów niepoprawnych $m=m_1+m_2+m_3$ jest dla każdego przypadku inna. Łatwo zauważyć, że w każdym z trzech przypadków rozpatrywanych w przykładzie 3. strukturalne testowanie losowe jest efektywniejsze od testowania w pełni losowego. Widać także, że wzrost liczby testów niepoprawnych $m=m_1+m_2+m_3$ pociąga za sobą wzrost wartości obu prawdopodobieństwa P_p i P_r.

Warto zauważyć, że wartości prawdopodobieństw P_p w przykładzie 3. są tylko nieznacznie większe od wartości prawdopodobieństw P_r. Przyczyny tego efektu zostaną wyjaśnione w dalszej części pracy.

Przykład 3.

	$k=3$										
Przypadek	d_i		m_i		n_i		θ_i		θ	P_r	P_p
1	d_1	*500*	m_1	*10*	n_1	*30*	θ_1	*0,0200*	*0,0095*	*0,6996*	*0,7019*
	d_2	*300*	m_2	*7*	n_2	*18*	θ_2	*0,0233*			
	d_3	*1300*	m_3	*3*	n_3	*78*	θ_3	*0,0023*			
Razem		*2100*		*20*		*126*					
2	d_1	*500*	m_1	*10*	n_1	*30*	θ_1	*0,0200*	*0,0143*	*0,8371*	*0,8377*
	d_2	*300*	m_2	*10*	n_2	*18*	θ_2	*0,0333*			
	d_3	*1300*	m_3	*10*	n_3	*78*	θ_3	*0,0077*			
Razem		*2100*		*30*		*126*					
3	d_1	*500*	m_1	*20*	n_1	*30*	θ_1	*0,0400*	*0,0286*	*0,9742*	*0,9747*
	d_2	*300*	m_2	*20*	n_2	*18*	θ_2	*0,0667*			
	d_3	*1300*	m_3	*20*	n_3	*78*	θ_3	*0,0154*			
Razem		*2100*		*60*		*126*					

3 Najlepsze i najgorsze przypadki dla strukturalnego testowania losowego

W celu oceny schematów partycjonowania, określonych przez Weyuker i Jeng [3] oraz Chen i Yu [4-5] w dalszej części pracy określone zostaną warunki służące maksymalizacji prawdopodobieństwa P_p, będące głównym jej wynikiem.

Dla potrzeb określenia tych warunków przydatny będzie następujący lemat.

Lemat 1. Niech $k=2$, $n_1 \geq 1$, $n_2 \geq 1$ a partycje są ponumerowane w taki sposób, że $\theta_1 \leq \theta_2$. Wówczas zachodzi $(1-\theta_1)^{n_1}(1-\theta_2)^{n_2} \geq (1-\theta_1)(1-\theta_2)^{n_1+n_2-1}$.

Dowód. Logarytmując wyrażenie stanowiące tezę lematu oraz wykorzystując podstawowe własności logarytmu otrzymuje się

$$ln[(1-\theta_1)^{n_1}(1-\theta_2)^{n_2}] \geq ln[(1-\theta_1)(1-\theta_2)^{n_1+n_2-1}]$$

a następnie

$$n_1 ln(1-\theta_1) + n_2 ln(1-\theta_2) \geq ln(1-\theta_1) + (n_1+n_2-1) ln(1-\theta_2).$$

Po przekształceniach mamy $(n_1-1) ln(1-\theta_1) \geq (n_1-1) ln(1-\theta_2)$.

Ponieważ $n_1 - 1 > 0$, więc $ln(1-\theta_1) \geq ln(1-\theta_2)$, czyli $(1-\theta_1) \geq (1-\theta_2)$.

Ostania nierówność jest prawdziwa, ponieważ $\theta_1 \leq \theta_2$.

Z lematu 1 wynika natychmiast, że jeśli $k=2$, $n_1 + n_2 = n$ oraz $\theta_1 \leq \theta_2$, to zachodzi

$$P_p(\mathbf{n}) = 1-(1-\theta_1)^{n_1}(1-\theta_2)^{n-n_1} \leq 1-(1-\theta_1)(1-\theta_2)^{n-1}.$$

Lemat 1 pomocny będzie w dowodzie następującego twierdzenia.

Twierdzenie 1. Niech $k > 1$, $\sum_{i=1}^{k} n_i = n$ a partycje są ponumerowane w taki sposób, że $\theta_1 \leq \theta_2 \leq ... \leq \theta_k$. Wówczas zachodzi

$$\prod_{i=1}^{k}(1-\theta_i)^{n_i} \geq \prod_{i=1}^{k-1}(1-\theta_i)(1-\theta_k)^{n-k+1}.$$

Dowód. Lewa strona ostatniej nierówności może być przedstawiona w postaci

$$\prod_{i=1}^{k}(1-\theta_i)^{n_i} = (1-\theta_1)^{n_1}(1-\theta_2)^{n_2}\prod_{i=3}^{k}(1-\theta_i)^{n_i}.$$

Wykorzystując lemat 1 można napisać

$$\prod_{i=1}^{k}(1-\theta_i)^{n_i} = (1-\theta_1)^{n_1}(1-\theta_2)^{n_2}\prod_{i=3}^{k}(1-\theta_i)^{n_i} \geq (1-\theta_1)(1-\theta_2)^{n_1+n_2-1}\prod_{i=3}^{k}(1-\theta_i)^{n_i} \geq$$

$$\geq (1-\theta_1)(1-\theta_2)(1-\theta_3)^{n_1+n_2+n_3-1}\prod_{i=4}^{k}(1-\theta_i)^{n_i} \geq \cdots \geq \prod_{i=1}^{k-1}(1-\theta_i)(1-\theta_k)^{n-k+1}.$$

Następujący wniosek jest natychmiastową konsekwencją twierdzenia 1.

Wniosek 1. Jeśli $k>1$, $\sum_{i=1}^{k} n_i = n$ a partycje są ponumerowane w taki sposób, że $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_k$, to

$$P_p(\boldsymbol{n}) = 1-\prod_{i=1}^{k}(1-\theta_i)^{n_i} \leq 1-\prod_{i=1}^{k-1}(1-\theta_i)(1-\theta_k)^{n-k+1}.$$

Wniosek 1 może być przedstawiony w następującej formie

$$P_p(1,1,\ldots 1,n-k+1) = \max_{\boldsymbol{n}} P_p(\boldsymbol{n}),$$

gdzie $\boldsymbol{n} = (n_1, n_2, \ldots, n_k)$.

Twierdzenie 2. Niech $k>1$, $\sum_{i=1}^{k} n_i = n$ a partycje są ponumerowane w taki sposób, że $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_k$. Wówczas zachodzi

$$\prod_{i=1}^{k}(1-\theta_i)^{n_i} \leq \prod_{i=1}^{k-1}(1-\theta_i)(1-\theta_k)^{n-k+1}.$$

Dowód twierdzenia 2 można przeprowadzić w sposób bardzo podobny do dowodu twierdzenia 1.
Z twierdzenia 2 wynika następujący wniosek.

Wniosek 2. Jeśli $k>1$, $\sum_{i=1}^{k} n_i = n$ a partycje są ponumerowane w taki sposób, że $\theta_1 \geq \theta_2 \geq \ldots \geq \theta_k$, to

$$P_p(\boldsymbol{n}) = 1-\prod_{i=1}^{k}(1-\theta_i)^{n_i} \geq 1-\prod_{i=1}^{k-1}(1-\theta_i)(1-\theta_k)^{n-k+1}.$$

Zatem

$$P_p(1,1,\ldots 1,n-k+1) = \min_{\boldsymbol{n}} P_p(\boldsymbol{n}),$$

gdzie $\boldsymbol{n} = (n_1, n_2, \ldots, n_k)$.
Prawdziwość powyższych wniosków ilustrują przykłady 4 i 5.

Przykład 4.

n	d	m	θ
30	1200	60	0,05

m_1	10
m_2	20
m_3	30

d_1	400
d_2	600
d_3	200

θ_1	0,025
θ_2	0,033
θ_3	0,150

n_1	n_2	n_3	P_p	P_r
1	1	28	0,9900	
1	2	27	0,9887	
1	3	26	0,9871	
1	4	25	0,9854	
1	5	24	0,9833	
...				
10	12	8	0,8592	
10	13	7	0,8398	
10	14	6	0,8178	
10	15	5	0,7928	0,7854
10	16	4	0,7644	
10	17	3	0,7321	
10	18	2	0,6953	
...				
26	2	2	0,6504	
26	3	1	0,6025	
27	1	2	0,6474	
27	2	1	0,5990	
28	1	1	0,5956	

Przykład 4 przedstawia odpowiednio najbardziej i najmniej korzystny przypadek prawdopodobieństwa P_p dla $k=3$, $n=30$ i $\theta_1 \le \theta_2 \le \theta_3$. Zgodnie z twierdzeniami 1 i 2 prawdopodobieństwo P_p osiąga maksimum dla $\mathbf{n}=(1, 1, 28)$ oraz minimum dla $\mathbf{n}=(28, 1, 1)$ odpowiednio.

Warto zauważyć, że dla $\mathbf{n}=(10, 15, 5)$ zachodzi $n_1/d_1=n_2/d_2=n_3/d_3=0,025$, co oznacza zgodność z warunkami sformułowanymi przez Chen i Yu [4-5]. Wiersz, który odpowiada tym warunkom został zacieniowany. Zgodnie z wcześniejszymi uwagami spełnienie tego warunku gwarantuje, że zachodzi $P_p \ge P_r$, jednakże wartość prawdopodobieństwa P_p jest tylko nieznacznie większa od prawdopodobieństwa P_r. Z przykładu 4 wynika bowiem, że $P_p(10,15,5)=0,7928$, podczas gdy $P_r(30)=0,7854$. Jednocześnie jednak widać, że prawdopodobieństwo $P_p(10,15,5)=0,7928$ jest znacząco mniejsze od prawdopodobieństwa $P_p(1,1,28)=0,9900$. Z kolei $\mathbf{n}=(28, 1, 1)$ jest najmniej korzystnym przypadkiem prawdopodobieństwa P_p, bowiem $P_p(1,1,28)=0,9900$.

Przykład 5.

k	2
m_1	80
m_2	320
d_1	5000
d_2	15000

θ_1	0,0160
θ_2	0,0213

n_1	n_2	P_p	P_r	n_1/d_1	n_2/d_2
1	99	0,8836		0,0002	0,0066
2	98	0,8830		0,0004	0,0065
3	97	0,8824		0,0006	0,0065
4	96	0,8817		0,0008	0,0064
5	95	0,8811		0,0010	0,0063
…				…	
21	79	0,8703		0,0042	0,0053
22	78	0,8696		0,0044	0,0052
23	77	0,8689		0,0046	0,0051
24	76	0,8681		0,0048	0,0051
25	75	0,8674	0,8674	0,0050	0,0050
26	74	0,8667		0,0052	0,0049
27	73	0,8660		0,0054	0,0049
28	72	0,8652		0,0056	0,0048
29	71	0,8645		0,0058	0,0047
30	70	0,8638		0,0060	0,0047
…				…	
31	69	0,8630		0,0062	0,0046
96	4	0,8050		0,0192	0,0003
97	3	0,8039		0,0194	0,0002
98	2	0,8029		0,0196	0,0001
99	1	0,8018		0,0198	0,0001

Przykład 5 charakteryzuje odpowiednio najbardziej i najmniej korzystny przypadek prawdopodobieństwa P_p dla $k=2$, $\theta_1 \le \theta_2$ oraz $n = n_1 + n_2 = 100$. Zgodnie z twierdzeniami 1 i 2 prawdopodobieństwo P_p osiąga maksimum dla $\mathbf{n}=(1, 99)$ oraz minimum dla $\mathbf{n}=(99, 1)$ odpowiednio. Warto zwrócić uwagę, że dla $\mathbf{n}=(25, 75)$ zachodzi $n_1/d_1 = n_2/d_2 = 0{,}005$. Odpowiadający temu warunkowi wiersz został zacieniowany.

Analiza przykładu 5 pozwala na sformułowanie następujących spostrzeżeń: jeśli mamy tylko dwie partycje ($k=2$) oraz $\theta_1 \le \theta_2$, to:

1) $P_p > P_r$ wtedy i tylko wtedy, gdy $n_1/d_1 < n_2/d_2$ (powyżej zacieniowanego wiersza);
2) $P_p = P_r$ wtedy i tylko wtedy, gdy $n_1/d_1 = n_2/d_2$ (zacieniowany wiersz);
3) $P_p < P_r$ wtedy i tylko wtedy, gdy $n_1/d_1 > n_2/d_2$ (poniżej zacieniowanego wiersza).

Jeśli $\theta_1 \ge \theta_2$, wówczas powyższe relacje są odwrócone.

4 Podsumowanie

Można ogólnie stwierdzić, że prawie wszystkie strategie testowania programów opierają się na następującym podejściu: zbiór danych wejściowych programu jest dzielony na podzbiory (partycje), z których następnie wybiera się określoną liczbę przypadków testowych. Tak utworzony zbiór danych testowych jest następnie wykorzystywany w procesie testowania programu. Przyjęcie założenia, że wybór przypadków testowych z poszczególnych partycji realizowany jest w sposób losowy prowadzi do strategii, która w przedstawionych rozważaniach nazywana była strukturalnym testowaniem losowym (*partition testing*). Strategia ta konfrontowana była ze strategią opartą na tzw. testowaniu w pełni losowym (*random testing*), zgodnie z którą poszczególne przypadki testowe losowane były z całego zbioru danych wejściowych rozpatrywanego programu.

Symulacje i empiryczne prace Durana i Ntafosa [1], mające na celu porównanie efektywności strukturalnego testowania losowego i testowania w pełni losowego pokazują, że efektywność obu metod jest porównywalna, a w pewnych warunkach testowanie w pełni losowe może być nawet bardziej efektywne, zwłaszcza w aspekcie kosztowym, niż strukturalne testowanie losowe.

Weyuker i Jeng [3] sformułowali warunki, dla których efektywność strukturalnego testowania losowego nie będzie mniejsza od efektywności testowania w pełni losowego. Dla spełnienia tych warunków wystarczy, aby zachodziły równości: $d_1=d_2=\ldots=d_k$ oraz $n_1 = n_2 = \ldots = n_k$. Chen i Yu [4-5] dokonali rozszerzenia i uogólnienia wyników uzyskanych przez Weyuker i Jeng [3], formułując bardziej ogólne warunki, dla których jedna strategia testowania jest lepsza od drugiej. Najważniejszym rezultatem Chen i Yu [4-5] jest spostrzeżenie, że jeśli liczba przypadków testowych losowanych z poszczególnych partycji jest proporcjonalna do liczności tych partycji, to efektywność strukturalnego testowania losowego nie będzie mniejsza od efektywności testowania w pełni losowego. Zasadniczym rezultatem niniejszej pracy jest określenie warunków doboru przypadków testowych z poszczególnych partycji, dla których prawdopodobieństwo P_p przyjmuje wartości ekstremalne, tj. maksymalne i minimalne. Uzyskane wyniki pozwalają na stwierdzenie, że dla określonych przez Weyuker i Jeng [3] oraz Chen i Yu [4-5] warunków, dla których efektywność strukturalnego testowania losowego nie jest mniejsza od efektywności testowania w pełni losowego, wartości prawdopodobieństw P_p i P_r różnią się jedynie nieznacznie. Zostało to zilustrowane w przykładzie 4, w którym dla $k=3$, $n=30$, $d=1200$ i $m=60$ otrzymano $P_r=0,7854$, podczas gdy dla warunków określonych przez Chen i Yu [4-5], tj. dla $m_1=10$, $m_2=20$, $m_3=30$ i $d_1=400$, $d_2=600$, $d_3=200$ oraz $n_1=10$, $n_2=15$, $n_3=5$ (wówczas zachodzi $n_1/d_1 = n_2/d_2 = n_3/d_3 = 0,025$) $P_p = 0,7928$. Zatem $P_p - P_r = 0,0074$, podczas gdy $P_p^{max} - P_r = 0,2046$.

Literatura

1. Duran J. W., Ntafos S.C.: *An evaluation of random testing.* IEEE Transactions on Software Engineering, vol. SE-IO, July 1984, pp. 438-444
2. Hamlet R., Taylor R.: *Partition testing does not inspire confidence*. IEEE Transactions on Software Engineering, vol. SE-16, December. 1990, pp. 1402-1411

3. Weyuker E. J., Jeng B.: *Analyzing partition testing strategies.* IEEE Transactions on Software Engineering, vol. SE-17, July 1991, pp. 703-711
4. Chen T.Y., Yu Y.T.: *On the relationship between partition and random testing*. IEEE Transactions on Software Engineering, vol. SE-20, December 1994, pp. 977-980
5. Chen T.Y., Yu Y.T.: *A more general sufficient condition for partition testing to be better than random testing*. Information Processing Letters, vol. 57, 1996, pp. 145-149

Streszczenie

W pracy przedstawione są wyniki analizy porównawczej dwóch, najczęściej wykorzystywanych w praktyce, strategii losowego tworzenia zbioru danych testowych: testowania w pełni losowego (*random testing*) i strukturalnego testowania losowego (*partition testing*). Zawarte w pracy rozważania mają na celu określenie warunków, dla których jedna z ww. strategii testowania losowego jest lepsza od drugiej, w sensie prawdopodobieństwa wykrycia co najmniej jednego błędu za pomocą utworzonych w oparciu o nie, tak samo licznych zbiorów danych testowych. Przedstawione rozważania zostały zilustrowane wynikami eksperymentów symulacyjnych, służących porównaniu efektywności analizowanych strategii testowania.

A Comparison Analysis of Some Random Software Testing Strategy

Summary

Both partition testing and random testing methods are commonly followed practice towards selection of test cases. For partition testing, the program's input domain is divided into subsets, called subdomains, and one or more representatives from each subdomain are selected to test the program. In random testing test cases are selected from the entire program's input domain randomly. The main aim of the paper is to compare the fault-detecting ability of partition testing and random testing methods. The results of comparing the effectiveness of partition testing and random testing may be surprising to many people. Even when partition testing is better than random testing at finding faults, the difference in effectiveness is marginal. Using some effectiveness metrics for testing and some partitioning schemes this paper investigates formal conditions for partition testing to be better than random testing and vice versa.