testerzy.pl

Priorytetyzacja przypadków testowych za pomocą macierzy

W niniejszym artykule przedstawiony został problem przyporządkowania priorytetów do przypadków testowych przed rozpoczęciem testów oprogramowania. Problem ten występuje w przypadku, gdy osoby testujące nie mają wystarczająco dużo czasu na przeprowadzenie wszystkich testów. Wówczas należy określić, które przypadki testowe (testy) są najważniejsze, które są „średnio ważne", a które można pominąć, ponosząc małe ryzyko wystąpienia awarii w środowisku produkcyjnym.

Praca przedstawia sposób priorytetyzacji przypadków testowych, za pomocą tabeli nazwanej macierzą raportów. Macierz tę można wykorzystać do wprowadzenia pewnych miar jakości oraz funkcjonalności wersji oprogramowania oraz umożliwienia porównań różnych wersji ze sobą.

Matematyczny model problemu priorytetyzacji przypadków testowych

Załóżmy, że mamy do rozwiązania problem doboru priorytetów do przypadków testowych. Niech zatem $T = \{t_1, \dots, t_m\}$ będzie zbiorem wszystkich przypadków testowych, które opisują testy funkcjonalności w konkretnej wersji testowanego systemu. Definiujemy odwzorowanie:

$$g: T \rightarrow \{l, m, h\}, \quad (1)$$

które przypisuje priorytety do przypadków testowych w taki sposób, że h oznacza priorytet wysoki, m średni natomiast l niski. W praktyce zawsze istnieją w systemach funkcje na tyle istotne, że muszą zadziałać zawsze. Istnieje więc potrzeba podziału zbioru T w taki sposób, że:

$$T = T_1 \cup T_2,\ T_1 \cap T_2 = \emptyset, \quad (2)$$

gdzie T_1 oznacza przypadki testowe z wysokim priorytetem, który nie może zostać zmieniony; T_2 jest zbiorem przypadków testowych, w których priorytety mogą być zmieniane w zależności od przyjętych kryteriów. Niech $R = \{R_1, \dots, R_p\}$ będzie zbiorem raportów z testów. Dla raportu R_j definiujemy odwzorowanie $r_j: T \rightarrow \{0,1\}$, które definiuje akceptację przypadku testowego: 1 – test zaliczony (warunki akceptacji zostają spełnione), 0 – test niezaliczony (warunki akceptacji nie zostają spełnione). Jeśli liczba przypadków ze stałym wysokim priorytetem jest równa s w danej iteracji (np. miesiącu) wytwarzania oprogramowania, to zbiór T_2 ma postać $T_2 = \{t_{s+1}, \dots, t_m\}$ i tylko te przypadki będą mogły mieć zmienione priorytety. Może zaistnieć sytuacja, że pewien test ze zbioru T_1 zostanie przeniesiony do zbioru T_2 z pewnych powodów, np. zmiana systemu pracy czy organizacji w firmach, dla których dedykowane jest oprogramowania. Dlatego ma sens umieszczanie przypadków ze zbioru T_1. Zgodnie z iteracyjnym modelem wytwarzania, do oprogramowania co pewien czas dochodzą nowe funkcje. Umownie przyjmujemy, że priorytety nadajemy dla przypadków testowych, które znalazły się przynajmniej w trzech raportach. Definiujemy macierz raportów $A(T) = [a_{ij}]$, gdzie:

$$a_{ij} = r_j(t_i). \quad (3)$$

Dla każdego wiersza z macierzy A definiujemy funkcję $f: T \rightarrow [0,1],\ c: T \rightarrow \{1,2,3, \dots\},$ gdzie:

$$f(t_i) = \sum_{j=p+1-c(t_i)}^{p} \frac{a_{ij}}{c(t_i)}. \quad (4)$$

natomiast $c(t_i)$ oznacza ilość wystąpień przypadku testowego t_i we wszystkich dotychczasowych raportach. Liczba p oznacza ilość wszystkich wykonanych raportów. Wprowadzenie funkcji c jest konieczne, ponieważ przypadek testowy może pojawić się w trzecim, czwartym lub późniejszym raporcie lub zostać pominięty z jakiegoś powodu w pewnym raporcie. Może się to zdarzyć na przykład, gdy nowy klient nie wymaga istnienia konkretnej funkcjonalności i nie ma czasu, żeby przetestować każdą funkcję. Funkcję f można interpretować jako miarę jakości testowanej funkcjonalności, która wprowadza pewne prawdopodobieństwo wystąpienia awarii systemu. Taka awaria może być spowodowana przez nieprawidłowe zadziałanie pewnych funkcji w nowej wersji oprogramowania.

Zastosowanie macierzy raportów do rozwiązania problemu priorytetyzacji przypadków testowych

Załóżmy, że należy przeprowadzić testy oprogramowania, które ma za sobą 10 iteracji wytwarzania oraz 30 testów, z czego 10 ma wysoki priorytet. Przyjmiemy również, że czas wykonania wszystkich testów jest stosunkowo długi, więc o kolejności przeprowadzenia tych testów decydowały będą ich priorytety. Raport z poprzednich testów przedstawia tabela 1. Wiersze od pierwszego do dziesiątego przedstawiają przypadki testowe ze stałym wysokim priorytetem (czyli ze zbioru T_1):

Numer przypadku testowego	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
1	1	1	1	0	1	1	0	1	0	1
2	1	1	1	1	1	1	1	1	1	1
3	0	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	0	1	1	1	0
5	1	0	1	1	1	1	0	0	1	1
6	1	1	1	0	0	0	0	1	1	1
7	1	1	1	1	1	0	1	1	1	1
8	1	1	1	1	1	1	1	1	0	0
9	1	1	1	1	0	1	0	1	1	1
10	1	0	0	1	1	1	1	1	1	1
11	0	1	1	1	1	1	1	1	1	1
12	0	1	1	1	0	1	1	1	1	0
13	1	1	1	0	1	1	0	0	1	1
14	1	0	0	0	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1
16	1	0	0	1	1	1	0	1	1	1
17	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	0	1	0	1	1
19	1	1	1	1	0	0	1	1	1	1
20	X	0	1	1	1	1	1	1	1	1
21	X	0	0	1	1	1	1	0	1	1
22	X	1	0	1	1	1	1	1	1	1
23	X	1	1	1	1	1	1	1	1	1
24	X	0	1	0	1	1	1	1	1	1
25	X	1	1	0	1	1	1	1	1	1
26	X	1	1	0	1	1	0	1	1	0
27	X	X	1	1	1	1	1	1	1	1
28	X	X	1	1	1	1	0	1	1	1
29	X	X	X	1	0	1	0	1	1	1
30	X	X	X	1	1	1	0	0	0	1

Tabela 1: Macierz raportów z poprzednich dziesięciu iteracji wytwarzania oprogramowania

Na podstawie wzoru (4) obliczamy dla każdego wiersza wartość funkcji f, którą przedstawia następująca tabela:

Numer przypadku testowego	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Wartość funkcji f	*0,7*	*1*	*0,9*	*0,8*	*0,7*	*0,6*	*0,9*	*0,8*	*0,8*	*0,8*	*0,9*	*0,7*	*0,7*	*0,7*	*1*

Tabela 2: Wartości funkcji f dla przypadków testowych (1)

Numer przypadku testowego	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*
Wartość funkcji f	*0,7*	*1*	*0,8*	*0,8*	*0,89*	*0,67*	*0,89*	*1*	*0,78*	*0,89*	*0,67*	*1*	*0,88*	*0,71*	*0,57*

Tabela 3: Wartości funkcji f dla przypadków testowych (2)

Mając tabele z policzonymi wartościami funkcji f z poprzednich raportów należy przyjąć kryteria funkcji g, wg których przyporządkowane zostaną priorytety do przypadków testowych. Taką czynność wykonuje zwykle kierownik, menadżer czy analityk testów (w większych firmach). Przykładowy wzór funkcji g został przedstawiony poniżej:

$$g(t_i) = \begin{cases} l, & f(t_i) \geq 0{,}9 \\ m, & f(t_i) \in (0{,}7; 0{,}9) \\ h, & f(t_i) \leq 0{,}7 \end{cases} \quad (5)$$

Wobec tego po dziesięciu poprzednich iteracjach, testy (przypadki testowe) będą miały priorytety przedstawione w tabeli 4 i tabeli 5. Zgodnie z funkcją g, kolejność wykonywania testów w następnej iteracji będzie następująca:

1) **1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 21, 26, 30 - priorytet wysoki;**
2) **18, 19, 20, 22, 24, 25, 28, 29 - priorytet średni;**
3) **11, 15, 17, 23, 27 - priorytet niski.**

Numer przypadku testowego	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
Wartość priorytetu	*h*	*l*	*h*	*m*	*h*	*h*	*l*	*m*	*m*	*m*	*l*	*h*	*h*	*h*	*l*

Tabela 4: Wyznaczone wartości priorytetów dla przypadków testowych (1)

Numer przypadku testowego	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*
Wartość priorytetu	*h*	*l*	*m*	*m*	*m*	*h*	*m*	*l*	*m*	*m*	*h*	*l*	*m*	*m*	*h*

Tabela 5: Wyznaczone wartości priorytetów dla przypadków testowych (2)

Mając macierz $A(T)$ ze wszystkimi przypadkami, można postępować analogicznie. Przedziały, zgodnie z którymi funkcja g nadaje priorytety przypadkom testowym, mogą być dowolne dobierane. Mogą one zależeć od tego, jak dużą pewność co do jakości testowanego oprogramowania chce mieć osoba za nie odpowiedzialna. Nasuwa się pytanie: co zrobić, jeśli czas przeznaczony dla testów jest ograniczony, podczas gdy istnieje dużo przypadków z wysokim lub średnim priorytetem? Podjęcie decyzji należy do osoby odpowiedzialnej za zapewnienie odpowiedniej jakości.

Wprowadzenie miar jakości oprogramowania i funkcjonalności za pomocą macierzy raportów

Miary jakości oprogramowania i porównywanie testów z poprzednimi iteracjami są bardzo ważne, ponieważ pozwalają zdecydować, czy testy oraz rozwój oprogramowania idzie w dobrym kierunku. Wykorzystanie macierzy $A(T)$ może wprowadzić pewne miary jakości oprogramowania i umożliwić porównanie różnych wersji ze sobą. Patrząc na macierz jako całość, miarę jakości $m(A(T)): A(T) \rightarrow [0; 1]$ można zdefiniować wzorem:

$$m\big(A(T)\big) = \frac{\sum_{i=1}^{p_k}\sum_{j=1}^{m_k}\gamma(a_{ij})}{p_k * m_k - \partial(A(T)_k)}, \quad (6)$$

gdzie:

p_k - ilość raportów po iteracji k;

m_k - ilość przypadków testowych po iteracji k;

$\partial(A(T)_k)$ – liczba elementów macierzy $A(T)$ po iteracji k, które mają wartość X;

$$\gamma(a_{ij}) = \begin{cases} 1, & jeśli\ a_{ij} = 1 \\ 0, & jeśli\ a_{ij} \in \{0, X\}. \end{cases}$$

Załóżmy, że chcemy porównać wersję oprogramowania między 7 oraz 8 iteracją wykorzystując tabelę 1 oraz przedstawiony wyżej wzór (6). Wobec tego należy obliczyć następujące wyrażenia:

$$m_7\big(A(T)\big) = \frac{20}{30} = 66{,}66\%,$$

$$m_8\big(A(T)\big) = \frac{26}{30} = 86{,}66\%.$$

Po iteracji 8 jakość wersji po pierwszych testach była lepsza o 20%.

Bardzo łatwo na podstawie powyższych wzorów wprowadzić sposób porównania konkretnych funkcji lub obszarów. Wystarczy ograniczyć się do przypadków związanych tylko z obszarem, który należy porównać. Wprowadzając oznaczenie $m_k(A(T|S)): A(T) \rightarrow [0; 1]$ można zdefiniować miarę obszaru po iteracji k:

$$m_k(T|S) = \frac{\sum_{j=1}^{h}\gamma(a_{kj})}{h - \partial(A(T|S)_k)}, \quad (7)$$

gdzie:

$T|S$ - podzbiór przypadków testowych związanych z konkretną funkcją/obszarem;

h - liczba elementów zbioru S.

Łatwo zauważyć, że wzór (7) to uogólnienie wzoru (6). Porównując kolejne wersje pod kątem konkretnych obszarów można również zestawić umiejętności programistów (w dużych projektach), co może pomóc w podjęciu decyzji o szkoleniu lub reorganizacji pracy programistów.

Zalety stosowania macierzy raportów

Z pewnością łatwo zauważyć kilka zalet zastosowania macierzy w celu priorytetyzacji przypadków testowych:

1. Metoda jest prosta i łatwo ją zaimplementować poprzez odpowiedni program komputerowy lub arkusz kalkulacyjny.
2. Można porównać ze sobą testy z różnych iteracji.
3. Istnieje możliwość porównania różnych funkcji i zaobserwowania na podstawie tego zestawienia jak zachowuje się jedna funkcja naprzeciwko innych w każdej iteracji oraz jak te funkcje zachowują się średnio względem siebie.
4. Można wprowadzić miarę jakości pojedynczej funkcji i jej zmian w cyklu życia oprogramowania.
5. Poszczególne wersje testowanej aplikacji mogą być porównywane jako całość z wersjami z innych iteracji.

Autor

Marek Żukowicz jest absolwentem matematyki na Uniwersytecie Rzeszowskim. Obecnie pracuje jako tester. Jego zainteresowania skupiają się wokół testowania, matematyki, zastosowania algorytmów ewolucyjnych oraz zastosowania matematyki w procesie testowania. Interesuje się również muzyką, grą na akordeonach oraz na perkusji.