Kumulowanie się defektów jest możliwe - analiza i potwierdzenie tezy

Marek Żukowicz

14 marca 2018

Streszczenie

Celem napisania artykułu jest próba podania konstruktywnego dowodu, który wyjaśnia, że niewielka liczba modułów zwykle zawiera większość usterek znalezionych przed wydaniem lub jest odpowiedzialna za większość awarii na środowisku produkcyjnym.

1 Wstęp

Zasada kumulowania się błędów zgodnie z sylabusem ISTQB Foundation Level mówi: "Pracochłonność testowania jest dzielona proporcjonalnie do spodziewanej oraz zaobserwowanej gęstości błędów w modułach. Niewielka liczba modułów zwykle zawiera większość usterek znalezionych przed wydaniem lub jest odpowiedzialna za większość awarii na produkcji" [1]. W celu wykazania prawdziwości tej zasady przedstawimy testowane funkcjonalności w programie jako zbiór trójek (f, X, Y), gdzie:

- $f-$ funkcjonalność w oprogramowaniu (prościej mówiąc funkcja),
- $X-$ dziedzina funkcji f (dane wejściowe),
- $Y-$ wartości funkcji f (oczekiwany zbiór zachowań).

Defekt, z definicji, to *wprowadzona statyczna wada modułu lub systemu* [2]. Prościej mówiąc, jest to framgemt kodu powodujący nieprawidłowe działanie. Defekt z kolei wywołuje **awarię** - *dowolny warunek, który odbiega od oczekiwań bazujących na specyfikacji wymagań, dokumentacji projektowej, dokumentacji użytkownika (...)*[2]. W celu potwierdzenia tezy, o której mowa w streszczeniu, wprowadzimy następującą definicję:

Definicja 1 *Jeśli dla funkcji $f : X \to Y$ istnieje taki argument $x \in X$, że $f(x) \notin Y$, to mówimy, że funkcja f zawiera defekt.*

Takie sformułowanie definicji oznacza, że istnieje pewne wejście dla funkcjonalności, które powoduje działanie niepożądane. Inaczej mówiąc, zachowanie po przetworzeniu wejścia nie trafiło do oczekiwanego zbioru zachowań.

System informatyczny może być modelowany na wiele sposobów, natomiast w artykule przedstawimy testowany system zgodnie z modelem:

- cały system to zbiór takich trójek: (f_i, X_i, Y_i), $i \in \{1, 2, ...n\}, n \in N$, które połączone są relacjami,
- X_i dziedzina funkcji f_i (dane wejściowe),
- Y_i wartości funkcji f_i (oczekiwany zbiór zachowań).

Zgodnie z matematyczną definicją, relacja to dowolny podzbiór iloczynu kartezjańskiego skończonej liczby zbiorów. Definicja ta oddaje intuicję pewnego związku, czy zależności między elementami wspomnianych zbiorów, które pozostają w związku albo łączy je pewna zależność, czy też własność [3]. Niech S będzie systemem informatycznym oraz niech $(f_1, X_1, Y_1) \in S$, $(f_2, X_2, Y_2) \in S$. Dla potrzeb artykułu wprowadzimy kolejną definicję:

Definicja 2 *Mówimy, że pomiędzy f_1 oraz f_2 istnieje relacja, jeśli*

$$f_1(X_1) \subseteq Y_2,$$

albo

$$f_2(X_2) \subseteq Y_1.$$

Zapis $f_1 \sim f_2$ będzie oznaczał, że $f_1(X_1) \subseteq Y_2$.

Wyżej opisaną definicję należy rozumieć w ten sposób, że po dostarczeniu pewnych danych wejściowych do funkcji działającej na pewnym obiekcie (formularzu, stronie, oknie) otrzymamy oczekiwane dane wyjściowe. Takim przykładem może być dokument w programie do obsługi gospodarki magazynowej, który musi przejść przez kilka statusów w aplikacji, np. dokument magazynowy, dokument oferty sprzedaży. Przycisk "Dalej" lub "Next" to funkcja, która przekształca dokument o statusie k w dokument o statusie $k + 1$. Dane wejściowe to dane widocznego ekranu z tłem i dostąpnymi akcjami (X_i). Natomiast dane wyjściowe to kolejny ekran w procesie (Y_i).

2 Zjawisko kumulacji defektów

System informatyczny może być modelowany na wiele sposobów. W tym celu pomocne są modele matematyczne podczas wnioskowania na temat jakości [6]. W artykule przedstawimy go zgodnie z taką zasadą, że dostępne funkcjonalności to zbiór takich trójek (f_i, X_i, Y_i), $i \in \{1, 2, ...n\}, n \in N$, pomiędzy którymi istnieją relacje.

Definicja 3 *Ciągiem kroków (ścieżką) w testowanym systemie będziemy nazywali dowolną skończoną relację* $f_1 \sim f_2 \sim ... \sim f_n, n \in N$, *gdzie* $f_1(X_1) \subseteq X_2, f_2(X_2) \subseteq X_3, ..., f_{n-1}(X_{n-1}) \subseteq X_n$.

Wyżej przedstawiona definicja jest niezbędna do wykazania faktu kumulowania się błędów.

Przykład 1. Załóżmy, że w oprogramowaniu istnieje defekt, czyli dla pewnej trójki (f_i, X_i, Y_i) istnieje takie $a \in X_i$, że $f_i(a) \notin Y_i$. Zjawisko kumulowania się błędów w oprogramowaniu zaczyna się od tego, że użyta jest funkcjonalność f_i dla argumentu a w celu wykonania ścieżki. Oczywiście dana funkcjonalność może korzystać z jednego lub kilku modułów, natomiast argument a może posiadać wiele danych składowych przetwarzanych przez jeden lub więcej modułów. Dowolny ciąg kroków nie będzie możliwy do zrealizowania, jeśli użyjemy argumentu a w tym ciągu kroków. Przykładowo mając zaimplementowane ścieżki:

- 1) $f_3 \sim ... \sim f_i \sim ... \sim f_{n-3}$
- 2) $f_4 \sim ... \sim f_i \sim ... \sim f_{n-1}$
- ...
- k) $f_7 \sim ... \sim f_i \sim ... \sim f_{n-5}$

nie zadziała **co najmniej** k ścieżek w oprogramowaniu, a takie zdarzenia będą wywołane tym samym defektem.

Przykład 2. Załóżmy, że wystąpi takie zjawisko dla pewnych funkcjonalności $(f_i, X_i, Y_i), (f_j, X_j, Y_j)$, że istnieją $a \in X_i$, $b \in X_j$, dla których $f_i(a) \notin Y_i$ oraz $f_j(b) \notin Y_j$. Podczas implementowania ścieżek zawierających funkcje f_i, f_j liczba możliwych awarii będzie dwa razy większa od liczby wymaganych ścieżek, które muszą zadziałać.

Z wyżej przedstawionych przykładów wynika, że liczbę możliwych awarii (niedziałających ścieżek) można obliczyć poprzez pomnożenie argumentów powodujących awarię przez ilość wymaganych ścieżek. Z tego wynika, że kilka defektów leżących w niewielkim obszarze może powodować sporo problemów w aplikacji.

Wyżej zaprezentowany model funkcjonalności zachowa swoje własności również na poziomie kodu i klas.

Przykład 3. Przykładem kumulowania się defektów jest dziedziczenie stosowane w programowaniu obiektowym. Ta cecha polega na umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów pochodnych nie trzeba redefiniować całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy [4]. Jeżeli klasa bazowa zawiera funkcjonalności z defektami, to wszystkie klasy pochodne (specjalistyczne) również bądą zawierały defekty. Liczba wszystkich defektów będzie

co najmniej równa iloczynowi defektów w klasie bazowej oraz liczbie klas, które z niej dziedziczą.

Przykład 4. Podobnym przykładem do dziedziczenia jest agregacja danych np. kompozycja. Proces ten polega na tym, że tworzy się nową klasę używając klas już istniejących (często nazywa się to "tworzeniem obiektu składowego"). Nowa klasa może być zbudowana z dowolnej liczby obiektów (obiekty te mogą być dowolnych typów) i w dowolnej kombinacji, by uzyskać pożądany efekt [5]. Agregacja tworzy nowy typ danych z nowymi funkcjonalnościami. Jeśli klasy, z których powstał nowy typ będą łącznie zawierały k defektów, to liczba defektów powstałych w ten sposób w nowym typie będzie większa lub równa liczbie k. Może również zaistnieć taka sytuacja, że pewna składowa klasa zawiera defekt, czyli dla pewnej trójki (g, X, Y) istnieje takie $a \in X$, że $g(a) \notin Y$. Załóżmy, że nowy typ zawiera n nowych funkcjonalności. Mamy takie przypadki:

1. Funkcja g w nowym typie nie jest wywołana w każdej funkcjonalności (nie wszystkie funkcje używają funkcji g jako część składowej),
2. Funkcja g w nowym typie wywoływana jest n razy,
3. Funkcja g w nowym typie wywoływana jest w każdej funkcjonalności co najmniej jeden raz (może być kilka razy).

Pierwsze dwa przypadki są proste pod względem liczenia przyszłych awarii. Z ostatniego przypadku wynika, że ilość nowych defektów może być większa niż n. Przykładowo, gdy $n = 10$, a g będzie wywołane średnio 2 razy w każdej nowej funkcjonalności, to liczba możliwych awarii będzie wynosiła 20 (a powoduje to tylko jeden argument a). Stąd wynika, że defekty są skumulowane tylko w jednym niewielkim obszarze.

Przykład 5. Przykładem kumulowania się błędów jest również zastosowanie funkcji rekurencyjnych. Są one implementowane nie tylko na liczbach, ale dla potrzeb artykułu skupimy się tylko na funkcjach liczbowych. Zgodnie z naszą definicją dla trójki (g, X, Y) z funkcją rekurencyjną g istnieje takie $a \in X$, że $g(a) \notin Y$. Jeśli funkcja rekurencyjna zawiera defekt, to nie będzie możliwe wyznaczenie wszystkich jej wartości, które wymagają użycia argumentu a - w przypadku funckji liczbowo-liczbowych jeden argument powoduje, że nie zadziała większość przypadków.

Literatura

[1] Sylabus ISTQB Foundation Level

[2] A. Roman, Testowanie i jakość oprogramowania, PWN W-wa 2015

[3] https://pl.wikipedia.org/wiki/Relacja_(matematyka)

[4] https://pl.wikipedia.org/wiki/Programowanie_obiektowe

[5] https://pl.wikipedia.org/wiki/Agregacja_(programowanie_obiektowe)

[6] http://testerzy.pl/baza-wiedzy/matematyk-testerem-oprogramowania

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.

Recenzenci:

Anna Justyna Rejrat, Radosław Smilgin