Testowanie oprogramowania

Piotr Hofman
University of Warsaw

Plan

1. Podstawowa terminologia.
2. Systematyka testów.
3. Tworzenie testów
4. Mierzenie jakości testów.

failure, fault, error

failure

Zachowanie systemu niezgodne ze specyfikacją.

fault

Element kodu powodujący failure.

error

To błąd człowieka, który doprowadza do powstania “fault”.

Zgodnie z nomenklaturą IEEE[a] (IEEE Standard GlossaryofSoftware EngineeringTerminology. ANSI/IEEEStd610.12-1990,1996).

[a] The Institute of Electrical and Electronics Engineers

Test suit- Pakiet testów

Pakiet testów $\mathbb{T}$ to trójka uporządkowana $\langle \Delta_0, \langle \mathbb{T}_1, \mathbb{T}_2 \ldots \mathbb{T}_k \rangle, \langle \Delta_1, \Delta_2 \ldots \Delta_k \rangle \rangle$, gdzie

- Δ_0 to stan początkowy.
- $\mathbb{T}_i$ to poszczególne operacje testowe.
- Δ_i to stan po wykonaniu testu i testu $\mathbb{T}_i$.

Przez stan rozumiemy tu zbiór par nazwa zmiennej i jej wartość.

Test suit- Pakiet testów

Pakiet testów $\mathbb{T}$ to trójka uporządkowana $\langle \Delta_0, \langle \mathbb{T}_1, \mathbb{T}_2 \ldots \mathbb{T}_k \rangle, \langle \Delta_1, \Delta_2 \ldots \Delta_k \rangle \rangle$, gdzie

- Δ_0 to stan początkowy.
- $\mathbb{T}_i$ to poszczególne operacje testowe.
- Δ_i to stan po wykonaniu testu i testu $\mathbb{T}_i$.

Przez stan rozumiemy tu zbiór par nazwa zmiennej i jej wartość.

Independent - Pakiet testów jest niezależny jeśli dla każdego i zachodzi $\Delta_i = \Delta_0$.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

To jak to jest z tym błędem.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

To jak to jest z tym błędem.

1. Test musi wykonać kod zawierający “fault”.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

To jak to jest z tym błędem.

1. Test musi wykonać kod zawierający “fault”.
2. Efekty tego błędu muszą przetrwać do momentu, w którym sprawdzamy prawidłowość wartościowania zmiennych.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

To jak to jest z tym błędem.

1. Test musi wykonać kod zawierający “fault”.
2. Efekty tego błędu muszą przetrwać do momentu, w którym sprawdzamy prawidłowość wartościowania zmiennych.
3. Zmienne zawierające niepoprawne wartości muszą zostać sprawdzone.

Kiedy obserwujemy błąd?

Test to sekwencja operacji, część operacji pochodzi z testowanego systemu a część to sekwencje testujące.

To jak to jest z tym błędem.

1. Test musi wykonać kod zawierający “fault”.
2. Efekty tego błędu muszą przetrwać do momentu, w którym sprawdzamy prawidłowość wartościowania zmiennych.
3. Zmienne zawierające niepoprawne wartości muszą zostać sprawdzone.

Przykład. Testy1 (testy od 00,0,1,2)

Model procesu tesotowania oprogramowania.

Wejście

1. Program $\mathbb{P}$
2. Kryterium jakości testów $\mathbb{C}$

W pętli

1. Wygeneruj testy.
2. Zapuść testy.
3. Jeśli testy nie przeszły szukaj błędu i po poprawieniu wróć do punktu 2.
4. Jeśli przeszły, to sprawdź czy testy spełniają kryterium adekwatności (jakości) testów $\mathbb{C}$?
5. Jeśli nie spełniają to popraw testy i przejdź do punktu 2.
6. Jeśli nastąpiły zmiany w kodzie to wróć do punktu 2.

Kluczowe pytania to:

1. Gdzie szukać błędu?
2. Jak generować testy?
3. Jak wyglądają kryteria jakości testów?

Gdzie szukać błędu?[a]

1. Błąd w kodzie.
2. Błąd w testach.
3. Błąd w specyfikacji.
4. Błąd w OS, w kompilatorze, w bibliotekach, w sprzęcie.
5. Utknęliśmy...

---

[a] Materiały do zajęć : https://www.youtube.com/watch?v=OWbRNOBDwEk&list=PLAwxTw4SYaPkWVHeC_8aSIbSxE_NXI76g&index=3

Przykład

Mars Climate Orbiter.

failure

Orbiter spalił się w atmosferze.

Ale co do tego doprowadziło? NASA i Lockheed Martin nie dogadali się. NASA używa metra jako podstawowej jednostki, a oprogramowanie przygotowane przez inżynierów z Lockheed Martin działało w oparciu o stopy(Angielska jednostka długości).

Remember the Mars Climate Orbiter incident from 1999?

Rysunek: (Źródło: Slideplayer.com)

Jakiego rodzaju to błąd?

Podział testów

- białą i czarną skrzynkę,
- poziomy testów.

White i black box

White box

Testy, które piszemy znając implementację.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Po co?

White i black box

White box

Testy, które piszemy znając implementację.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.

White i black box

White box

Testy, które piszemy znając implementację.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.

White i black box

White box

Testy, które piszemy znając implementację.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.

White i black box

White box

Testy, które piszemy znając implementację.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

White i black box

White box

Testy, które piszemy znając implementację.

- Mogą powstawać równolegle z implementacją.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

White i black box

White box

Testy, które piszemy znając implementację.

- Mogą powstawać równolegle z implementacją.
- Mogą być na niskim poziomie, pozwalają na testowanie małych Kawałków kodu, a nie dużych feature-ów projektu.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

White i black box

White box

Testy, które piszemy znając implementację.

- Mogą powstawać równolegle z implementacją.
- Mogą być na niskim poziomie, pozwalają na testowanie małych Kawałków kodu, a nie dużych feature-ów projektu.
- Łatwiej je automatyzować.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

White i black box

White box

Testy, które piszemy znając implementację.

- Mogą powstawać równolegle z implementacją.
- Mogą być na niskim poziomie, pozwalają na testowanie małych Kawałków kodu, a nie dużych feature-ów projektu.
- Łatwiej je automatyzować.
- Pozwalają na wykrywanie błędów na wczesnym etapie.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

White i black box

White box

Testy, które piszemy znając implementację.

- Mogą powstawać równolegle z implementacją.
- Mogą być na niskim poziomie, pozwalają na testowanie małych Kawałków kodu, a nie dużych feature-ów projektu.
- Łatwiej je automatyzować.
- Pozwalają na wykrywanie błędów na wczesnym etapie.
- Trzeba umieć programować.

Black box

Testy, które powstają tylko w oparciu o specyfikację.

- Mogą powstawać przed implementacją.
- Tworzenie ich pomaga uściślić specyfikację.
- Pozwalają na stworzenie jasnych i obiektywnych kryteriów odbioru produktu.
- Mogą je tworzyć osoby z minimalna znajomością programowania.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy).

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.
3. Testy systemowe. Sprawdzamy działanie całego systemu, często analizując przypadki użycia.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.
3. Testy systemowe. Sprawdzamy działanie całego systemu, często analizując przypadki użycia.
4. Testy akceptacyjne. Testy po stronie klienta.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.
3. Testy systemowe. Sprawdzamy działanie całego systemu, często analizując przypadki użycia.
4. Testy akceptacyjne. Testy po stronie klienta.

1. „smoke testing“ sprawdzenie czy program się uruchamia.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.
3. Testy systemowe. Sprawdzamy działanie całego systemu, często analizując przypadki użycia.
4. Testy akceptacyjne. Testy po stronie klienta.

1. „smoke testing“ sprawdzenie czy program się uruchamia.
2. α testy niestabilnej wersji u producenta.

Poziomy testów

1. Testy jednostkowe (ang. unit-testy). *Zaślepka* to kawałek kodu zwracający odpowiedź która docelowo będzie pochodzić z innego modułu. Służy do testowania, czy ta odpowiedź jest poprawnie interpretowana. Może być użyta kiedy drugi moduł nie jest jeszcze napisany.
2. Testy integracyjne. Sprawdzamy współdziałanie modułów.
3. Testy systemowe. Sprawdzamy działanie całego systemu, często analizując przypadki użycia.
4. Testy akceptacyjne. Testy po stronie klienta.

1. „smoke testing“ sprawdzenie czy program się uruchamia.
2. α testy niestabilnej wersji u producenta.
3. β testy już stabilnej wersji w środowisku klienta, częste przy grach komputerowych.

Porównywanie testów

Definicja

Powiemy, że kryterium testów C_1 pokrywa (subsumes) kryterium C_2 jeśli dla każdego zestawu testów spełniających kryterium C_1 spełniają one też kryterium C_2.

Powyższa obserwacja nie ma dużego przełożenia na praktykę, ale pozwala ustawiać kryteria w formalną hierarchię.

Black Box

The Category-Partition Method

Idea/Balck Box

Wykorzystując specyfikację i własne doświadczenie podzielić możliwe input-y na podzbiory. Stworzyć po jednym teście dla każdego podzbioru. Zakładamy, że zaproponowany podział jest na tyle dokładny, że jeśli działa jedno wejście dla danego podzbioru, to zadziałają wszystkie.

Miarą pokrycia jest procent zdefiniowanych podzbiorów pokrywanych przez testy.
W pracy *The Category-Partition Method for Specifying and Generating Functional Tests* autorzy opisują metodologią podziału na podzbiory i tworzenia testów.

The Category-Partition Method

Analizujemy specyfikację. Tester identyfikuje pojedyncze jednostki funkcyjne które mogą być testowane oddzielnie. Dla każdej jednostki tester identyfikuje:

- parametry wejścia do jednostki funkcyjnej,
- charakterystykę każdego parametru,
- elementy środowiska, które mogą mieć wpływ na działanie jednostki funkcyjnej,
- charakterystykę każdego z elementów środowiska.

Opisujemy powyższe elementy przy pomocy istotnych kategorii, np. jeśli na wejściu mamy sekwencję znaków to do jej opisu będziemy używali długości, zbioru znaków występujących, kodowania.

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.
2. Opisz sprzeczne zależności, np. nie ma możliwości aby string był długości 0 i zawierał znaki specjalne.

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.
2. Opisz sprzeczne zależności, np. nie ma możliwości aby string był długości 0 i zawierał znaki specjalne.
3. Zapisz kategorie, przedziały i zależności w formalnym języku do opisu specyfikacji np. TSL

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.
2. Opisz sprzeczne zależności, np. nie ma możliwości aby string był długości 0 i zawierał znaki specjalne.
3. Zapisz kategorie, przedziały i zależności w formalnym języku do opisu specyfikacji np. TSL
4. Wygeneruj automatycznie zestawy danych testowych.

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.
2. Opisz sprzeczne zależności, np. nie ma możliwości aby string był długości 0 i zawierał znaki specjalne.
3. Zapisz kategorie, przedziały i zależności w formalnym języku do opisu specyfikacji np. TSL
4. Wygeneruj automatycznie zestawy danych testowych.
5. Przeanalizuj wygenerowane testy i zdecyduj, czy specyfikacja testów powinna być poprawiona. Możliwe przyczyny:
   - brak jakiegoś oczywistego przypadku,
   - informacja, że generacja jakiegoś przypadku jest niemożliwa,
   - zbyt duża liczba testów.

The Category-Partition Method

1. Każdą kategorię dzielimy na przedziały, zbiory wartości, które należy przetestować. Np. dla listy sprawdzamy długości $0, 1$, coś pomiędzy 1 a 10000 i coś powyżej 10000 aby sprawdzić czy np. zamiast listy ktoś nie używa tablicy o stałej wielkości.
2. Opisz sprzeczne zależności, np. nie ma możliwości aby string był długości 0 i zawierał znaki specjalne.
3. Zapisz kategorie, przedziały i zależności w formalnym języku do opisu specyfikacji np. TSL
4. Wygeneruj automatycznie zestawy danych testowych.
5. Przeanalizuj wygenerowane testy i zdecyduj, czy specyfikacja testów powinna być poprawiona. Możliwe przyczyny:
   - brak jakiegoś oczywistego przypadku,
   - informacja, że generacja jakiegoś przypadku jest niemożliwa,
   - zbyt duża liczba testów.
6. Produkcja skryptów testowych.

The Category-Partition Method

Czemu warto generować zestawy danych testowych automatycznie?

Przy dużych projektach testów może być dużo.
Osób piszących testy też będzie dużo, a co za tym idzie, warto je wygenerować i rozdać do napisania.

The Category-Partition Method

Czemu warto generować zestawy danych testowych automatycznie?

Przy dużych projektach testów może być dużo.
Osób piszących testy też będzie dużo, a co za tym idzie, warto je wygenerować i rozdać do napisania.
Przy systemach krytycznych ilość kodu sprawdzającego poprawność często przekracza kilkukrotnie wielkość systemu np. seL4 10k linie kodu, weryfikacja 480 k linii dowodów poprawności.

The Category-Partition Method

All-pairs testing

Założenie:

1. większość błędów jest powodowanych przez pojedynczy parametr lub parę parametrów,

The Category-Partition Method

All-pairs testing

Założenie:

1. większość błędów jest powodowanych przez pojedynczy parametr lub parę parametrów,
2. jeśli błąd występuje tylko dla kombinacji większej ilości parametrów to ciężko go znaleźć i mała szansa, że będzie faktycznie występował w działaniu.

The Category-Partition Method

All-pairs testing

Założenie:

1. większość błędów jest powodowanych przez pojedynczy parametr lub parę parametrów,
2. jeśli błąd występuje tylko dla kombinacji większej ilości parametrów to ciężko go znaleźć i mała szansa, że będzie faktycznie występował w działaniu.

Rozwiązanie:
generujemy minimalny zestaw testów pokrywających wszystkie możliwe pary wartości wejściowych.

White Box

Pierwsza przymiarka

Procent/zbiór linii kodu odwiedzonych w czasie wykonywania testów.

Przykład

Python : coverage

`https://pypi.org/project/coverage/`

Pierwsza przymiarka

Procent/zbiór linii kodu odwiedzonych w czasie wykonywania testów.

Przykład

Python : coverage
`https://pypi.org/project/coverage/`

Problemy i zalety

- Linia kodu to nie jest sensowna miara bo w jednej linii kodu można umieszczać jedną lub wiele instrukcji, jak liczyć makra w C?
- Ale zwraca bardzo czytelny i łatwy do zrozumienia komunikat dla progrmisty.

Control flow graph

Podstawowy blok

Sekwencja instrukcji, posiadająca tylko jedno wejście (pierwsza wykonywana instrukcja) i jedno wyjście (ostatnia wykonywana instrukcja). Dobrze brać bloki maksymalnej długości.

Control flow graph

To graf skierowany, którego wierzchołkami są bloki programu a krawędzie obrazują możliwe przejścia pomiędzy nimi.

Control flow graph

Podstawowy blok

Sekwencja instrukcji, posiadająca tylko jedno wejście (pierwsza wykonywana instrukcja) i jedno wyjście (ostatnia wykonywana instrukcja). Dobrze brać bloki maksymalnej długości.

Control flow graph

To graf skierowany, którego wierzchołkami są bloki programu a krawędzie obrazują możliwe przejścia pomiędzy nimi.

Przykład

Pythonpycfg `https://www.geeksforgeeks.org/draw-control-flow-graph-using-pycfg-python/`

Miary pokrycia testami oparte na “control flow graph”.

Pokrycie stanów

Procent/zbiór bloków przez, które przechodzą testy.

Pokrycie krawędzi

Procent/zbiór krawędzi przez, które przechodzą testy.

Przykład

Python: Coverage `https://pypi.org/project/coverage/`

Pokrycie ścieżkami

Wszystkie ścieżki w grafie są pokryte (może ich być nieskończenie wiele, wtedy nie da się spełnić kryterium).

Miary pokrycia testami oparte na przepływie danych.

Wystąpienia zmiennych

Wyróżniamy 3 sytuacje, w których występuje zmienna:

1. przypisanie do zmiennej,
2. użycie w obliczeniu,
3. użycie w warunku.

Zakładamy, że wartość zmiennej nie zmienia się jeśli jej nie używamy.

Czysta ścieżka dla zmiennej x

To ścieżka w grafie sterowania, która zaczyna się w wierzchołku, w którym przypisujemy wartość do x kończy w wierzchołku, w którym używamy x. Pomiędzy przypisaniem a użyciem nie następuje inne przypisanie do zmiennej x. Wyróżniamy czyste ścieżki zakończone obliczeniem i czyste ścieżki zakończone warunkiem.

Miary pokrycia testami oparte na przepływie danych.

Miary

1. Dla każdej pary przypisanie i użycie w (obliczeniu/warunku) takiej, że istnieje pomiędzy nimi czysta ścieżka, istnieje test, który pokrywa stan, w którym następuje przypisanie i stan, w którym nastąpiło użycie.
2. all-du-test pokrywa wszystkie ścieżki czyste.

Mutacje

Koncepcja

Wprowadzam sztuczne pojedyncze błędy do kodu i sprawdzam czy moje testy je wykrywają.

Założenia metody

- Założenie kompetentnego programisty - kod napisany przez programistę robi prawie to co miał robić, albo inaczej to co robimy ma sens.
- coupling (sprzeżenie) effect - jeśli testy wykrywają proste błędy, to powinny też wykrywać bardziej skomplikowane.

Jak wprowadzać błędy?

Definujemy sposoby na mutowanie programu. I aplikujemy mutacje aby otrzymać zestaw programów.

Mutacje 2

Przykłady

1. W if-ach zmieniamy == na =, albo odwracamy nierówności.
2. zamieniamy warunek na true lub false
3. podmieniamy zmienną na inną o tym samym typie
4. W pętlach dodajemy 1 do zakresu, albo odejmujemy jeden od zakresu.
5. mutacje zależą od języka programowania.

Narzędzia

- Na wikipedi jest długa lista.
- do pythona np. mutmut, zaraz się nim pobawimy

Mutacje 3. Problemy!

Część mutacji przechodzi testy?

Możliwości są dwie:

1. testy są za mało dokładne,
2. mutacje tworzą programy równoważne.

Sprawdzanie równoważności programów jest nierozstrzygalne. A zatem trzeba ręcznie przejrzeć kod mutantów, dla których testy przechodzą i zrozumieć czy coś jest nie tak z testami, czy mutant robi to samo co program?

Wydajność

Załóżmy że wykonanie testów zajmuje minutę. Generator mutantów produkuje 500 mutantów.

- Odpalenie testów zajmie 500 minut $\sim$ 8.5 godziny.
- Okaże się, że 120 mutantów przechodzi testy.
- Ile czasu mozna poświęcić na ich przejrzenie i sprawdzenie?

Fatal Error!