Testowanie

Jakość testów

Plan

- Jakość kodu, a jakość przypadków testowych,
- Analiza pokrycia kodu testami,
- Testowanie mutacyjne.

Jakość kodu/testów

- Jakość kodu:
  - Czy program robi to co powinien?
  - Czy kod jest dobrze napisany?
- Jakość testu:
  - Czy test testuje to, do czego został przeznaczony?
  - Czy test testuje dobrze i efektywnie?
  - Czy kod testu jest dobrze napisany?

Jakość kodu/testów

- Jakość zbioru testów:
  - Jaki zakres kodu testuje zbiór testów?

Jakość kodu/testów

- Przypadek testowy – np. napisany w JUnit – jest także kawałkiem kodu,
- Techniczną jakość kodu produkcyjnego jak i testowego można badać tymi samymi metodami – np. stosując metryki kodu.

Jakość testu

- Nie jest możliwe automatyczne sprawdzenie, czy test sprawdza to do czego został zaprojektowany,
- Można sprawdzić efektywność testu za pomocą techniki testowania mutacyjnego.

Jakość zbioru testów

- Nie jest możliwe automatyczne sprawdzenie, czy zbiór testów sprawdza odpowiedni zakres funkcjonalności (np. wszystkie kluczowe lub krytyczne funkcje),
- Można sprawdzić jakie obszary kodu są pokryte testami.

Pokrycie kodu

- Ile kodu jest pokryte przez testy?
- Modele pokrycia kodu:
  - pokrycie instrukcji,
  - pokrycie gałęzi,
  - pokrycie instrukcji bytecode'u,
  - pokrycie metod,
  - pokrycie typów, …

Pokrycie kodu

- Nie zawsze jest możliwe uzyskanie pokrycia 100% (np. martwy kod),
- 100% pokrycie kodu nie gwarantuje tego, że testowany kod jest bezbłędny.

Pokrycie kodu

- EclEmma – plugin do IDE Eclipse,
- Pozwala analizować kod pod kątem kilku modeli pokrycia,
- Udostępnia osobny widok z obliczonymi statystykami.

Pokrycie kodu

- Pozwala oznaczać linie kodu w zależności od stopnia pokrycia,
- Udostępnia szereg dodatkowych opcji ułatwiających analizę.

Testowanie mutacyjne

- Sprawdzenie efektywności testu,
- Testowanie mutacyjne polega na wprowadzaniu do testowanego kodu niewielkich modyfikacji i ponownym wykonaniu testów.

Testowanie mutacyjne

- Test jest efektywny jeśli wykrywa mutację,
- Test potencjalnie wymaga poprawy jeśli nie wykrywa mutacji.

Testowanie mutacyjne

- Przykład: niech będzie dana funkcja dodająca dwa argumenty, np. a1 + a2;
- Test: a1 == 2 i a2 == 2, wynik oczekiwany i uzyskany jest równy 4.

Testowanie mutacyjne

- Mutacja: a1 * a2,
- Test dalej przechodzi poprawnie,
- Konkluzja: konieczna jest zmiana danych testowych lub dopisanie kolejnego przypadku testowego.

Operatory mutacji

- Relacyjne, np.: z > na >=
- Negacje, np.: z == na !=
- Usuwanie warunków, np.: z if (a==b) {} na if (true) {}
- Matematyczne, np.: z + na –
- Inkrementacyjne, np.: z ++ na --

Operatory mutacyjne

- Zamiana znaku, np..: +1 na -1
- Zamiana wartości stałej,
- Zamiana wartości zwracanej z metody,
- Usuwanie wywołań funkcji i procedur,
- Usuwanie wywołań konstruktorów,
- …

Narzędzia do mutacji

- PIT,
- Narzędzie konsolowe,
- Integracja z Maven, ant,
- Integracja z IDE Eclipse, IntelliJ,
- Działa na poziomie bytecode.

Narzędzia do mutacji

- Testy są uruchamiane automatycznie na zmutowanych klasach,
- PIT wykrywa stany mutantów: trafiony, nietrafiony, bez pokrycia, timeout, niepoprawny dla VM, błąd pamięciowy, błąd uruchomienia,
- Raport z wykonania.

Podsumowanie

- Jakość kodu, a jakość testów,
- Wady i zalety pokrycia kodu,
- Ocena efektywności testów,
- http://www.eclemma.org/index.html
- http://pitest.org/

Q&A