Testowanie oprogramowania

Sztuka Wytwarzania Oprogramowania, w. 10

Konrad Grochowski

Testowanie oprogramowania

- Ta część analizy, w której rzeczywiste wyniki zderza się z *założonymi* oczekiwaniami
  - Uwaga – moda na LLMowe generowanie testów jest ekstremalnie niebezpieczna

Testowanie oprogramowania

- Z punktu widzenia klienta – najważniejsza część procesu wytwarzania oprogramowania
  - Nawet jeśli klient nie zdaje sobie z tego sprawy...
  - Może warto nazywać to *zapewnianiem jakości*?
- Weryfikuje zgodność programu z oczekiwaniami
- *Nic innego* nie daje informacji, *co* program *naprawdę* robi
  - Istnieją metody weryfikacji formalnej, jednak nie zawsze daje się je zastosować, są drogie, i często mogą być użyte do weryfikacji fragmentów „odseparowanych od świata zewnętrznego”.
  - Modelowanie może pomagać, ale zależy od procesu (czy model jest *przetwarzany* czy *przepisywany*) ale i jakości samego modelu

Testowanie oprogramowania – dwa użycia

- Oddzielny element cyklu, dedykowana czynność
  - Czasem zwany „walidacją”, „kwalifikacją”, „certyfikacją” etc.
- Integralny element „implementacji” / codziennej pracy
  - Nie można ocenić wykonanej pracy bez jej przetestowania
- Oba mają sens
  - Ale uwaga na nieporozumienia

Testowanie oprogramowania

- Testowanie może tylko zweryfikować,
  czy nie ma ***znanych błędów***
- Testy robi się *na coś* i tak naprawdę *„zielone testy”* to *„negatywne wyniki”*
- Czyli testowanie nie daje 100% gwarancji poprawności działania
- Ale to nie znaczy, że klient ma testować „na produkcji”...

Testowanie oprogramowania

- Rodzajów testów oprogramowania jest bardzo dużo
- W pewnym sensie każde uruchomienie programu i obserwacja jego zachowania to test
  - Tylko najlepiej, żeby klient obserwował tylko oczekiwane działanie...
- Istnieją dedykowani specjaliści zajmujący aspektami testowania
  - Product Assurance / Quality Assurance
    (choć w tych terminach jest więcej, niż tylko testy)
- Testowanie nie musi dotyczyć wyłącznie kodu
  - Np. zgodność instrukcji użytkownika z rzeczywistym oprogramowaniem

Podział testów

- Ze względu na to czy program jest fizycznie uruchamiany:
  - Statyczne
  - Dynamiczne
- Ze względu na obiekty objęte testem (poziom testu):
  - Jednostkowe
  - Integracyjne
  - Systemowe
  - *Akceptacyjne*
- Testy bezpośrednio weryfikujące wymagania czasem nazywa się *testami walidacyjnymi*

Podział testów

- Ze względu na podejście do obserwacji wyniku:
  - White-box
  - Black-box
  - Grey-box
- Ze względu na sposób przeprowadzenia:
  - Automatyczne
  - Manualne

Podział testów

- Ze względu na badane aspekty:
  - Testy funkcjonalne (functional testing)
  - Testy wydajnościowe (performance testing)
  - Testy przeciążeniowe (stress testing)
  - Testy bezpieczeństwa (security testing)
  - Testy bezpieczeństwa (safety testing)
  - Testy niezawodności (dependability testing)
  - Testy odporności (recovery testing)
  - Testy zgodności (compatibility testing)
  - Testy dokumentacji (documentation testing)
  - ...

Inne przykłady rodzajów testów

- Testy dymne (smoke tests)
- Testy regresji (regression testing)
- Testy destrukcyjne (destructive testing)
- Testy „interakcji z człowiekiem”
  - Testy tłumaczeń i regionalizacji (localization and internationalization)
  - Testy używalności (usability)
  - Testy dostępności (accessibility)

Dobry test

- Jednoznacznie ustalony cel testu
- Czytelny scenariusz testu
- Określony warunek końca testu
- Określony warunek oceny testu (zaakceptowania)
- Jeśli możliwe – odwołanie do wymagań, z których test wynika.
- *Idealnie:*
  - Automatyczny
  - Deterministyczny i powtarzalny
  - Jednoznacznie i łatwo oceniany
  - Szybki

Testowanie jednostkowe

- „Najwyższa stopa zwrotu w jakości”
  - Krótka ścieżka pomiędzy inwestycją/czasem a wynikiem i wpływem na kod
- „Najlepszy przyjaciel programisty”
  - Tylko test mówi, co kod naprawdę robi, więc test jednostkowy pozwala wiedzieć co jest zrobione „na koniec dnia”
- Wymusza lepszą architekturę (decoupling)
- Brak formalnej definicji jednostki
- Nie wszystko da się unit-testować
- Nie zawsze odpowiada bezpośrednio wymaganiom
  - Usunie drobne błędy, ale nie poważne naruszenia funkcjonalności

Najlepsze praktyki tworzenia testu (jednostkowego)

- Automatyczny
- Powtarzalny i deterministyczny
- Szybki
- Jednoznaczny
- Skoncentrowany na pojedynczym aspekcie
- Czytelny (testy to najlepsza dokumentacja kodu)
- Niezależny

- *Nieidealny* test *zazwyczaj* jest lepszy niż brak testu
- *Zły* test obniża morale i często *przeszkadza*

Dziękuję za uwagę

Konrad.Grochowski@pw.edu.pl