Inżynieria oprogramowania 1

Testowanie dynamiczne, część druga

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska

Kielce, 23 stycznia 2025

Plan

1. Wprowadzenie
2. Testowanie kodu obiektowego
3. Metryki
   - Metryki pokrycia kodu
   - Metryki pokrycia wymagań
   - Inne metryki

Motto

„Debugowanie jest dwukrotnie trudniejsze niż pisanie kodu. Jeśli więc piszesz kod najinteligentniej jak potrafisz, to z definicji nie jesteś wystarczająco inteligentny by go debugować.”

— Brian W. Kernighan

„Zawsze pisz kod tak, jakby gość, który ma się nim zajmować, był agresywnym psychopatą, który wie, gdzie mieszkasz.”

— Martin Golding

„Mówienie komuś, że nie ma racji nazywa się krytyką. Czynienie tego publicznie nazywa się testowaniem.”

— Gaurav Khurana

Wprowadzenie

Opisane na poprzednim wykładzie poziomy testowania zostały opracowane w czasach, kiedy dominującym paradygmatem było programowanie proceduralne. W przypadku programowania obiektowego wymagają one niewielkiego dostosowania, które będzie pierwszym zagadnieniem poruszanym w ramach tego wykład. Kolejnym będą metryki używane w testowaniu. Pozwalają one ocenić jakość przeprowadzanych testów oraz oprogramowania, które tym testom podlega.

Testowanie kodu obiektowego

Kod obiektowy różni się od kodu proceduralnego w kliku aspektach:

- Obiekt, jako niezależny komponent oprogramowania, zazwyczaj jest większy niż pojedynczy podprogram.
- Obiekty zintegrowane w podsystem są zazwyczaj luźno powiązane, nie mają wspólnego „punktu wejścia.”
- Jeśli obiekty są ponownie używane, to testerzy mogą nie mieć dostępu do kodu źródłowego ich klas i w związku z tym nie mogą go przeanalizować.

Testowanie kodu obiektowego

Poziomy testowania

W testowaniu kodu obiektowego można wyróżnić następujące poziomy:

testowanie jednostkowe Jeśli to możliwe, to metody obiektu powinny być testowane osobno. Można w tym celu zastosować zarówno testowanie funkcjonalne jak i strukturalne.

testowanie klas To jest poziom nadrzędny testowania jednostkowego, w którym sprawdzane są sekwencje wzajemnie zależnych operacji obiektów. Obie metody testowania mogą być w tym celu stosowane.

testy integracyjne Na tym poziomie powinny być testowane grona wspólnie używanych obiektów. Testy te powinny być oparte na scenariuszach.

testowanie systemowe Sposób testowania oprogramowania obiektowego na tym poziomie jest taki sam, jak w przypadku kodu proceduralnego.

testowanie akceptacyjne Również na tym poziomie nie ma zmian.

Należy zauważyć, że granica między testowaniem jednostkowym, a integracyjnym w przypadku oprogramowania obiektowego jest rozmyta.

Testowanie kodu obiektowego

Testowanie klas

Są trzy cele *testowania klas*:

1. Weryfikacja metod obiektów w izolacji — jest to podstawowe wymaganie testowania jednostkowego.
2. Weryfikacja metod zmieniających i odczytujących (ang. *setters and getters*) wszystkie atrybut obiektu — konieczna tylko w niektórych przypadkach.
3. Weryfikacja wszystkich możliwych stanów obiektów — wszystkie zdarzenia skutkujące zmianą stanu obiektów powinny być sprawdzone.

Testowanie kodu obiektowego

Testowanie klas

Załóżmy, że klasa implementująca następujący interfejs musi zostać poddana testom:

```
interface LightControl {
    void switchOn();
    void makeBrighter();
    void makeDimmer();
    void switchOff();
}
```

Testowanie kodu obiektowego

Testowanie klas

Aby sprawdzić wszystkie możliwe stany obiektu tej klasy należy poddać testom następujące sekwencje wywołania jego metod:

`switchOn()` → `switchOff()`
`switchOn()` → `makeBrighter()` → `switchOff()`
`switchOn()` → `makeDimmer()` → `switchOff()`
`switchOn()` → `makeBrighter()` → `makeDimmer()` → `switchOff()`
`switchOn()` → `makeDimmer()` → `makeBrighter()` → `switchOff()`

Testowanie kodu obiektowego

Testy integracyjne

Następujące metody mogą zostać wykorzystane w testach integracyjnych kodu obiektowego:

Testowanie przypadków użycia lub scenariuszy Przypadki użycia lub scenariusze opisują w jaki sposób oprogramowanie będzie używane, zatem przypadki testowe mogą być opracowane na ich podstawie.

Testowanie wątków Oprogramowanie obiektowe zazwyczaj jest sterowane zdarzeniami (ang. *event-driven*). Ta metoda zakłada sprawdzenie reakcji oprogramowania na zdarzenia. Testerzy muszą wiedzieć jak wewnątrz oprogramowania są przetwarzane zdarzania, aby je przetestować dokładnie.

Testowanie interakcji obiektów Jest to metoda podobna do testowania wątków. Różnica polega na tym, że przypadki testowe powinny weryfikować ścieżki *metoda* — *komunikat*, które rozpoczynają się zdarzeniem wejściowym i kończą zdarzeniem wyjściowym.

Metryki

Testowanie dynamiczne wymaga wielu zasobów i jest kosztowne. Aby ocenić jego postęp i jakość opracowano specjalne *metryki*. Niektóre z nich służą także do szacowania jakości testowanego oprogramowania. Część z nich została zaprojektowana na potrzeby określonej metody testowania lub nawet poziomu testów.

Metryki pokrycia kodu

Metryki *pokrycia kodu* określają ile testowanego kodu jest uruchamiane w wyniku wykonania przypadków testowych. Powszechnie zaleca się, aby podczas testowania zostało wykonane około 80% sprawdzanego kodu. Ta liczba jest jednak dobrana arbitralnie. Ustalenie poziomu pokrycia kodu powinno być osadzone w kontekście. W przypadku oprogramowania o zastosowaniach krytycznych może być wyższe, podczas gdy w pozostałych przypadkach może być niższe. Metryki pokrycia kodu pozwalają testerom wykryć nadmiarowe przypadki testowe, czyli takie, które nie wykrywają nowych defektów. Istnieją trzy takie metryki:

- metryka pokrycia bloków instrukcji (ang. *block coverage metric*),
- metryka pokrycia decyzji (ang. *decision coverage metric*),
- metryka pokrycia ścieżek (ang. *path coverage metric*).

Metryki pokrycia kodu

Metryka pokrycia bloków instrukcji

Blok instrukcji jest ciągiem instrukcji sekwencyjnych, które w przeciwieństwie np. do pętli i instrukcji warunkowych nie zmieniają przepływu sterowania. Metryka *pokrycia bloków instrukcji* to liczba sprawdzonych bloków do całkowitej ich liczby w testowanym komponencie. Szczególnym przypadkiem tej metryki jest *metryka pokrycia wierszy*, w której bloki zawierają jeden wiersz kodu. Te metryki są łatwe do zrozumienia i mogą być stosowane zarówno na poziomie kodu źródłowego, jak i wykonywalnego. Nie pozwalają one jednak ocenić na ile dobrze zostały sprawdzone wyrażenia warunkowe w instrukcjach zmieniających przepływ sterowania.

Metryki pokrycia kodu

Metryka pokrycia bloków instrukcji — Przykład

```
if(a==3) {
    function_1(...);
} else {
    function_2(...);
}
```

Zaprezentowany fragment kodu wymaga tylko dwóch przypadków testowych, aby w metryce pokrycia bloków instrukcji uzyskać 100% pokrycia kodu — po jednym na każdą gałąź instrukcji warunkowej.

Metryki pokrycia kodu

Metryki pokrycia kodu — Przykład

```
int a=-2;
if(b>0)
    a=1;
double x=sqrt(a);
```

Przedstawiony fragment kodu w metryce pokrycia bloków instrukcji wymaga tylko jednego przypadku testowego, aby uzyskać 100% pokrycia kodu — dla `b` większego od `0`. Jednakże ten przypadek testowy nie pozwoli wykryć poważnego defektu, który ujawnia się, gdy $b \leqslant 0$.

Metryki pokrycia kodu

Metryka pokrycia decyzji

Metryka *pokrycia decyzji* pozwala ocenić na ile dobrze zostały przetestowane instrukcje sterujące. Jest ona zdefiniowana jako stosunek liczby przetestowanych gałęzi instrukcji sterujących do całkowitej liczby tych gałęzi w kodzie. Jej zaletą jest równie dobra weryfikacja pokrycia bloków instrukcji jak we wcześniej omówionych metrykach i lepsze sprawdzenie pokrycia wyrażeń warunkowych w instrukcjach sterujących. Do jej wad zalicza się konieczność opracowania większej liczby przypadków testowych oraz to, że nie uwzględnia ona skracania obliczeń wartości wyrażeń warunkowych (ang. *short-circuit evaluation*).

Metryki pokrycia kodu

Metryka pokrycia decyzji — Przykład

```
if(a>0&&(b<0 || function(a,3)>0)) {
        statement_1;
} else {
        statement_2;
}
```

Zgodnie z metryką pokrycia decyzji, zaprezentowany fragment kodu może być w pełni przetestowany przy użyciu tylko dwóch przypadków testowych. Niestety, nie muszą one spowodować wywołania funkcji w wyrażeniu warunkowym.

Metryki pokrycia kodu

Metryka pokrycia ścieżek

Metryka *pokrycia ścieżek* jest powiązana ze ścieżkami pierwotnymi, przedstawionymi na poprzednim wykładzie. Uwzględnia ona skracanie obliczeń wartości wyrażeń warunkowych, ale wymaga opracowania jeszcze większej liczby przypadków testowych niż metryka pokrycia decyzji. Jeśli testowany komponent oprogramowania zawiera wiele pętli, to przypadki testowe powinny być zdefiniowane tak, aby tylko ograniczona liczba ich iteracji była wykonywana.

Metryki pokrycia wymagań

Metryki *pokrycia wymagań* stosowane są wraz z metodą testowania funkcjonalnego na poziomie testów systemowych i akceptacyjnych. Są one używane do sprawdzania na ile dobrze weryfikowane są w testach wymagania określone dla badanego oprogramowania. Ogólnie można przyjąć, że te metryki mierzą stosunek liczby przetestowanych wymagań do całkowitej ich liczby. Jednakże wymagania mogą być zdefiniowane w różny sposób, w zależności od wybranej metody ich określania. Sprawdzenie niektórych z nich może wymagać użycia więcej niż jednego przypadku testowego.

Metryki pokrycia wymagań

Metryka pokrycia błędów

Metryka *pokrycia błędów* (ang. *error coverage metric*) jest zdefiniowana jako stosunek liczby błędów obsłużonych przez oprogramowanie w trakcie testów, do całkowitej liczby błędów jakie to oprogramowanie powinno obsługiwać. Błędy w tym wypadku oznaczają wyjątki, takie jak nieodpowiedni format danych, brak połączenia z siecią lub brakujące inne zasoby.

Metryki pokrycia wymagań

Metryk pokrycia przypadków użycia

Przypadek użycia jest zbiorem scenariuszy, które opisują typowe interakcje z oprogramowaniem. Istnieją dwie kategorie takich przypadków: *biznesowe*, które związane są z wysokopoziomowymi wymaganiami i *systemowe*, które określają techniczne szczegóły procesów zachodzących w oprogramowaniu. Można opracować metryki dla obu typów przypadków.

Inne metryki

W tej części wykładu zostaną przedstawione metryki, których nie można zaliczyć do wcześniejszych kategorii. Mogą one być stosowane do jednoczesnej oceny jakości kodu i testów.

Inne metryki

Liczba wykrytych defektów

Liczba wykrytych defektów (ang. *number of detected defects*) lub *skumulowana liczba wykrytych defektów* (ang. *cumulated number of detected defects*) pozwala oszacować początkową jakość kodu i jej przyrost w kolejnych sesjach testowania i usuwania defektów. Używając jej inżynierowie kontroli jakości mogą ocenić również efektywność metod testowania, które stosują i technik poprawy defektów używanych przez programistów. Testowanie można zakończyć, kiedy liczba wykrytych defektów się stabilizuje.

Inne metryki

Liczba defektów komponentów

Liczba defektów komponentów jest po prostu liczbą defektów odkrytych w testowanym komponencie oprogramowania. Używa się jej do szacowania jakości wszystkich komponentów. Choć ta metryka jest łatwa do obliczenia, to jej interpretacja zależy od wielu czynników. Komponent z wieloma defektami może być bardziej złożony niż pozostałe lub opracowany przez zespół programistów przeciążonych pracą. Wysoka wartość tej metryki może także oznaczać, że komponent został przetestowany przy użyciu lepszej jakości przypadków testowych niż pozostałe komponenty. Te przypadki mogą zatem być używane w fazie utrzymania oprogramowania, celem wykrywania regresji.

Inne metryki

Gęstość defektów

Metryka *gęstości defektów* (ang. *density of defect*) jest zdefiniowana jako liczba defektów przypadających na liczbę wierszy lub tysiąc wierszy (ang. *kilolines of code*). Pozwala ona ocenić przyrost jakości testowanego oprogramowania w stosunku do przyrostu liczby wierszy kodu. Mimo, że nie ma ogólnie przyjętej definicji wiersza kodu, to ta metryka jest powszechnie stosowana.

Inne metryki

Procent wykrytych defektów

Procent wykrytych defektów (ang. *percentage of detected defects*) używany jest do oceny efektywności testowania. Jest on zdefiniowany jako stosunek liczby wykrytych w testach defektów do szacowanej całkowitej liczby defektów w testowanym oprogramowaniu. Aby oszacować tę drugą wartość wykonuje się *zasiewanie* (ang. *seeded*) defektów tj. sztucznie wprowadza się je do weryfikowanego oprogramowania. Całkowita liczba defektów jest szacowana przy użyciu następującego wyrażenia: $N_t = N_s \cdot \frac{n_t}{n_s}$, gdzie N_s jest całkowitą liczbą zasianych defektów, N_t jest szacowaną całkowitą liczbą prawdziwych defektów, n_t jest liczbą prawdziwych defektów wykrytych w testach, a n_s jest liczbą zasianych defektów znalezionych w testowaniu. Im bardziej typy zasianych defektów odpowiadają typom prawdziwych defektów, tym bardziej wiarygodna jest ta metryka.

Metryki

Podsumowanie

W inżynierii oprogramowania opracowano znacznie więcej metryk, niż to zaprezentowano w ramach tego wykładu. Jednakże zastosowanie tych omówionych powinno być wystarczające do oszacowania jakości oprogramowania i testów.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!