Dr Katarzyna Grzesiak-Kopeć

Inżynieria oprogramowania

9.1 Wprowadzenie do testowania

Plan wykładu

- Tworzenie oprogramowania
- Najlepsze praktyki IO
- Inżynieria wymagań
- Technologia obiektowa i język UML
- Techniki IO
- Metodyki zwinne
- Refaktoryzacja
- Mierzenie oprogramowania
- Jakość oprogramowania
- Programowanie strukturalne
- Modelowanie analityczne
- Wprowadzenie do testowania

10 Worst Software Bugs

Wired News

- 28.07.1962: Mariner I
  - Kropka zamiast przecinka FORTRAN DO-Loop
- 1982: Soviet gas pipeline
  - Największa nienuklearna eksplozja w historii planety
- 1985-1987: Therac-25
  - Co najmniej 5 ofiar śmiertelnych
  - Wiele osób poważny uszczerbek na zdrowiu

10 Worst Software Bugs

- 1988: Buffer overflow in Berkeley Unix finger daemon
  - 1 internetowy robak (Morris Worm)
- 1988-1996: Kerberos Random Number Generator
  - Przez 8 lat trywialne jest złamanie zabezpieczeń każdego komputera korzystającego z tego protokołu uwierzytelniania i autoryzacji

10 Worst Software Bugs

- 15.01.1990: AT&T Network Outage
  - NY, 114 switchów pada i wstaje co 6 sekund, ~60 000 ludzi nie ma dostępu do połączeń zamiejscowych przez 9h
- 1993: Intel Pentium floating point divide
  - Od 3 do 5 milionów urządzeń w obiegu, kosztował Intel $475 000 000
- 1995/1996: The Ping of Death

10 Worst Software Bugs

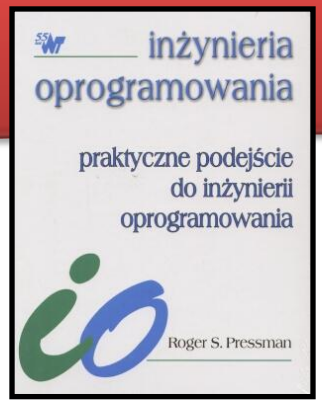
Wired News

- 4.06.1996: Ariane 5 Flight 501
  - Konwersja 64-bit floating point do 16-bit signed integer
  - Budowa $7 000 000 000, Straty $500 000 000
  - Brak ubezpieczenia
- listopad 2000: National Cancer Institute, Panama City
  - Co najmniej 8 ofiar śmiertelnych
  - ~20 osób naświetlonych
  - Wyrok skazujący za morderstwo

Definicje

- Pomyłka, błąd (error)
  - Działanie człowieka
- Defekt, usterka (defect, fault, bug)
  - Skutek błędu programisty
  - Może spowodować awarię
- Awaria (failure)
  - Odchylenie od spodziewanego działania/wyniku

Weryfikacja i walidacja



- Weryfikacja
  - Czy budujemy produkt tak, jak trzeba?
  - Czy oprogramowanie poprawnie realizuje wyspecyfikowaną funkcję?
  - Zgodność z dokumentacją
- Walidacja
  - Czy budujemy ten produkt, co trzeba?
  - Porównanie działania gotowego oprogramowania z wymaganiami klientów
  - Zgodność z oczekiwaniami klienta

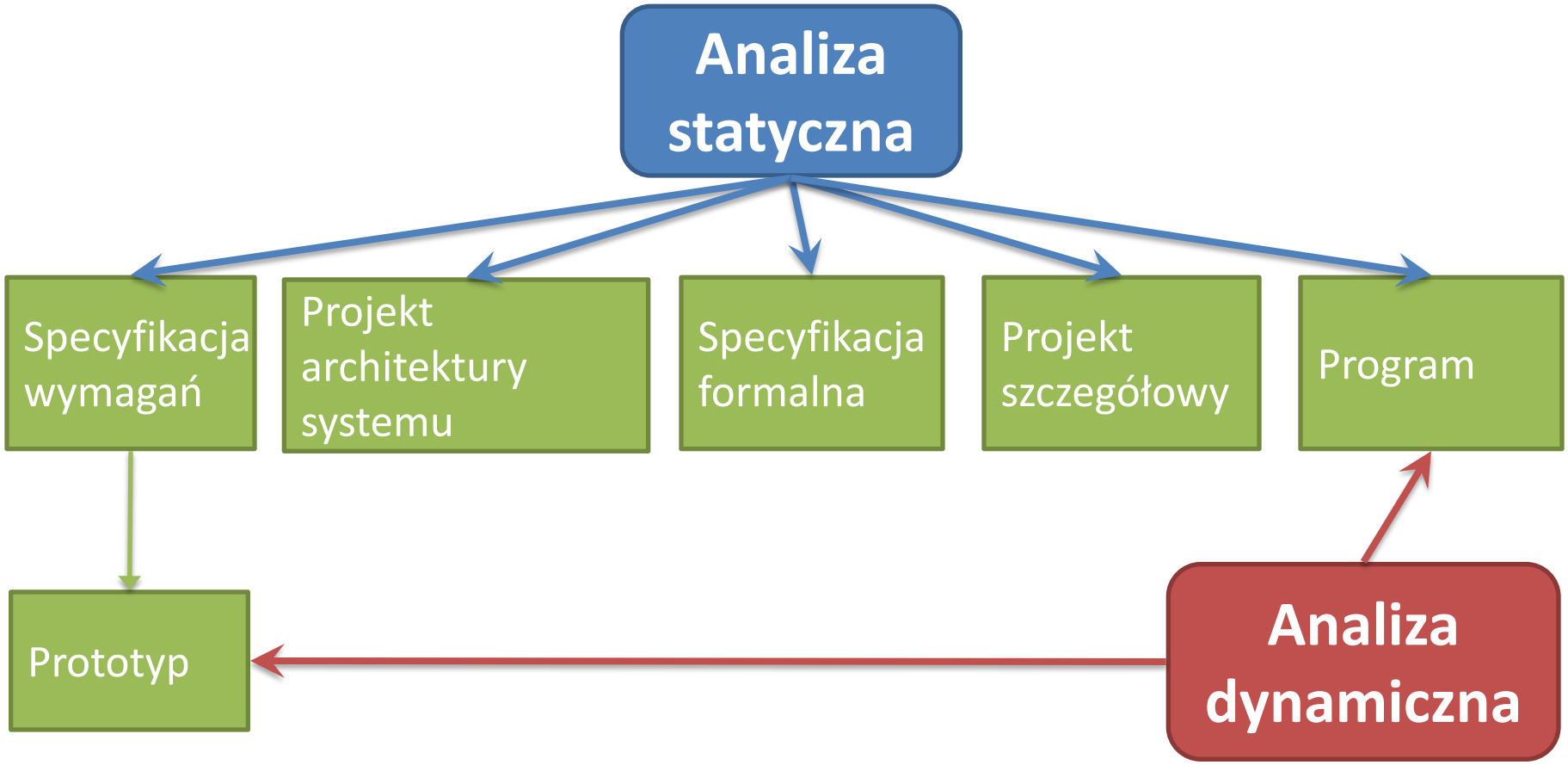
Weryfikacja i walidacja

- Powinny być wykonywane w każdym kroku procesu tworzenia oprogramowania
- Zaczynają się od przeglądów wymagań, poprzez przeglądy projektów i kontrolę kodu, a kończą się testowaniem produktu

Analiza statyczna i dynamiczna

- Statyczna – inspekcje kodu
  - Praca z kodem źródłowym
- Dynamiczna – testowanie oprogramowania
  - Eksperymentowanie z działającym kodem programu

Analiza statyczna i dynamiczna



Weryfikacja i walidacja

- Całkowita (pełna) weryfikacja i walidacja
  - Testowanie
  - Statyczna weryfikacja
- Testowanie to jedyna technika walidacji wymagań niefunkcjonalnych

Cele testowania

- Testowanie polega na

  uruchamianiu programu w celu wykrycia błędów

- Dobry test to taki, który

  z dużym prawdopodobieństwem pozwala znaleźć błąd wcześniej nie wykryty

- Test można uznać za skuteczny, jeżeli

  doprowadził do wykrycia nowych błędów

Zasady testowania

- Buduj testy na podstawie wymagań klienta
- Planuj!
- Zasada Pareto (80/20)
- Testuj bottom-up
- Gruntowne testowanie jest niewykonalne
- Powierz testowanie niezależnym osobom

!Gruntownie » Poziom zaufania

- Funkcja oprogramowania
  - Jak bardzo krytyczny jest system dla firmy
- Oczekiwania użytkowników
  - Jak wiele potrafi znieść użytkownik ☺
- Rynek
  - Lepiej wypuścić szybciej z błędami

Dobry test

- Daje duże prawdopodobieństwo wykrycia błędu
- Testy nie dublują się
- Wybieramy testy „najlepsze w swoim rodzaju”
- Ani zbyt prosty, ani zbyt skomplikowany

Poziomy testowania

- Testowanie jednostkowe
  - Pojedynczych funkcji lub innych fragmentów kodu w oderwaniu od reszty systemu
- Testowanie integracyjne
  - Przetestowane jednostki kodu są stopniowo łączone w większą całość, a następnie ponownie testowane już jako grupa jednostek, proces łączenia i testowania jest powtarzany aż do powstania całego systemu

Poziomy testowania

- Testowanie systemowe
  - Czy system jako całość spełnia wymagania funkcjonalne i jakościowe postawione przez klienta
- Testy akceptacyjne
  - Testy w środowisku docelowym

Typy testów

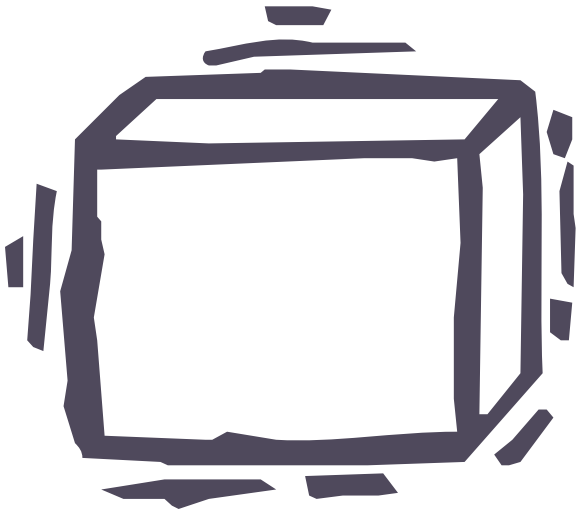
- Funkcjonalne (F)
  - Co robi?
- Niefunkcjonalne (NF)
  - Jak działa?
- Strukturalne (S)
  - Jaka jest architektura?
- Regresywne (R)
  - Czy nadal działa?

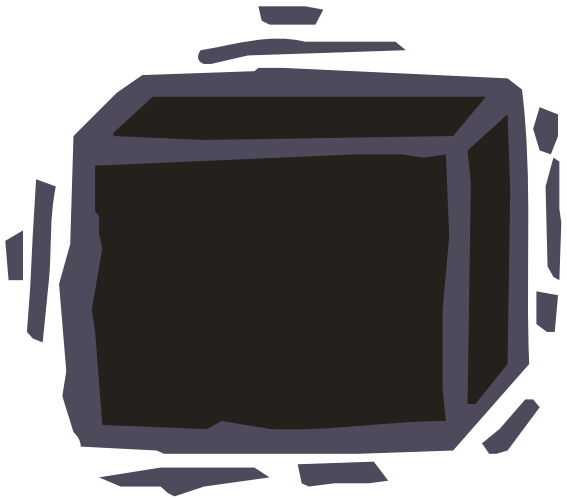
Poziomy a typy testów

F – funkcjonalne
W – wydajnościowe
R – regresywne
B – bezpieczeństwa
Ins – instalacyjne
Int - interfejsu

Projektowanie testów

- Metoda białej skrzynki
  - Logiczna struktura algorytmów
- Metoda czarnej skrzynki
  - Badanie interfejsów, funkcjonalności





Metoda białej skrzynki

- Metoda szklanej skrzynki, testowanie strukturalne
- Projektowanie testów na podstawie logicznej struktury algorytmu
  - Wszystkie niezależne ścieżki przepływu
  - Wszystkie możliwe kombinacje decyzji logicznych
  - Jedno i wielokrotne wykonanie pętli z uwzględnieniem warunków brzegowych

Po co biała skrzynka?

- Literówki są rozmieszczone losowo
- Liczba błędów logicznych i błędnych założeń jest odwrotnie proporcjonalna do częstości wykonywania danego fragmentu kodu
- To, co w założeniu miało być rzadko, w rzeczywistości jest bardzo często wykonywane

Biała skrzynka 100% OK?

- Gruntowne testowanie
  - Identyfikacja wszystkich ścieżek logicznych
  - Sprawdzenie działania każdej z nich
  - Ocena ich poprawności
- Ale…

Przykład: 100 wierszy kodu

- Deklaracja danych
- 2 zagnieżdżone pętle od 1 do 20 zależnie od wejścia
- W wewnętrznej pętli program sprawdza 4 warunki
- Ile jest możliwych ścieżek?

$\sim 10^{14}$

10^{14}

- Załóżmy, że istnieje procesor, który może przygotować test, wykonać go i ocenić wynik w ciągu 1 milisekundy
- Pracując 24h/dobę przez 365 dni w roku
- Potrzebowałby do przetestowania naszego przykładu…

3170 lat

Pokrycie kodu

- Ile kodu jest pokryte przez testy?
- Pokrycie instrukcji (ang. statement coverage)
  - Każda instrukcja jest sprawdzana
- Pokrycie gałęzi (ang. branch coverage)
- Każda gałąź była odwiedzona
  - Instrukcja warunkowa musi być przynajmniej raz prawdziwa i przynajmniej raz fałszywa

Pokrycie instrukcji

```
void myFunction(int number) {
        while(liczba<10)
                liczba++;
        if(10==liczba)
                System.out.println(liczba);
}
```

number=3

Pokrycie gałęzi

```
void myFunction(int number) {
        while(liczba<10)
                liczba++;
        if(10==liczba)
                System.out.println(liczba);
}
```

number=3, number=13

100 % Pokrycia kodu ☹

```
int add(int no1, int no2) {
        return no1-no2;
}
```

no1=3, no2=0

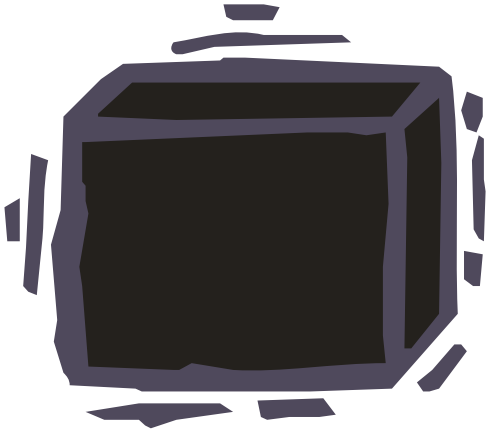
100 % Pokrycia kodu ☹

```
int add(int no1, int no2) {
        return no1-no2;
        System.out.println(„Martwy kod”);
}
```

Nie zawsze 100% jest możliwe!

Metoda czarnej skrzynki

- Testowanie zachowania (behawioralne), funkcjonalne
- Badanie interfejsów
- Nie zwracamy uwagi na wewnętrzną strukturę logiczną



Cel to wykrycie…

- Pominiętych lub błędnie zaimplementowanych funkcji
- Błędnych interfejsów
- Błędów w strukturach danych lub metodach dostępu do zewnętrznych baz danych
- Niewłaściwego zachowania lub zbyt małej efektywności systemu
- Błędów powodujących niewłaściwe uruchamianie się lub wyłączanie systemu

Metoda czarnej skrzynki

- Stosuje się pod koniec testowania
- Pomija się strukturę sterowania
- Koncentracja na dziedzinie informacyjnej
- Umożliwia przygotowanie testów, które:
  - Zmniejszają (>1) liczbę dodatkowych testów koniecznych do przetestowania programu
  - Mogą wykryć lub wykluczyć całe grupy błędów jednocześnie

Testowanie vs debugowanie

- Testowanie ma na celu
  - Wykrycie błędów
- Debugowanie ma na celu
  - Lokalizację i poprawę błędów

Kiedy koniec testowania?

- Nigdy, jest tylko przekazywane klientowi
- Brakuje na nie czasu lub pieniędzy
- Nie ma idealnej odpowiedzi
- Istnieją praktyczne wskazówki
  - Kryteria statystyczne: „z 95% pewnością możemy powiedzieć, że prawdopodobieństwo bezbłędnego działania systemu przez 1000h pracy procesora jest równe co najmniej 0,995”

KONIEC