Analiza kodu statycznego

Przeglądanie kodu oraz obsługa błędów

Opracowanie: inż. Grzegorz Petri

„No amount of testing can prove a software right,
a single test can prove a software wrong”
(Amir Ghahrai)

01 Oczekiwania do oprogramowania

Jak zwiększyć jakość kodu oprogramowania?

01 Niezawodność

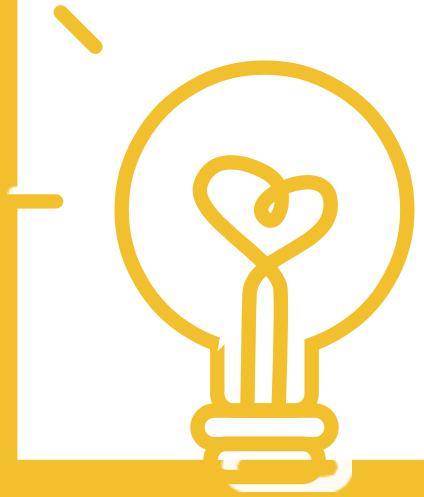
- integralność
- efektywność
- opracowane algorytmy
- używane wzorce projektowe

Jakość oprogramowania:

- *Zarządzanie ryzykiem*
- *Zarządzanie kosztami*

02 Funkcjonalność

- użyteczność
- elastyczność
- wydajność



Czy oprogramowanie spełnia wymagania

03 Bezpieczeństwo

Walidacja wejścia
Ochrona przed SQL injection
Ochrona przed Scross-Site Scripting
Kontrola dostępu
Dobre praktyki programistyczne
Obsługa błędów i wyjątków

04 Wsparcie/Utrzymywalność

Dokumentacja użytkownika
Czas wsparcia
Przenośność
Architektura programu
Dokumentacja osadzona w kodzie
Czytelność kodu

Problemy

- Analiza
- Potrzeby
- Projekt
- Szacowanie

- Niejasności
- Błędy ludzkie
- Problemy zewnętrzne

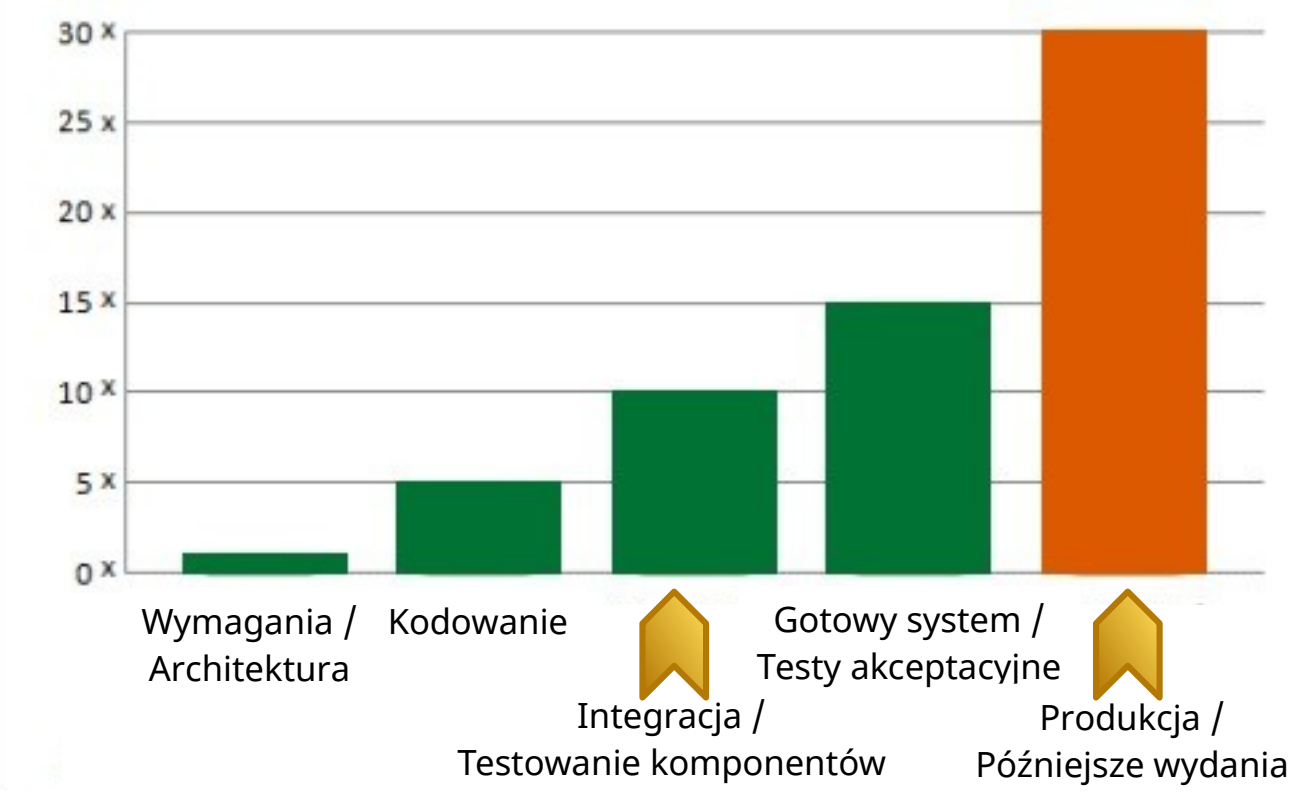
- Czas
- Koszty
- Błędy
- Oczekiwania

- Zasoby ludzkie
- Zmiany technologii
- Nowe błędy

- Wdrożenie
- Szkolenie
- Utrzymanie
- Rozwój
- Wsparcie

Problemy

Względny koszt naprawy błędu w zależności od momentu detekcji



Źródło: National Institute of Standards and Technology

Zwiększanie jakości

Code review
Przegląd kodu

- wykonywany ręcznie
- powinien być cykliczny
- czasochłonny i wymagający
- wymaga min. 2-ch osób
- można wykonywać w trakcie kodowania, również w parach

Static code analysis
Analiza kodu statycznego

- wykonywana automatycznie
- pokrywa tylko pewien zakres
- złożona i omylna (*false positive*)
- narzędzia zewnętrzne (+IDE)
- prostota wdrożenia w cykl
- łatwe użycie, szybkie testy

Unit/Pen. tests
Testowanie kodu

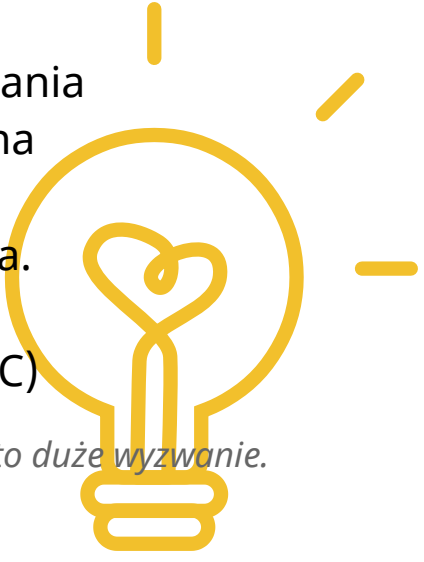
- rozwiązanie półautomatyczne
- zakres, który oprogramujemy
- czasochłonne, w miarę proste
- wymaga napisania testów i użycia narzędzi testowych
- proste wdrożenie w cykl
- łatwe użycie, czas testów różny

Analiza statyczna

Zadania:

- znajdywanie nieefektywnych konstrukcji oraz fragmentów kodu
- odnajdywanie potencjalnie niebezpiecznych konstrukcji kodu
- zwiększenie wydajności i stabilności oprogramowania
- wymuszenie wdrożenia dobrych praktyk lub własnych zasad pisania kodu
- dostarczenie struktury do zarządzania standardami kodu
- edukacja programisty poprzez analizę

- jest wykonywana automatycznie...
- ... choć uruchamiana przez człowieka
- nie wymaga uruchamiania programu ...
- ... choć wymaga kompilacji
- jest wykonywana na kodzie źródłowym...
- ... więc często narzędzie jest częścią IDE
- pozwala szybko wykryć błędy...
- ... więc ich eliminacja jest prosta i tania
- analizę można też przeprowadzić na skompilowanym kodzie...
- ... więc są też zewnętrzne narzędzia.
- Stąd SAST stanowi element cyklu wydawania oprogramowania (SDLC)



Najłatwiej wykonywana na językach silnie typowanych – dla słabo i dynamicznie typowanych jest to duże wyzwanie.

Metody analizy

leksykograficzna analiza kodu

- zestawy reguł
- różne heurystyki
- dopasowania do wzorców błędów

metody formalne

- modele matematyczne
- reguły logiczne
- analiza przepływu

metryki kodu źródłowego

ocena kodu na podstawie danych statystycznych

Metody analizy

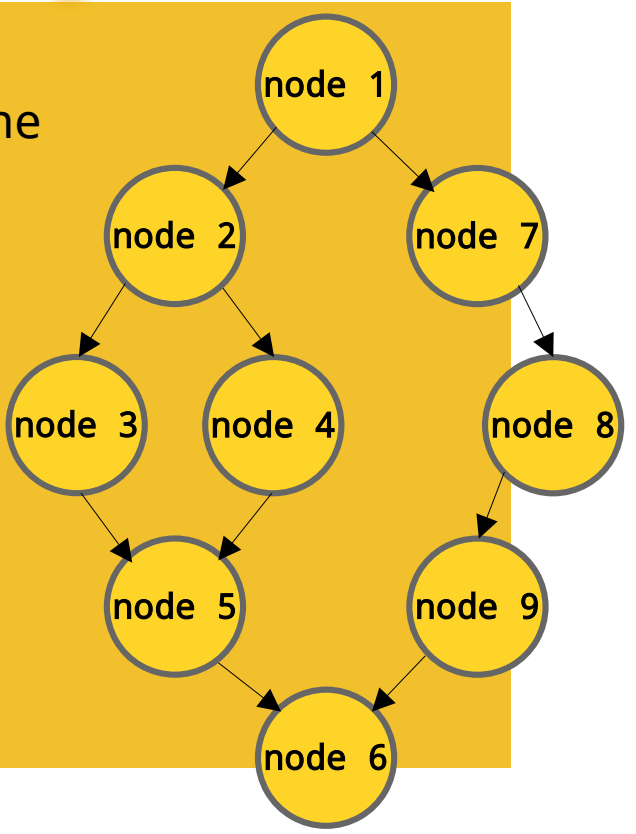
leksykograficzna analiza kodu

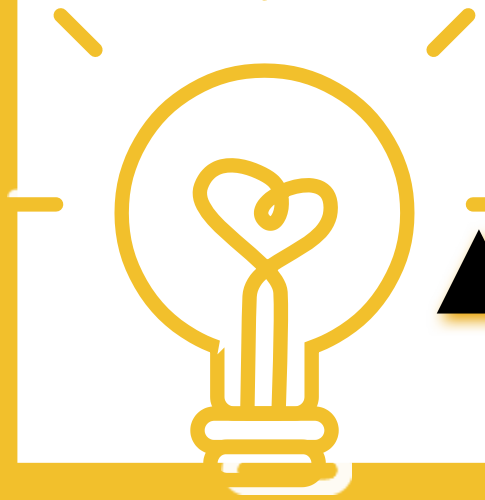
```
<?php
$name = „Grigorio”;
?>
```

```
T_OPEN_TAG
T_VARIABLE
=
T_CONSTANT_ENCAPSED_STRING
;
T_CLOSE_TAG
```

metody formalne

```
$a = 0;
$b = 1;
if ($a == $b)
{  # start bloku
echo „a oraz b są takie same”;
}  # koniec bloku
else
{  # start bloku
echo „a oraz b są różne”;
}  # koniec bloku
```





Zalety ▲

+01 Szybko i łatwo
Szybkość działania, możliwość zrównoleglenia, łatwość użycia

+02 Automatyzacja
integracja z narzędziami *continius integrations* oraz możliwość rozszerzania

+03 Elastyczne narzędzie
dużo darmowych i integracja z innymi: serwery automatyzacji, IDE, kontrola wersji

Wady ▼

-01 Wiele narzędzi
1 narzędzie pokrywa zakres testów, IDE oraz inne narzędzia, wymagany dostęp do źródeł

-02 Prosty kod, proste błędy
nie jest alternatywą dla Code Review, często też rezultaty są nieprawdziwe: *False positive*

-03 Złożone architektury
konfiguracje, logika biznesowa, skomplikowane algorytmy, zależności, konteksty frameworków, mieszane technologie, np.. C#,PHP,HTML,JS

Automatyzacja testów

Automatyzacja testów

- kiedy analizować:
  - przed/po commicie,
  - po każdym buildzie,
  - po zapisaniu pliku
- co analizuje:
  - edytor kodu IDE
  - rozszerzenie
  - zewnętrzne narzędzie (darmowe/płatne)
- wykonanie testów:
  - po stronie developera
  - po stronie repozytorium
  - serwery automatyzacji
- skrypty uruchamiane po zatwierdzeniu zmiany + powiadomienia (email, inne alerty)

Narzędzia / Platformy

- **CodeClimate**
  #CSS, #Go, #JS, #PHP, #Python, #Ruby
- **Codacy** //Platforma
  #CSS, #Java, #JS, #PHP, #Python, #Ruby, #Scala, coffescript
- **SonarQube** //Platforma
  20+ języków, m.in. Java, PL/SQL, C/C++ i C#
- **Understand Scitools** //Program
  m.in. Java, Python, C/C++, C#, Delphi/Pascal
- **Inne:**
  PMD, Findbugs, Checkstyle, jlint
  pixy, php-sat, RIPS,
  Cppcheck, splint,
  pylint, jslint, csslint, tidy

Issues?

report@gplweb.pl
edu.gplweb.pl