Inżynieria oprogramowania 1

Zatwierdzanie i weryfikacja oprogramowania

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska

Kielce, 26 listopada 2024

Plan

Motto

„Inżynier zapewniania jakości (QA) wchodzi do baru i zamawia piwo. Zamawia 0 piw. Zamawia 99999999999 piw. Zamawia jaszczurkę. Zamawia −1 piw. Zamawia ueicbksjdhd. Pierwszy rzeczywisty klient wchodzi do baru i pyta gdzie jest toaleta. Bar staje w płomieniach zabijając wszystkich wewnątrz.”

Źródło: Thomas’ Digital Garden

Wprowadzenie

Zatwierdzanie i weryfikacja (ang. *Validation and Verification — V&V*) jest procesem w inżynierii oprogramowania, którego celem jest zapewnienie jakości tworzonego produktu. W *Zatwierdzaniu* sprawdzane jest czy oprogramowanie spełnia potrzeby klienta, a w *Weryfikacji*, czy jest ono zgodne ze specyfikacją. Innymi słowy, *Zatwierdzanie* odpowiada na pytanie:

Czy tworzone jest odpowiednie oprogramowanie?

Z kolei *Weryfikacja* udziela odpowiedzi na pytanie:

Czy oprogramowanie jest tworzone w odpowiedni sposób?

Jakość oprogramowania

Przewodnik SWEBOK 3.0 [2] definiuje *jakość oprogramowania* jako zdolność produktu do zaspokojenia wyrażonych i domniemanych potrzeb, pod określonymi warunkami. Opisuje on również jakość jako stopień spełnienia przez oprogramowanie ustalonych wymagań, z zastrzeżeniem, że te wymagania reprezentują w sposób precyzyjny potrzeby, życzenia i oczekiwania interesariuszy.

Kontrola jakości (ang. *Quality Control* — QC) to proces sprawdzania poziomu jakości, a *zapewnianie jakości* (ang. *Quality Assurance* — QA) jest zbiorem działań zmierzających do poprawienia jakości w cyklu rozwojowym oprogramowania [1]. Te terminy nie są zamienne, więc ten użyty w motto jest (prawdopodobnie) niepoprawny ☺

Testowanie

Procesy QA, QC i V&V są przeprowadzane z wykorzystaniem *testowania*. Wyróżniamy dwa rodzaje tej czynności: *testowanie statyczne* i *testowanie dynamiczne*.

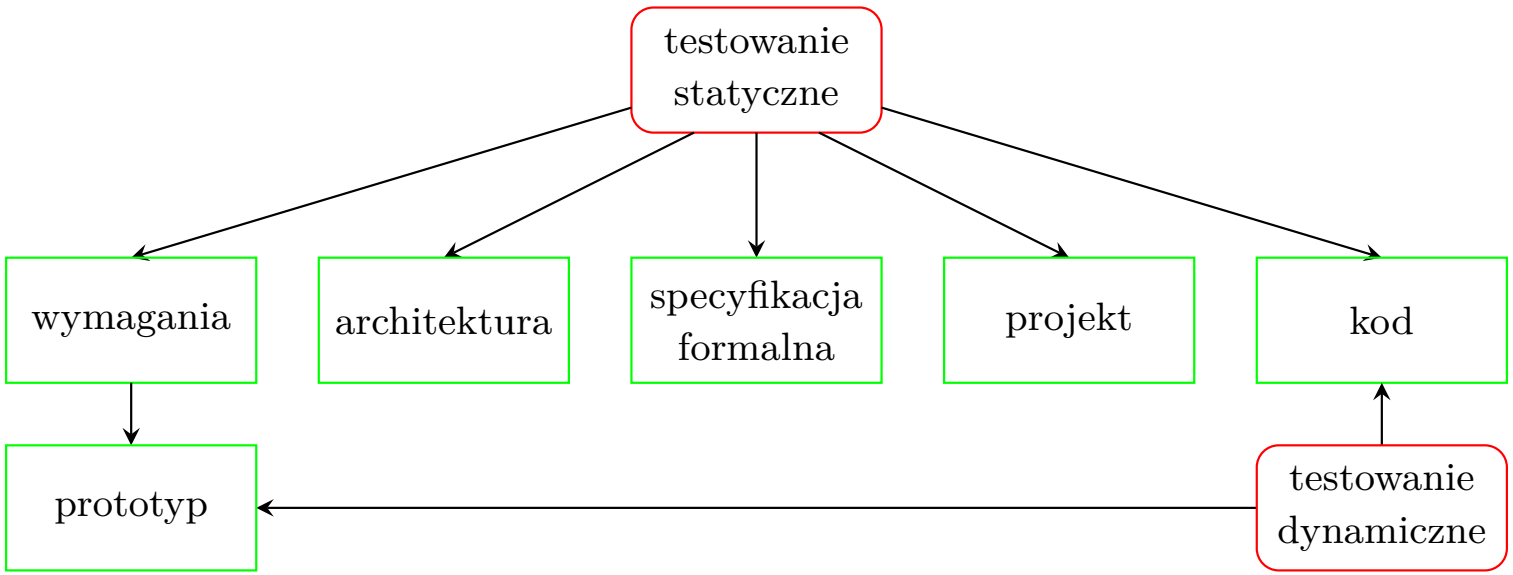
Testowanie statyczne jest systematycznym badaniem kodu i dokumentacji. W tej czynności kod nie jest uruchamiany. Ten rodzaj testowania może być wykonywany ręcznie lub przy pomocy odpowiednich programów narzędziowych. To ostatnie rozwiązanie nazywa się *analizą statyczną* (ang. *static analysis*). Najczęstszą formą testowania statycznego jest *przegląd*, który może być *formalny* lub *nieformalny*, zwanym także *przeglądem koleżeńskim* (ang. *peer review*). Jeśli wykonanie kodu jest symulowane podczas przeglądu, to ta czynność nazywa się *sprawdzeniem przy biurku* (ang. *desk checking*). Najbardziej formalnymi typami przeglądów są *inspekcja* (ang. *inspection*) wykonywana przez twórców kodu i *audyt* (ang. *audit*) przeprowadzany przez podmiot zewnętrzny (ang. *third party*).

Testowanie dynamiczne wymaga wykonywalnej formy kodu. W tym przypadku test jest eksperymentem, w którym oprogramowanie jest wykonywane dla pewnego starannie wybranego zbioru danych wejściowych i w którym badana jest poprawność jego danych wyjściowych oraz zachowania.

Zakres testowania statycznego i dynamicznego

Jak wynika z Rysunku 1, testowanie statyczne może być zastosowane dla prawie każdego artefaktu, który powstaje w trakcie tworzenia oprogramowania. Pozawala ono programistom wykryć defekty wcześniej, niż testowanie dynamiczne, które może być zastosowane jedynie do wykonywalnej postaci kodu. Jednakże obie te formy testowania **nie wykluczają się wzajemnie**. Co więcej, **powinny być stosowane razem**. Tylko za pomocą testowania dynamicznego można zweryfikować pewne atrybuty oprogramowania, które mogą być zbadane jedynie w trakcie jego wykonania.

Zakres statycznego i dynamicznego testowania

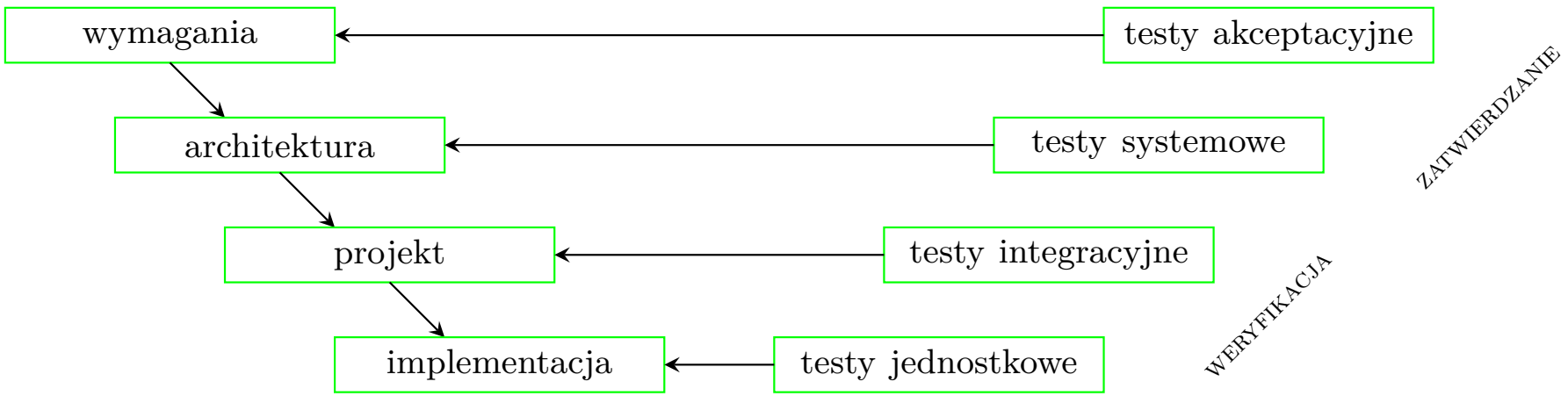


Rysunek: Zakresy różnych form testowania

Planowanie testów

Jak stwierdził Edsger Dijkstra, testy mogą ujawnić defekty w oprogramowaniu, ale nie potrafią zapewnić, że jest ono od nich wolne. To dlatego testowanie jest jedną z najdroższych czynności w tworzeniu oprogramowania. Początkowy koszt testowania statycznego jest wysoki, ale korzyści w późniejszych fazach projektu przewyższają wydatki. Z kolei na początku rozwoju oprogramowania testowanie dynamiczne prawie nie powoduje kosztów (ponieważ nie istnieje nic do testowania), ale później one gwałtownie rosną. Jednym z najistotniejszych zagadnień w testowaniu jest planowanie. W oryginalny modelu kaskadowym testy były wykonywane po koniec projektu. To szybko okazało się problematyczne, bo programiści nie otrzymywali informacji zwrotnej, wynikającej z wyników testów, która pozwoliłaby im usuwać defekty najwcześniej, jak to możliwe. Jedną z pierwszych prób włączenia testowania w cały proces tworzenia oprogramowania jest model V (Rysunek 2). W tym modelu z każdym artefaktem jest związany określony poziom testowania.

Model V



Rysunek: Model V

Wprowadzenie do testowania dynamicznego

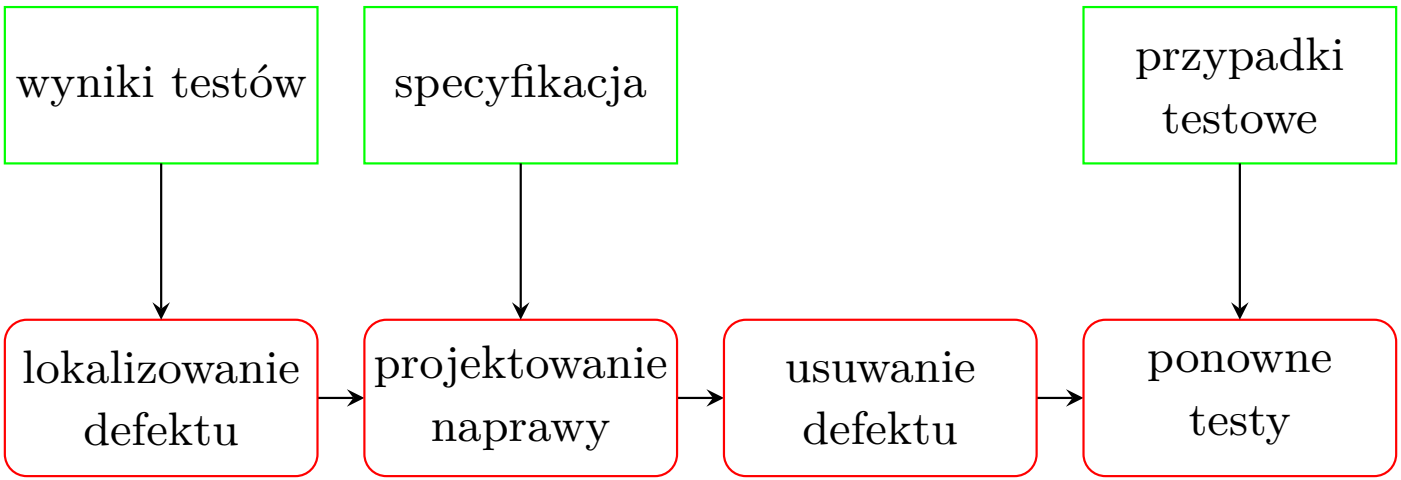
Głównym celem testowania dynamicznego jest wykrycie *defektów*. *Defekt*, to wada w kodzie oprogramowania, która powoduje, że pojawiają się *błędy* (ang. *error*), zwane także *usterkami*, *awariami* lub potocznie *pluskwami* (ang. *bugs*). *Błąd* jest nieoczekiwanym zachowaniem lub wynikiem pracy oprogramowania. Wynik testu ujawniający istnienie defektu nazywamy *pozytywnym*, a wynik nieujawniający *negatywnym*.
Testowanie dynamiczne może być też stosowane w celu sprawdzenia atrybutów, które ma oprogramowanie w czasie wykonania, a które są zdefiniowane przez wymagania niefunkcjonalne. Zalicza się do nich wydajność, dostępność i niezawodność. Tego rodzaju testowanie nazywane jest *testowaniem statystycznym*[1].
Granica między testami statystycznymi, a testowaniem w poszukiwaniu defektów jest rozmyta. Niektóre defekty, które nie zostały ujawnione przez inne rodzaje testów, mogą być wykryte w trakcie testowania statystycznego. Z kolei doświadczeni testerzy podczas *testowania defektów* potrafią oszacować wydajność i inne atrybuty czasu wykonania oprogramowania.

[1] Proszę nie mylić go z testowaniem statycznym. To są dwa różne pojęcia.

Lokalizowanie defektów

Wykrywanie defektów jest inną czynnością niż znajdowanie ich w kodzie, choć są one ze sobą ściśle powiązane. Wyniki testów dynamicznych pozwalają programiście przeanalizować symptomy defektu. Zlokalizowanie go w kodzie jest zależne od dziedziny zastosowania i użytego języka programowania. Wymaga także pewnego doświadczenia w używaniu takich narzędzi jak debuggery i symulatory. Rysunek 3 pokazuje związki między dynamicznym testowaniem oraz znajdywaniem i usuwaniem defektów.
Po usunięciu defektu oprogramowanie powinno zostać poddane ponownym testom. Celem takiego testowania jest sprawdzenie, czy defekt faktycznie został usunięty, czy tylko ukryty. W teorii, należałoby powtórzyć wszystkie testy, które zostały wykonane do chwili wykrycia defektu. To byłoby jednak zbyt kosztowne. Lepszym rozwiązaniem jest użycie testów, które pozwoliły wykryć defekt, oraz tych, które związane są z komponentami oprogramowania, na których działanie mogła mieć wpływ naprawa defektu. Taką metodę określa się mianem *testowania regresji* (ang. *regression testing*).

Lokalizowanie defektów



Rysunek: Testowanie, znajdowanie i usuwanie defektów

Testowanie statyczne

Porównanie z testowaniem dynamicznym

+ całkowity koszt testowania statycznego jest mniejszy niż całkowity koszt testowania dynamicznego,
+ podczas jednej sesji przeglądu można wykryć zazwyczaj więcej defektów, niż podczas jednej sesji testowania dynamicznego,
+ (potencjalne) defekty są nie tylko wykrywane, ale także lokalizowane,
+ przeglądy umożliwiają sprawdzenie także dodatkowych atrybutów kodu, takich jak czytelność oraz wykrycie problemów związanych z dziedziną zastosowania,
+ oprócz kodu przeglądowi mogą podlegać inne artefakty,
- przeglądy nie pozwalają sprawdzić atrybutów kodu związanych z jego wykonaniem (tj.: niezawodności, efektywności i innych),
- ręczne testowanie statycznie nie nadaje się do zastosowania dla dużej ilości kodu.

Inspekcje

Testowanie statyczne, w postaci *inspekcji*, wprowadził połowie lat 1970. do procesu tworzenia oprogramowania pracownik IBM, Michael Fagan. Zespół przeprowadzający inspekcję jest na ogół mały (do 10 osób). Każdy z jego członków otrzymuje co najmniej jedną z następujących ról:

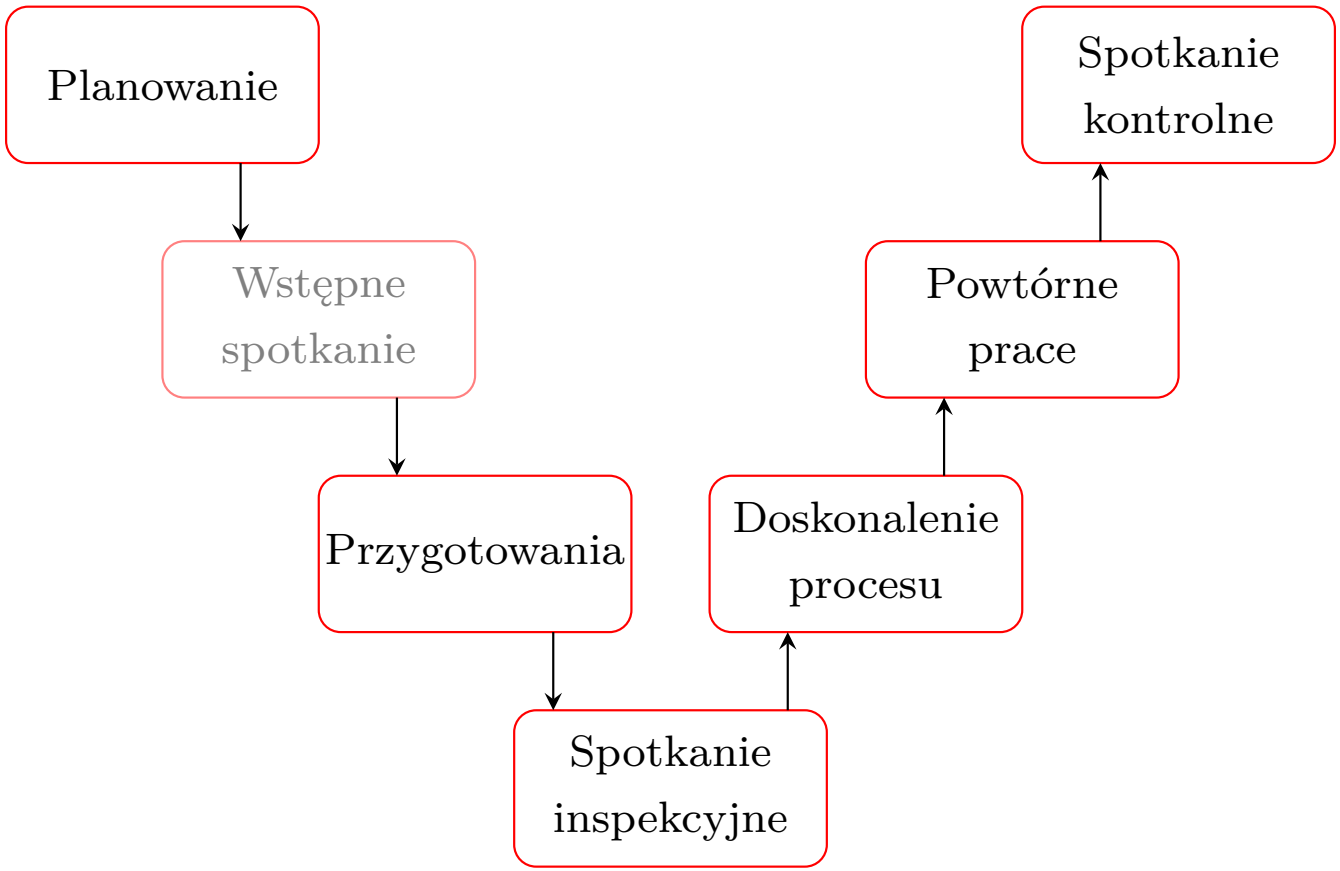
Rola	**Obowiązki**
Autor (Właściciel)	Dostarcza artefakt poddawany inspekcji. Usuwa wykryte defekty.
Recenzent	Znajduje defekty i inne problemy.
Czytelnik	Interpretuje sprawdzany kod lub dokument.
Sekretarz (Pisarz)	Zapisuje wyniki spotkania.
Moderator	Planuje i zarządza spotkaniami inspekcyjnymi.
Główny Moderator	Ustala standardy inspekcji w firmie.

Warunki wstępne

Niektóre warunki wstępne dla inspekcji zależą od typu badanych artefaktów. Inne są takie same we wszystkich przypadkach. *Moderator* jest odpowiedzialny za sprawdzenie, czy wszystkie warunki wstępne są spełnione. W przypadku, gdy badany jest kod, należy potwierdzić, że:

1. Dostępna jest bieżąca i poprawna specyfikacja tego kodu.
2. Członkowie zespołu znają standardy inspekcji.
3. Rozwój kodu podlegającego przeglądowi został ukończony.
4. Jest dostępna lista kontrolna (ang. *checklist*) typów defektów.

Proces inspekcji



Rysunek: Przebieg inspekcji

Testowanie statyczne — wykrywane defekty

Klasa defektów	**Przykłady pytań kontrolnych**
Defekty danych	Czy wszystkie zmienne zostały zainicjowane przed użyciem? Czy wszystkie stałe mają nazwy? Czy jest możliwe przepełnienie bufora?
Defekty sterowania	Czy każda pętla się kończy? Czy warunki w instrukcjach warunkowych są poprawne?
Defekty wejścia-wyjścia	Czy nieoczekiwane dane wejściowe mogą spowodować awarię?
Defekty interfejsu	Czy liczba argumentów funkcji jest prawidłowa? Czy typy danych argumentów i parametrów są zgodne?
Defekty zarządzania pamięcią	Czy wskaźniki są używane poprawnie? Czy przydzielona pamięć jest zwalniana, kiedy staje się niepotrzebna?

Testowanie statyczne — podsumowanie

Przeglądy nie są czasochłonne. Spotkanie inspekcyjne trwa około godziny, a pozostałe czynności w sumie 1–2 godzin. W czasie każdej inspekcji można przeanalizować do 500 wierszy kodu. Badania wykazały, że w trakcie przeglądów można znaleźć do 60% defektów, które musiałby być wykryte później, na drodze testowania dynamicznego, które jest bardziej kosztowne. Jeśli testowanie statyczne jest używane w połączeniu z metodami formalnymi, to liczba wykrytych defektów może sięgać nawet 90%. Przeglądy są trudne do wprowadzenia w firmach, w których panuje kultura (niezdrowej) rywalizacji.

Statyczna analiza kodu

Statyczne testowanie kodu można przeprowadzić za pomocą *analizatorów statycznych* (ang. *Static Code Analysis Tools* — SCA *Tools*). Te narzędzia nie uruchamiają oprogramowania, ale analizują jego kod źródłowy w poszukiwaniu *potencjalnych* defektów. **Muszą one być używane ostrożnie, gdyż nie znają kontekstu kodu i mogą dawać zarówno fałszywie pozytywne, jaki i fałszywie negatywne wyniki.** Zazwyczaj narzędzia SCA są bardziej dokładne niż kompilatory. Wykrywają one te same klasy defektów, które znajdywane są w trakcie manualnego przeglądu, ale również mogą sprawdzać dodatkowe własności kodu, na przykład jego strukturę i styl. Te z nich, które służą do weryfikowania bezpieczeństwa w badanym oprogramowaniu nazywane są narzędziami SAST (ang. *Static Application Security Testing*).

Przykłady narzędzi do analizy statycznej

LINT jest pierwszym programem, który został stworzony na potrzeby analizy statycznej. Wykonuje on weryfikację oprogramowania napisanego w języku C. Opracowano również inne analizatory statyczne, dla innych języków programowania. Niektóre z nich przeznaczone są dla więcej niż jednego języka programowania. Pierwszym reprezentantem narzędzi SAST jaki powstał jest *split*. Narzędzie *checkstyle* jest przykładem analizatora statycznego, który sprawdza czy kod programu jest zgodny z przyjętymi standardami jego tworzenia (w tym z formatowaniem). Z kolei *ikos* jest wolnym analizatorem statycznym, przeznaczonym dla języków C/C++, którego autorami są pracownicy NASA.
Współcześnie analizatory statyczne są używane głównie w potokach CI/CD, aby sprawdzać kod dodany (ang. *committed*) przez programistów zanim trafi on do repozytorium. Przykładami takich narzędzi są SonarQube (narzędzie typu *open source*), Coverity (oprogramowanie komercyjne firmy Synopsys), Klocwork (również narzędzie komercyjne, opracowane przez firmę Perforce). Wszystkie wymienione narzędzia obsługują wiele języków programowania i są ogólnego przeznaczenia.

Analiza statyczna — wykrywane typy defektów

Analizatory statyczne wykrywają te same klasy defektów co recenzenci. Jednakże mogą one również być zastosowane do obliczania niektórych *metryk kodu* i wykonywania *analizy przepływu danych* (ang. *data flow analysis*) i *analizy ścieżek* (ang. *code path analysis*). Pierwsza z wymienionych analiz śledzi związki między danymi wejściowymi i wyjściowymi, natomiast druga znajduje wszystkie ścieżki wykonania w kodzie i instrukcje, które tworzą te ścieżki. Analiza ścieżek jest szczególnie przydatna w metodzie testowania dynamicznego, która nazywa się *testowaniem strukturalnym* lub *testowaniem białej skrzynki*, albo *testowaniem przezroczystej skrzynki*.

Bibliografia

Adam Roman. *Thinking-Driven Testing: The Most Reasonable Approach to Quality Control*. Cham, Switzerland: Springer International Publishing AG, 2017.

Pierre Bourque i Richard E. Fairley, red. *SWEBOK 3.0: Guide to the Software Engineering Body of Knowledge*. URL: https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!