Inżynieria Programowania — Weryfikacja i zatwierdzanie

Arkadiusz Chrobot

Katedra Systemów Informatycznych, Politechnika Świętokrzyska w Kielcach

Kielce, 8 maja 2020

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Plan wykładu

Motto

„Pod koniec lat sześćdziesiątych zapowiadano pojawienie się programów dowodzących poprawności innych programów. Niestety, pod koniec lat osiemdziesiątych, oprócz kilku cennych wyjątków nadal nie pojawiło się nic poza opowieściami o systemach automatycznej weryfikacji."

Jon Bentley „Perełki oprogramowania"

Wstęp

Weryfikacja i zatwierdzanie (ang. *Verification and Validation - V&V*) to procesy, których celem jest zapewnienie, że tworzone oprogramowanie odpowiada specyfikacji i spełnia oczekiwania klientów. Terminy te nie są równoznaczne. Rozróżnienie między nimi jest następujące:

- **Zatwierdzanie** - określenie, czy produkt odpowiada potrzebom klientów,
- **Weryfikacja** - określenie, czy produkt jest budowany zgodnie ze specyfikacją.

Uwaga: Użyta w ramach tego wykładu terminologia jest odmienna od tej, którą stosuje organizacja ISTQB (ang. *International Software Testing Qualifications Board*). Więcej szczegółów na temat opracowań tej organizacji można znaleźć w książce Adama Romana pt. „Testowanie i jakość oprogramowania" oraz na stronach https://sjsi.org i www.istqb.org

Metody sprawdzania i analizy

Istnieją dwie główne metody, które można wykorzystać w procesie weryfikacji i zatwierdzania:

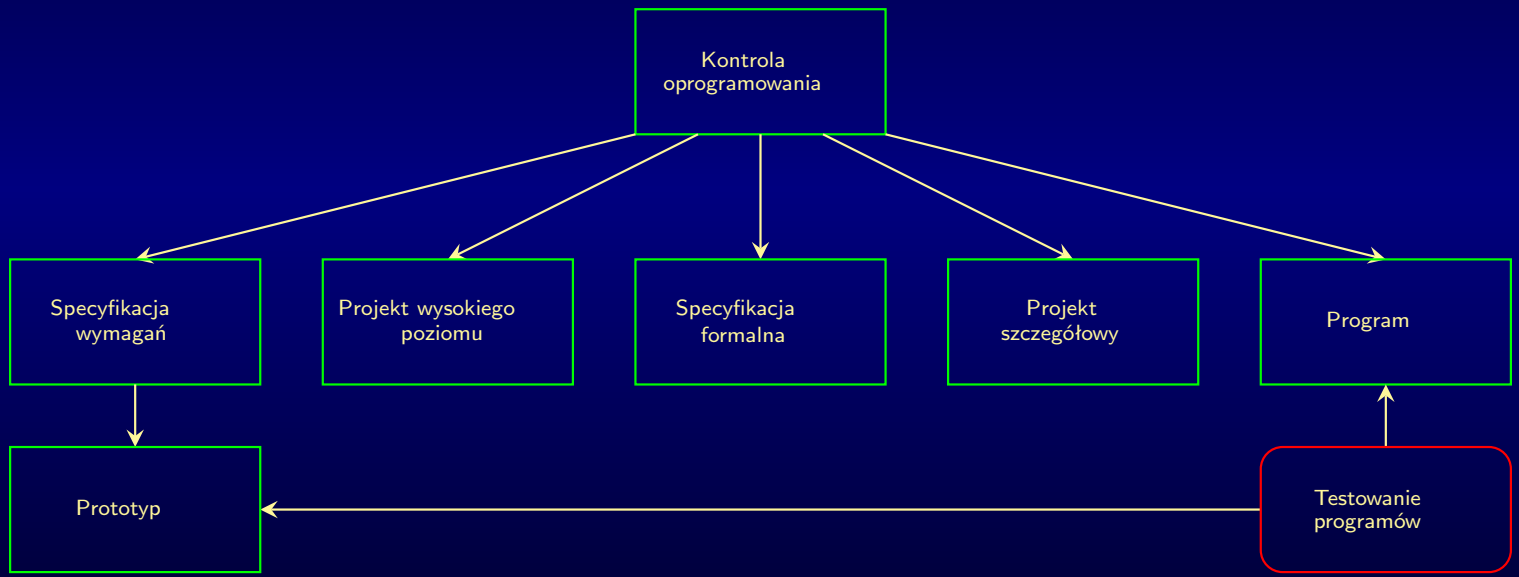
1. *Kontrole (inspekcje) oprogramowania* - polegają na analizie i sprawdzeniu przedstawień systemu, takich jak: dokumentacja wymagań, diagramy projektowe i kod źródłowy. Można je przeprowadzić w każdym kroku projektu, można je również wspomagać odpowiednim oprogramowaniem. Kontrole oprogramowania i inspekcje kodu są statycznymi metodami V&V, ponieważ nie trzeba do ich przeprowadzenia uruchamiać tworzonego programu.
2. *Testowanie oprogramowania* - sprowadza się do uruchomienia oprogramowania na danych testowych i badaniu wyników wygenerowanych przez to oprogramowanie oraz na badaniu jego zachowania celem sprawdzenia, czy jest zgodne ze specyfikacją. Testowanie jest dynamiczną metodą V&V, ponieważ dotyczy jedynie wykonywalnej reprezentacji tworzonego systemu.

Metody sprawdzania i analizy

Istnieją dwie główne metody, które można wykorzystać w procesie weryfikacji i zatwierdzania:

1. *Kontrole (inspekcje) oprogramowania* - polegają na analizie i sprawdzeniu przedstawień systemu, takich jak: dokumentacja wymagań, diagramy projektowe i kod źródłowy. Można je przeprowadzić w każdym kroku projektu, można je również wspomagać odpowiednim oprogramowaniem. Kontrole oprogramowania i inspekcje kodu są statycznymi metodami V&V, ponieważ nie trzeba do ich przeprowadzenia uruchamiać tworzonego programu.
2. *Testowanie oprogramowania* - sprowadza się do uruchomienia oprogramowania na danych testowych i badaniu wyników wygenerowanych przez to oprogramowanie oraz na badaniu jego zachowania celem sprawdzenia, czy jest zgodne ze specyfikacją. Testowanie jest dynamiczną metodą V&V, ponieważ dotyczy jedynie wykonywalnej reprezentacji tworzonego systemu.

Weryfikacja i zatwierdzanie dynamiczne i statyczne



Testowanie

Rodzaje testów:

1. *Testowanie defektów* - jego celem jest znalezienie niezgodności między programem, a jego specyfikacją. Testy przygotowuje się w celu wykrycia obecności usterek, a nie symulowania działania systemu.
2. *Testowanie statystyczne* - służy do zbadania efektywności i niezawodności programu, oraz sprawdzeniu jak działa w warunkach normalnego użytkowania.

Granica między tymi rodzajami testów jest płynna.

Testowanie

Rodzaje testów:

1. *Testowanie defektów* - jego celem jest znalezienie niezgodności między programem, a jego specyfikacją. Testy przygotowuje się w celu wykrycia obecności usterek, a nie symulowania działania systemu.
2. *Testowanie statystyczne* - służy do zbadania efektywności i niezawodności programu, oraz sprawdzeniu jak działa w warunkach normalnego użytkowania.

Granica między tymi rodzajami testów jest płynna.

Poziom zaufania

Celem V&V jest zapewnienie odpowiedniego *poziomu zaufania* w stosunku do tworzonego systemu. Poziom zaufania może dotyczyć:

1. *Funkcji oprogramowania* - zależy od stopnia krytyczności danego systemu dla firmy w której ma zostać zainstalowany. Ogólnie przyjmuje się, że poziom zaufania dla systemu dostarczonego powinien być wyższy niż dla jego prototypu.
2. *Oczekiwań użytkowników* - badania wykazały, że poziom oczekiwania użytkowników w stosunku do bezawaryjności oprogramowania jest niewielki, ale oczekują oni, że korzyści z użytkowania systemu będą większe niż wady. Od lat dziewięćdziesiątych dwudziestego wieku ten trend ma tendencję malejącą!
3. *Środowiska rynkowego* - firmy w obliczu konkurencji mogą dostarczać oprogramowanie, które jest zawodne, ale szybciej pojawiło się na rynku i jest tańsze.

Poziom zaufania

Celem V&V jest zapewnienie odpowiedniego *poziomu zaufania* w stosunku do tworzonego systemu. Poziom zaufania może dotyczyć:

1. *Funkcji oprogramowania* - zależy od stopnia krytyczności danego systemu dla firmy w której ma zostać zainstalowany. Ogólnie przyjmuje się, że poziom zaufania dla systemu dostarczonego powinien być wyższy niż dla jego prototypu.
2. *Oczekiwań użytkowników* - badania wykazały, że poziom oczekiwania użytkowników w stosunku do bezawaryjności oprogramowania jest niewielki, ale oczekują oni, że korzyści z użytkowania systemu będą większe niż wady. Od lat dziewięćdziesiątych dwudziestego wieku ten trend ma tendencję malejącą!
3. *Środowiska rynkowego* - firmy w obliczu konkurencji mogą dostarczać oprogramowanie, które jest zawodne, ale szybciej pojawiło się na rynku i jest tańsze.

Poziom zaufania

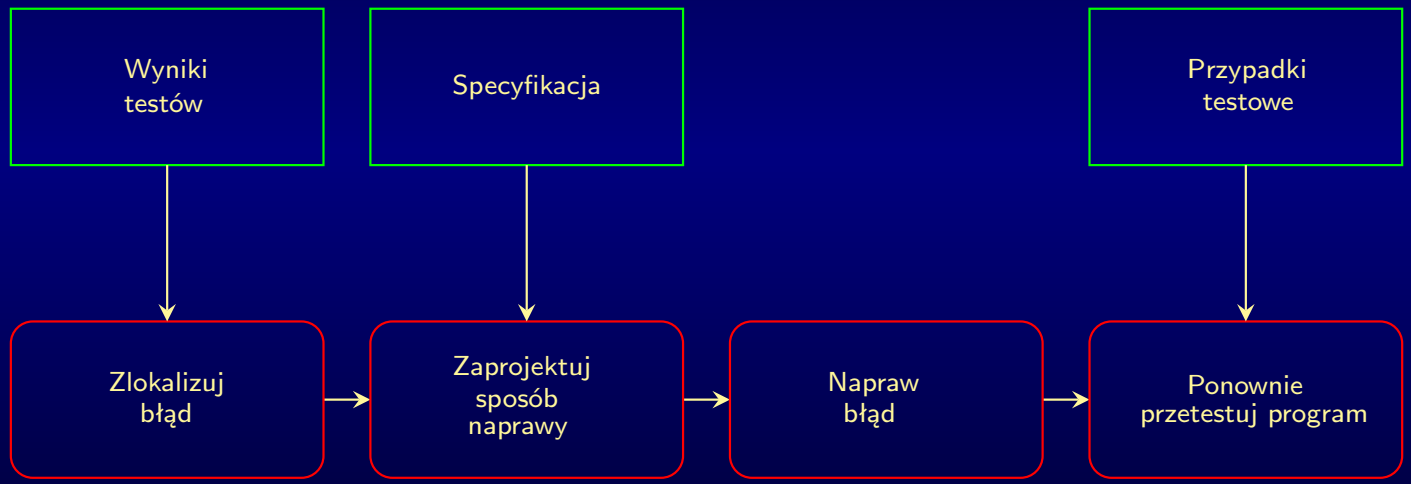
Celem V&V jest zapewnienie odpowiedniego *poziomu zaufania* w stosunku do tworzonego systemu. Poziom zaufania może dotyczyć:

1. *Funkcji oprogramowania* - zależy od stopnia krytyczności danego systemu dla firmy w której ma zostać zainstalowany. Ogólnie przyjmuje się, że poziom zaufania dla systemu dostarczonego powinien być wyższy niż dla jego prototypu.
2. *Oczekiwań użytkowników* - badania wykazały, że poziom oczekiwania użytkowników w stosunku do bezawaryjności oprogramowania jest niewielki, ale oczekują oni, że korzyści z użytkowania systemu będą większe niż wady. Od lat dziewięćdziesiątych dwudziestego wieku ten trend ma tendencję malejącą!
3. *Środowiska rynkowego* - firmy w obliczu konkurencji mogą dostarczać oprogramowanie, które jest zawodne, ale szybciej pojawiło się na rynku i jest tańsze.

Weryfikacja i zatwierdzanie, a usuwanie błędów

Należy jawnie rozgraniczyć procesy weryfikacji i zatwierdzania, a usuwania błędów. Celem pierwszego jest znalezienie defektów w tworzonym oprogramowaniu, a drugi ma na celu ich dokładną lokalizację i usunięcie.

Proces usuwania błędów



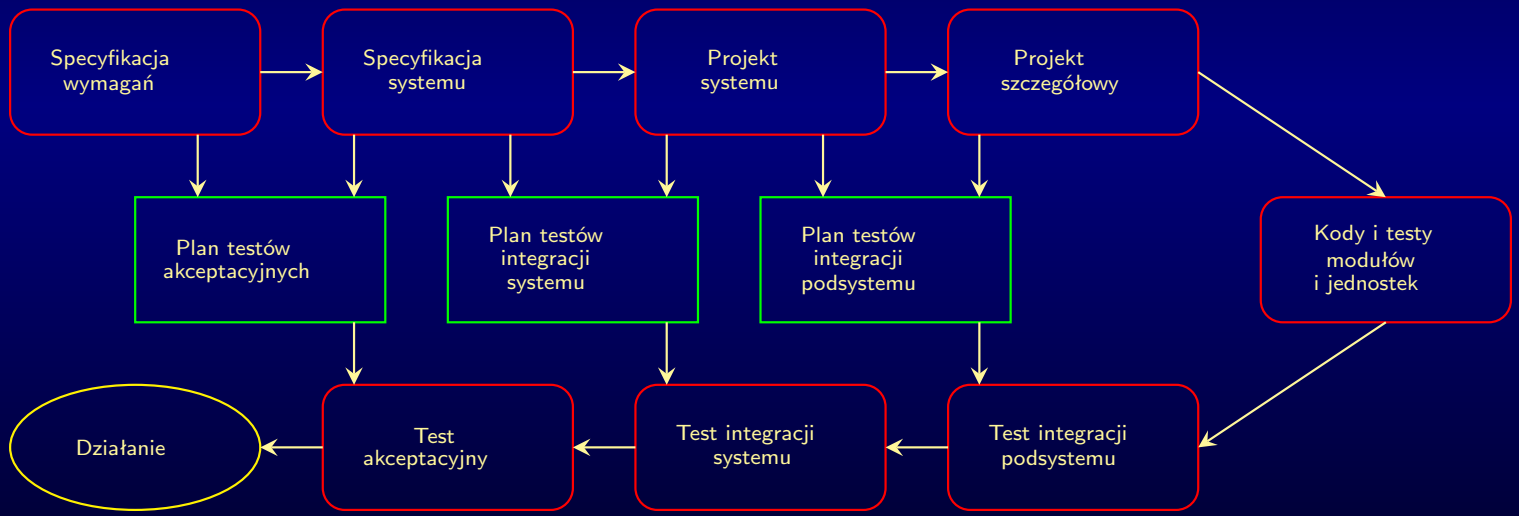
Usuwanie błędów

Nie ma prostej, jednolitej metody usuwania błędów. W tym procesie przydaje się doświadczenie osób, które dysponują wiedzą na temat naprawiania błędów w systemach o podobnej dziedzinie zastosowania i pisanych przy użyciu tego samego języka programowania. Proces ten należy w miarę możliwości wspomagać przez użycie odpowiednich programów narzędziowych. Po wprowadzeniu poprawek należy przeprowadzić *testy regresyjne*.

Planowanie weryfikacji i zatwierdzania

Weryfikacja i zatwierdzanie jest kosztownym procesem, dlatego należy staranie go zaplanować. Planowanie to odbywa się we wczesnych fazach procesu budowania.

Plany testowania jako pomost między tworzeniem a testowaniem



Planowanie testów

Struktura planu testów oprogramowania

Proces testowania
Opis zasadniczych faz procesu testowania.
Ślad wymagań
Testowanie należy zaplanować tak, aby sprawdzić wszystkie wymagania oddzielnie.
Testowane byty
Należy wskazać, które produkty tworzenia oprogramowania mają być poddane testom.
Harmonogramy testów
Ogólny harmonogram testowania i przydziału zasobów do jego realizacji. Należy go odnieść do bardziej ogólnego harmonogramu przedsięwzięcia.
Procedury zapisywania wyników testów
Wynik testów trzeba systematycznie zapisywać. Musi również istnieć możliwość kontroli procesu testowania, celem sprawdzenia, czy przebiega on w odpowiedni sposób.
Wymagany sprzęt i oprogramowanie
W tym punkcie określa się niezbędne narzędzia programowe do przeprowadzenie testów i szacuje się użycie sprzętu.
Ograniczenia
W tym punkcie należy określić przewidywane ograniczenia procesu testowania, np. brak personelu.

Inspekcje oprogramowania

Testowanie oprogramowania jest metodą czasochłonną i kosztowną. Najczęściej po przeprowadzeniu pojedynczego testu można wykryć co najwyżej jeden defekt. Aby zredukować koszt testów przeprowadza się kontrole statyczne reprezentacji systemu oraz statyczne inspekcje kodu źródłowego. Takie inspekcje są mniej kosztowne niż testowanie, a co najmniej równie skuteczne. Należy podkreślić, że nie eliminują one całkowicie konieczności przeprowadzania testów. Jedynie testy mogą być zastosowane na poziomie systemu i jedynie one pozwalają zbadać zachowanie dynamiczne systemu.

Przyczyny skuteczności inspekcji

1. Wiele różnych defektów można wykryć podczas jednej sesji inspekcji. Testowanie zwykle pozwala w trakcie jednej sesji wykryć jeden defekt. Ponadto symptomy defektów mają tendencję do nakładania się na siebie.
2. Inspekcje umożliwiają zastosowanie wiedzy dziedzinowej i dotyczącej języka programowania, co pozwala skoncentrować się na określonej grupie błędów.

Przyczyny skuteczności inspekcji

1. Wiele różnych defektów można wykryć podczas jednej sesji inspekcji. Testowanie zwykle pozwala w trakcie jednej sesji wykryć jeden defekt. Ponadto symptomy defektów mają tendencję do nakładania się na siebie.
2. Inspekcje umożliwiają zastosowanie wiedzy dziedzinowej i dotyczącej języka programowania, co pozwala skoncentrować się na określonej grupie błędów.

Role w procesie inspekcji

Autor lub właściciel	Programista lub projektant odpowiedzialny za opracowanie programu lub dokumentu. Odpowiada za usunięcie defektów wykrytych w trakcie procesu inspekcji.
Kontroler	Znajduje w programach i dokumentach błędy, pominięcia oraz niespójności. Może również rozpoznawać szersze zagadnienia spoza zakresu prac zespołu kontrolującego.
Czytelnik	Interpretuje kod lub dokument w trakcie spotkania kontrolnego.
Pisarz	Odnotowuje rezultaty sprawdzania końcowego.
Przewodniczący lub moderator	Zarządza procesem i ułatwia inspekcję. Informuje naczelnego moderatora o wynikach procesu.
Naczelny moderator	Odpowiada za ulepszenie procesu inspekcji, aktualizację list kontrolnych, opracowanie standardów itd.

Wymagania

Zanim rozpocznie się inspekcja programów należy upewnić się że:

1. Istnieje precyzyjna specyfikacja kodu podlegającego inspekcji. Bez pełnej specyfikacji nie uda się wykonać inspekcji komponentu na poziomie szczegółowości wystarczającym do wykrycia defektów.
2. Członkowie zespołu kontrolującego znają standardy firmowe.
3. Jest dostępna aktualna, poprawna składniowo wersja kodu. Inspekcja kodu „niemal ukończonego” nie ma sensu, nawet, jeśli opóźnienie spowoduje niedotrzymanie harmonogramu.

Wymagania

Zanim rozpocznie się inspekcja programów należy upewnić się że:

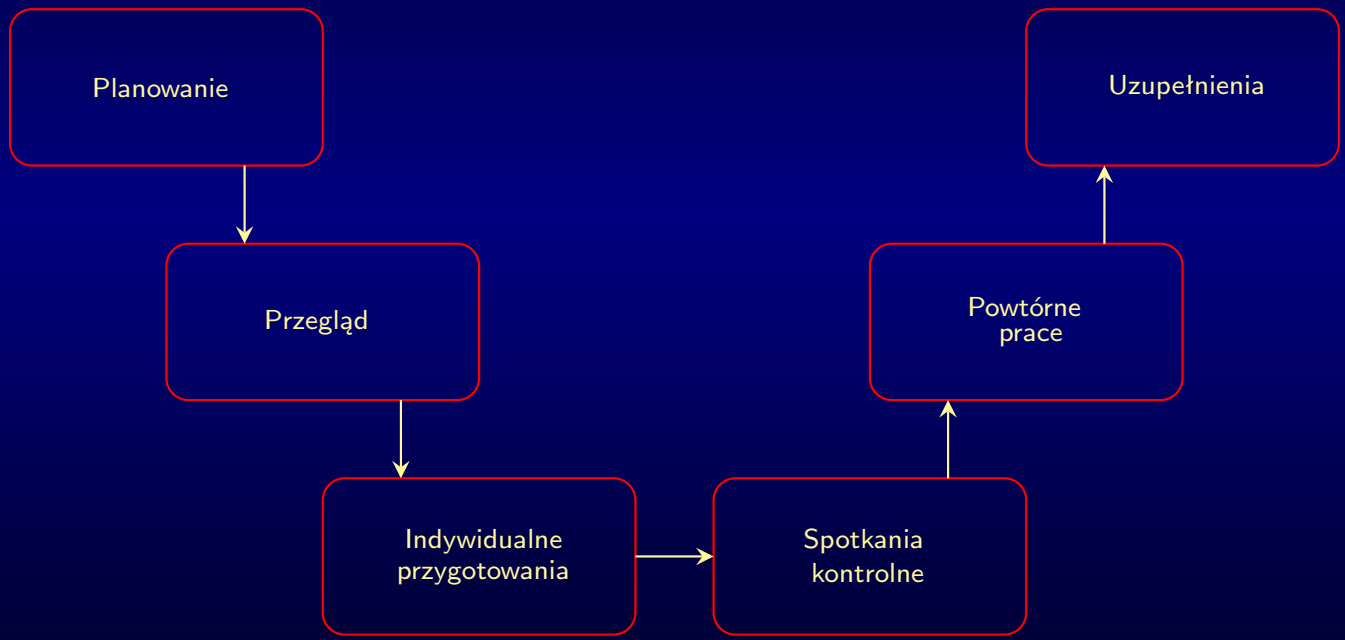
1. Istnieje precyzyjna specyfikacja kodu podlegającego inspekcji. Bez pełnej specyfikacji nie uda się wykonać inspekcji komponentu na poziomie szczegółowości wystarczającym do wykrycia defektów.
2. Członkowie zespołu kontrolującego znają standardy firmowe.
3. Jest dostępna aktualna, poprawna składniowo wersja kodu. Inspekcja kodu „niemal ukończonego” nie ma sensu, nawet, jeśli opóźnienie spowoduje niedotrzymanie harmonogramu.

Wymagania

Zanim rozpocznie się inspekcja programów należy upewnić się że:

1. Istnieje precyzyjna specyfikacja kodu podlegającego inspekcji. Bez pełnej specyfikacji nie uda się wykonać inspekcji komponentu na poziomie szczegółowości wystarczającym do wykrycia defektów.
2. Członkowie zespołu kontrolującego znają standardy firmowe.
3. Jest dostępna aktualna, poprawna składniowo wersja kodu. Inspekcja kodu „niemal ukończonego” nie ma sensu, nawet, jeśli opóźnienie spowoduje niedotrzymanie harmonogramu.

Proces inspekcji



Sprawdzenia kontrolne

Klasa usterek	**Sprawdzenie kontrolne**
Usterki danych	Czy wszystkie zmienne programu zainicjowano przed użyciem ich wartości? Czy wszystkie stałe mają nazwy? Czy górna granica zakresu tablicy jest równa rozmiarowi tablicy, czy rozmiarowi minus jeden? Czy tam, gdzie użyto stałych napisowych, jawnie przypisano ogranicznik? Czy istnieje jakakolwiek możliwość przepełnienia buforów?
Usterki sterowania	Czy warunek każdej instrukcji warunkowej jest poprawny? Czy każda pętla na pewno się zakończy? Czy instrukcje złożone są poprawnie ujęte w nawiasy? Czy w instrukcjach wyboru uwzględniono wszystkie możliwości? Czy tam gdzie jest konieczna instrukcja *break* w instrukcji wyboru, uwzględniono ją?
Usterki wejścia-wyjścia	Czy wszystkie zmienne wejściowe są używane? Czy wszystkie zmienne wyjściowe mają przypisaną wartość, zanim staną się daną wyjściową? Czy nieoczekiwane dane wejściowe mogą spowodować uszkodzenia?
Usterki interfejsu	Czy wszystkie wywołania funkcji i metod mają odpowiednią liczbę parametrów? Czy pasują do siebie typy parametrów formalnych i aktualnych? Czy parametry są podane w odpowiednim porządku? Czy komponenty korzystające z pamięci dzielonej zakładają ten sam model struktury pamięci dzielonej?
Usterki zarządzania pamięcią	Czy przy modyfikacji wskaźnikowej struktury danych wszystkie wiązania są właściwie przełączane? Czy tam, gdzie korzysta się z dynamicznego przydziału pamięci, jest ona przydzielana poprawnie? Czy pamięć jest jawnie zwalniana, gdy już nie jest potrzebna?

Automatyczna analiza statyczna

Automatyczną analizę kodu przeprowadza się przy pomocy narzędzi, które *mogą* wskazać miejsca potencjalnych usterek. Rodzaj narzędzi jakie należy użyć zależny jest od języka programowania, który służy do stworzenia systemu. Przykładami takich narzędzi są: LINT (język C), Flowfinder (język C), splint (język C), Perl::Critic (język Perl), RATS (C,C++,PHP,Perl,Python), Jlint (Java), JSlint (JavaScript). Kompleksowym narzędziem do analizy statycznej kodu jest SonarQube, który domyślnie obsługuje oprogramowanie napisane w języku Java, ale posiada też wtyczki (ang. *plugins*) umożliwiające mu analizę kodu aplikacji stworzonych przy pomocy innych języków programowania.

Kroki analizy statycznej

1. *Analiza przepływu sterowania* - rozpoznanie i oznaczenie pętli z wieloma punktami wejścia lub wyjścia oraz kodem nieosiągalnym.
2. *Analiza użycia danych* - badanie użycia zmiennych programu. Wykrywa się zmienne niezainicjowane przed użyciem, zmienne zapisywane dwukrotnie bez odczytu, zmienne zadeklarowane, ale nie użyte i warunki nadmiarowe.
3. *Analiza interfejsu* - sprawdzenie spójności deklaracji podprogramów i ich użycia. Wykrywane są również podprogramy zdefiniowane, ale nigdy nie użyte, oraz nie wykorzystane wyniki funkcji.
4. *Analiza przepływu informacji* - analiza zależności między zmiennymi wejściowymi i wyjściowymi.
5. *Analiza ścieżek* - określa się wszelkie możliwe ścieżki w programie i ustala instrukcje wykonywane w każdej z nich.

Kroki analizy statycznej

1. *Analiza przepływu sterowania* - rozpoznanie i oznaczenie pętli z wieloma punktami wejścia lub wyjścia oraz kodem nieosiągalnym.
2. *Analiza użycia danych* - badanie użycia zmiennych programu. Wykrywa się zmienne niezainicjowane przed użyciem, zmienne zapisywane dwukrotnie bez odczytu, zmienne zadeklarowane, ale nie użyte i warunki nadmiarowe.
3. *Analiza interfejsu* - sprawdzenie spójności deklaracji podprogramów i ich użycia. Wykrywane są również podprogramy zdefiniowane, ale nigdy nie użyte, oraz nie wykorzystane wyniki funkcji.
4. *Analiza przepływu informacji* - analiza zależności między zmiennymi wejściowymi i wyjściowymi.
5. *Analiza ścieżek* - określa się wszelkie możliwe ścieżki w programie i ustala instrukcje wykonywane w każdej z nich.

Kroki analizy statycznej

1. *Analiza przepływu sterowania* - rozpoznanie i oznaczenie pętli z wieloma punktami wejścia lub wyjścia oraz kodem nieosiągalnym.
2. *Analiza użycia danych* - badanie użycia zmiennych programu. Wykrywa się zmienne niezainicjowane przed użyciem, zmienne zapisywane dwukrotnie bez odczytu, zmienne zadeklarowane, ale nie użyte i warunki nadmiarowe.
3. *Analiza interfejsu* - sprawdzenie spójności deklaracji podprogramów i ich użycia. Wykrywane są również podprogramy zdefiniowane, ale nigdy nie użyte, oraz nie wykorzystane wyniki funkcji.
4. *Analiza przepływu informacji* - analiza zależności między zmiennymi wejściowymi i wyjściowymi.
5. *Analiza ścieżek* - określa się wszelkie możliwe ścieżki w programie i ustala instrukcje wykonywane w każdej z nich.

Kroki analizy statycznej

1. *Analiza przepływu sterowania* - rozpoznanie i oznaczenie pętli z wieloma punktami wejścia lub wyjścia oraz kodem nieosiągalnym.
2. *Analiza użycia danych* - badanie użycia zmiennych programu. Wykrywa się zmienne niezainicjowane przed użyciem, zmienne zapisywane dwukrotnie bez odczytu, zmienne zadeklarowane, ale nie użyte i warunki nadmiarowe.
3. *Analiza interfejsu* - sprawdzenie spójności deklaracji podprogramów i ich użycia. Wykrywane są również podprogramy zdefiniowane, ale nigdy nie użyte, oraz nie wykorzystane wyniki funkcji.
4. *Analiza przepływu informacji* - analiza zależności między zmiennymi wejściowymi i wyjściowymi.
5. *Analiza ścieżek* - określa się wszelkie możliwe ścieżki w programie i ustala instrukcje wykonywane w każdej z nich.

Kroki analizy statycznej

1. *Analiza przepływu sterowania* - rozpoznanie i oznaczenie pętli z wieloma punktami wejścia lub wyjścia oraz kodem nieosiągalnym.
2. *Analiza użycia danych* - badanie użycia zmiennych programu. Wykrywa się zmienne niezainicjowane przed użyciem, zmienne zapisywane dwukrotnie bez odczytu, zmienne zadeklarowane, ale nie użyte i warunki nadmiarowe.
3. *Analiza interfejsu* - sprawdzenie spójności deklaracji podprogramów i ich użycia. Wykrywane są również podprogramy zdefiniowane, ale nigdy nie użyte, oraz nie wykorzystane wyniki funkcji.
4. *Analiza przepływu informacji* - analiza zależności między zmiennymi wejściowymi i wyjściowymi.
5. *Analiza ścieżek* - określa się wszelkie możliwe ścieżki w programie i ustala instrukcje wykonywane w każdej z nich.

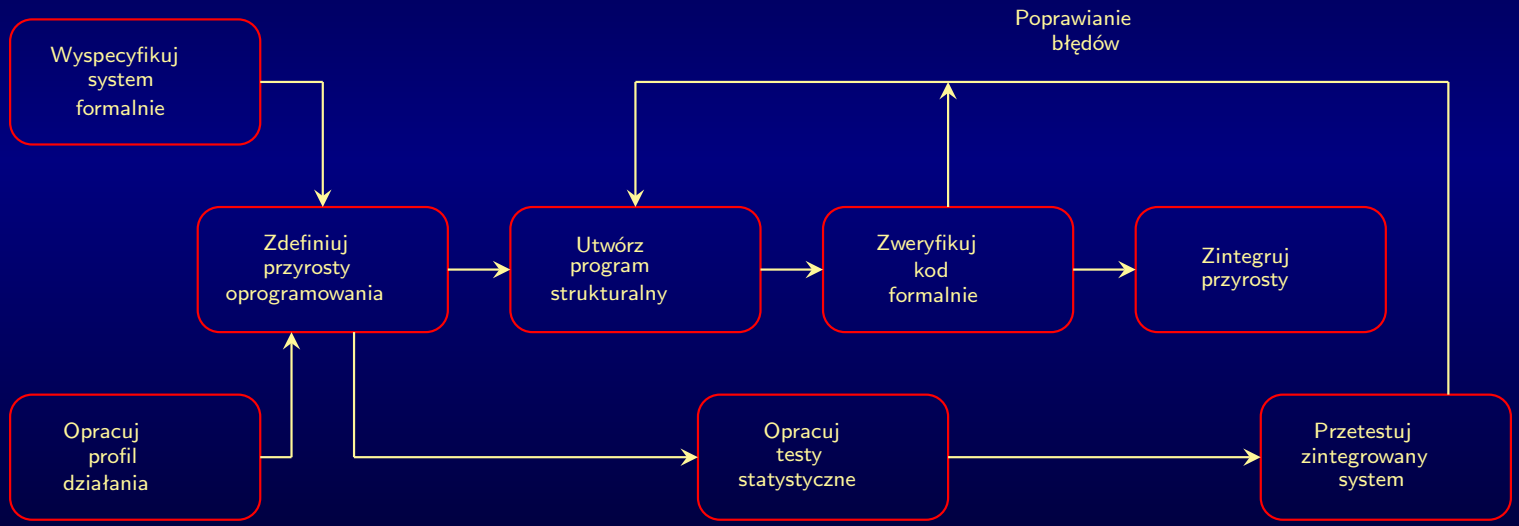
Wykrywane usterki

Klasa usterek	**Sprawdzenie analizy statycznej**
Usterki danych	Zmienne użyte przed inicjacją. Zmienne zadeklarowane, ale nigdy nie użyte. Zmienne, którym wartość przypisano dwukrotnie bez jej odczytu między przypisaniami. Potencjalne przekroczenie zakresu tablic. Nie zadeklarowane zmienne.
Usterki sterowania	Kod nieosiągalny. Bezwarunkowe odgałęzienia w pętlach.
Usterki wejścia-wyjścia	Zmienne wypisywane dwukrotnie bez przypisania im wartości między wypisaniami.
Usterki interfejsu	Niezgodność typów parametrów. Niezgodność liczby parametrów. Niewykorzystanie wyników funkcji. Nie wywołane podprogramy.
Usterki zarządzania pamięcią	Wskaźniki, którym nie przypisano wartości. Arytmetyka wskaźników.

Metoda Cleanroom tworzenia oprogramowania

Metoda Cleanroom to sposób tworzenia oprogramowania opracowany w IBM, którego podstawą jest unikanie defektów oprogramowania dzięki rygorystycznemu procesowi inspekcji.

Model procesu Cleanroom



Cechy metody Cleanroom

1. *Specyfikacja formalna* - oprogramowanie, które należy stworzyć jest specyfikowane formalnie. Tę specyfikację zapisuje się w postaci modelu stanu, z którego wynika, jak system reaguje na bodźce.
2. *Tworzenie przyrostowe* - oprogramowanie jest dzielone na przyrosty, które tworzy się oddzielnie i poddaje zatwierdzeniu w procesie Cleanroom. Te przyrosty specyfikuje się z udziałem klienta we wczesnej fazie procesu.
3. *Programowanie strukturalne* - korzysta się jedynie z ograniczonego zestawu konstrukcji sterowania i abstrakcji danych. Proces tworzenia programów jest procesem stopniowego udoskonalania specyfikacji. Używa się niewielkiej liczby konstrukcji. W celu utworzenia kodu programu do specyfikacji stosuje się jedynie te przekształcenia, które zachowują poprawność.
4. *Weryfikacja statyczna* - utworzone oprogramowanie poddaje się weryfikacji statycznej w rygorystycznych inspekcjach oprogramowania. Nie ma procesu testowania jednostkowego ani modułowego kodu komponentów.
5. *Testowanie statystyczne systemu* - zintegrowany przyrost oprogramowania jest testowany statystycznie w celu określenia jego niezawodności. Podstawą testów statystycznych jest profil działania opracowany równolegle ze specyfikacją systemu.

Cechy metody Cleanroom

1. *Specyfikacja formalna* - oprogramowanie, które należy stworzyć jest specyfikowane formalnie. Tę specyfikację zapisuje się w postaci modelu stanu, z którego wynika, jak system reaguje na bodźce.
2. *Tworzenie przyrostowe* - oprogramowanie jest dzielone na przyrosty, które tworzy się oddzielnie i poddaje zatwierdzeniu w procesie Cleanroom. Te przyrosty specyfikuje się z udziałem klienta we wczesnej fazie procesu.
3. *Programowanie strukturalne* - korzysta się jedynie z ograniczonego zestawu konstrukcji sterowania i abstrakcji danych. Proces tworzenia programów jest procesem stopniowego udoskonalania specyfikacji. Używa się niewielkiej liczby konstrukcji. W celu utworzenia kodu programu do specyfikacji stosuje się jedynie te przekształcenia, które zachowują poprawność.
4. *Weryfikacja statyczna* - utworzone oprogramowanie poddaje się weryfikacji statycznej w rygorystycznych inspekcjach oprogramowania. Nie ma procesu testowania jednostkowego ani modułowego kodu komponentów.
5. *Testowanie statystyczne systemu* - zintegrowany przyrost oprogramowania jest testowany statystycznie w celu określenia jego niezawodności. Podstawą testów statystycznych jest profil działania opracowany równolegle ze specyfikacją systemu.

Cechy metody Cleanroom

1. *Specyfikacja formalna* - oprogramowanie, które należy stworzyć jest specyfikowane formalnie. Tę specyfikację zapisuje się w postaci modelu stanu, z którego wynika, jak system reaguje na bodźce.
2. *Tworzenie przyrostowe* - oprogramowanie jest dzielone na przyrosty, które tworzy się oddzielnie i poddaje zatwierdzeniu w procesie Cleanroom. Te przyrosty specyfikuje się z udziałem klienta we wczesnej fazie procesu.
3. *Programowanie strukturalne* - korzysta się jedynie z ograniczonego zestawu konstrukcji sterowania i abstrakcji danych. Proces tworzenia programów jest procesem stopniowego udoskonalania specyfikacji. Używa się niewielkiej liczby konstrukcji. W celu utworzenia kodu programu do specyfikacji stosuje się jedynie te przekształcenia, które zachowują poprawność.
4. *Weryfikacja statyczna* - utworzone oprogramowanie poddaje się weryfikacji statycznej w rygorystycznych inspekcjach oprogramowania. Nie ma procesu testowania jednostkowego ani modułowego kodu komponentów.
5. *Testowanie statystyczne systemu* - zintegrowany przyrost oprogramowania jest testowany statystycznie w celu określenia jego niezawodności. Podstawą testów statystycznych jest profil działania opracowany równolegle ze specyfikacją systemu.

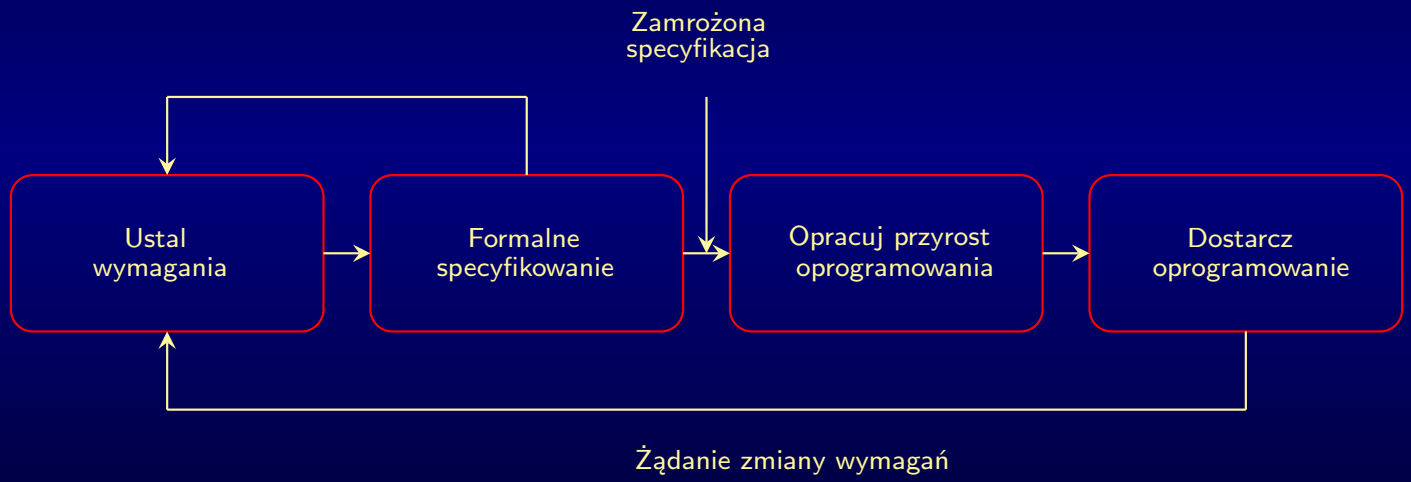
Cechy metody Cleanroom

1. *Specyfikacja formalna* - oprogramowanie, które należy stworzyć jest specyfikowane formalnie. Tę specyfikację zapisuje się w postaci modelu stanu, z którego wynika, jak system reaguje na bodźce.
2. *Tworzenie przyrostowe* - oprogramowanie jest dzielone na przyrosty, które tworzy się oddzielnie i poddaje zatwierdzeniu w procesie Cleanroom. Te przyrosty specyfikuje się z udziałem klienta we wczesnej fazie procesu.
3. *Programowanie strukturalne* - korzysta się jedynie z ograniczonego zestawu konstrukcji sterowania i abstrakcji danych. Proces tworzenia programów jest procesem stopniowego udoskonalania specyfikacji. Używa się niewielkiej liczby konstrukcji. W celu utworzenia kodu programu do specyfikacji stosuje się jedynie te przekształcenia, które zachowują poprawność.
4. *Weryfikacja statyczna* - utworzone oprogramowanie poddaje się weryfikacji statycznej w rygorystycznych inspekcjach oprogramowania. Nie ma procesu testowania jednostkowego ani modułowego kodu komponentów.
5. *Testowanie statystyczne systemu* - zintegrowany przyrost oprogramowania jest testowany statystycznie w celu określenia jego niezawodności. Podstawą testów statystycznych jest profil działania opracowany równolegle ze specyfikacją systemu.

Cechy metody Cleanroom

1. *Specyfikacja formalna* - oprogramowanie, które należy stworzyć jest specyfikowane formalnie. Tę specyfikację zapisuje się w postaci modelu stanu, z którego wynika, jak system reaguje na bodźce.
2. *Tworzenie przyrostowe* - oprogramowanie jest dzielone na przyrosty, które tworzy się oddzielnie i poddaje zatwierdzeniu w procesie Cleanroom. Te przyrosty specyfikuje się z udziałem klienta we wczesnej fazie procesu.
3. *Programowanie strukturalne* - korzysta się jedynie z ograniczonego zestawu konstrukcji sterowania i abstrakcji danych. Proces tworzenia programów jest procesem stopniowego udoskonalania specyfikacji. Używa się niewielkiej liczby konstrukcji. W celu utworzenia kodu programu do specyfikacji stosuje się jedynie te przekształcenia, które zachowują poprawność.
4. *Weryfikacja statyczna* - utworzone oprogramowanie poddaje się weryfikacji statycznej w rygorystycznych inspekcjach oprogramowania. Nie ma procesu testowania jednostkowego ani modułowego kodu komponentów.
5. *Testowanie statystyczne systemu* - zintegrowany przyrost oprogramowania jest testowany statystycznie w celu określenia jego niezawodności. Podstawą testów statystycznych jest profil działania opracowany równolegle ze specyfikacją systemu.

Tworzenie przyrostowe



Zespoły uczestniczące w metodzie Cleanroom

1. *Zespół specyfikujący* - jest grupą odpowiedzialną za opracowanie i pielęgnację specyfikacji systemu. Tworzy specyfikacje przeznaczone dla klienta (definicje wymagań) i specyfikacje matematyczne dla weryfikacji. W niektórych wypadkach po ukończeniu specyfikacji zespół specyfikujący przejmuje także odpowiedzialność za tworzenie.
2. *Zespół wytwarzający* - ten zespół odpowiada za utworzenie i zweryfikowanie oprogramowania. Oprogramowania nie uruchamia się w trakcie procesu tworzenia. Stosuje się formalne podejście do weryfikacji, którego podstawą jest inspekcja kodu uzupełniona uzasadnieniem poprawności.
3. *Zespół certyfikujący* - ten zespół jest odpowiedzialny za opracowanie zbioru testów statystycznych, za pomocą których bada się oprogramowanie po jego stworzeniu. Te testy są budowane na podstawie specyfikacji formalnej. Testy opracowuje się równolegle z tworzeniem oprogramowania. Przypadki testowe służą do wystawienia certyfikatu niezawodności oprogramowania.

Zespoły uczestniczące w metodzie Cleanroom

1. *Zespół specyfikujący* - jest grupą odpowiedzialną za opracowanie i pielęgnację specyfikacji systemu. Tworzy specyfikacje przeznaczone dla klienta (definicje wymagań) i specyfikacje matematyczne dla weryfikacji. W niektórych wypadkach po ukończeniu specyfikacji zespół specyfikujący przejmuje także odpowiedzialność za tworzenie.
2. *Zespół wytwarzający* - ten zespół odpowiada za utworzenie i zweryfikowanie oprogramowania. Oprogramowania nie uruchamia się w trakcie procesu tworzenia. Stosuje się formalne podejście do weryfikacji, którego podstawą jest inspekcja kodu uzupełniona uzasadnieniem poprawności.
3. *Zespół certyfikujący* - ten zespół jest odpowiedzialny za opracowanie zbioru testów statystycznych, za pomocą których bada się oprogramowanie po jego stworzeniu. Te testy są budowane na podstawie specyfikacji formalnej. Testy opracowuje się równolegle z tworzeniem oprogramowania. Przypadki testowe służą do wystawienia certyfikatu niezawodności oprogramowania.

Zespoły uczestniczące w metodzie Cleanroom

1. *Zespół specyfikujący* - jest grupą odpowiedzialną za opracowanie i pielęgnację specyfikacji systemu. Tworzy specyfikacje przeznaczone dla klienta (definicje wymagań) i specyfikacje matematyczne dla weryfikacji. W niektórych wypadkach po ukończeniu specyfikacji zespół specyfikujący przejmuje także odpowiedzialność za tworzenie.
2. *Zespół wytwarzający* - ten zespół odpowiada za utworzenie i zweryfikowanie oprogramowania. Oprogramowania nie uruchamia się w trakcie procesu tworzenia. Stosuje się formalne podejście do weryfikacji, którego podstawą jest inspekcja kodu uzupełniona uzasadnieniem poprawności.
3. *Zespół certyfikujący* - ten zespół jest odpowiedzialny za opracowanie zbioru testów statystycznych, za pomocą których bada się oprogramowanie po jego stworzeniu. Te testy są budowane na podstawie specyfikacji formalnej. Testy opracowuje się równolegle z tworzeniem oprogramowania. Przypadki testowe służą do wystawienia certyfikatu niezawodności oprogramowania.

Pytania

?

Koniec

Dziękuję Państwu za uwagę.