Inżynieria oprogramowania II

Wykład 9:

"UPEDU: Testowanie
(ang. *Testing discipline*)"

Marek Krętowski
e-mail: mkret@wi.pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

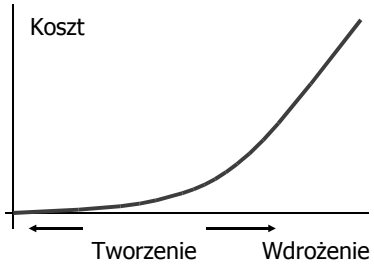
Na podstawie podręcznika: „Software Engineering Process with the UPEDU" P. Robillard, P. Kruchten, P. d'Astous, Addison-Wesley, 2003

Wprowadzenie

- Dwa najbardziej powszechne (a zarazem kosztowne) błędy związane z testowaniem:
  - traktowanie testowania jako końcowego zadania procesu wytwórczego
  - wprowadzanie niezależnego zespołu testującego, gdy implementacja jest "na ukończeniu", tuż przed dostarczeniem aplikacji do klienta
- Co więcej w wielu projektach w końcowej fazie realizacji brakuje czasu (lub zasobów), co wymusza ograniczenie nawet takich testów
- W procesie opartym o iteracje testowanie jest częścią procesu i tak jak inne czynności musi być zaplanowane, zaprojektowane, zaimplementowane a następnie wykonane przez członków zespołu
- Testowanie przeprowadzane powinno być podczas tworzenia oprogramowania, jedynie testy akceptacyjne wykonywane są podczas fazy przekazania (ang. *transition*)

(Kiedy) testowanie czas zacząć ...

- Odpowiednio wczesne rozpoczęcie testowania oraz jego poprawne przeprowadzenie pozwala na znaczącą redukcję kosztów ukończenia i utrzymywania oprogramowania
- Właściwie przemyślane testy redukują ryzyka oraz problemy związane z dostarczeniem oprogramowania niskiej jakości (niska produktywność użytkowników końcowych, błędy wprowadzania danych i obliczeń, nieakceptowalne dysfunkcjonalne zachowania, ...)
- Iteracyjność narzuca pewne wymagania na opracowywanie testów, które muszą rozwijać się w odpowiedzi na zmiany zachodzące w tworzonym oprogramowaniu



- Jednym z celów testowania jest sprawdzenie kompletności wykonania iteracji poprzez zweryfikowanie czy dodatkowe lub rozszerzone wymagania są już spełnione
- Planowanie i projektowanie testów może prowadzić do wykrycia błędów lub słabych punków w wymaganiach

Ocena jakości poprzez testowanie

- Choć często widziane jako sposób na poprawę jakości oprogramowania w rzeczywistości powinno być raczej traktowane jako sposób pomiaru jakości osiągniętej w procesie wytwórczym
- Jakość oprogramowania i testy są powiązane w takim sensie, że testowanie pozwala na potwierdzenie jakość, lecz samo jej nie tworzy
- Atrybuty jakości, które mogą być oceniane podczas testowania
  - funkcjonalność - typowe czynności testowania, w oparciu o utworzone testy dla poszczególnych scenariuszy
  - niezawodność, solidność (ang. *reliability*) - nie w oparciu o przypadki użycia, raczej przy użyciu narzędzi kompilacji i analizy
  - "osiągi aplikacji" (ang. *application performance*) - zachowanie się aplikacji uruchomionej samodzielnie
  - "osiągi systemu" (ang. *system performance*) - zachowanie się aplikacji w ramach całego systemu

Atrybuty jakości

Type	Why?	How?
Functionality	Is the application doing what is required?	Create test cases for each scenario implemented
Reliability	Is the application leaking memory?	Use analysis tools and code instrumentation
Application Performance	Is the application responding acceptably?	Check performance for each use-case/scenario implemented
System Performance	Is the system performing under production load?	Test performance of all use-cases under authentic and worst-case load

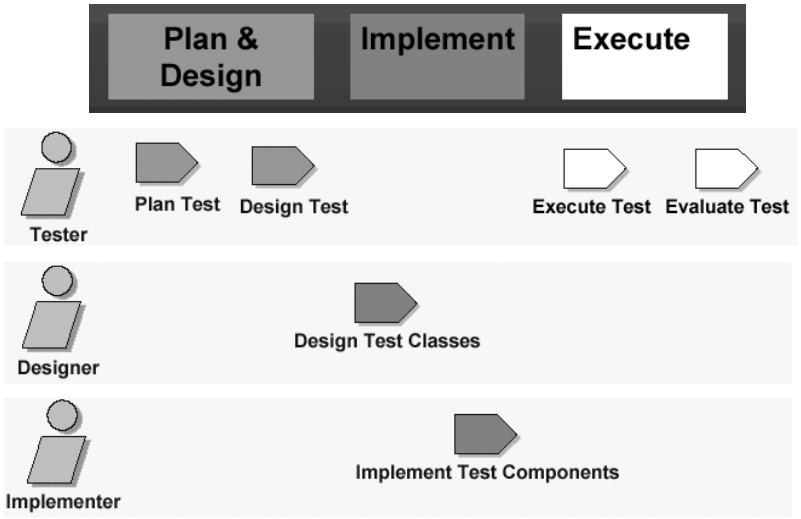
Rodzaje testowania

- **Testy integracyjne** (ang. *integration testing*) - wykonywane w celu sprawdzenia czy komponenty współpracują ze sobą poprawnie np. podczas realizacji przypadku użycia; służy do wychwycenia niekompletności lub błędów w specyfikacjach interfejsów pakietów
- **Testy systemu** (ang. *system testing*) - rozpoczynają się gdy oprogramowanie jako całość lub jego jasno określona część jest funkcjonalna (sprawna)
- **Testy akceptacyjne** (ang. *acceptance testing*) -celem jest sprawdzenie czy oprogramowanie jest gotowe i może być przekazane użytkownikowi, aby realizować postawione wymagania
  - ALFA - wykonywane w środowisku wytwórczym przez wytwórców oraz klientów, aby sprawdzić czy stworzony system jest akceptowalną implementacją wymagań
  - BETA - wymaga dostarczenia systemu do pewnej liczby potencjalnych klientów, którzy zgodzili się wykorzystywać system i przekazywać pojawiające się problemy

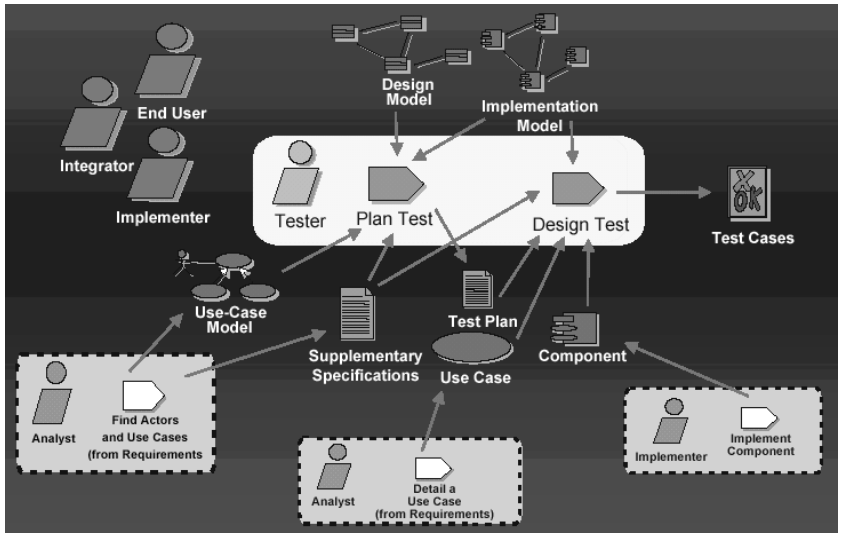
Poziomy testowania

Level	Product Target	Activity Tested
Debugging	Chunk of source code	Programmer workmanship
Unit	Designed product unit	Implementer design realization
Integration	Architectured product units	Implementer product realization
System	Product environment	Implementer product operation
Acceptance Alpha testing	Product functionality	Client product understanding
Beta testing	Product usability	Users product

Czynności dyscypliny testowania



Planowanie i projektowanie testów



Planowanie i projektowanie testów

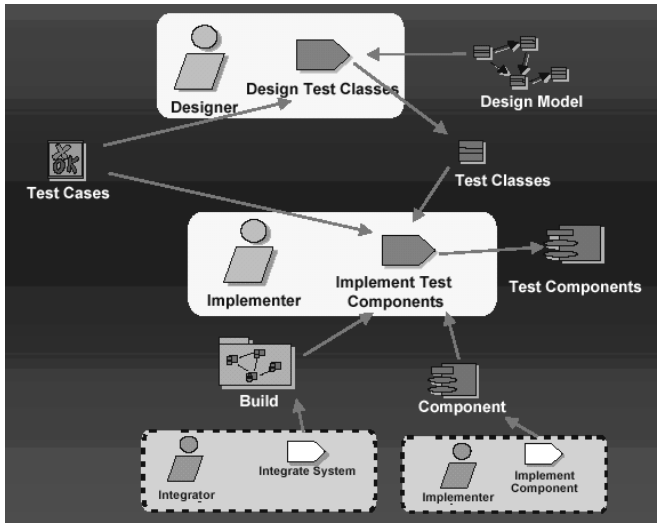
- Celem jest rozpoznanie wymagań, które będą testowane, wskazanie zakresu testowania oraz ustalenie zasobów niezbędnych do przeprowadzenia testów
- Dowolne aspekty zachowania oprogramowania mogą być badane pod warunkiem, że uzyskiwane są obserwowalne i mierzalne wyniki
- Wymagania do testów są najczęściej wyprowadzane z istniejących list wymagań, przypadków użycia, ich modeli i realizacji, dodatkowych specyfikacji, wymagań projektowych, rozmów z użytkownikami oraz przeglądów istniejących systemów
- Niezbędne jest odpowiednie zrównoważenie ryzyka z ograniczeniami zasobów w celu maksymalizacji efektywności testów i minimalizacji wysiłku związanego z testowaniem; zwykle powiązane z przypisaniem priorytetów, które określają sekwencję testów
- **Strategia** testów (kolejne kroki, planowanie i wykonanie, czas i zasoby), stosowane **podejście** (różne wykorzystywane techniki: *biała i czarna skrzynka*) oraz **kryteria** (obiektywne wyrażenia określające zakończenie i sukces testu) powinny być zdefiniowane i przedstawione członkom zespołu

Planowanie i projektowanie testów (2)

- Celem projektowania testu jest rozpoznanie i opisanie akcji aktora, gdy wchodzi on w interakcję z systemem => tworzone są przypadki testowe
- Przypadek testowy jest opisem konkretnego przeprowadzanego testu; składają się z specyficznych danych niezbędnych do przeprowadzenia testu oraz spodziewanych rezultatów; ponadto macierz zawierająca warunki testu, dane, stany systemu tworzące warunki, które są testowane
- Procedury testowe mogą być wykonywane ręcznie lub zaimplementowane w celu zautomatyzowanego wykonania; tworzone są skrypty testowe

Testy powinny być rygorystycznie zaplanowane, wyznaczone, udokumentowane i śledzone. Wykonywane *ad hoc* (spontanicznie) wiążą się zwykle ze stratą zasobów (czasu)

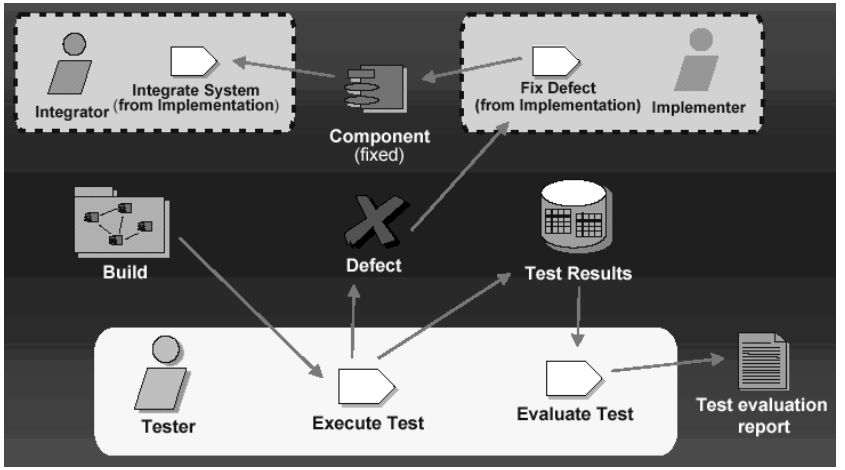
Implementacja testów



Implementacja testów

- Rodzaje komponentów testowych:
  - jednorazowy (ang. *throwaway*)- wykorzystywane jeden raz, tylko w celu testowania określonej funkcjonalności
  - skrypt (ang. *script*) - wykorzystywany wielokrotnie, zwykle zautomatyzowany w celu zapewnienia łatwości wykorzystania
  - "żywy trup" (ang. *zombie*) - stworzony aby symulować rzeczywisty komponent; jedynie minimalna funkcjonalność (*driver* lub *stub*)
- Dwa rodzaje pełnomocników (*stub*):
  - proste - zwracają jedynie określoną wartość, może być ich potrzebnych wiele
  - złożone - wykonują pewne przetwarzanie i zwykle symulują bardziej złożone systemy lub zachowania
- Dwa rodzaje driver-ów:
  - symulujące i kontrolujące wszystkie zewnętrzne interakcje z aplikacją
  - dostarczające GUI, który nie został jeszcze zaimplementowany

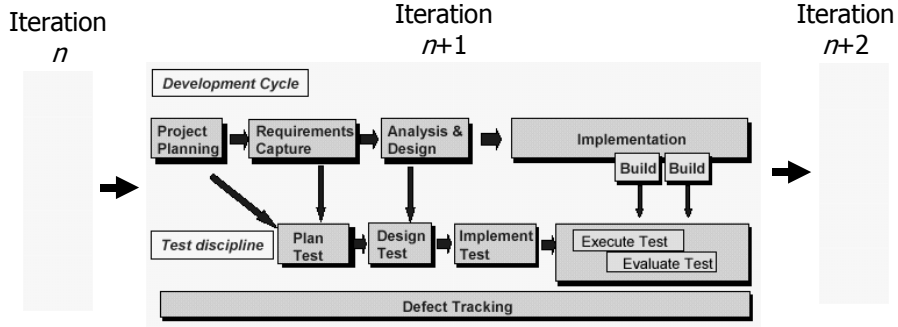
Wykonanie testów



Wykonanie testów

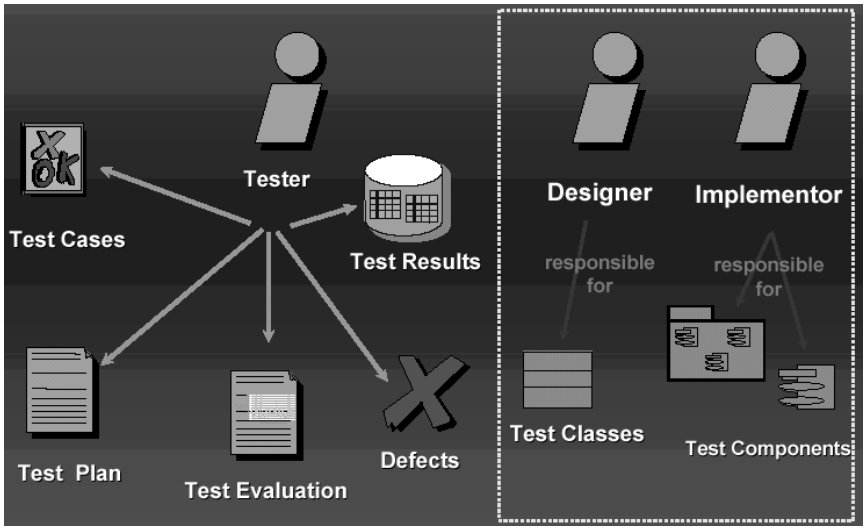
- Aby wykonać testy, wszystkie niezbędne elementy (sprzęt, oprogramowanie, narzędzia, dane, ...) muszą być przygotowane i dostępne w środowisku testowym
- Ważne jest aby dokonać oceny testów, aby stwierdzić czy zostały one przeprowadzone kompletnie i tak jak zaplanowano; w przypadku anomalii podczas czynności testowych, odpowiednie akcje poprawiające powinny być przedsięwzięte
- Testy mogą zakończyć się przedwcześnie wskutek problemów z wykonaniem lub ograniczeń czasowych
- **Testowanie regresyjne** (ang. *regression testing*) - po wprowadzeniu zmian do oprogramowania wykonywane w celu zapewnienia, że defekty nie zostały wprowadzone jako rezultat zmian; ten typ testów porównuje aktualną z poprzednią wersją i identyfikuje różnice jako potencjalne defekty (przy założeniu, że zachowanie nie powinno ulec zmianie)
- Nacisk na tego typu testowanie jest kładziony podczas kolejnych iteracji, gdy większość z testów z *n*-tej iteracji jest wykorzystywana w *n*+1 iteracji jako testy regresyjne (ponadto każdorazowo dodawane są jednak nowe testy, które nie muszą być regresyjne)

Testowanie regresyjne
(ang. regression testing)



- Każdy nowy przypadek testowy może być potencjalnie włączony do testów regresyjnych; pamiętać trzeba jednak o kosztach utrzymania i wykonywania testów w odniesieniu do korzyści z ich przeprowadzenia
- Wykorzystanie narzędzi automatyzacji testów znacząco poprawia stosunek korzyści do kosztów w testowaniu regresyjnym

Różnorodność artyfaktów

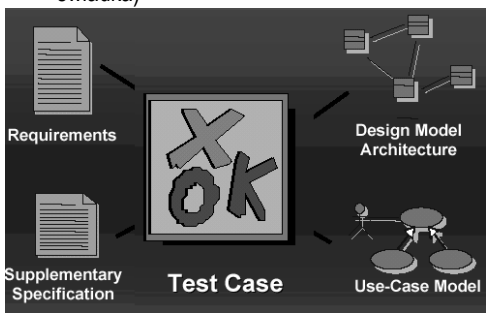


Artefakty

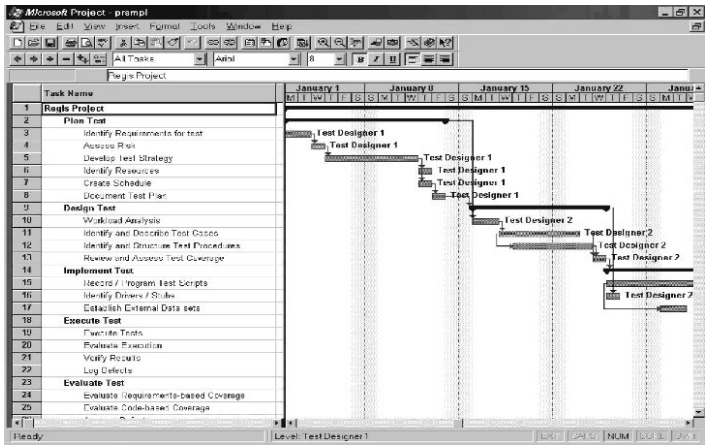
- Przypadek testowy (ang. *test case*) - podstawowy artefakt, zawiera opis wszystkich informacji niezbędnych do wykonania testu
- Skrypt testowy - pozwala na automatyzację przypadków testowych, wykorzystywane przede wszystkim podczas testów regresyjnych
- Model testowy - zbiór przypadków testowych wraz z odpowiadającymi im skryptami
- Plan test - identyfikuje strategię testów oraz określa niezbędne zasoby
- Rezultat testów - wyjście wszystkich czynności testowych
- Defekty - listuje anomalie znalezione w wyniku testów
- Zapotrzebowanie na zmiany (ang. *request for change*) - zawiera anomalie powstałe podczas testów
- Ocena testów (ang. *test evaluation*) - prezentuje wykresy i inne dane dotyczące niezawodności, solidności (ang. *reliability*) czynności testowych
- Klasy i komponenty testowe

Przypadki testowe
(ang. test cases)

- Tworzone dla każdego przypadku użycia (scenariusza) związanego z iteracją, w przypadku zbyt dużej liczby wprowadza się priorytety (co najmniej standardowy i alternatywne przepływy podczas normalnego użytkowania)
- Kolejne przypadki testowe mogą być tworzone jako **warianty** już istniejących (taki sam przebieg testu, ale inne dane wejściowe i wyniki)
- Drugim ważnym źródłem przypadków testowych są wymagania niefunkcjonalne nie manifestujące się w przypadkach użycia (np. obciążenie - zdolność przetwarzanie dużych danych)
- Akceptacyjne przypadki testowe - podzbiór wzajemnie uzgodnionych przez klienta i dostawcę przypadków testowych na bazie których dokonywany jest test akceptacyjny (wykonywany w określonych warunkach, np. obecność eksperta - świadka)



Plan testów

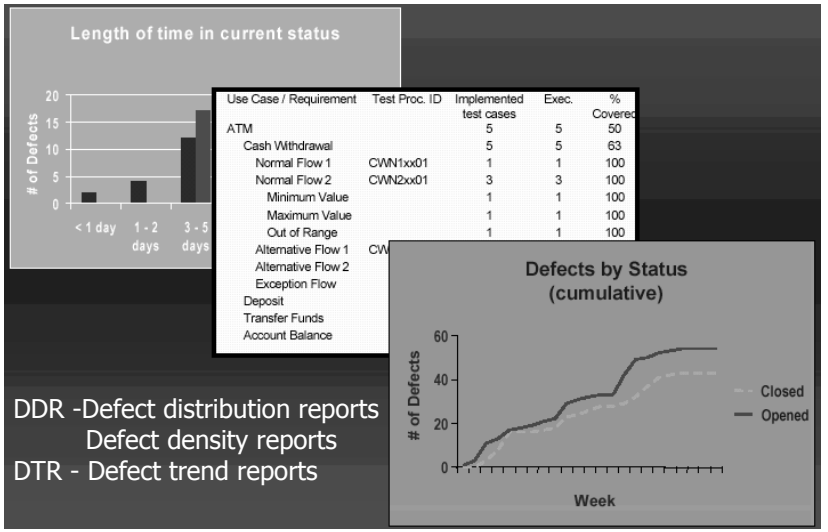


- Strategia testów - zarysowuje ogólne podejście, cele (w ramach danej iteracji) oraz harmonogram czynności testowych

Defekty

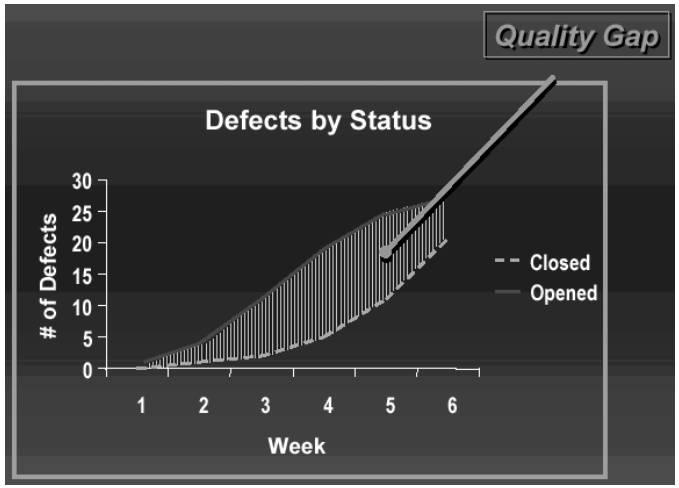
- Zawiera dokładny opis anomalii oraz sposób w jaki może być on odtworzony
  - status - otwarty, poprawiany, zamknięty, ...
  - priorytet - natychmiastowe wykonanie, wysoki priorytet, normalna kolejka, niski
  - istotność (ang. severity), krytyczność (ang. criticality) - błąd fatalny (ang. fatal error), podstawowa funkcja nie została wykonana, pomniejsza niedogodność
  - źródło - wymagania, architektura, moduł N, biblioteka
- Standardy opisywania anomalii np. taksonomia z IEEE 10444
- Należy uważać na sygnalizowane defekty, które mogą być w rzeczywistości dodatkowymi wymaganiami rozbudowy funkcjonalności; ponadto mogą pojawiać się powtórzenia
- Niezbędna jest dokładna analiza zebranych defektów - raporty oceny defektów (ang. *defect evaluation report*) oraz śledzenie ich losu: wykorzystanie specjalizowanego oprogramowania

Raporty oceny defektów
(ang. defect evaluation reports)



DDR -Defect distribution reports
Defect density reports
DTR - Defect trend reports

Raport trendu
(ang. defect trends report)



Raport gęstości defektów
(ang. defect density report)

Defect density by Use-Case Flow and Severity