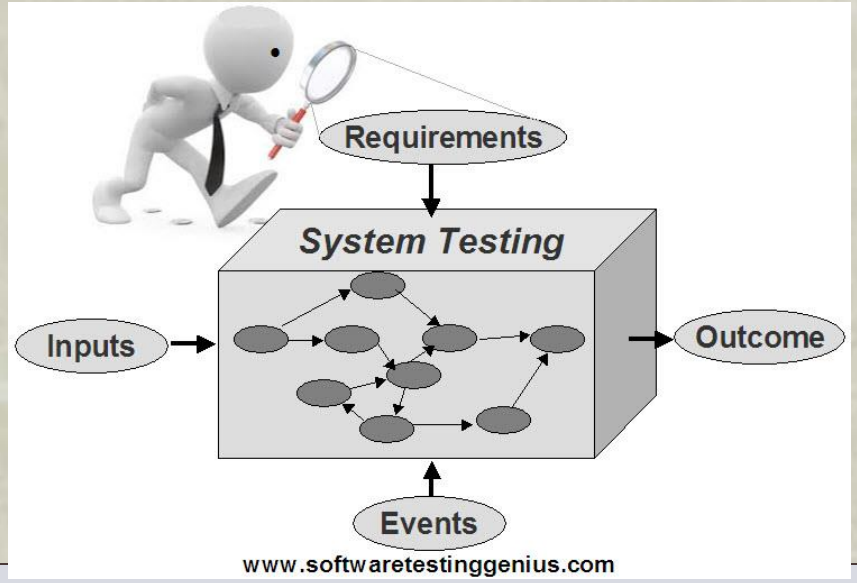
Metody zarządzania projektami informatycznymi

Wykład 7

Testowanie oprogramowania – cz. 2



dr hab. inż. Aneta Poniszewska-Marańda
Instytut Informatyki Politechniki Łódzkiej

Testowanie – testy wydajnościowe

- testy wydajnościowe – *performance testing*
  - badanie czasu odpowiedzi krytycznych dla biznesu funkcji systemu
  - sprawdzenie, czy poszczególne akcje wykonywane przez aplikację w akceptowalnym czasie

Testowanie – testy wydajnościowe

- niska wydajność lub brak dostępu do usługi może powodować znaczne straty finansowe (np. klienci przeniosą się do konkurencji)
- wysokowydajne systemy (serwery, łącza o dużej przepustowości) są kosztowne
- konieczna jest równowaga pomiędzy tymi dwoma czynnikami
- testowanie wydajności:
  - jest skomplikowane i kosztowne związku z ilością zasobów, jakich wymaga oraz czasu, jakiego należy na nie poświęcić
  - nie jest jednoznaczne – różne osoby mogą mieć inne oczekiwania co do wydajności
  - duża liczba defektów odsłoniętych podczas testowania może wymagać zmiany projektu systemu

Testowanie – testy wydajnościowe

- wykonywane w celu zapewnienia, że aplikacja:
  - przetwarza określoną liczbę transakcji w założonym przedziale czasu
  - jest dostępna i ma możliwość uruchomienia w różnych warunkach obciążeniowych
  - umożliwia wystarczająco szybką odpowiedź w różnych warunkach obciążeniowych
  - umożliwia dostęp do zasobów w zależności od potrzeb aplikacji
  - jest porównywalna pod względem parametrów wydajnościowych do konkurencyjnych aplikacji

Testowanie – testy wydajnościowe

Parametry wydajności

- *Przepustowość*
  - liczba transakcji/żądań obsługiwanych przez system w określonym przedziale czasu
  - mierzy, ile transakcji może być przetwarzanych w określonym przedziale czasu przy zadanym obciążeniu (np. liczbie wykonywanych transakcji, liczbie korzystających z systemu klientów)
  - przepustowość optymalna to maksymalna liczba wykonywanych transakcji
- *Czas odpowiedzi*
  - opóźnienie pomiędzy momentem wysłania żądania a pierwszą odpowiedzią serwera
- *Opóźnienie*
  - nie każdy dłuższy czas oczekiwania na odpowiedź jest spowodowany działaniem aplikacji
  - zwłoka spowodowana przez aplikację, system operacyjny, środowisko uruchomieniowe

Testowanie – testy wydajnościowe

Metodyka testowania wydajnościowego

1. Zbieranie wymagań
2. Zapis przypadków testowych
3. Automatyzacja przypadków testowych
4. Wykonanie przypadków testowych
5. Analiza otrzymanych wyników
6. Wykonanie benchmarków
7. Dostrajanie wydajności
8. Rekomendacja właściwej konfiguracji dla klienta (planowanie obciążenia)

Testowanie – testy wydajnościowe

1. Zbieranie wymagań
   - wymagane wyniki mogą zależeć od ustawień środowiska, mogą również nie być znane z góry
   - wymagania powinny być możliwe do przetestowania
   - wymagania muszą jasno określić, jakie parametry mierzone i ulepszane
   - wymagania testowe muszą być skojarzone z pożądanym stopniem ulepszenia wydajności aplikacji
2. Zapis przypadków testowych – Definiowanie przypadku testowego
   - lista operacji lub transakcji przewidzianych do testowania
   - opis sposobu wykonania tych operacji
   - lista produktów oraz parametrów wpływających na wydajność
   - szablon obciążenia
   - spodziewany wynik
   - porównanie wyników z poprzednimi wersjami / konkurencyjnymi aplikacjami

Testowanie – testy wydajnościowe

3. Automatyzacja przypadków testowych
   - testy wydajności są wielokrotnie powtarzane
   - testy muszą być zautomatyzowane, w najgorszym przypadku niemożliwe jest testowanie bez tego ułatwienia
   - testy wydajności muszą dawać dokładne informacje, ręczne przeliczanie może wprowadzać dodatkowy błąd pomiaru
   - testowanie jest uzależnione od wielu czynników – mogą one występować w różnych kombinacjach, które trudno wyprowadzić ręcznie
   - analiza wyników wymaga dostarczenia wielu informacji pomocniczych – logów, sposobów wykorzystania zasobów w określonych przedziałach czasu, co jest trudne lub niemożliwe do uzyskania ręcznie

Testowanie – testy wydajnościowe

4. Wykonanie przypadków testowych
   - zapis czasu początku i końca testu
   - zapis plików zawierających informacje o produkcie i systemie operacyjnym, ważnych dla powtarzalności testów i przyszłego debugowania
   - zużycie zasobów (CPU, pamięć, obciążenie sieci)
   - zapis konfiguracji testowej zawierający wszystkie czynniki środowiskowe / zbieranie informacji o wymaganych parametrach

Testowanie – testy wydajnościowe

5. Analiza otrzymanych wyników
   - obliczenia statystyczne dla otrzymanych wyników – średnia i odchylenie standardowe
   - usunięcie szumu informacyjnego i ponowne obliczenie statystyk
   - zróżnicowanie wyników uzyskanych z *cache* od wyników bezpośrednio uzyskanych z aplikacji
   - odróżnienie wyników testowania uzyskanych w sytuacji, gdy wszystkie zasoby były dostępne od wyników, gdy były uruchomione dodatkowe usługi w tle

Testowanie – testy wydajnościowe

5. Analiza otrzymanych wyników

- Czy zachowana jest wydajność testowanego systemu, gdy testy są wykonywane wielokrotnie?
- Jaka wydajność może być spodziewana w zależności od różnych konfiguracji (np. dostępnych zasobów)?
- Jakie parametry wpływają na wydajność?
- Jaki jest efekt scenariuszy z udziałem kilku różnych operacji dla czynników wydajnościowych?
- Jaki jest wpływ technologii (np. zastosowanie *cache*) na testy wydajnościowe?
- Jaki jest optymalny czas odpowiedzi / przepustowość dla różnego zbioru czynników wpływających na system – obciążenie, liczba zasobów i innych parametrów?
- Jakie obciążenie jest jeszcze możliwe do zaakceptowania i w jakich okolicznościach dochodzi do załamania wydajności?
- Czy wymagania wydajnościowe są spełnione i jak wygląda wydajność w porównaniu z poprzednimi wersjami lub spodziewanymi wynikami?
- Gdy nie ma jeszcze dostępu do wysokowydajnych konfiguracji, można na podstawie dostępnych wyników przewidzieć rezultaty na takich maszynach?

Testowanie – testy wydajnościowe

6. Wykonywanie benchmarków
   - identyfikacja transakcji / scenariuszy i konfiguracji testowych
   - porównanie wydajności różnych produktów
   - dostrajanie parametrów porównywanych produktów w celu uzyskania najlepszej wydajności
   - publikacja wyników testów
7. Dostrajanie wydajności
   - forkowanie procesów w celu umożliwienia równoległych transakcji
   - wykonywanie operacji w tle
   - powiększenie *cache* i pamięci operacyjnej
   - dostarczenie wyższego priorytetu dla często używanych operacji
   - zmiana sekwencji operacji

Testowanie – testy wydajnościowe

8. Planowanie obciążenia
   - wyszukiwanie, jakie zasoby i konfiguracja jest konieczna w celu osiągnięcia najlepszych rezultatów
   - ustalenie, jaką wydajność osiągnie klient przy aktualnie dostępnych zasobach zarówno sprzętowych, jak i softwerowych

Narzędzia do testowania wydajnościowego

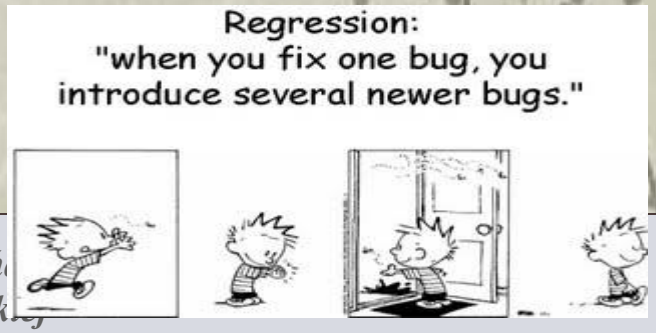
- narzędzia do testowania funkcjonalnego – zapis i odtwarzanie operacji w celu uzyskania parametrów wydajnościowych
- narzędzia do testowania obciążeniowego – symulacja warunków obciążeniowych bez potrzeby wprowadzania wielu użytkowników lub maszyn
- narzędzia do mierzenia zużycia zasobów

Testowanie – testy obciążeniowe

- testy przeciążeniowe – *stress testing*
  - założenie: zbyt wielu użytkowników, danych, czasu oraz malejące zasoby systemowe
  - badanie, czy system "zawiedzie" w oczekiwany sposób
  - wyszukiwanie defektów w aplikacji działającej w trybie awaryjnym
  - sprawdzanie konsekwencji utraty danych po awarii wywołanej nadmiernym obciążeniem
- testy obciążeniowe – *load testing*
  - duża liczba jednocześnie działających użytkowników / przeprowadzanych transakcji
  - utrzymanie takiego stanu przez określony w scenariuszu czas
  - jak wiele zapytań (*requests*) jest w stanie obsłużyć system w określonym przedziale czasu

Testowanie – testy regresyjne

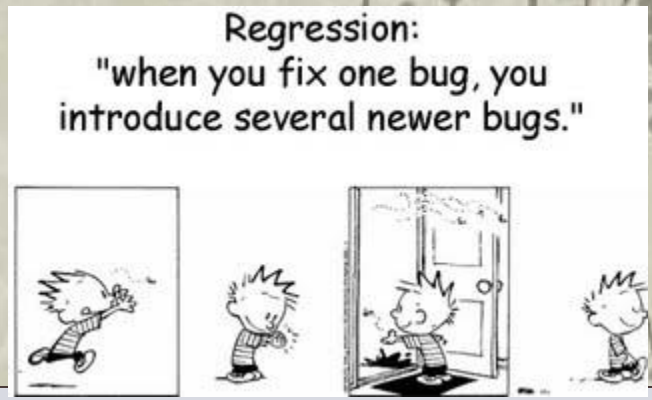
- testy regresyjne
- celem upewnienie się, że aplikacja działa po dokonaniu w niej modyfikacji, poprawieniu błędów lub po dodaniu nowej funkcji
  - upewnienie się, że wprowadzone nowe elementy lub naprawa defektów nie wpłynęła niekorzystnie na zbudowane wcześniej elementy systemu
- cecha: *powtarzalność*
- co dają?
  - wyszukanie błędów powstałych w wyniku zmian kodu / środowiska
  - ujawnienie wcześniej nie odkrytych błędów
  - dobry kandydat do automatyzacji ze względu na swoją powtarzalność
  - iteracyjne metodologie oraz krótkie cykle, w których dostarczane są kolejne funkcje powodują, że testy regresywne pozwalają upewnić się, czy nowe funkcje nie wpłynęły negatywnie na istniejące już i działające części systemu



Testowanie – testy regresyjne

Typy testów regresyjnych

- regularne testy regresyjne – wykonywane pomiędzy cyklami testowania w celu zapewnienia, że po wprowadzonych poprawkach elementy aplikacji testowane wcześniej nadal działają poprawnie
- końcowe testy regresyjne – wykonywane w celu walidacji ostatniej wersji aplikacji przed jej wypuszczeniem na rynek
  - uruchamiane na pewien okres czasu, ponieważ niektóre defekty ujawniają się dopiero po pewnej chwili (np. wycieki pamięci)



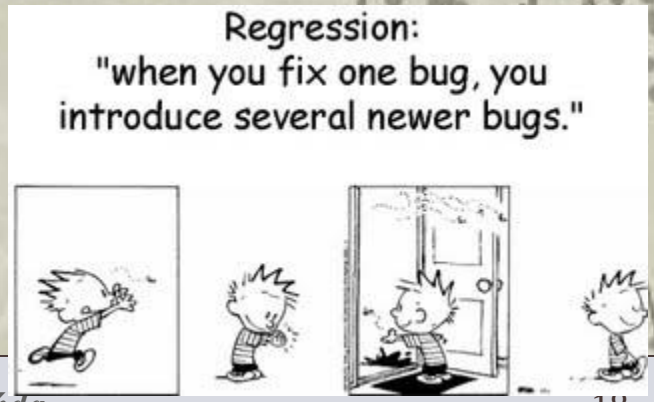
Testowanie – testy regresyjne

- wybór momentu testowania regresyjnego
  - znacząca liczba wstępnych testów została już przeprowadzona
  - została wprowadzona duża liczba poprawek
  - poprawki mogły wprowadzić efekty uboczne

Testowanie – testy regresyjne

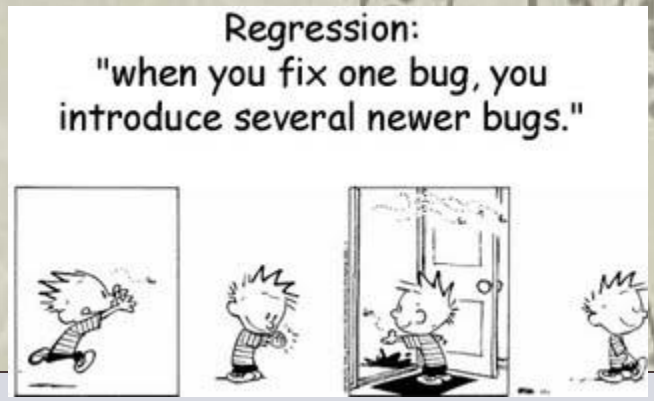
Metodyka testowania regresyjnego

1. Wykonanie początkowych testów *smoke* lub *sanity*
2. Zrozumienie kryteriów wyboru przypadków testowych
3. Klasyfikacja przypadków testowych pod względem różnych priorytetów
4. Metodyka selekcji przypadków testowych
5. Ponowne ustawianie przypadków testowych do wykonania
6. Podsumowanie cyklu regresyjnego

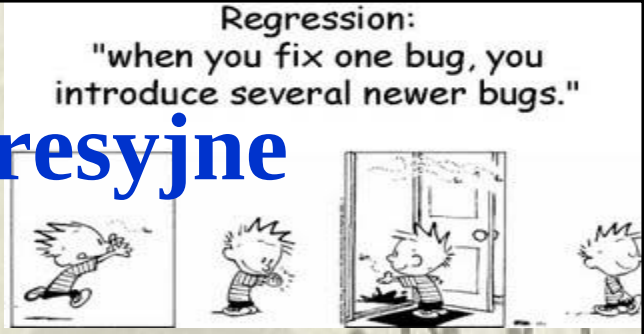


Testowanie – testy regresyjne

1. *Smoke test* – testy „dymne” składają się z:
   - identyfikacji podstawowych funkcji, jakie musi spełniać przygotowywany produkt
   - przygotowanie przypadku/ów testowego w celu zapewnienia, że podstawowe funkcje działają i dodanie go do zestawu testów
   - zapewnienie, że ten zestaw przypadków testowych będzie wykonywany po każdej budowie systemu
   - w przypadku niepowodzenia tego testowania poprawienie lub wycofanie zmian



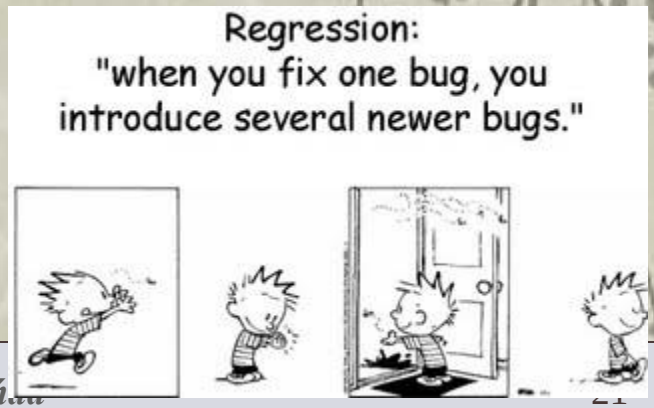
Testowanie – testy regresyjne



2. Zrozumienie wyboru przypadków testowych
   - wymaga informacji na temat:
     - wprowadzonych poprawek i zmian do bieżącej wersji aplikacji
     - sposobów testowania wprowadzonych zmian
     - wpływu, jaki mogą mieć wprowadzone poprawki na resztę systemu
     - sposobu testowania elementów, na które mogły mieć wpływ wprowadzone poprawki
3. Klasyfikacja przypadków testowych
   - podział ze względu na priorytet dla użytkownika
     - priorytet 0 – testy sprawdzają podstawowe funkcje; najważniejsze z punktu widzenia użytkownika i twórców systemu
     - priorytet 1 – testy używają podstawowych i normalnych ustawień dla aplikacji; ważne z punktu widzenia użytkownika i twórców systemu
     - priorytet 2 – testy dostarczają poglądowych wartości odnośnie projektu

Testowanie – testy regresyjne

4. Metodyka wyboru przypadków testowych
   - gdy wpływ wprowadzonych poprawek na resztę kodu jest mały można wybrać tylko wybrane testy ze zbioru przypadków testowych
     - można wybrać testy o priorytecie 0, 1 lub 2
   - gdy wpływ wprowadzonych poprawek jest średni, należy wybrać wszystkie testy o priorytecie 0 i 1
     - wybór testów o priorytecie 2 jest możliwy, lecz nie jest konieczny
   - gdy wpływ wprowadzonych poprawek jest duży, należy wybrać wszystkie testy o priorytecie 0 i 1 oraz starannie wybrane testy o priorytecie 2

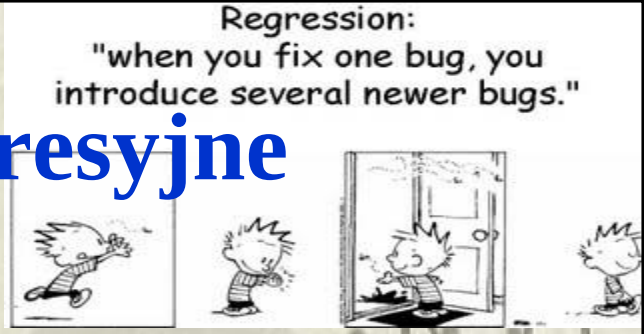


Testowanie – testy regresyjne

5. Ustawianie przypadków testowych – wykonywane, gdy:
   - wystąpiły duże zmiany w produkcie
   - nastąpiła zmiana w procedurze budowania wpływająca na system
   - występują duże cykle wprowadzania kolejnych wersji, podczas których pewne przypadki testowe nie były wykonywane przez dłuższy czas
   - produkt jest w finalnej wersji z kilkoma wybranymi przypadkami testowymi
   - wystąpiła sytuacja, gdy spodziewane wyniki testów są zupełnie inne w porównaniu z poprzednim cyklem

1. Przypadki testowe będące w relacji z wprowadzonymi poprawkami mogą być usunięte, gdy kolejne wersje nie powodują zgłaszania błędów.
2. Usunięto z aplikacji jakąś funkcję, przypadki testowe powiązane z nią również powinny być usunięte.
3. Przypadki testowe kończą się za każdym razem wynikiem pozytywnym.
4. Przypadki testowe powiązane z kilkoma negatywnymi warunkami testowymi (nie powodującymi wykrywanie defektów) mogą być usunięte.

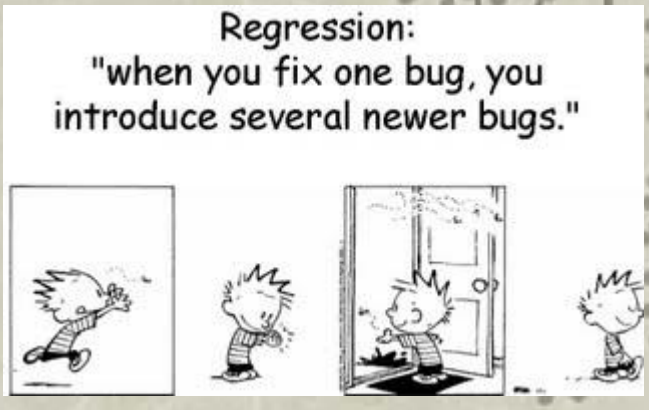
Testowanie – testy regresyjne



6. Podsumowanie cyklu regresyjnego
   - Przypadek testowy zakończył się sukcesem w poprzedniej wersji i niepowodzeniem przy obecnej – regresja zakończona niepowodzeniem.
   - Przypadek testowy zakończył się niepowodzeniem w poprzedniej wersji i przeszedł pomyślnie test w obecnej wersji – można założyć, że wprowadzone poprawki rozwiązały problem.
   - Przypadek testowy kończy się niepowodzeniem w obu wersjach (poprzedniej i obecnej) przy braku wprowadzonych poprawek dla tego przypadku testowego – można oznaczać, że test nie powinien być wliczany do wyników testowania.
   - Przypadek testowy kończy się niepowodzeniem w poprzedniej wersji, a w obecnej działa z dobrze udokumentowanym obejściem problemu, to regresję można uznać za zakończoną sukcesem.

Testowanie – testy regresyjne

- porady dotyczące testowania regresyjnego
  - regresje mogą być używane ze wszystkimi wersjami (wprowadzającymi poprawki, nową funkcję)
  - powiązanie problemu z przypadkiem testowym zwiększa jakość regresji
  - testy regresyjne powinny być wykonywane codziennie
  - lokalizacja i usunięcie problemu powinna ochronić produkt przed wpływem problemu i poprawki



Testowanie – testy ad hoc

- **Metody testowania ad hoc**
  - Testowanie losowe
  - Testowanie koleżeńskie
  - Testowanie parami
  - Testowanie badawcze
  - Testowanie iteracyjne
  - Testowanie zwinne i ekstremalne
  - Zasiew defektów

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie koleżeńskie
  - dwaj członkowie zespołu (zazwyczaj programista i tester) pracują nad identyfikacją i usunięciem problemu w oprogramowaniu
  - sprawdza się np. standardy kodowania, definicje zmiennych, kontrolę błędów, stopień udokumentowania kodu
  - używa się testów białej i czarnej skrzynki
- Testowanie parami
  - wykonywane przez dwóch testerów na tej samej maszynie
  - celem jest wymiana pomysłów pomiędzy osobami, lepsze zrozumienie wymagań stawianych projektowi, zwiększenie umiejętności testowania oprogramowania
  - podział na role – *tester* i skryba – jedna osoba wykonuje testy, druga robi notatki

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie badawcze
  - na podstawie poprzednich doświadczeń z technologią, podobnymi systemami wnioskuje się, jakie błędy program może zawierać
  - technika użyteczna szczególnie w przypadku systemów nieprzetestowanych wcześniej, nieznanych albo niestabilnych
  - używana, gdy nie do końca wiadomo, jakie testy należy jeszcze dołożyć do istniejącej bazy testów

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie badawcze – techniki testowania badawczego
  - zgadywanie – na podstawie wcześniejszych doświadczeń z podobnymi produktami tester zgaduje, która część programu zawiera najwięcej bledów
  - użycie diagramów architektonicznych i przypadków użycia – tester używa do testowania diagramów – architektoniczne przedstawiają, jak wyglądają powiązania pomiędzy modułami, przypadki użycia natomiast dostarczają informacji o systemie z punktu widzenia użytkownika
  - studiowanie poprzednich defektów – pozwala na ustalenie, jakie części systemu są najbardziej narażone na defekty

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie badawcze – techniki testowania badawczego
  - obsługa błędów – system po wykonaniu niepoprawnej operacji użytkownika może być niestabilny
    - obsługa błędów powinna dostarczać czytelnych komunikatów diagnostycznych i/lub sposobów zaradzenia problemowi
    - testuje się sytuacje, w których dochodzi do tego typu błędów
  - dyskusja – szczegóły implementacyjne użyteczna dla testowania systemu są omawiane na spotkaniach
  - użycie kwestionariuszy i wykazów – odpowiedzi na pytania (np. jak, gdzie, kto i dlaczego) pozwalają ustalić, jakie obszary należy przebadać w systemie

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie iteracyjne
  - używany, gdy dodawane są wymagania systemowe z każdą wersją oprogramowania (np. model spiralny)
  - ma zapewniać, że zaimplementowana wcześniej funkcja nadal jest testowana – wymagane powtarzalne testy
  - zmiany w testach wykonywane w każdej iteracji
  - nie wszystkie rodzaje testów są wykonywane w wcześniejszych iteracjach (np. testy wydajnościowe)

Testowanie – testy ad hoc

Metody testowania ad hoc

- Testowanie zwinne i ekstremalne
  - ma na celu zapewnienie, że wymagania użytkownika są spełnione w odpowiednim czasie
  - użytkownik jest częścią zespołu przygotowującego oprogramowanie (odpowiada na pytania)
  - oprogramowanie rozwija się w iteracjach
  - testerzy nie są traktowani jako osobna grupa wykonująca swoje zadania
  - testerzy nie muszą wysyłać raportów i czekać na odpowiedź innych członków zespołu
  - testerzy traktowani jako pomost pomiędzy programistami (znającymi technologie i sposoby implementacji pomysłów) a użytkownikami (znającymi swoje wymagania) – wyjaśniają różne punkty widzenia

Testowanie – testy ad hoc

Metody testowania ad hoc

- Zasiew defektów
  - polega na tym, że część członków zespołu wprowadza do programu błędy, które pozostała część zespołu próbuje zidentyfikować
  - wprowadzane błędy są podobne do rzeczywistych defektów
  - w czasie wyszukiwania zasianych błędów można zidentyfikować również błędy, które nie były wprowadzone celowo
  - może być wskazówką efektywności procesu testowania, może mierzyć liczbę usuniętych błędów i pozostających w kodzie

$$liczukb = (liczbz / liczbwzb) * liczwab$$

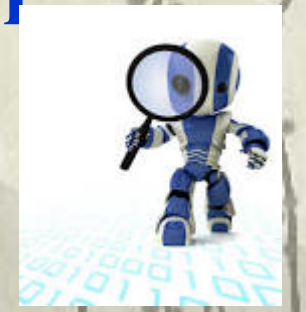
liczukb – liczba ukrytych błędów

liczbz – liczba błędów zasianych

liczbwzb – liczba wykrytych zasianych błędów

liczwab – liczba wykrytych autentycznych błędów

Testowanie – testy użyteczności



Testowanie użyteczności i dostępności

- testy użyteczności są subiektywne
- postrzeganie dobrej użyteczności zależy od użytkownika
- interfejs użytkownika może być stworzony w czasie projektowania systemu – tworzenie interfejsu użytkownika nie wpływa jednak w bezpośredni sposób na wymagania funkcjonalne stawiane aplikacji
- użyteczność testuje się z punktu widzenia użytkowników
- sprawdza, czy produkt jest łatwy do użycia przez różne grupy użytkowników
- proces sprawdzający rozbieżności pomiędzy zaprojektowanym interfejsem a oczekiwaniami użytkownika

Testowanie – testy użyteczności

- łatwość użycia
  - może być różna dla różnych grup użytkowników
- szybkość działania
  - zależy od wielu czynników – np. systemowych lub sprzętowych
- estetyka aplikacji
  - zależna od postrzegania aplikacji przez użytkownika
- testowanie użyteczności obejmuje
  - aplikację – pozytywne i negatywne testowanie
  - dokumentację programu
  - komunikaty aplikacji
  - media, z jakich korzysta aplikacja

Testowanie – testy użyteczności

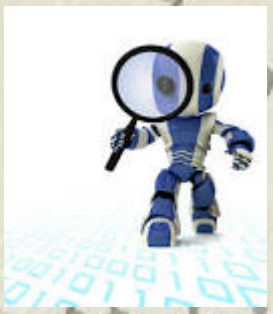
- czynniki obiektywne w testowaniu użyteczności, np.:
  - liczba kliknięć myszy
  - liczba klawiszy do naciśnięcia
  - liczba menu do przejścia
  - liczba poleceń potrzebnych do wykonania zadania
- dwie grupy najbardziej przydatne do testowania użyteczności
  - typowi przedstawiciele aktualnej grupy użytkowników – umożliwiają wyodrębnienie wzorców korzystania z aplikacji
  - nowi użytkownicy nieposiadający przyzwyczajeń – mogą zidentyfikować problemy z użytecznością niedostrzegalne przez użytkowników korzystających z aplikacji dłuższy czas
- moment wykonania testów użyteczności
  - planowanie ↔ uwzględnienie testów użyteczności w planie testowania
  - projektowanie ↔ projektowanie walidacji użyteczności
  - kodowanie ↔ testowanie użyteczności

Testowanie – testy użyteczności

Projektowanie użyteczności

- arkusze stylów
  - grupują elementy interfejsu użytkownika
  - zapewniają spójność interfejsu użytkownika
- prototypy ekranów aplikacji
  - projektowane tak, jak będą wyglądały dla użytkownika, bez połączenia z funkcjami produktu – umożliwia odrębne testowanie
  - symulują wykonywanie różnych ścieżek programu – wyświetlanie komunikatów i okien
- projektowanie na papierze
  - projekt okien aplikacji, układu elementów, menu narysowany na papierze jest wysyłany do użytkownika w celu akceptacji
- projektowanie *layoutu*
  - pomocne przy układaniu różnorodnych elementów aplikacji na ekranie
  - zmiana układu elementów aplikacji może prowadzić do błędów użytkownika

Testowanie – testy użyteczności



- problemy z testowaniem użyteczności
  - grupy użytkowników posiadające niezgodne ze sobą wymagania
  - użytkownicy mogą nie być w stanie wyrazić użycia produktu prowadząc do problemów
  - różne grupy użytkowników
    - eksperci – mogą znajdować obejścia istniejących problemów
    - początkujący – nie mają odpowiedniej wiedzy o użyciu produktu
- obserwacja wzorców zachowania użytkowników
  - W jaki sposób użytkownicy wykonują operacje w celu osiągnięcia określonego zadania?
  - Ile zajmuje im to czasu?
  - Czy czas odpowiedzi systemu jest satysfakcjonujący?
  - W jakich okolicznościach napotykają na problemy?
  - Jak je omijają?
  - Jakie zachowania aplikacji powodują zmieszanie użytkownika?

Testowanie – testy użyteczności

Czynniki jakości użyteczności

- czytelność
  - produkt i jego dokumentacja posiada prostą i logiczną strukturę
  - operacje są grupowane na podstawie scenariuszy uzyskanych od użytkownika
  - często wykonywane operacje są bardziej widoczne w interfejsie użytkownika
  - komponenty programu używają terminologii użytkownika
- spójność z
  - uznanymi standardami
  - wyglądem aplikacji dla określonej platformy
  - poprzednimi wersjami aplikacji
  - innymi produktami danego producenta

Testowanie – testy użyteczności

Czynniki jakości użyteczności

- „nawigowalność”
  - łatwość wyboru różnych operacji wykonywanych przed produkt
  - np. minimalizacja liczby kliknięć myszy potrzebnych do wykonania operacji
- czas odpowiedzi aplikacji
  - szybkość reakcji produktu na żądania użytkownika – różne od wydajności aplikacji
  - np. wizualizacja, ile procent danych zostało przetworzonych przez aplikację

Testy automatyczne i manualne

- testy automatyczne – zalety:
  - testy automatyczne świetnie uzupełniają testy manualne
  - obliczenia, które człowiekowi mogą sprawić wiele trudności przez maszynę zostaną przetworzone w ułamku sekundy
  - powtarzalność, np. regresja
  - wraz z upływem czasu organizm człowieka się męczy, spada koncentracja i coraz trudniej jest mu ustrzec się błędów, automat się nie męczy
  - mogą być uruchamiane cyklicznie (np. codziennie wraz z *daily buildem* oprogramowania)
  - pozwalają oszczędzić czas testerów
  - błędy związane z czynnikiem ludzkim praktycznie wyeliminowane

Testy automatyczne i manualne

- testy automatyczne – wady i zarazem zalety testów manualnych
  - nie mogą zastąpić testów manualnych
  - komputery mają trudności z przetwarzaniem i interpretowaniem pewnych informacji, które są naturalne dla człowieka
  - obsługa automatycznego środowiska testowego wymaga sporych nakładów
  - testy manualne często nie wymagają wiedzy z programowania

Testowanie oprogramowania

Planowanie testowania

1. Wybór zakresu testowania wraz z identyfikacją, co powinno być przetestowane, a czego można nie testować
2. Wybór sposobu wykonania testów – podział zadania na mniejsze zadania i przyjęcie strategii przeprowadzenia zadania
3. Wybór zasobów, jakie należy przetestować – zarówno komputerowe, jak i te związane z ludźmi
4. Przyjęcie ram czasowych, w których będą wykonywane testy
5. Ocena ryzyka, jakie musi być podjęte z wykonaniem wcześniej wymienionych zadań, wraz z odpowiednim planem awaryjnym i migracyjnym

Testowanie oprogramowania

- zasięg testowania
  - wyodrębnienie podstawowych składowych końcowego produktu
  - podział produktu na zbiór podstawowych składowych
  - ustawienie priorytetów testów dla różnych aspektów oprogramowania
  - zadecydowanie, co należy testować, a co nie
  - estymacja ilości zasobów potrzebnych do testowania na podstawie zebranych wcześniej danych

Testowanie oprogramowania

Wybór priorytetów

- cechy, które są nowe i krytyczne dla nowego wydania produktu
  - spełnienie oczekiwań użytkownika
  - w nowym kodzie spodziewane wiele nowych błędów
- cechy, których nieprawidłowe działanie może spowodować katastrofalne skutki – np. mechanizm odzyskiwania bazy danych
- cechy oprogramowania, dla których spodziewane jest skomplikowane testowanie
- cechy, które są rozszerzeniem mechanizmów z obecnej wersji oprogramowania, które były źródłem wielu błędów
- wymienione cechy rzadko można zidentyfikować jako występujące samodzielne, częściej spotykane są w różnych kombinacjach

Testowanie oprogramowania

- wybór strategii testowania
  - wybór testów potrzebnych do przetestowania funkcjonalności produktu
  - wybór konfiguracji i scenariuszy potrzebnych do testowania funkcjonalności produktu
  - wybór testów integracyjnych, koniecznych do sprawdzenie, czy zestawienie elementów prowadzi do poprawnej ich pracy
  - wybór potrzebnej walidacji lokalizacji produktu
  - wybór koniecznych testów niefunkcjonalnych
- ustanowienie kryteriów testowania
  - wybór jasnych kryteriów dla danych testowych i dla spodziewanych wyników
  - wybór warunków, w których testy powinny zostać wstrzymane
    - wykryto szereg błędów
    - napotkano na problem, który uniemożliwia dalsze testowanie
    - wydano nową wersję, która ma na celu naprawienie błędów w obecnej wersji

Testowanie oprogramowania

- identyfikacja dostarczanych wyników testowania
  1. planowanie samych testów
  2. identyfikacja przypadków testowych
  3. przypadki testowe wraz z ich automatyzacją
  4. logi wyprodukowane przez wykonane testy
  5. raporty podsumowujące testy
- ocena rozmiaru i wysiłku koniecznego do wykonania testów
  - ocena rozmiaru – oszacowanie liczby koniecznych do wykonania testów
  - rozmiar produktu przeznaczonego do testowania
    - liczba linii kodu
    - liczba punktów funkcyjnych
    - liczba ekranów, raportów i transakcji
  - zakres wymaganej automatyzacji
  - liczba platform i środowisk koniecznych do rozważenia

Testowanie – proces testowania

- specyfikacja przypadku testowego
  - cel testu
  - elementy testowane
  - konieczne środowisko uruchomieniowe
  - dane wejściowe dla testu
  - kroki potrzebne do wykonania testu
  - spodziewane wyniki
  - kroki potrzebne do porównania wyników ze spodziewanymi wynikami
  - relacje pomiędzy testem a innymi testami
- identyfikacja potencjalnych kandydatów dla automatyzacji
- zaprogramowanie i przyłączenie przypadków testowych
- wykonanie przypadków testowych
- zbieranie i analiza metryk
- przygotowanie raportu podsumowującego

Testowanie – automatyzacja

- automatyzacja pozwala zaoszczędzić czas – testy mogą być uruchamiane i wykonywane szybciej
- testowanie automatyczne może być bardziej wiarygodne
- automatyzacja otwiera możliwość lepszego wykorzystania globalnych zasobów
  - testy mogą być uruchamiane przez większość czasu w różnych strefach czasowych
- niektóre testy nie mogą być uruchomione bez automatyzacji, np. testy obciążeniowe i wydajnościowe systemu, wymagającego zalogowania tysięcy użytkowników

Testowanie – automatyzacja

- nagrywanie i odtwarzanie
  - nagrywanie czynności użytkownika – kliknięcia myszą, akcje wykonywane klawiaturą – i ich późniejsze odtwarzanie w tej samej kolejności
- testy sterowane danymi
  - metoda pozwala na generowanie testów na podstawie zestawów danych wejściowych i wyjściowych
- testy sterowane akcjami
  - wszystkie akcje, jakie mogą być wykonane przez aplikacje są generowane automatycznie na podstawie zbioru kontrolek, jakie zostały zdefiniowane do testowania

Testowanie – automatyzacja

Identyfikacja testów podatnych na automatyzację

- testowanie wydajności, niezawodności, obciążeniowe
  - testy wymagające wielu użytkowników i dużego czasu przeznaczonego na ich wykonanie
- testowanie regresyjne
  - powtarzalne, uruchamiane wielokrotnie
- testowanie funkcjonalne
  - może wymagać skomplikowanych przygotowań – automatyzacja umożliwi uruchamianie przygotowanych przez ekspertów testów mniej doświadczonym testerom

Testowanie – automatyzacja

Architektura dla automatyzacji testowania

- moduły zewnętrzne
  - baza przypadków testowych
  - baza błędów
- moduły plików konfiguracyjnych i scenariuszy
  - scenariusze – informacja, jak uruchomić określony test
  - pliki konfiguracyjne – zawierają zbiór zmiennych używanych w automatyzacji
- moduły narzędzi i wyników
  - narzędzia wspierają proces tworzenia przypadków testowych
  - wyniki są zachowywane w celu dalszej analizy, umożliwiają również przygotowanie przypadków testowych dla dalszych faz tworzenia oprogramowania
- moduły generacji raportów i metryk
  - generowanie raportów podsumowujących daną fazę testowania oprogramowania

Testowanie – wymagania

Wymagania stawiane testom

- brak ustawionych na sztywno wartości dla zestawów testowych
- zestawy testowy powinny mieć możliwość dalszej rozbudowy
  - dodany test nie powinien wpływać na inne przygotowane wcześniej testy
  - dodanie nowego testu nie powinno wymagać ponownego przeprowadzenia już przeprowadzonych testów
  - dodanie nowego zestawu testowego nie powinno wpływać na inne zestawy testowe
- zestawy testowe powinny mieć możliwość ponownego użycia
  - test powinien robić jedynie to, co jest spodziewane, że robi
  - testy powinny być modularne
- automatyczne ustawianie i czyszczenie po teście
- niezależność przypadków testowych

Testowanie – wymagania

Wymagania stawiane testom

- izolacja testów w czasie wykonania
  - elementy środowiska mogą wpływać na wyniki testów – blokowanie niektórych zdarzeń
- standardy kodowania i struktury katalogowej
  - ułatwia zrozumienie nowym pracownikom przypadków testowych
  - ułatwia przenoszenie kodu pomiędzy platformami
  - wymuszenie struktury katalogowej może ułatwić zrównoleglenie testów
- selektywne wykonanie przypadków testowych
  - ułatwienie w wyborze z: wielu zestawów testowych → wielu programów testujących → wielu przypadków testowych
- losowe wykonanie przypadków testowych
  - wybór niektórych testów z zestawu przypadków testowych
- równoległe wykonanie przypadków testowych

Testowanie – wymagania

Wymagania stawiane testom

- zapętlenie przypadków testowych
  - testy wykonywane iteracyjnie dla znanej z góry liczby iteracji
  - testy wykonywane w określonym przedziale czasu
- grupowanie przypadków testowych
  - umożliwia wykonanie testów w przedefiniowanej kombinacji scenariuszy
- wykonanie przypadków testowych w oparciu o wcześniejsze wyniki
- zdalne wykonanie przypadków testowych
  - testy mogą wymagać więcej niż jednej maszyny do ich uruchomienia
  - umożliwienie uruchomienia testów, zbieranie logów, informacji o postępie testów
- automatyczne zachowywanie danych testowych
- schematy raportowania
  - jakie informacje mają być uwzględnione w raporcie

Testowanie – metryki

Metryki przydatne przy testowaniu

- metryki wyprowadzają informacje z surowych danych w celu ułatwienia podejmowanie decyzji
  - relacje pomiędzy danymi
  - przyczyny i efekty korelacji pomiędzy zaobserwowanymi danymi
  - wskaźniki, jak dane mogą być użyte w dalszym planowaniu i ulepszaniu produktu
- wysiłek – czas spędzony na jakiejś częściowej aktywności lub fazie
- kroki w zdobywaniu metryki programu
  - identyfikacji tego, co należy zmierzyć
  - transformacja tego, co zostało zmierzone do metryki
  - podjęcie decyzji, co do wymagań operacyjnych
  - wykonanie analizy metryk
  - wybór i wykonanie akcji
  - sprecyzowanie miar i metryk

Testowanie – metryki

Przydatność metryk w testowaniu

- mierzenie postępu w testowaniu
  - produktywność wykonywanych testów
  - data zakończenia testów (oszacowana)
    - na podstawie ilorazu liczby testów do wykonania przez liczbę testów wykonywanych na dzień
  - liczba dni potrzebnych na usunięcie błędów

    (liczba błędów do usunięcia + liczba przewidywanych błędów) / zdolność usuwania błędów
- oszacowanie daty wydania finalnej wersji
  - max (liczba dni potrzebnych na testy, liczba dni potrzebnych do usunięcia błędów)
- oszacowanie jakości wydanego oprogramowania
- wybór elementów, jakie powinny być wzięte pod uwagę w planowaniu wydania oprogramowania

Testowanie – metryki

Typy metryk

- metryki projektu – jak projekt jest planowany i wykonywany
  - wariancja włożonego wysiłku
  - wariancja harmonogramu
  - rozkład wysiłku
- metryki postępu – śledzące, jak wygląda postęp w różnych rodzajach działalności związanych z projektem
  - wskaźnik odnalezionych błędów
  - wskaźnik naprawionych błędów
  - wskaźnik pozostających błędów
  - wskaźnik priorytetu pozostających błędów
    - aktywność związana z programowaniem
    - aktywność związana z testowaniem

Testowanie – metryki

Typy metryk

- metryki produktywności – zbierające różnego rodzaju miary, które mają wpływ na planowanie testowania
  - liczba błędów na 100 godzin testowania
  - liczba przypadków testowych na 100 godzin testowania
  - liczba zaimplementowanych przypadków testowania na 100 godzin
  - liczba błędów na 100 przypadków testowych
- różnego rodzaju aktywności, podstawowy wkład do projektu i harmonogram prac są wejściem dla początku projektu
- aktualny wkład i czas poświęcony na działalność są dodawane w miarę jak jest rozwijany projekt
- skorygowany wkład i harmonogram – ponownie obliczane, gdy jest to potrzebne

Testowanie oprogramowania

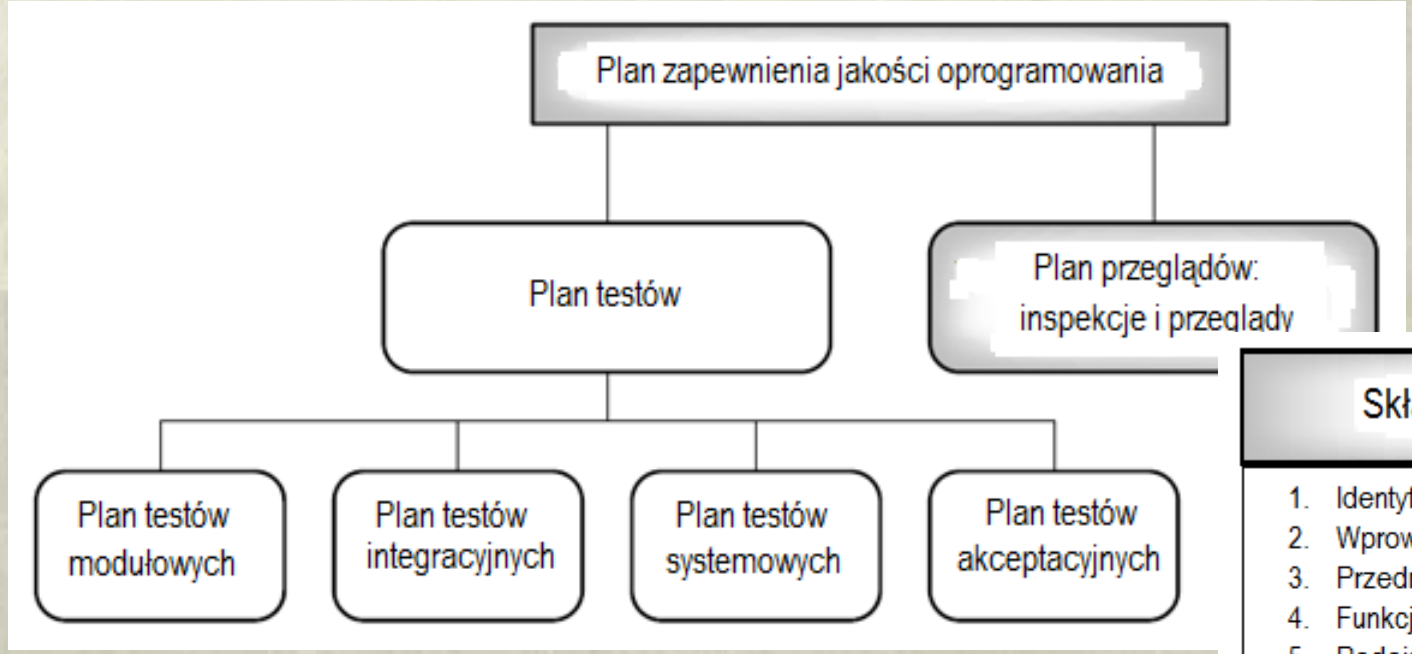
- Kiedy zakończyć testy ?
  - terminy, np. termin wydania *release*, termin zakończenia testowania
  - przypadki testowe ukończone z określonym procentem „przejścia”
  - budżet na testowanie „wyczerpany”
  - pokrycia kodu, funkcji lub wymagań osiągnęło określony poziom
  - wskaźnik błędów poniżej ustalonego poziomu
  - poziom ryzyka w projekcie poniżej akceptowanego poziomu

Testowanie – dokumentacja testowa

- **plan testów** – dokument IEEE 829 sugeruje rozdziały:
  1. Test plan identifier
  2. Introduction
  3. Test items
  4. Features to be tested
  5. Features not to be tested
  6. Approach
  7. Item pass/fail criteria
  8. Test deliverables
  9. Suspension criteria and resumption requirements
  10. Testing tasks
  11. Environmental needs
  12. Responsibilities
  13. Staffing and training needs
  14. Schedule
  15. Risks and contingencies
  16. Approvals

Testowanie oprogramowania

Hierarchia planów testowych



Składniki Test Planu

1. Identyfikator planu testów
2. Wprowadzenie
3. Przedmiot testów
4. Funkcje do przetestowania
5. Podejście
6. Kryteria zaliczenia/oblania testów
7. Kryteria zawieszenia i wznowienia
8. Produkty testowania
9. Zadania testowe
10. Środowisko testów
11. Obowiązki
12. Kadry i szkolenia
13. Grafiki
14. Ryzyko
15. Koszty testowania
16. Zatwierdzenie

Testowanie – dokumentacja testowa

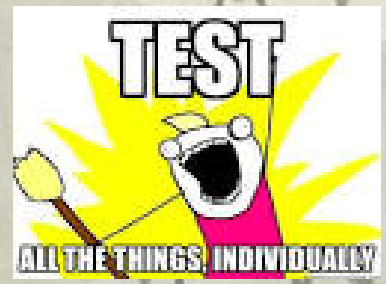
- dokumentacja testowa – **test case**
  - Test Script Title
  - Test Design Title
  - ID
  - Version
  - Creation Date
  - Intended Tester
  - Product
  - Product version
  - Intended Platform/OS
  - Priority
  - Summary
  - Preconditions
  - Test Steps
  - Expected Results

Testowanie – dokumentacja testowa

- dokumentacja testowa – **bug report**
  - Title
  - ID
  - Version
  - Creation Date
  - Intended Tester
  - Product
  - Product version
  - Intended Platform/OS
  - Priority
  - Classification
  - Can it be reproduced
  - Attachments
  - Summary
  - Preconditions
  - Test Steps
  - Expected Results
  - Results
  - Comments

Testowanie oprogramowania – tester

- podejście do testowania – sposób myślenia podczas testowania różny od nastawienia podczas tworzenia oprogramowania:
  - dociekliwość, powątpiewanie, dbałość o szczegóły
  - ciekawość, nastawienie „co jeśli?”
  - „profesjonalny pesymizm”, krytyczne spojrzenie
  - poszukiwanie, badanie, zagłębianie się



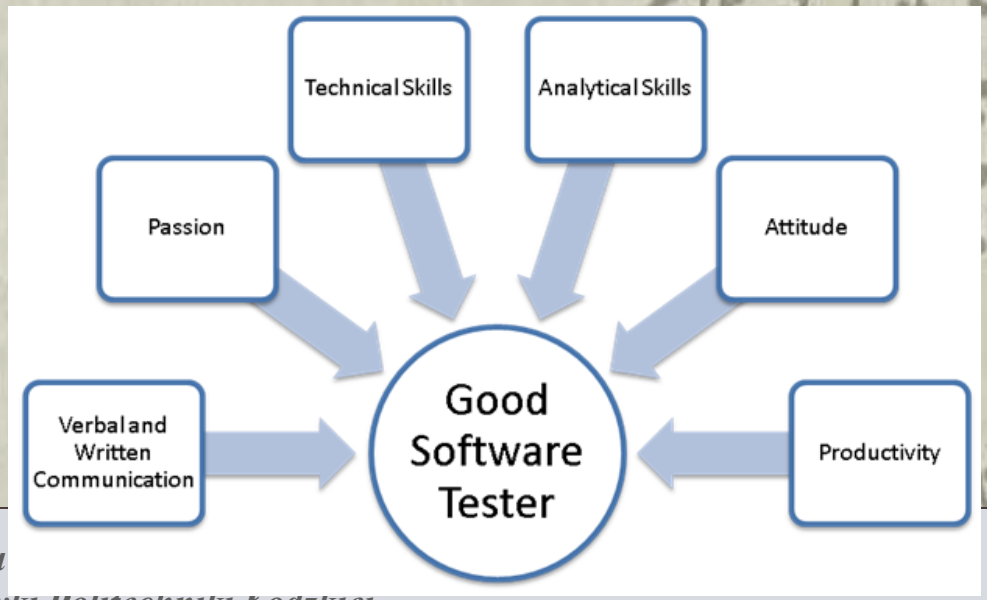
- podejście testera może być kontrowersyjne dla programistów
- ryzyko związane z testowaniem własnego kodu – rozwiązanie: **niezależność testowania** – korzyści:
  - wzrost efektywności znajdowania błędów
  - spojrzenie na oprogramowanie pod innym kątem
  - specjalistyczna wiedza o testowaniu
  - brak powiązania z produktem
  - weryfikacja założeń poczynionych na etapie specyfikacji wymagań i implementacji oprogramowania

Testowanie oprogramowania – tester

- **niezależność testowania** – poziomy niezależności
  1. testy projektowane przez autora kodu (programistę) – brak niezależności
  2. testy projektowane przez inną osobę z zespołu programistów
  3. testy projektowane przez osoby z odrębnego zespołu w organizacji (np. zespołu testowego)
  4. testy projektowane przez inną organizację lub firmę (*outsourcing*)
  5. testy projektowanie przez testerów z organizacji klienta bądź użytkowników
  6. niezależni specjaliści testowi do wybranych typów testów np. testów użyteczności, wydajności
- rozwiązania niezależności
  - dla dużych skomplikowanych projektów najlepiej kilka poziomów testowania realizowanych przez niezależnych testerów
  - programiści w testowaniu szczególnie na niższych poziomach testów
  - niezależni testerzy mogą mieć autorytet do definiowania procesu testowego, ale tylko przy wyraźnej zgodzie kierownictwa

Testowanie oprogramowania – tester

- cechy dobrego testera według *testerzy.pl*
- Steven Miller w artykule "*The Seven Habits of Highly Effective Testers*" opisuje różnice między testerem a dobrym testerem:
  1. Bądź proaktywny
  2. Zaczynając myśl już o końcu
  3. Najważniejsze rzeczy na początku
  4. Myśl w kategoriach *Win/Win*
  5. Na początku zrozum, potem postaraj się być zrozumianym
  6. Staraj się o synergię
  7. Bądź ostrzejszy niż brzytwa

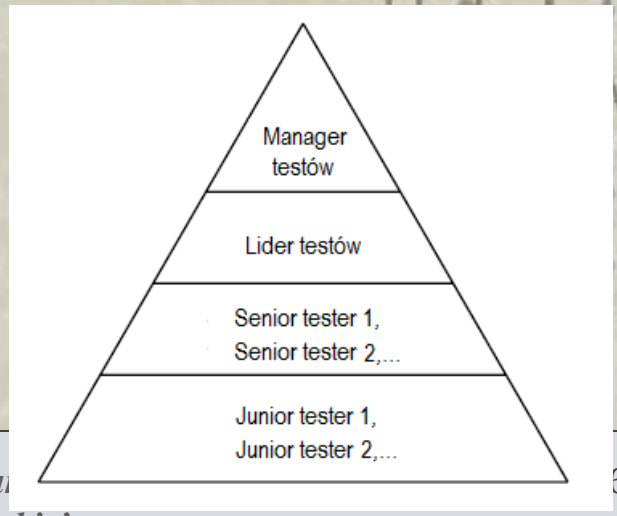


Testowanie oprogramowania – tester

- Manager testów – jest osobą odpowiedzialną za wszelkie sprawy związane z testowaniem i zapewnieniem jakości
- obowiązki: ustalenie strategii testów, interakcja z klientem, planowanie testów, dokumentacja testów, kontrola i monitorowanie postępów, szkolenia, pozyskanie odpowiednich narzędzi, udział w inspekcjach i przeglądach, audyt testów, repozytorium testów
- w sprawach kadrowych: zatrudnianie, zwalnianie i ocena pracy członków zespołu
- Lider testów – pełni funkcję asystenta managera testów i współpracuje z grupą testerów przy konkretnych projektach
- obowiązki: planowanie testów, nadzór kadr oraz wykonywanie bieżących raportów

Testowanie oprogramowania – tester

- Senior tester – specjalista ds. testów, projektuje, tworzy i przeprowadza testy oraz zajmuje się tworzeniem środowiska testów (jarzma testowego)
- ma swój udział w planowaniu testów, pomocy technicznej i tworzeniu repozytoriów błędów
- Junior tester – młodszy tester, zazwyczaj osoby nowo zatrudnione
- głównym celem jest nabieranie doświadczenia podczas uczestnictwa przy projektowaniu i przeprowadzaniu testów
- mogą brać udział w przeglądaniu instrukcji, pomocy technicznej oraz utrzymywaniu repozytoriów testów i błędów



Testowanie oprogramowania – tester

Cechy dobrego testera

Umiejętności techniczne

Wiedza:
Zasad i praktyk tworzenia oprogramowania
Strategie i metodologie testowania
Sieci, bazy danych i systemy operacyjne
Zagadnienia procesu testowania
Zarządzanie konfiguracją
Dokumentacja testowa
Zaganienia utrzymania jakości

Umiejętności:
Planowanie, tworzenie i przeprowadzanie przypadków testowych
Definiowanie, zbieranie i analizowanie danych testowych
Chęć i motywacja do korzystania na narzędzi testowych

Tester

Umiejętności osobiste i kierownicze

Zdolności organizacyjne i planowania
Szczegółowość
Determinacja w znajdowaniu i roziązywaniu problemów
Umiejętność pracy w zespole
Dzielenie się wiedzą
Praca z klientem / użytkownikiem
Rozwinięte zdolności komunikacyjne
Kreatywność

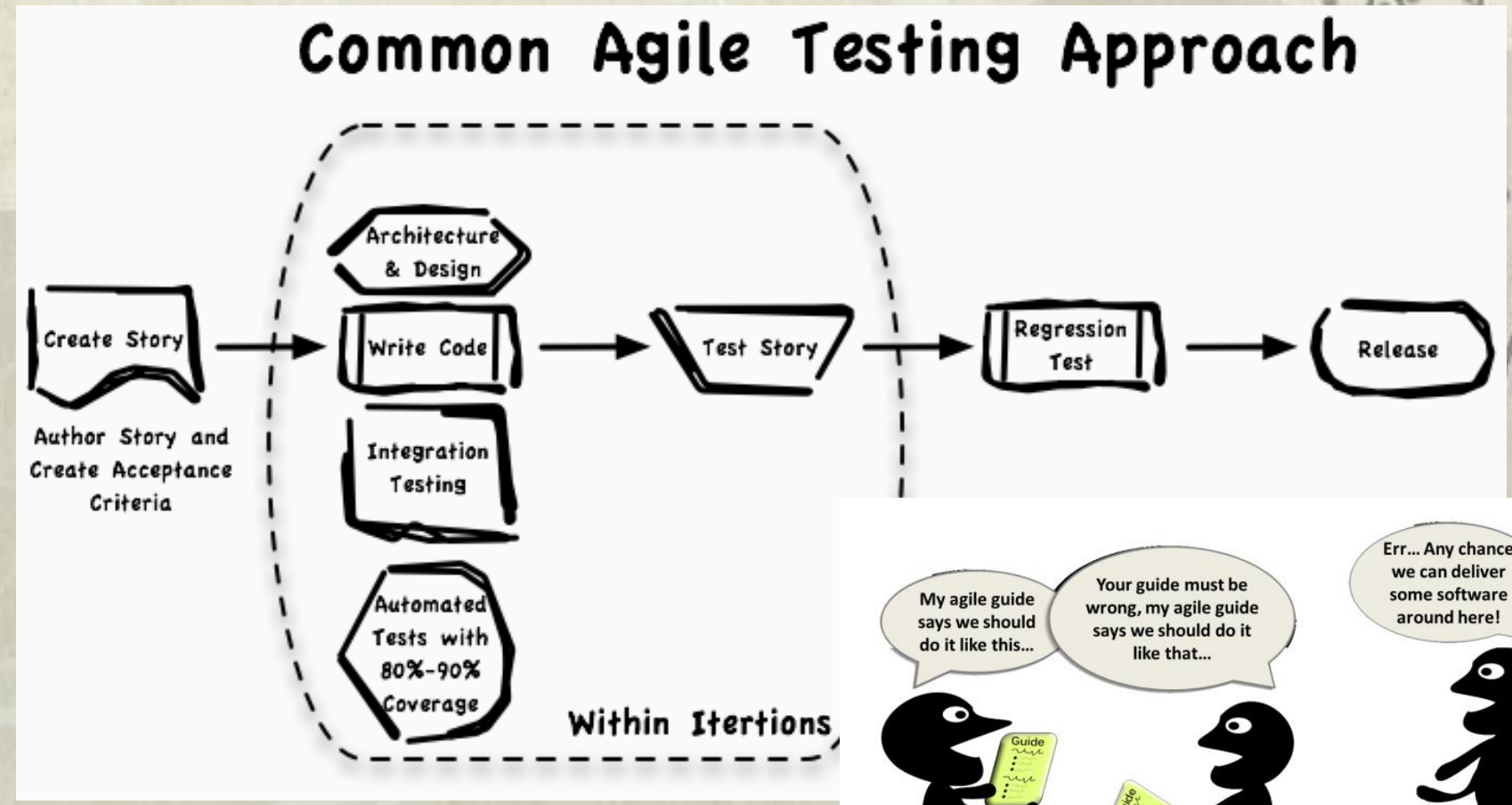
Testowanie oprogramowania

- dobry przypadek testowy to taki, po którego przeprowadzeniu znaleziony błąd
  - przypadki powinny być tak dobrane, by jak najlepiej się tylko da pokryć funkcje systemu
- każdy przypadek testowy powinien zawierać

Identyfikator	**Nazwa testu, czynności**
Dotyczy	Funkcja, procedura
Utworzył	Osoba odpowiedzialna za PT (może być login)
Data utworzenia	Data
Typ	Manualny / Automatyczny
Opis	Wymagania
Cel	
Warunki początkowe	
Status	Wykonany czy nie
Etapy:	
Krok	Nr etapu
Opis	Opis kroku
Oczekiwania	Co się powinno zdarzyć . . .

Testowanie oprogramowania

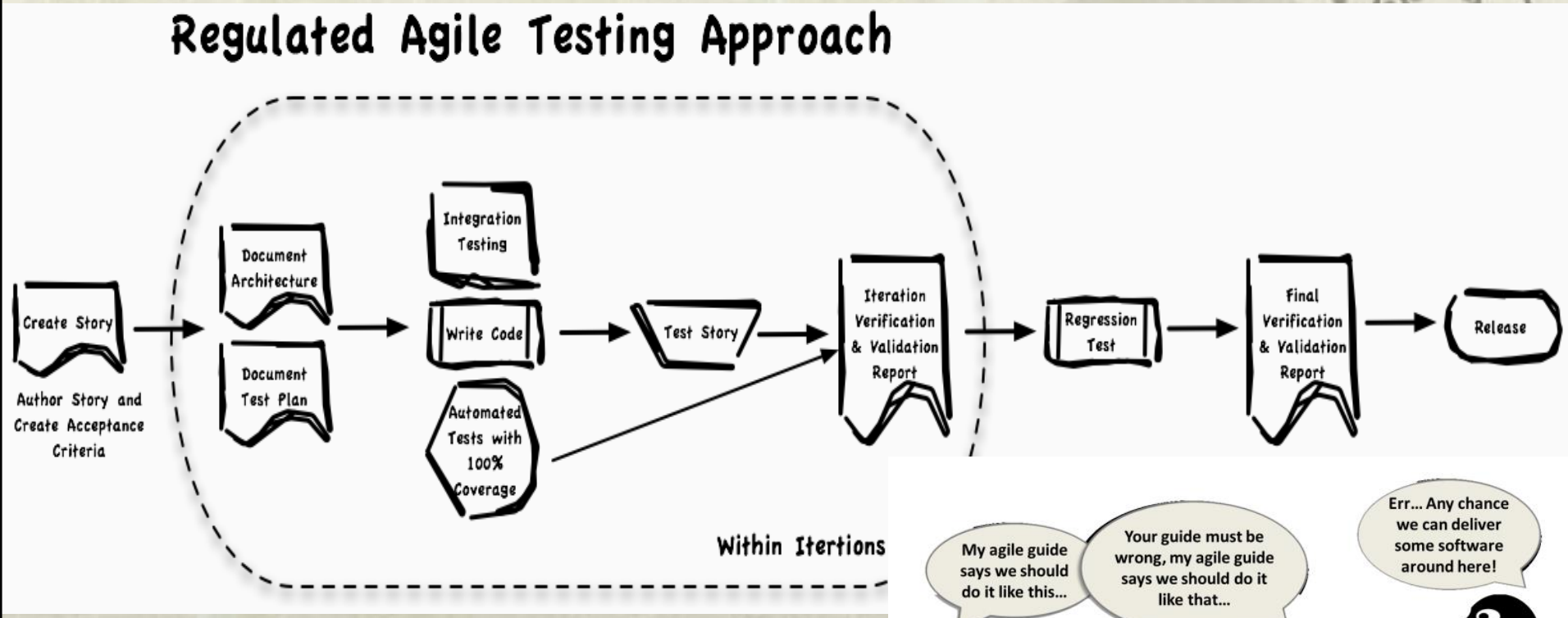
Testowanie zwinne

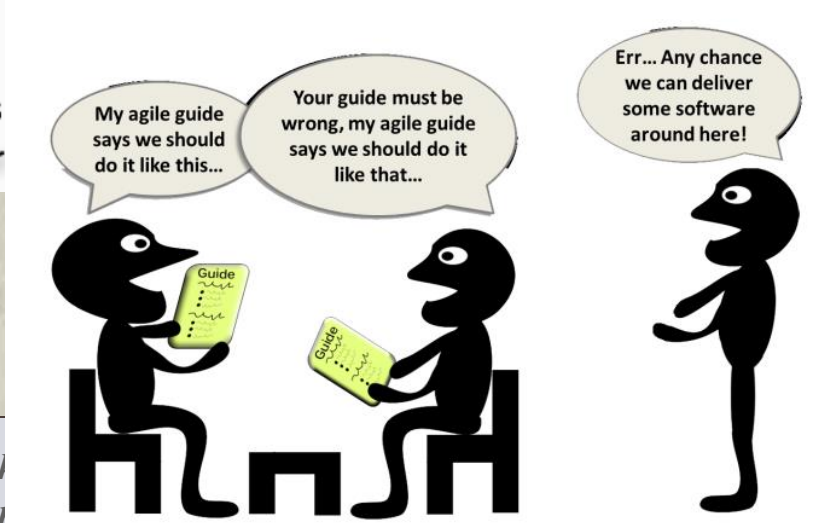


dr hab. inż. Aneta Poniszewsk
Instytut Informatyki Politechn

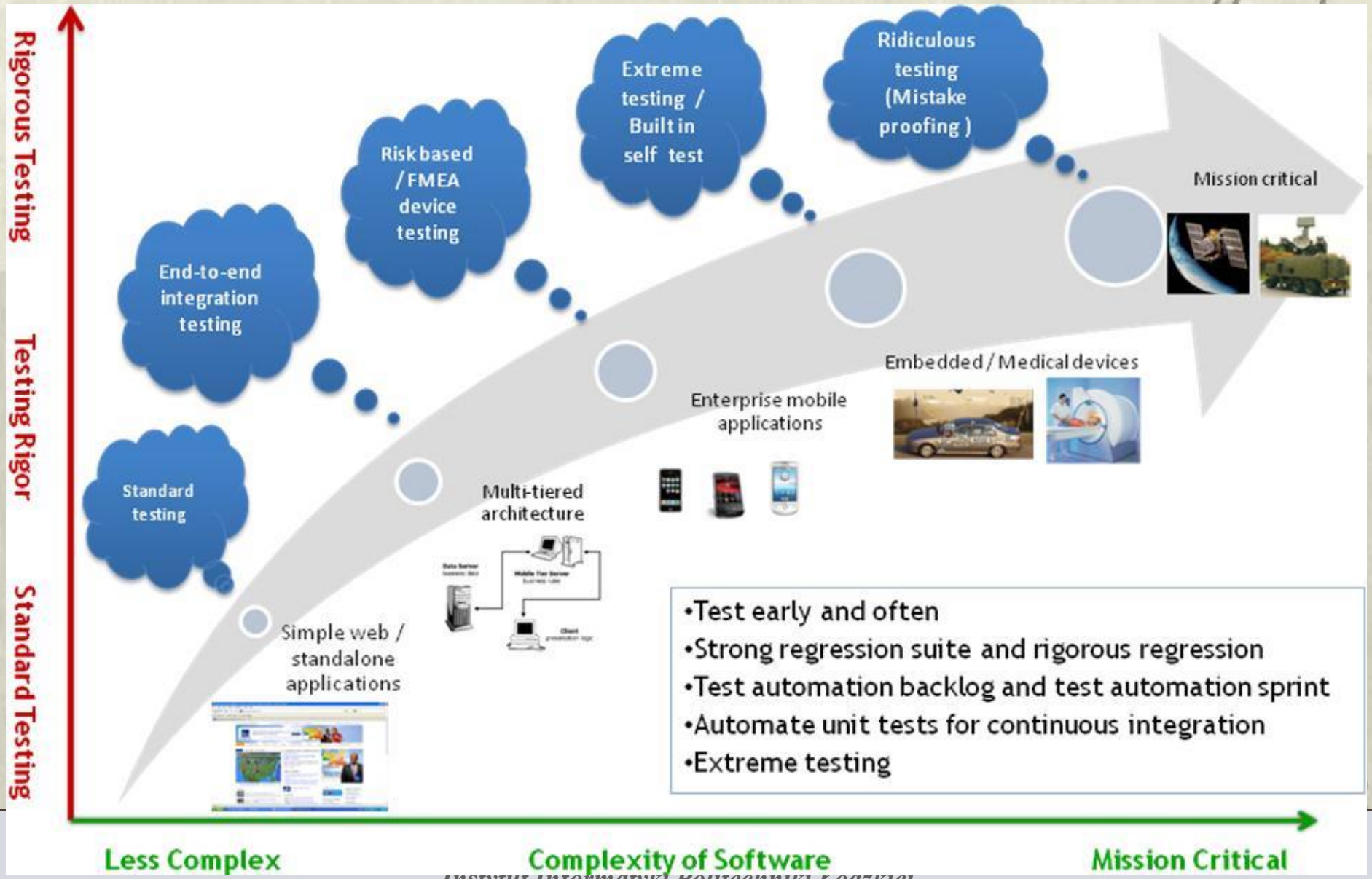
Testowanie oprogramowania

Testowanie zwinne



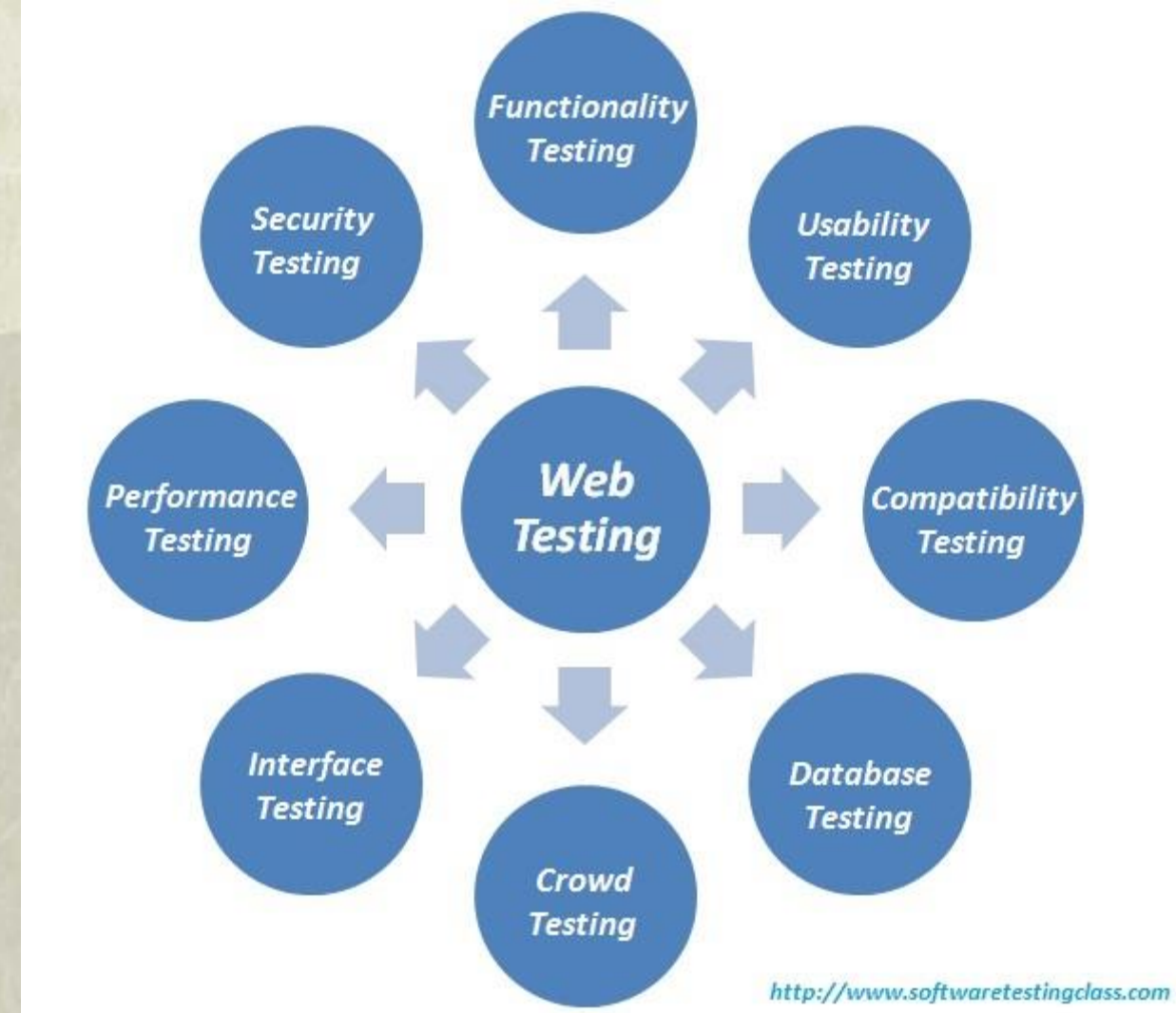


Testowanie oprogramowania



Testowanie oprogramowania

Testowanie oprogramowania



Narzędzia do testowania – Selenium

- **Selenium** – narzędzie do automatyzacji *testów akceptacyjnych* dla aplikacji internetowych
- integruje się z przeglądarką internetową i umożliwia nagrywanie akcji wykonywanych przez użytkownika
  - nagrywanie aplikacji napisanych w językach: C#, Java, Perl, PHP, Python, Ruby
- zapisane akcje można przekonwertować do wybranego języka programowania (np. Java, C#) i wykonywać dowolną ilość razy jako *testy regresyjne*

Fragment nagranego scenariusza testowego

Command	Target	Value
open	/	
waitForPageToLoad		
clickAndWait	xpath=id('menu_download')/a	
assertTitle	Downloads	
verifyText	xpath=id('mainContent')/h2	Downloads

Najważniejsze polecenia:

- open
- click/clickAndWait
- verifyTitle/assertTitle
- verifyTextPresent
- verifyElementPresent
- verifyText
- verifyTable
- waitForPageToLoad
- waitForElementPresent

Narzędzia do testowania – JUnit

- twórcy: K. Beck, E. Gamma
- http://junit.org
- wsparcie dla wielu języków:
  - SUnit (Smalltalk)
  - NUnit (C#)
  - PyUnit (Python)
  - CPPUnit (C++)
  - fUnit (Fortran)
  - JSUnit (JavaScript)

Narzędzia do testowania – JUnit

- framework o otwartych źródłach przeznaczony do testowania jednostkowego kodów napisanych w języku Java
- integracja z popularnymi środowiskami programistycznymi – *Eclipse*, *NetBeans*, *IntelIiJ*
- umożliwia odseparowanie testów, co pozwala na uniknięcie efektów ubocznych
- ADNOTACJE *@Before*, *@BeforeClass*, *@After* i *@AfterClass* umożliwiają inicjowanie i odzyskiwanie zasobów
- zbiór metod *assert* umożliwia łatwe sprawdzanie wyników testowanie

Narzędzia do testowania – JUnit

Prosty test

```
import static org.junit.Assert.*;
import org.junit.Test;
public class AddTest {
@Test
public void TestAdd() {
double result = 15+ 25;
assertEquals(40, result, 0);
}
}
```

Narzędzia do testowania – JUnit

- JUnit umożliwia sprawdzanie, czy dany wyjątek został wyrzucony w określonej sytuacji

```
@Test(expected=RuntimeException.class)
public void testExceptionMethod()
{//...}
```

- JUnit umożliwia sprawdzanie czasu wykonania danego fragmentu kodu
- parametr *timeout* pozwala na ustawienie czasu, którego przekroczenie spowoduje, że test nie zostanie zaliczony
  - wprowadzona bariera czasowa jest zapisywana w milisekundach

```
@Test(timeout = 120)
public void testTimeMethod()
{//...}
```

Narzędzia do testowania – JUnit

- czasami istnieją powody, żeby zignorować wyniki niektórych testów – np. w sytuacji, gdy do końca nie wiadomo, jaki będzie trzeba ustalić limit czasowy
- JUnit pozwala na ignorowanie wyników niektórych testów za pomocą adnotacji

```
@Ignore
@Test
@Ignore(value="Zignorowany do ustalenia czasu")
public void testIgnore(){}
```

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

1. **Działania wstępne**
   1. ***poznanie architektury systemu***
      - wstępnie określić na ile skomplikowany będzie proces testowania
   2. ***określenie jakiego typu testy są potrzebne***
      - związane z charakterystyką testowanego systemu i z wymaganiami klienta, standardami zapewniania jakości przyjętymi w firmie itp.
   3. ***weryfikacja testowalności systemu***
      - istotne przy dużych, wielowarstwowych aplikacjach
      - tester musi mieć dostęp do wszelkich potrzebnych mu danych
   4. ***poznanie dodatkowych źródeł informacji*** o działaniu aplikacji
      - np. możliwości dostępu do logów aplikacji, możliwości uruchomienia aplikacji w konfiguracji debugowania, która dostarcza więcej informacji o wewnętrznym działaniu oprogramowania

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

2. **Definicja przypadków testowych**
   1. przegląd zebranych wymagań oraz przypadków użycia (jeśli są spisane)
   2. dla każdego z wybranych wymagań powinien zostać określony jeden lub więcej przypadków użycia (o ile nie jest to już zrobione albo nie jest oczywiste), następnie należy utworzyć scenariusze testowe dla wymagań/przypadków użycia
   3. podjęcie decyzji, co testować
   4. wybór technik tworzenia testów, wyboru danych testowych
   5. wybór sposobu wykonywania testów (manualne, automatyczne)

- ***nie należy pomijać testowania wymagań niefunkcjonalnych***
- często przypadki użycia i wymagania niedostatecznie precyzyjne, aby opracować testy samodzielnie, bez udziału osób, które zajmowały się zbieraniem wymagań

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

3. **Określenie terminologii**
   - określić terminologię stosowaną do klasyfikowania występujących problemów
   - od tego zależy sposób traktowania zgłaszanych błędów: inaczej reagować przy literówce w nazwie przycisku, a inaczej w przypadku problemów z bazą danych
   - skategoryzować błędy ze względu na ich pochodzenie
     - np. błędy jako wynik automatycznych funkcjonalnych testów regresji, zgłoszonych przez odpowiednie oprogramowanie albo błędy wykryte w czasie testów jednostkowych, zgłoszone przez użytkowników systemu, czy przez testerów, którzy testowali oprogramowanie manualnie

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

4. **Określenie standardów dokumentowania testów**
   - dokumentacja testów odzwierciedlać działania, podejmowane w czasie testowania
   - **opis każdego testu powinien zawierać**
     1. Identyfikator, zgodny z wybraną notacją
     2. Opis testu: krótka informacja, co dany test weryfikuje
     3. Data wykonania testu
     4. Informacje o testerze, jeśli test jest wykonywany manualnie
     5. Autor testu (skryptu testowego)
     6. Cel testu
     7. Związane z danym testem wymagania, przypadki użycia: identyfikatory odpowiednich przypadków użycia bądź wymagań, które weryfikowane przez test
     8. Warunki początkowe, założenia wstępne, zależności – warunki, w jakich testować aplikację (np. podłączeni do Internetu); może się zdarzyć, że dany test musi być poprzedzony innym lub dwa testy nie mogą być wykonane w tym samym czasie

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

4. **Określenie standardów dokumentowania testów**
   - opis każdego testu powinien zawierać
     9. Sposób weryfikacji: jak weryfikować wynik testu; w przypadku testów niefunkcjonalnych dość złożone i polega np. na analizie logów aplikacji, zebraniu opinii użytkowników na temat używalności
     10. Działania użytkownika: kroki, jakie wykonać w ramach testu, nazwę skryptu testowego (lub jego zawartość); jeśli test manualny, to dokładnie kroki, jakie powinien wykonać tester
     11. Oczekiwany wynik testu: określamy stan aplikacji, w jakim powinna się ona znaleźć po przeprowadzeniu testu; inny stan będzie oznaczać niepowodzenie testu
     12. Zawartość logów aplikacji
     13. Faktyczny wynik testu – jeśli test się powiódł, to wpisujemy „tak, jak oczekiwany”, w przeciwnym razie podajemy informację, jaki rezultat testu, załączmy widok ekranu, numer błędu itp.
     14. Opis danych, jakich należy użyć do testowania

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

4. **Określenie standardów dokumentowania testów**
   - nie zawsze musimy uwzględniać wszystkie elementy
   - pełna dokumentacja testów może być bardzo rozległa
   - więcej szczegółów w standardzie IEEE: **IEEE Standard for Software Test Documentation (IEEE829-98)**
   - dokumenty zawierające informacje muszą być uwzględnione w ramach systemu zarządzania zmianami – testy będą ewoluowały wraz z oprogramowaniem, więc trzeba utrzymać synchronizację pomiędzy tymi dwoma elementami
5. **Metryki i raporty**
   - zadecydować, jakiego rodzaju informacje zbierać i zachowywać
     - liczba wszystkich wykrytych błędów
     - liczba błędów należących do określonej kategorii (np. liczba błędów krytycznych)
     - liczba wykrytych błędów przez dany test
     - liczba błędów danego komponentu oprogramowania
     - czas potrzebny na przeprowadzenie pojedynczych testów lub serii testów
     - liczba poprawionych błędów

Wdrażanie procedury testowania

Przykładowy scenariusz wdrożenia procesu testowania w projekcie

6. **Plan testów**
   - utworzenie planu testów i budowa bazy testów (ang. *tests inventory)*
   - **plan testów** – dokument, który zawiera informację o celach, zakresie i przyjętych technikach testowania
   - przygotowywanie planu testów dobrą okazją do tego, żeby określić warunki pozwalające uznać oprogramowanie za wystarczająco sprawne i gotowe do wdrożenia
   - celem dokumentu umożliwienie osobom spoza zespołu projektowego zrozumienie, jak i dlaczego oprogramowanie sprawdzane, kiedy można bezpiecznie przyjąć, że produkt jest gotowy
   - określa, jakie elementy oprogramowanie będą testowane, na jakim poziomie, w jakiej kolejności, jakie techniki testowania zostaną zastosowane i jakie założenia zostały przyjęte
   - plan testów powinien stać się częścią dokumentacji projektowej, podlegać kontroli zmian