

# **INŻYNIERIA OPROGRAMOWANIA**

**wykład 9:  
TESTOWANIE  
NIEZAWODNOŚĆ  
TESTY cd ..**

**dr inż. Leszek Grocholski  
Zakład Inżynierii Oprogramowania  
Instytut Informatyki  
Uniwersytet Wrocławski**

# Oszacowanie niezawodności

Niekiedy (umowy dot. świadczenia usług - QoS, umowy outsourcingowe) poziom niezawodności (wartość pewnej miary lub miar) jest określany w wymaganiach klienta.

Częściej, jest on jednak wyrażony w terminach jakościowych, co bardzo utrudnia obiektywną weryfikację. **Jednakże informacja o niezawodności jest przydatna również wtedy, gdy klient nie określił jej jednoznacznie w wymaganiach.**

## Dlaczego?

- Częstotliwość występowania błędnych wykonów ma duży wpływ na koszt konserwacji oprogramowania (serwis telefoniczny + wizyty u klienta).
- Znajomość niezawodności pozwala oszacować koszt serwisu, liczbę personelu, liczbę zgłoszeń telefonicznych, łączny koszt serwisu.
- Znajomość niezawodności pozwala ocenić i polepszyć procesy wytwarzania pod kątem zminimalizowania łącznego kosztu wynikającego z kosztów wytwarzania, kosztów utrzymania, powodzenia na rynku, reputacji firmy.

# Niezawodność oprogramowania

## Miary niezawodności:

- **Prawdopodobieństwo błędного wykonania podczas realizacji transakcji.** Każde błędne wykonanie powoduje zerwanie całej transakcji. Miarą jest częstotliwość występowania transakcji, które nie powiodły się wskutek błędów.
- **Częstotliwość występowania błędnych wykonań:** ilość błędów w jednostce czasu. Np. 0.1/h oznacza, że w ciągu godziny ilość spodziewanych błędnych wykonań wynosi 0.1. Miara ta jest stosowana w przypadku systemów, które nie mają charakteru transakcyjnego.
- **Średni czas między błędnymi wykonaniami** - odwrotność poprzedniej miary.
- **Dostępność:** prawdopodobieństwo, że w danej chwili system będzie dostępny do użytkowania. Miarę tę można oszacować na podstawie **stosunku czasu, w którym system jest dostępny, do czasu od wystąpienia błędu do powrotu do normalnej sytuacji**. Miara zależy nie tylko od błędnych wykonań, ale także od narzutu błędów na niedostępność systemu (czasu interwencji).

# Wzrost niezawodności oprogramowania

Rezultatem wykrycia przyczyn błędów jest ich usunięcie.

Jeżeli przy tym nie wprowadza się nowych błędów, to można mówić o wzroście niezawodności.

Jeżeli wykonywane są testy czysto statystyczne to wzrost niezawodności określa się następującym wzorem (logarytmiczny wzrost niezawodności):

$$\text{Niezawodność} = \text{Niezawodność początkowa} \times \exp(-C \times \text{liczba testów})$$

Miarą niezawodności jest częstotliwość występowania błędnych wykonań.

**Stała C zależy od konkretnego systemu.** Można ją określić na podstawie obserwacji statystycznych niezawodności systemu, stosując np. metodę najmniejszych kwadratów.

Szybszy wzrost niezawodności można osiągnąć jeżeli dane testowe są dobierane nie w pełni losowo, lecz w kolejnych przebiegach testuje się sytuacje, które dotąd nie były testowane.

# **Wykrywanie błędów**

## **- rodzaje testów wykrywających błędne wykonania, błędy**

Dynamiczne testy zorientowane na wykrywanie błędnych wykonień dzieli się na:

- **Testy funkcjonalne** (*functional tests, black-box tests*), które zakładają znajomość jedynie wymagań wobec testowanej funkcji. System jest traktowany jako czarna skrzynka, która w nieznany sposób realizuje wykonywane funkcje. Testy powinny wykonywać osoby, które nie były zaangażowane w realizację testowanych fragmentów systemu.
- **Testy strukturalne** (*structural tests, white-box tests, glass-box tests*), które zakładają znajomość sposobu implementacji testowanych funkcji.

Po wykonaniu testu należy znaleźć błąd w kodzie i go poprawić

# Testy funkcjonalne

## Techniki projektowania testów

Testy można tak zaprojektować, żeby pokrywały akceptowalne i nieakceptowalne klasy równoważności.

Istnieje większe prawdopodobieństwo, że oprogramowania będzie się błędnie zachowywać dla wartości na krawędziach klas równoważności niż w ich środku, więc testowanie tych obszarów najprawdopodobniej wykryje błędy.

**Minimum i maksimum klasy równoważności to jej wartości brzegowe.** Wartość brzegowa poprawnego przedziału jest nazywana poprawną wartością brzegową, a wartość brzegowa niepoprawnego przedziału –niepoprawną wartością brzegową.

Testy można zaprojektować tak, żeby pokrywały zarówno poprawne jak i niepoprawne wartości brzegowe. Podczas projektowania testów tworzy się przypadek testowy dla każdej wartości brzegowej.

# Testy funkcjonalne

Pełne przetestowanie rzeczywistego systemu jest praktycznie niemożliwe z powodu ogromnej liczby kombinacji danych wejściowych i stanów. Nawet dla stosunkowo małych programów ta liczba kombinacji jest tak ogromna, że pełne testowanie wszystkich przypadków musiałoby rozciągnąć się na miliardy lat.

Zwykle zakłada się, że jeżeli dana funkcja działa poprawnie dla **kilku** danych wejściowych, to działa także poprawnie dla **całej klasy** danych wejściowych. Jest to wnioskowanie czysto heurystyczne. Fakt poprawnego działania w kilku przebiegach nie gwarantuje zazwyczaj, że błędne wykonanie nie pojawi się dla innych danych z tej samej klasy.

Podział danych wejściowych na klasy odbywa się na podstawie opisu wymagań, np.

*Rachunek o wartości do 1000 zł może być zatwierdzony przez kierownika.*

*Rachunek o wartości powyżej 1000 zł musi być zatwierdzony przez prezesa.*

Takie wymaganie sugeruje podział danych wejściowych na dwie klasy w zależności od wysokości rachunku. Jednak przetestowanie tylko dwóch wartości, np. 500 i 1500 jest zazwyczaj niewystarczające. Konieczne jest także przetestowanie wartości **granicznych**, np. 0, dokładnie 1000 oraz maksymalnej wyobrażalnej wartości.

# Kombinacja elementarnych warunków

Z poprzedniego przykładu widać, że testy tylko dla jednej danej wejściowej muszą być przeprowadzone dla pięciu wartości: np. 0, 500, 1000, 1500, max. Jeżeli takich danych jest wiele, to mamy do czynienia z kombinatoryczną eksplozję przypadków testowych.

Dzieląc dane wejściowe na klasy należy więc brać pod uwagę rozmaite kombinacje elementarnych warunków. Np. do wymienionego warunku dołączony jest następujący:

*Kierownik może zatwierdzić miesięcznie rachunek o łącznej wartości do 10 000 zł.  
Każdy rachunek przekraczający tę wartość musi być zatwierdzony przez prezesa.*

Wśród danych wyjściowych można teraz wyróżnić następujące klasy:

- rachunek do 1000 zł nie przekraczający łącznego limitu 10 000 zł.
- rachunek do 1000 zł przekraczający łączny limit 10 000 zł.
- rachunek powyżej 1000 zł nie przekraczający łącznego limitu 10 000 zł.
- rachunek powyżej 1000 zł przekraczający łączny limit 10 000 zł.

Uwzględnienie przypadków granicznych powoduje dalsze rozmnożenie przypadków testowych: (0, 500, 1000, 1500, max)  $\times$  (<10000, 10000, >10000)

# Eksplozja kombinacji danych testowych

W praktyce przetestowanie wszystkich kombinacji danych wejściowych (nawet zredukowanych do “typowych” i “granicznych”) jest najczęściej niemożliwe. Konieczny jest wybór tych kombinacji.

**Ogólne zalecenia takiego wyboru są następujące:**

- **Możliwość wykonania funkcji jest ważniejsza niż jakość jej wykonania.** Brak możliwości wykonania funkcji jest poważniejszym błędem niż np. niezbyt poprawne wyświetlenie jej rezultatów na ekranie.
- **Funkcje systemu znajdujące się w poprzedniej wersji są istotniejsze niż nowo wprowadzone.** Użytkownicy, którzy posługiwali się dana funkcją w poprzedniej wersji systemu będą bardzo niezadowoleni jeżeli w nowej wersji ta funkcja przestanie działać.
- **Typowe sytuacje są ważniejsze niż wyjątki lub sytuacje skrajne.** Błąd w funkcji wykonywanej często lub dla danych typowych jest znacznie bardziej istotny niż błąd w funkcji wykonywanej rzadko dla nietypowych danych.

# Inne techniki testów funkcjonalnych

**Stosowane są również inne techniki testów funkcjonalnych:**

- **Testowanie w oparciu o tablicę decyzyjną**

**Tablice decyzyjne są bardzo popularne w: bankach, firmach ubezpieczeniowych, telekomunikacyjnych, czy np. Wizardach.**

Tablice często dotyczą postępowania umożliwiającego stosowanie reguł (biznesowych ) tzn ograniczeń specyficzne dla danej organizacji, zdefiniowane dla całego jej obszaru funkcjonowania.

- **Testowanie przejść między stanami**

System może różnie odpowiadać w zależności od aktualnych warunków oraz od historii (od stanu). W takim przypadku zachowanie systemu można opisać diagramem przejść stanów (automatem skończonym).

- **Testowanie przypadków użycia**

# Testy strukturalne

W przypadku testów strukturalnych, dane wejściowe dobiera się na podstawie analizy struktury programu realizującego testowane funkcje.

Kryteria doboru danych testowych są następujące:

- **Kryterium pokrycia wszystkich instrukcji.** Zgodnie z tym kryterium dane wejściowe należy dobierać tak, aby każda instrukcja została wykonana co najmniej raz. Spełnienie tego kryterium zwykle wymaga niewielkiej liczby testów. To kryterium może być jednak bardzo nieskuteczne.

```
if x > 0 then begin ... end; y := ln( x);
```

Dla  $x > 0$  wykonane będą wszystkie instrukcje, ale dla  $x \leq 0$  program jest błędny.

- **Kryterium pokrycia instrukcji warunkowych.** Dane wejściowe należy dobierać tak, aby każdy elementarny warunek instrukcji warunkowej został co najmniej raz spełniony i co najmniej raz nie spełniony. Testy należy wykonać także dla każdej wartości granicznej takiego warunku. Zastosowanie tego warunku pozwoli wykryć błąd z poprzedniego przykładu, gdyż zmusi do testu dla  $x = 0$  oraz  $x < 0$ .

Istnieje szereg innych kryteriów prowadzących do bardziej wymagających testów.

# Testowanie programów zawierających pętle

**Kryteria doboru danych wejściowych mogą opierać się o następujące zalecenia:**

- Należy dobrać dane wejściowe tak, aby nie została wykonana żadna iteracja pętli, lub, jeżeli to niemożliwe, została wykonana minimalna liczba iteracji.
- Należy dobrać dane wejściowe tak, aby została wykonana maksymalna liczba iteracji.
- Należy dobrać dane wejściowe tak, aby została wykonana przeciętna liczba iteracji.

# Programy uruchamiające

*debuggers*

Mogą być przydatne dla wewnętrznego testowania jak i dla testowania przez osoby zewnętrzne. Zakładają testowanie na zasadzie białej skrzynki (znajomość kodu).

## Podstawowe funkcje debuggerów:

- Wyświetlenie stanu zmiennych programu i interakcja z testującym z użyciem symboli kodu źródłowego.
- Wykonywanie programów krok po kroku, z różną granulowością instrukcji
- Ustanowienie punktów kontrolnych w programie (zatrzymujących wykonanie)
- Ustanowienie obserwatorów wartości zmiennych
- Zarządzanie plikiem źródłowym podczas testowania i ewentualna poprawa wykrytych błędów w tym pliku.
- Tworzenie dziennika testowania, umożliwiającego powtórzenie testowego przebiegu.

# Analizatory pokrycia kodu

*coverage analysers*

**Analizatory pokrycia kodu** są to programy umożliwiające ustalenie obszarów kodu źródłowego, które były wykonane w danym przebiegu testowania. Umożliwiają wykrycie martwego kodu, kodu uruchamianego przy bardzo specyficznych danych wejściowych oraz (niekiedy) kodu wykonywanego bardzo często (co może być przyczyną wąskiego gardła w programie).

## Funkcje bardziej zaawansowanych analizatorów przykrycia kodu:

- Zsumowanie danych z kilku przebiegów (dla różnych kombinacji danych wejściowych) np. dla łatwiejszego wykrycia martwego kodu.
- Wyświetlenie grafów sterowania, dzięki czemu można łatwiej monitorować przebieg programu
- Wyprowadzenie informacji o pokryciu, umożliwiające poddanie pokrytego kodu dalszym testom.
- Operowanie w środowisku rozwoju oprogramowania.

# Programy porównujące

*comparators*

Są to narzędzia programistyczne umożliwiające porównanie dwóch programów, plików lub zbiorów danych celem wykrycia cech wspólnych i różnic. Często są niezbędne do porównania wyników testów z wynikami oczekiwanyimi. Programy porównujące przekazują w czytelnej postaci różnice pomiędzy aktualnymi i oczekiwanyimi danymi wyjściowymi.

Ekranowe programy porównujące mogą być bardzo użyteczne dla testowania oprogramowania interakcyjnego. Są niezastąpionym środkiem dla testowania programów z graficznym interfejsem użytkownika.

## Inne narzędzia wspomagające testowanie:

Duża różnorodność narzędzi stosowanych w różnych fazach rozwoju oprogramowania. Np. wspomaganie do planowania testów, automatyczne zarządzania danymi wyjściowymi, automatyczna generacja raportów z testów, generowanie statystyk jakości i niezawodności, wspomaganie powtarzalności testów, itd.

# Testy statyczne

Polegają na analizie kodu bez uruchomienia programu. Techniki są następujące:

- metody nieformalne - dowody poprawności
- metody formalne

Dowody poprawności nie są praktycznie możliwe dla rzeczywistych programów. Nie istnieją wyłącznie w ideach teoretyków informatyki. Stosowanie ich dla programów o obecnej skali i złożoności jest trudne.

(*Ale dowody poprawności stosuje się np.. w systemach krytycznych*)

**Statyczne metody nieformalne** polegają na analizie kodu przez programistów.

- czyli inspekcje

**Dwa niewykluczające się podejścia:**

- śledzenie przebiegu programu (wykonywanie programu "w myśli" przez analizujące osoby)
- wyszukiwanie typowych błędów

Testy nieformalne są niedocenione, chociaż bardzo efektywne w praktyce.

Uwaga:

Na ogół testy funkcjonalne są bardziej skuteczne niż testy strukturalne.

# Typowe błędy wykrywane statycznie

- **niezainicjowane** zmienne
- porównania na równość liczb **zmiennoprzecinkowych**
- **Indeksy** wykraczające poza tablice
- błędne operacje na **wskaźnikach**
- błędy w warunkach **instrukcji warunkowych**
- niekończące się **pętle**
- błędy popełnione dla **wartości granicznych** (np. `>` zamiast `>=`)
- błędne użycie lub pominięcie **nawiasów** w złożonych wyrażeniach
- nieuwzględnienie **błędnych danych**

## Postępowanie podczas statycznych testów nieformalnych:

Programista, który dokonał implementacji danego modułu w nieformalny sposób analizuje jego kod.

Kod uznany przez programistę za poprawny jest analizowany przez doświadczonego programistę. Jeżeli znajdzie on pewną liczbę błędów, moduł jest zwracany programiście do poprawy.

Szczególnie istotne moduły są analizowane przez grupę osób.

# Ocena liczby błędów – koszty konserwacji

Błędy w oprogramowaniu niekoniecznie są bezpośrednio powiązane z jego zawodnością. Oszacowanie liczby błędów ma jednak znaczenie dla producenta oprogramowania, gdyż ma wpływ na koszty konserwacji oprogramowania. Szczególnie istotne dla firm sprzedających oprogramowanie pojedynczym lub nielicznym użytkownikom (relatywnie duży koszt usunięcia błędu).

## Dane umożliwiające szacowanie kosztów konserwacji dot. usuwania błędów:

- Szacunkowa liczba błędów w programie
- Średni procent błędów zgłaszanych przez użytkownika systemu, na podstawie danych z poprzednich przedsięwzięć.
- Średni koszt usunięcia błędu na podstawie danych z poprzednich przedsięwzięć.

# Technika “posiewania błędów”

Polega na tym, że do programu celowo wprowadza się pewną liczbę błędów podobnych do tych, które występują w programie. Wykryciem tych błędów zajmuje się inna grupa programistów niż ta, która dokonała “posiania” błędów.

Założymy, że:

N oznacza liczbę posianych błędów

M oznacza liczbę wszystkich wykrytych błędów

X oznacza liczbę posianych błędów, które zostały wykryte

Wyniki mogą być mocno chybione, jeżeli “posiane” błędy nie będą podobne do rzeczywistych błędów występujących w programie.

Technika ta pozwala również na przetestowanie skuteczności metod testowania.  
Zbyt mała wartość X/N oznacza konieczność poprawy tych metod.

# Testy fragmentów systemu

**Techniki testowania fragmentów systemu:**

- **testowanie wstępujące**
- **testowanie zstępujące**

- **Testowanie wstępujące:** najpierw testowane są pojedyncze moduły, następnie moduły coraz wyższego poziomu, aż do osiągnięcia poziomu całego systemu. Zastosowanie tej metody nie zawsze jest możliwe, gdyż często moduły są od siebie zależne. Niekiedy moduły współpracujące można zastąpić implementacjami szkieletowymi ( atrapy, tzw „mocki”).

- **Testowanie zchodzące:** rozpoczyna się od testowania modułów wyższego poziomu. Moduły niższego poziomu zastępuje się implementacjami szkieletowymi ( atrapy, „mocki” ). Po przetestowaniu modułów wyższego poziomu dołączane są moduły niższego poziomu. Proces ten jest kontynuowany aż do zintegrowania i przetestowania całego systemu.

# Testy pod obciążeniem, testy odporności

**Testy obciążeniowe** (*stress testing*). Celem tych testów jest zbadanie wydajności i niezawodności systemu podczas pracy pod pełnym lub nawet nadmiernym obciążeniem. Dotyczy to szczególnie systemów wielodostępnych i sieciowych. Systemy takie muszą spełniać wymagania dotyczące wydajności, liczby użytkowników, liczby transakcji na godzinę. Testy polegają na wymuszeniu obciążenia równego lub większego od maksymalnego.

**Testy odporności** (*robustness testing*). Celem tych testów jest sprawdzenie działania w przypadku zajścia niepożądanych zdarzeń, np.

- zaniku zasilania
- awarii sprzętowej
- wprowadzenia niepoprawnych danych
- wydania sekwencji niepoprawnych poleceń

# Bezpieczeństwo oprogramowania

Pewne systemy są krytyczne z punktu widzenia bezpieczeństwa ludzi, np. Może to być także zagrożenie pośrednie, np. systemy eksperckie w dziedzinie medycyny, systemy informacji o lekach.

**Bezpieczeństwo niekoniecznie jest pojęciem tożsamym z niezawodnością.**

System zawodny może być bezpieczny, jeżeli skutki błędnych wykonań nie są groźne.

**Wymagania wobec systemu mogą być niepełne** i nie opisywać zachowania systemu we wszystkich sytuacjach. Dotyczy to zwłaszcza sytuacji wyjątkowych, np. wprowadzenia niepoprawnych danych. Ważne jest, aby system zachował się bezpiecznie także wtedy, gdy właściwy sposób reakcji nie został opisany.

Niebezpieczeństwo może także wynikać z awarii sprzętowych. Analiza bezpieczeństwa musi uwzględniać oba czynniki.

# Analiza bezpieczeństwa

Techniki:

- oparte na doświadczeniu,
- oparte na analizie ryzyka (dużo odmian)

Zaczyna się od określenia potencjalnych niebezpieczeństw związanych z użytkowaniem systemu: możliwości utraty życia, zdrowia, strat materialnych, złamania przepisów prawnych.

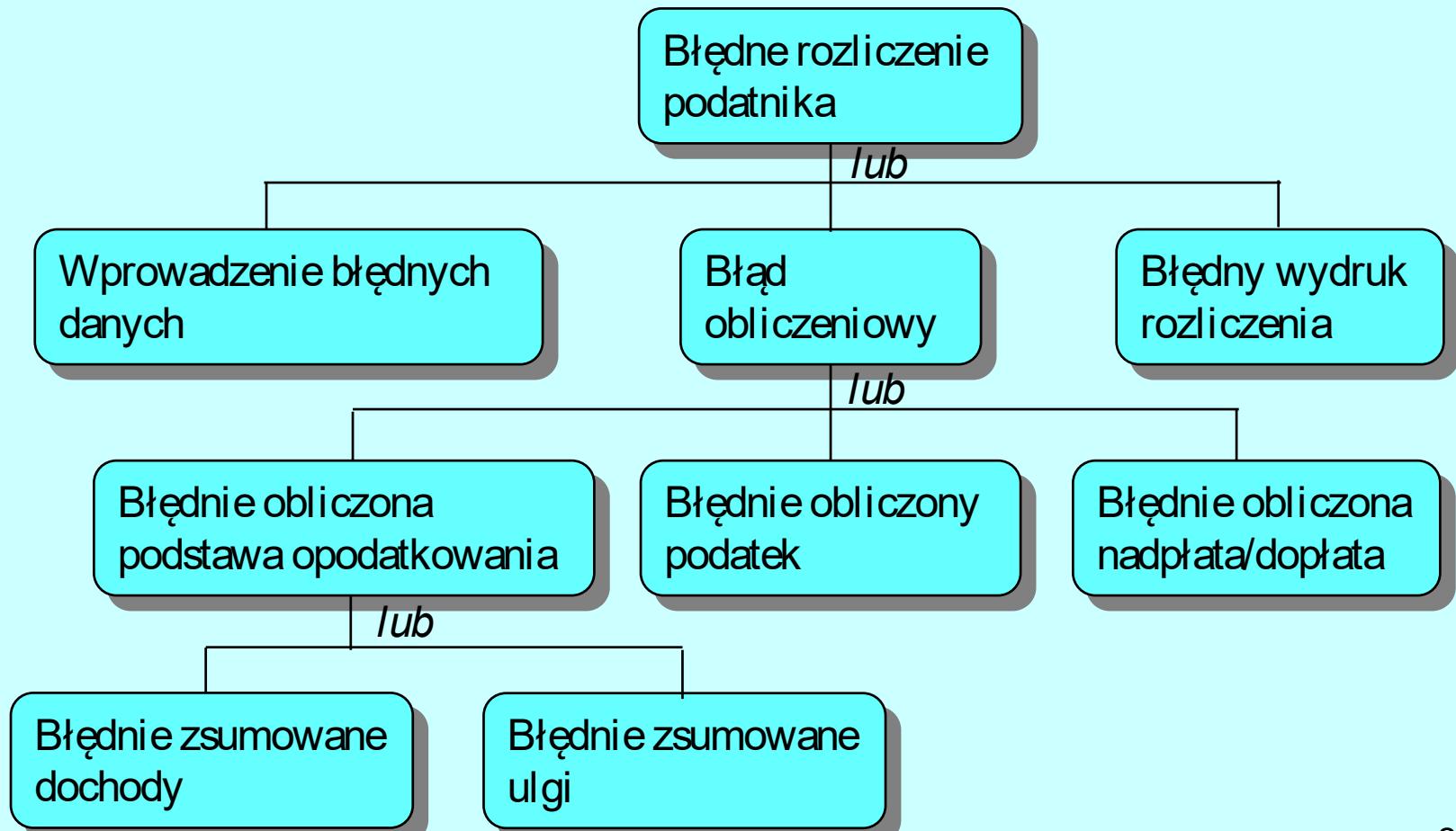
Np. dla programu podatkowego mogą wystąpić następujące niebezpieczeństwa:

- błędne rozliczenie się z urzędem podatkowym
- nie złożenie zeznania podatkowego
- złożenie wielu zeznań dla jednego podatnika

# Drzewo błędów

*fault tree*

Korzeniem drzewa są jedna z rozważanych niebezpiecznych sytuacji. Wierzchołkami są sytuacje pośrednie, które mogą prowadzić do sytuacji odpowiadającej wierzchołkowi wyższego poziomu.



# Techniki zmniejszania niebezpieczeństwa

- Położenie większego nacisku na unikanie błędów podczas implementacji fragmentów systemu, w których błędy mogą prowadzić do niebezpieczeństw.
- Zlecenie realizacji odpowiedzialnych fragmentów systemu bardziej doświadczonym programistom.
- Zastosowanie techniki programowania N - wersyjnego w przypadku wymienionych fragmentów systemu. Ten sam komponent zaprogramowany przez nizależne zespoły programistów. Wyniki działania komponentu są porównywalne.
- Szczególnie dokładne przetestowanie tych fragmentów systemu.
- Wczesne wykrywanie sytuacji, które mogą być przyczyną niebezpieczeństw i podjęcie odpowiednich, bezpiecznych akcji. Np. ostrzeżenie w pewnej fazie użytkownika o możliwości zajścia błędu (asercje + zrozumiałe komunikaty o niezgodności).

# Czynniki sukcesu, rezultaty testowania

## Czynniki sukcesu:

- Określenie fragmentów systemu o szczególnych wymaganiach wobec niezawodności.
- Właściwa motywacja osób zaangażowanych w testowanie. Np. stosowanie nagród dla osób testujących za wykrycie szczególnie groźnych błędów, zaangażowanie osób posiadających szczególny talent do wykrywania błędów

## Podstawowe wyniki testowania:

- Poprawiony kod, projekt, model i specyfikacja wymagań
- Raport przebiegu testów, zawierający informację o przeprowadzonych testach i ich rezultatach.
- Oszacowanie niezawodności oprogramowania i kosztów konserwacji.

# **INŻYNIERIA OPROGRAMOWANIA**

**Dziękuję za uwagę**

# **INŻYNIERIA OPROGRAMOWANIA**

**wykład 10:  
ZAPEWNIENIE JAKOŚCI OPROGRAMOWANIA**

**dr inż. Leszek Grocholski**  
Zakład Inżynierii Oprogramowania  
Instytut Informatyki  
Uniwersytet Wrocławski

# POJĘCIE JAKOŚCI

Krótka i jasna definicja:

Jakość (quality, die Qualität) to stopień realizacji wymagań.

Inne definicje

1. Pewien stopień doskonałości •Platon, IV wiek p.n.e.
2. Zgodność z wymaganiami (conformance to user requirements)
  - P.B. Crosby, Quality Is Free, McGraw-Hill, 1979
3. Osiągnięcie doskonałych poziomów zdatności do użycia (achieving excellent levels of fitness for use)
  - W. Humphrey, Managing the Software Process, Addison-Wesley, 1989
4. Jakość sterowana rynkiem (market-driven quality) •IBM, lata 80-te
5. Stopień, w jakim zbiór inherentnych charakterystyk spełnia wymagania (the degree to which a set of inherent characteristics fulfills requirements)
  - ISO 9001:2000, Quality Management Systems - Requirements, ISO, 2000
  - Przykładowe charakterystyki: funkcjonalność, niezawodność, użyteczność, wydajność, rozszerzalność

# **Co to jest “zapewnienie jakości oprogramowania” ?**

**Zapewnienie jakości jest rozumiane jako zespół działań zmierzających do wytworzenia u wszystkich zainteresowanych przekonania, że dostarczony produkt właściwie realizuje swoje funkcje i odpowiada aktualnym wymaganiom i standardom.**

Problem jakości, oprócz mierzalnych czynników technicznych, włącza dużą liczbę niemierzalnych obiektywnie czynników psychologicznych.

Podstawą obiektywnych wniosków co do jakości oprogramowania są **pomiary** pewnych parametrów użytkowych (niezawodności, szybkości, itd.) w realnym środowisku, np. przy użyciu metod statystycznych.

**Niestety, obiektywne pomiary cech produktów programistycznych są utrudnione lub niemożliwe.** Jakość gotowych produktów programistycznych jest bardzo trudna do zmierzenia ze względu na ich złożoność (eksplozja danych testowych), wieloaspektowość, identyczność (?) wszystkich kopii produktu, oraz niską przewidywalnością wszystkich aspektów ich zastosowań w długim czasie.

# Trudności z oceną jakości oprogramowania

- Oceny jakości najczęściej muszą być znane zanim powstanie gotowy, działający produkt, co wyklucza zastosowanie obiektywnych metod pomiarowych.
- Wiele czynników składających się na jakość produktu jest niemierzalna.
- Produkty programistyczne są złożone i wieloaspektowe, co powoduje trudności w wyodrębnieniu cech mierzalnych, które odzwierciedlałyby istotne aspekty jakości.
- Produkty programistyczne mogą działać w różnych zastosowaniach, o różnej skali. Pomiary jakości mogą okazać się nieadekwatne przy zmianie skali (np. zwiększonej liczbie danych lub użytkowników), w innym środowisku, itp.

Pomiary mogą okazać się bardzo kosztowne, czasochłonne lub niewykonalne (z powodu niemożliwości stworzenia środowiska pomiarowego przed wdrożeniem);

Nie ma zgody co do tego, w jaki sposób pomierzone cechy danego produktu składają się na syntetyczny wskaźnik jego jakości.

**Stąd oceny jakości produktów programistycznych są skazane na metody spekulacyjne, oparte na uproszczeniach oraz dodatkowych założeniach, algorytmach, wzorach i heurystykach.**

# **Kultura jakości**

**To LUDZIE tworzą jakość!**

**DOSKONAŁOŚĆ i ORIENTACJA na WYNIK KOŃCOWY kluczem do SUKCESU**

**SUKCES** - Kiedy naprawdę wiemy, że odnieśliśmy sukces ?

**DOSKONALOŚĆ** - Co robimy, aby rzeczywiście stawać coraz się lepszymi?

**ORIENTACJA na WYNIK KONCOWY**

- W jaki sposób dbamy o to, aby dostarczać produkt zgodny z wymaganiami?

# Kultura jakości

***Jakość jest jak kultura osobista...***

***Trzeba o nią dbać i ją kształtować – inaczej przepadnie!***

**Ideał jakości może pokazać tylko pewien kierunek;**

**Cele jakości są celami projektu**

W przypadku zarządzania jakością w IT (i nie tylko ) widać wyraźny wpływ

2 podejść:

1. TQM ( Total Quality Management)
2. ISO serii 9000

# TQM - zarządzanie przez jakość

Koncepcja wymyślona przez Japończyka **Eiji Toyodę** dla potrzeb naprawy japońskiego przemysłu motoryzacyjnego - 1950 r.

**Główna jej myśl mówiła o tym, że w związku z tym, że klient stanowi o rentowności przedsiębiorstwa, to należy tak sterować wszystkimi fazami procesu produkcyjnego wyrobu, aby klient był zadowolony z jakości tego wyrobu,**

TQM została sformalizowana przez Amerykanów (W.E.Deming, P.Crosby, J.M.Juran, A.V.Feigenbaum), Japończyków (E.Toyoda, M.Imai, K.Ishikawa) i Brytyjczyka J.Oaklanda,

Każdy z powyższych Autorów zdefiniował własne zasady TQM. Wszystkie one obracają się jednak wokół zasady Toyody: „**Jakość jest najważniejszym kryterium oceny przydatności produktów dla klienta, a to właśnie klient umożliwia funkcjonowanie wytwórcy tych produktów**”.

Stąd wniosek, że producent wytwarzający produkty kiepskie powinien wypaść z rynku.

# TQM w wytwarzaniu oprogramowania

Podstawowe działania:

1. Ciągłe badanie czy, Z PUNKTU WIDZENIA Klienta:
  - Budujemy właściwy system ?
  - Budujemy system dobrze ?
  - Właściwy system budujemy dobrze od samego początku?
2. W razie problemów czy niepewności wprowadzamy działania modyfikujące:
  - produkt,
  - ew. proces.
3. Badamy wyniki działań modyfikacyjnych.

# Cykl Deminga – podstawowy cykl zarządzania realizacja TQM, ISO 900...

**Cykl Deminga (znany także jako Koło Deminga, Cykl PDCA)** jest koncepcją zarządzania (jakością) opracowaną przez W.E. Deminga ( 14 zasad Deminga).

Koncepcja ta mówi o ciągłym doskonaleniu (pojęcie znane m.in. z TQM) przebiegającym w czterech następujących po sobie etapach: planowanie – wykonanie – sprawdzenie – poprawienie (ang. Plan – Do – Check – Act).

Poszczególne etapy polegają na:

1. **Planowanie** – w tym etapie określony zostaje sposób działania, który doprowadzić ma do określonego celu.
2. **Wykonanie** – ten etap polega na wykonaniu wcześniej zaplanowanych działań.
3. **Sprawdzenie** – w tym etapie bada się wyniki wcześniej podjętych działań. Sprawdza się stopień wykonania celów zawartych w planie.
4. **Poprawienie** – na podstawie wniosków wyciągniętych podczas sprawdzania doskonali się procesy oraz dostarcza pomysły i rozwiązania, które można zawrzeć w kolejnym planie.

źródło: [www.jakosc.biz/cykl-deminga/](http://www.jakosc.biz/cykl-deminga/)

# Jakość w terminologii ISO 9000

**jakość** - ogólny cel i właściwości wyrobu lub usługi decydujący o zdolności wyrobu lub usługi do zaspokojenia stwierdzonych lub przewidywanych potrzeb użytkownika wyrobu;

**system jakości** - odpowiednio zbudowana struktura organizacyjna z jednoznacznym podziałem odpowiedzialności, określeniem procedur, procesów i zasobów, umożliwiających wdrożenie tzw. *zarządzania jakością*;

**zarządzanie jakością** - jest związane z aspektem całości funkcji zarządzania organizacji, który jest decydujący w określaniu i wdrażaniu *polityki jakości*;

**polityka jakości** - ogólny zamierzenie i kierunków działań organizacji dotyczących jakości, w sposób formalny wyrażony przez najwyższe kierownictwo organizacji, będącej systemem jakości;

**audyt jakości** - systematyczne i niezależne badanie, mające określić, czy działania dotyczące jakości i ich wyniki odpowiadają zaplanowanym ustaleniom, czy te ustalenia są skutecznie realizowane i czy pozwalają na osiągnięcie odpowiedniego *poziomu jakości*.

# Polityka i system jakości

**Polityka jakości** to ogólne intencje i zamierzenia danej organizacji w odniesieniu do jakości [ISO8402] wyrażana w sposób formalny przez zarząd firmy.

- Musi być zdefiniowana i udokumentowana;
- Muszą być określone cele i zaangażowanie w jakość;
- Musi być zgodna z działaniami przedsiębiorstwa i oczekiwaniami klienta;
- Musi być zakomunikowana i rozumiana na wszystkich szczeblach zarządzania.

**System jakości** to struktura organizacyjna, przydział odpowiedzialności, procedury postępowania, zasoby użyte do implementacji polityki jakości w danej organizacji [ISO8402]

- **pełnomocnik** lub zespół do spraw jakości;
- **księga jakości**: udokumentowane procedury systemu jakości.

# Model jakości ISO 9126 – atrybuty jakości

## 1. Funkcjonalność

- odpowiedniość
- dokładność
- współdziałanie
- zgodność
- bezpieczeństwo

## 2. Niezawodność

- dojrzałość
- tolerancja błędów
- odtwarzalność

## 3. Użyteczność

- zrozumiałość
- łatwość uczenia
- łatwość posługiwania się

## 4. Efektywność

- charakterystyka czasowa
- wykorzystanie zasobów

## 5. Pielęgnowalność

- dostępność
- podatność na zmiany
- stabilność
- łatwość walidacji

## 6. Przenośność

- dostosowywalność
- instalacyjność
- zgodność
- zamienność

Atrybut jakościowy to cecha lub charakterystyka mająca wpływ na jakość danego wyrobu lub usługi

# Zasady zarządzania jakością

Istnieje kilka głównych zasad zarządzania jakością. Oto one:

1. **Ukierunkowanie na klienta** (również klient wewnętrzny)
2. **Przywództwo** (budowa wizji, identyfikacja wartości)
3. **Zaangażowanie** ludzi (satysfakcja, motywacja, szkolenia)
4. **Podejście procesowe** (koncentracja na poszczególnych krokach procesu i relacjach pomiędzy tymi krokami, pomiary)
5. **Podejście systemowe** (całe otoczenie procesu wytwórczego - wewnętrz i na zewnątrz firmy - dostawcy)
6. **Ciągłe doskonalenie** (doskonalenie stanu obecnego, ewolucja a nie rewolucja)
7. **Rzetelna informacja** (zbieranie i zabezpieczanie danych do podejmowania obiektywnych decyzji)
8. **Partnerstwo dla jakości** (bliskie związki producentów z klientami)

# Zapewnienie Jakości Oprogramowania (ZJO) *software quality assurance, SQA*

Zgodnie z normą jest to „**planowany i systematyczny wzorzec wszystkich działań potrzebnych dla dostarczenia adekwatnego potwierdzenia że element lub produkt jest zgodny z ustanowionymi wymaganiami technicznymi**”.

**ZJO oznacza sprawdzanie:**

1. Czy zdefiniowano standardy i procedury ?
2. Czy plany są zdefiniowane zgodnie ze standardami ?
3. Czy procedury są wykonywane zgodnie z planami ?
4. Czy produkty są implementowane zgodnie z planami ?

Kompletne sprawdzenie jest zwykle niemożliwe. Projekty bardziej odpowiedzialne powinny być dokładniej sprawdzane odnośnie jakości.

W ramach ZJO musi być ustalony plan ustalający czynności sprawdzające przeprowadzane w poszczególnych fazach projektu.

Najbardziej istotnym kryterium przy zapewnianiu jakości jest **ryzyko**.

# Ryzyko utraty jakości

Najbardziej istotnym kryterium przy zapewnianiu jakości jest **ryzyko**. Najczęstszymi czynnikami ryzyka utraty jakości są:

- innowacyjność projektu,
- złożoność projektu,
- niedostateczne wyszkolenie personelu,
- zbyt małe doświadczenie personelu,
- niesformalizowane (tworzone i zarządzane ad hoc) procedury
- niska dojrzałość organizacyjna wytwórcy.

**Dla zmniejszenia ryzyka personel ZJO powinien być zaangażowany w projekt programistyczny jak najwcześniej.**

Powinien on sprawdzać wymagania użytkownika, plany, procedury i dokumenty na zgodność ze standardami i przyjętymi procedurami postępowania.

Wynika to z faktu, że dodatkowe koszta związane z problemem lub błędem są tym większe, im później zostanie on zidentyfikowany.

# Zadania zapewniania jakości

## Firma

- ciągła pielęgnacja (doskonalenie) procesu wytwarzania
- definiowanie standardów
- nadzór i zatwierdzanie procesu wytwarzania

## Przedsięwzięcie (project)

- dostosowywanie standardów
- ocena planów wytwarzania i jakościowych
- przeglądy przedsięwzięcia
- testowanie i udział w inspekcjach
- audyt systemu zarządzania konfiguracją
- udział w komitecie sterującym projektu

# **Procesy obsługiwane przez personel ZJO**

## **Tworzenie standardowych wzorców produktów, działań, procesu**

- tworzenie standardu
- wdrażanie standardu

## **Kontrola jakości**

- ocena produktu
- ocena działania, procesu
- zatwierdzanie jakości

## **Analiza działalności firmy**

- zbieranie danych
- analiza danych

# Personel ZJO powinien ustalić, czy... (1)

- Projekt jest właściwie zorganizowany, z odpowiednim cyklem życiowym;
- Członkowie zespołu projektowego mają zdefiniowane zadania i odpowiedzialności;
- Plany w zakresie dokumentacji są implementowane;
- Dokumentacja zawiera to, co powinna zawierać;
- Przestrzegane są standardy dokumentacji i kodowania;
- Standardy, praktyki i konwencje są przestrzegane;
- Dane pomiarowe są gromadzone i używane do poprawy produktów i procesów;
- Przeglądy i audyty są przeprowadzane i są właściwie kierowane;
- Testy są specyfikowane i rygorystycznie przeprowadzane;

# **Personel ZJO powinien ustalić, czy... (2)**

- Problemy są rejestrowane i reakcja na problemy jest właściwa;
- Projekty używają właściwych metod, narzędzi i technik;
- Oprogramowanie jest przechowywane w kontrolowanych bibliotekach;
- Oprogramowanie jest przechowywane w chroniony i bezpieczny sposób;
- Oprogramowanie od zewnętrznych dostawców spełnia odpowiednie standardy;
- Rejestrowane są wszelkie aktywności związane z oprogramowaniem;
- Personel jest odpowiednio przeszkolony;
- Zagrożenia projektu są zminimalizowane.

# Zakres działań dla zapewnienia jakości

- Modele i miary służące ocenie kosztu i nakładu pracy
- Modele i miary wydajności ludzi
- Gromadzenie danych
- Modele i miary jakości
- Modele niezawodności
- Ocena i modelowanie wydajności oprogramowania
- Miary struktury i złożoności artefaktów: dokumentów, projektu, kodu
- Ocena dojrzałości technologicznej
- Zarządzanie z wykorzystaniem metryk
- Ocena metod i narzędzi

# Klasyfikacja zadań zapewnienia jakości

- Certyfikacja systemów przed skierowaniem do produkcji
- Wymuszanie standardów gromadzenia i przetwarzania danych
- Recenzowanie i certyfikacja wytwarzania i dokumentacji
- Opracowanie standardów dotyczących architektury systemu i praktyk programowania
- Recenzowanie projektu systemu pod względem kompletności
- Testowanie nowego lub zmodyfikowanego oprogramowania
- Opracowanie standardów zarządzania
- Szkolenie

## **Uwaga:**

**Pomiary odgrywają istotną rolę, jednakże są one postrzegane jako jedno z wielu specjalistycznych działań, a nie podstawa całego procesu zapewnienia jakości.**

# Normy dotyczące jakości

Oprogramowanie jest rozumiane jako jeden z rodzajów wyrobów

ISO 8402

**Terminologia**

ISO 9000

**Wytyczne wyboru modelu**

ISO 9001

ISO 9002

ISO 9003

**Modele systemu jakości**

ISO 9004

**Elementy systemu jakości**

IEC/TC 56

**Niezawodność oprogramowania systemów krytycznych**

ISO/IEC 1508

**Bezpieczeństwo oprogramowania systemów krytycznych**

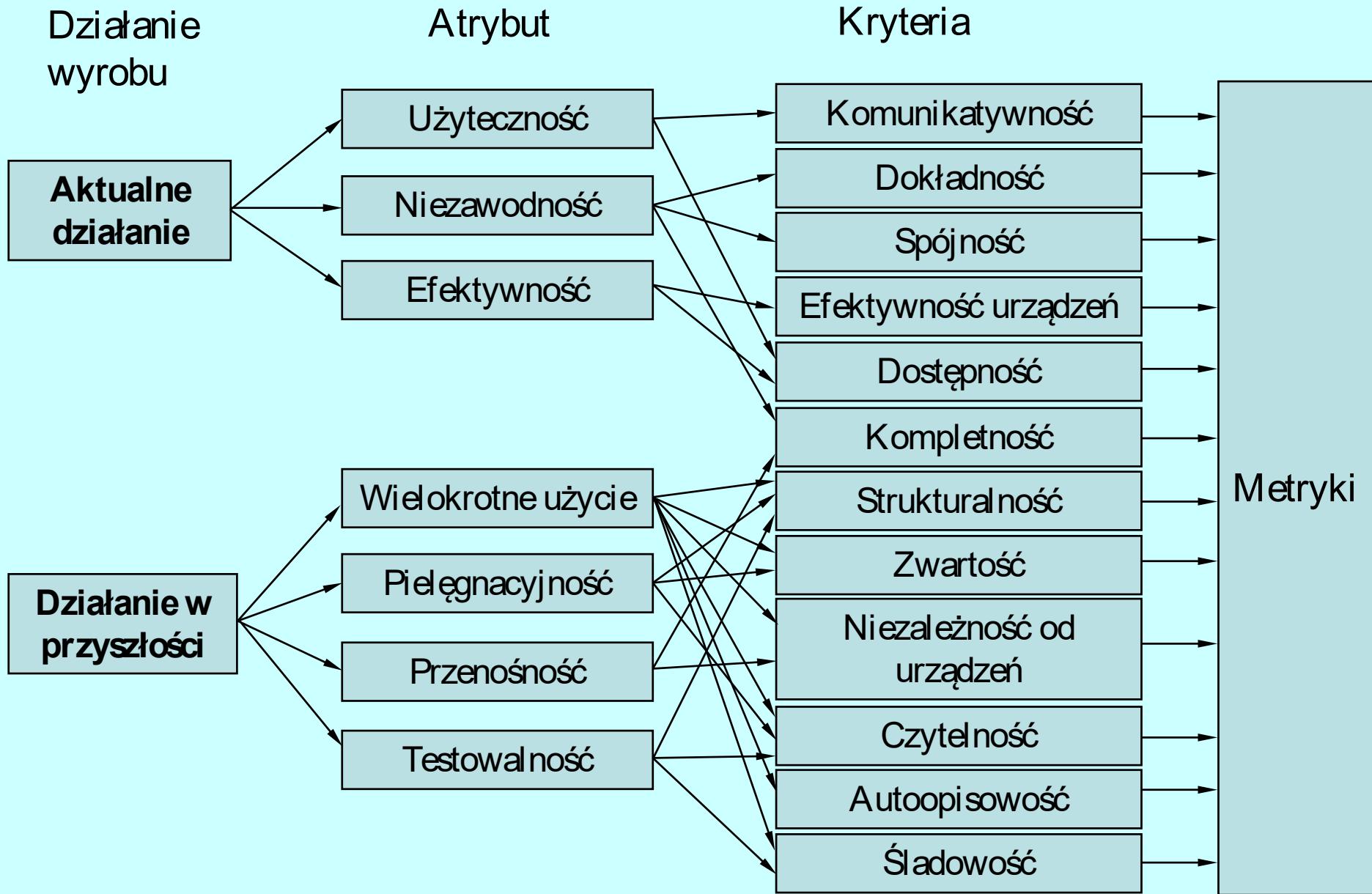
# Norma IEEE-730

Norma IEEE-730 podaje ogólne ramy planu zapewniania jakości. Powinien on obejmować następujące zagadnienia:

- analiza punktów widzenia
- referencje wykonawcy
- zarządzanie przedsięwzięciem informatycznym
- dokumentacja
- standaryzacja działań
- przeglądy i audyty
- zarządzanie konfiguracją oprogramowania
- raport napotykanych trudności i podjętych działań prewencyjnych
- wykorzystywane metody i narzędzia
- kontrola kodu, mediów, dostawców
- zarządzanie hurtowniami danych
- pielęgnacja

Norma IEEE-730 została uzupełniona i uszczegółowiona normą IEEE-983.

# Atrybuty jakości oprogramowania



# Dojrzałość procesów wytwórczych

## Niedojrzałość

1. Improwizacja podczas procesu wytwórczego
2. Proces jest wyspecyfikowany, ale specyfikacja nie jest stosowana
3. Doraźne reagowanie w sytuacji kryzysów
4. Harmonogram i budżet są przekraczane
5. Funkcjonalność jest stopniowo okrajana
6. Jakość produktu jest niska
7. Brak obiektywnych kryteriów oceny

## Dojrzałość

1. Zdolność do budowy oprogramowania jest cechą organizacji a nie personelu
2. Proces jest zdefiniowany, znany i wykorzystywany
3. Proces jest obserwowany i ulepszany
4. Prace są planowane i monitorowane
5. Role i odpowiedzialności są zdefiniowane
6. Obiektywna, ilościowa ocena

# CMM ( CMMI) - model dojrzałości procesu wytwarzczego

## CMM – Capability Maturity Model

- Wykorzystywany w procedurach klasyfikacji potencjalnych wykonawców oprogramowania dla Departamentu Obrony USA
- Wyróżniono 5 poziomów dojrzałości wytwórców (poczynając od poziomu najniższego):
  - poziom początkowy - 1 (proces chaotyczny)
  - poziom powtarzalny - 2 (proces zindywidualizowany)
  - poziom zdefiniowany - 3 (proces zinstytucjonalizowany)
  - poziom zarządzany - 4 (proces + informacje zwrotne dla sterowania procesem)
  - poziom optymalizujący - 5 (proces + informacje zwrotne wpływające na ulepszenie procesu )
- Niewiele firm uzyskało poziom 3-ci, umożliwiający uzyskanie prawa dostaw dla Departamentu Obrony USA. Poziom 5 posiadają: Motorola, Intel
- IBM w zakresie oprogramowania promu kosmicznego dla NASA uzyskał poziom 5-ty (najwyższy).

# Główne czynniki poprawy jakości

W zależności od występujących problemów dążymy do

- Poprawa zarządzania projektem
- Wzmocnienie inżynierii wymagań ( zarządzanie wymaganiami )
- Zwiększenie nacisku na kwalifikacje i wyszkolenie ludzi ( rola szkoleń !)

Możliwe do uzyskania wyniki dot. poprawy jakości

- Szybsze wykonywanie pracy (lepsze narzędzia, RAD, CASE) 10%
- Bardziej intelligentne wykonywanie pracy (lepsza organizacja i metody) 20%
- Powtórne wykorzystanie pracy już wykonanej (ponowne użycie) 65 %

# **Plan zapewnienia jakości oprogramowania (PZJO)**

**Plan zapewnienia jakości oprogramowania (PZJO) powinien być sporządzany i modyfikowany przez cały okres życia oprogramowania. Pierwsze jego wydanie powinno pojawić się na końcu fazy wymagań użytkownika.**

PZJO powinien ustalać i opisywać wszelkie aktywności związane z zapewnieniem jakości dla całego projektu. Odpowiednie sekcje planu jakości powinny dotyczyć wszystkich ustalonych w danym modelu rozwoju oprogramowania faz cyklu życia oprogramowania.

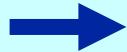
- Podane dalej zalecenia co do PZJO pochodzą z norm ISO serii 25 000.
- Zalecany (podany dalej) spis treści może i niekiedy powinien być uzupełniony o punkty specyficzne dla konkretnego projektu.

# Styl, odpowiedzialność, sekcje PZJO

- **Styl.** PZJO powinien być zrozumiały, lakoniczny, jasny spójny i modyfikowalny.
- **Odpowiedzialność.** PZJO powinien być wyprodukowany przez komórkę jakości zespołu podejmujący się produkcji oprogramowania. PZJO powinien być przejrzany i zrecenzowany przez ciało, któremu podlega dana komórka jakości oprogramowania.
- **Medium.** Zwykle PZJO jest dokumentem papierowym. Może być także rozpowszechniony w formie elektronicznej.
- **Zawartość.** PZJO powinien być podzielony na 4 rozdziały, każdy dla następujących faz rozwoju oprogramowania:
  1. PZJO dla etapu wymagań użytkownika i analizy;
  2. PZJO dla etapu projektu architektury;
  3. PZJO dla etapu projektowania i konstrukcji;
  4. PZJO dla etapu budowy, testowania i instalacji oprogramowania.
- **Ewolucja.** PZJO powinien być tworzony dla następnej fazy po zakończeniu fazy poprzedniej.

# Spis treści PZJO (1)

Informacje organizacyjne



- a - Streszczenie (maksymalnie 200 słów)
- b - Spis treści
- c - Status dokumentu (autorzy, firmy, daty, podpisy, itd.)
- d - Zmiany w stosunku do wersji poprzedniej

Zasadnicza zawartość dokumentu



- 1. Cel**
- 2. Referencje, odsyłacze do innych dokumentów**
- 3. Zarządzanie**
- 4. Dokumentacja**
- 5. Standardy, praktyki konwencje i metryki**
  - 5.1. Standardy dokumentacyjne
  - 5.2. Standardy projektowe
  - 5.3. Standardy kodowania
  - 5.4. Standardy komentowania
  - 5.5. Standardy i praktyki testowania
  - 5.6. Wybrane metryki pomocne przy ZJO
  - 5.7. Ustalenia dotyczące sposobu monitorowania zgodności z planem

..... na następnym slajdzie

# Spis treści PZJO (2)

Zasadnicza zawartość dokumentu



..... z poprzedniego slajdu

- 6. Przeglądy i audyty**
  - 7. Testowanie**
  - 8. Raportowanie problemów i akcje korygujące**
  - 9. Narzędzia, techniki i metody**
  - 10. Kontrola kodu**
  - 11. Kontrola mediów**
  - 12. Kontrola dostawców**
  - 13. Zbieranie, pielęgnacja i utrzymanie zapisów**
  - 14. Szkolenie**
  - 15. Zarządzanie ryzykiem**
  - 16. Przegląd pozostałej części projektu**
- Dodatek A: Słownik pojęć i akronimów**

Numeracja punktów nie powinna być zmieniana. Jeżeli pewien punkt nie ma treści, powinna tam znajdować się informacja „Nie dotyczy”.

Informacje nie mieszczące się w tym spisie treści powinny być zawarte w dodatkach. Punkty 3-15 powinny określać jak plany techniczne i zarządzania będą sprawdzane.

# Spis treści PZJO - omówienie (1)

- **Cel.** Sekcja ta powinna krótko określić: cel PZJO, rodzaj czytelnika, produkty programistyczne podlegające PZJO, zamierzone użycie oprogramowania, fazę cyklu życiowego, do którego PZJO się odnosi.
- **Zarządzanie.** Sekcja powinna opisywać organizację zarządzania jakością i związane z nią odpowiedzialności i role, bez określania przypisania ludzi do ról i bez określania pracochłonności i harmonogramu. Zalecana jest następująca struktura tego rozdziału:
- **Organizacja:** identyfikacja ról w organizacji (kierownik projektu, prowadzący zespołu, inżynierowie oprogramowania, bibliotekarze oprogramowania, prowadzący weryfikację i validację , inżynier ZJO), opis związków pomiędzy rolami, opis interfejsu z organizacją użytkownika.
- **Zadania:** Opisuje zadania ZJO, które będą wykonywane w tej fazie.
- **Odpowiedzialności:** Opisuje odpowiedzialność poszczególnych ról za poszczególne zadania oraz ustala kolejność wybranych zadań.

# Spis treści PZJO - omówienie (2)

- **Dokumentacja.** Identyfikuje wszystkie dokumenty, które będą wyprodukowane w tej fazie. Sekcja powinna ustalać jak te dokumenty będą sprawdzane na zgodność ze standardami.
- **Standardy, konwencje metryki.** Opisuje je detalicznie lub zawiera odsyłacze do innych dokumentów.
- **Przeglądy i audyty.** Identyfikuje techniczne przeglądy, przejścia, inspekcje, audyty mające zastosowanie w tej fazie, oraz cel każdego z nich. Opisuje sposoby monitorowania zgodności tych procedur z planem oraz rolę personelu ZJO w tych procedurach.
- **Testy.** Opisuje w jaki sposób czynności weryfikacji i walidacji oprogramowania będą monitorowane i w jaki sposób będą sprawdzane testy akceptacyjne .
- **Raportowanie problemów i akcje korygujące.** Identyfikuje procedury zgłaszania problemów oraz podejmowania akcji mających na celu usunięcie problemów. Może opisywać metryki stosowane do procedur zgłaszania problemów mające wpływ na jakość oprogramowania.

# **Spis treści PZJO - omówienie (3)**

- **Kontrola kodu.** Procedury stosowane do pielegnacji, przechowywania, zabezpieczania i dokumentowania kodu oprogramowania.
- **Kontrola mediów.** j.w., ale dotyczy mediów, na których oprogramowanie i dokumentacja będą przechowywane.
- **Kontrola dostawców.** Procedury stosowane do kontroli zewnętrznych organizacji lub osób, które rozwijają lub dostarczają oprogramowanie niezbędne dla projektu. Procedury powinny określać standardy, które mają być stosowane przez dostawców, oraz powinny określać sposoby kontroli przestrzegania tych standardów.
- **Zbieranie, pielegnacja i utrzymanie zapisów.** Identyfikuje procedury stosowane do przechowywania informacji zebranych ze wszelkich aktywności, takich jak spotkania, przeglądy, przejścia, audyty, sporządzanie notatek, korespondencji. Powinny określać gdzie te informacje/dokumenty są przechowywane i jak długo, oraz określać sposób dostępu do tych informacji.

# **INŻYNIERIA OPROGRAMOWANIA**

**Dziękuję za uwagę**

# **INŻYNIERIA OPROGRAMOWANIA**

**wykład 11:  
WDROŻENIE  
KONSERWACJA  
EKSPLOATACJA  
OPROGRAMOWANIA**

**dr inż. Leszek Grocholski**  
Zakład Inżynierii Oprogramowania  
Instytut Informatyki  
Uniwersytet Wrocławski

# **Wdrożenie - definicja**

## I.

**Wdrożenie systemu** – etap cyklu życia, polegający na **instalacji i dostosowaniu oprogramowania do wymagań użytkownika**, a także **migracji danych oraz testowaniu i uruchomieniu** systemu informatycznego.

Źródło: *Wikipedia*

## II. Wdrożenie

1. nauczenie kogoś wykonywania rutynowych czynności;
  2. podjęcie jakiegoś działania;
  3. rozpoczęcie stosowania czegoś w praktyce.
- 4. Wdrożyć = wprowadzić w stan używalności !**  
(najczęściej w odniesieniu do systemów informatycznych ale też ISO)

Źródło: Słownik języka polskiego [www.sjp.pl](http://www.sjp.pl)

# **Wdrożenie - pytania**

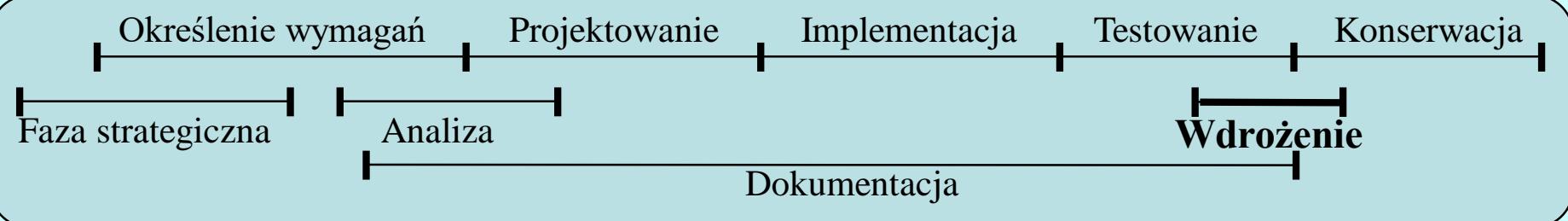
Dobrym przykładem wdrożeń są systemy klasy ERP ( Enterprise Resources Planning ). To oprogramowanie, które składa się w modułów.

Najpopularniejsze to: księgowość, środki trwałe, controlling, płace/kadry, sprzedaż, gospodarka magazynowa( logistyka), zarządzanie produkcją

**Pytania:**

- 1. Jaki jest koszt wdrożenia systemu w porównaniu z ceną licencji na system ?**
- 2. Ile trwało i ile kosztowało wdrożenie systemu ERP na Politechnice Wr. rozpoczęte w roku 2004? (2011? )**
- 3. Ile trwało wdrożenie systemu Kadry/Płace na Uniwersytecie Wr ?**
- 4. Ile kosztowało ile trwało wdrożenie systemy ERP na Uniwersytecie Wr ?**

# Wdrożenie ( etapy)



**Na fazę wdrożenia składają się:**

- Analiza specyficznych potrzeb a następnie dostosowanie do potrzeb klienta:  
cykle (minicykle) wytwarzania: analiza, projektowanie, kodowanie, testowanie
- Instalacja sprzętu i przeniesienie oprogramowania
- Wypełnienie baz danych
- Szkolenie użytkowników końcowych i administratorów systemu
- Nadzorowane korzystanie z systemu, często równolegle z tradycyjnym sposobem pracy
- Usuwanie błędów w oprogramowaniu i dokumentacji użytkowej
- Przekazanie systemu klientowi

# Główny problem firmy wdrażającej

**Wdrożenie trwa długo - NIESTETY NIE JEST MOŻLIWE precyzyjne szacowanie**

Trzy proste prawdy:

1. Nie jest możliwe zebranie i opisanie wszystkich wymagań na początku projektu !
2. Jakiekolwiek wymagania zostaną zebrane, na pewno się zmienią !
3. Zawsze będzie więcej do zrobienia, niż pozwalają czas i pieniądze !

Ale kierownictwo klienta potrzebuje i szuka w szacowaniu tego czego tam nie ma:

- Dokładnych przewidywań co do przyszłości: budżet i czas

Również w firmie wdrażającej trzeba zaplanować: ilu ludzi, czasu pieniędzy i innych zasobów będzie potrzebnych

**Zauważmy, że należy przygotować dokładne oszacowania: w sytuacji niepewności !:**

- Jeszcze niedokładnie opisanego końcowego systemu
- Często używając nowej albo nieprecyzyjnie określonej technologii
- Z zespołem różnych, często nieokreślonych ludzi
- W nieokreślonym nieznanym środowisku biznesowym
- Dla projektu, który rozpocznie się za 3,6 miesięcy albo za rok

# Najważniejsze czynniki sukcesu klienta

**W firmie, która kupuje i planuje wdrożyć oprogramowanie najważniejszym czynnikiem sukcesu jest oddelegowanie do wdrożenia osób, które:**

- Bardzo dobrze znają funkcjonowanie danej firmy
- Mają odpowiedni dużo czasu
- Mogą podejmować decyzje
- Potrafią skutecznie zarządzać ludźmi

Kolejne czynniki sukcesu to:

- Zrobienie i zapłaceniu za analizę przedwdrożeniową ( ale odpowiednią)
- Podział na projekty trwające co najwyżej 6 miesięcy, dostarczające konkretną wartość dla klienta
- Jasna i bezbłędna koncepcja identyfikatorów = kodów: pracowników, studentów, produktów, materiałów, magazynów itd. ...

# Problemy podczas wdrożenia

**Ważne jest planowanie i harmonogramowanie prac.** W tej fazie pojawia się szereg problemów, np. konieczność usunięcia błędów i wprowadzenia modyfikacji. **Z reguły, wykonawcy systemu nie mogą zarezerwować w pełni swojego czasu na prace związane z instalacją. Z drugiej strony, użytkownicy nie mogą zaniechać wykonywania przez nich bieżących prac.**

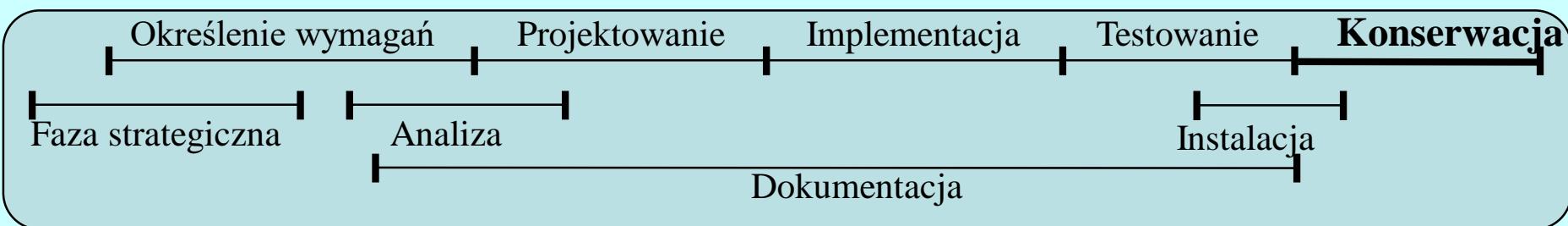
**Szkolenie użytkowników:** zaleca się, aby przeprowadzały je osoby, które bardzo dobrze znają damy obszar i były zaangażowane w określaniu wymagań (**tzw użytkownicy wiodący**).

**Wypełnienie bazy danych** jest często bardzo żmudnym procesem, wymagającym wprowadzenia danych z nośników papierowych. Niekiedy część danych jest w formie elektronicznej - wtedy z reguły potrzebne są specjalne programy konwersji. **Konwersja jest łatwiejsza, jeżeli znana jest specyfikacja struktury starej BD.**

**Opór klienta przed zmianą sposobu pracy.** Często użytkownicy systemu są to osoby po raz pierwszy stykający się z systemem (inni niż ci, którzy uczestniczyli w poprzednich fazach). Ważne jest uzyskanie ich akceptacji.

# Konserwacja oprogramowania

ang. maintenance



**Konserwacja oprogramowania polega na wprowadzeniu modyfikacji.**

Używa się również terminów „pielęgnacja”, „wsparcie” oraz „utrzymanie”.

Istnieją trzy główne rodzaje wprowadzanych w oprogramowaniu modyfikacji:

- **Modyfikacje środowiska: sprzętu i oprogramowania.**
- **Modyfikacje poprawiające:** polegają na usuwaniu z oprogramowania błędów popełnionych w fazach wymagań, analizy, projektowania i implementacji .
- **Modyfikacje ulepszające:** polegają na poprawie jakości oprogramowania.
- **Modyfikacje dostosowujące:** polegają na dostosowaniu oprogramowania do zmian zachodzących w wymaganiach użytkownika lub w środowisku komputerowym.

## **Modyfikacje ulepszające**

- Poprawa wydajności pewnych funkcji
- Poprawa ergonomii interfejsu użytkownika
- Poprawa przejrzystości raportów
- Przeniesienie na nowy sprzęt, instalacja nowego oprogramowania systemowego czy bazy danych

## **Modyfikacje dostosowujące** wynikają z:

- Zmian wymagań użytkowników
- Zmian przepisów prawnych dotyczących dziedziny problemu
- Zmian organizacyjnych po stronie klienta

**Jeżeli modyfikacja dostosowująca wprowadza wiele zmian, lub złożoną zmianę, to jej realizacja powinna mieć charakter cyklu (minicyklu) wytwarzania oprogramowania tzn: analiza, projektowanie, kodowanie, testowanie.**

# **Analiza potrzeb w modyfikacji dostosowującej**

**Analiza powinna uwzględniać:**

- Powód zmiany ( np. zmiany przepisów prawnych, rozszerzenie działalności)
- Znaczenie wprowadzenia zmiany dla użytkowników
- Koszt wprowadzenia zmiany
- Wpływ zmiany na poszczególne składowe systemu
- Wpływ zmiany na poszczególne składowe dokumentacji technicznej.

Dopiero po dokonaniu oceny zmiany podejmowana jest decyzja o jej ewentualnej realizacji. W przypadku bardzo dużych przedsięwzięć może zostać powołana w tym celu specjalna komisja.

**Uwaga: nie należy wprowadzać każdej zmiany natychmiast. Zaleca się grupowanie zmian, których wykonanie prowadzi do nowej wersji systemu.**

# Koszty konserwacji oprogramowania

Pytanie:

Ile kosztuje konserwacja oprogramowania ?

Panuje przekonanie, aby niżej oceniać koszt konserwacji niż koszt wytworzenia oprogramowania. Koszty konserwacji są jednak często duże !  
Niedocenianie nakładów pracy na fazę konserwacji jest jedną z głównych przyczyn opóźnień przedsięwzięć programistycznych dot. konserwacji.

**Obiektywne czynniki wpływające na koszty konserwacji oprogramowania:**

- **Stabilność domeny (otoczenia) w którym pracuje system.** Zmiany zachodzące w przepisach prawnych, zmiany struktury organizacyjnej i sposobów działania po stronie klienta prowadzą do zmian wymagań wobec systemu.
- **Stabilność platformy sprzętowej i oprogramowania systemowego**
- **Czas użytkowania systemu.** Całkowite koszty konserwacji rosną, gdy system jest eksploatowany przez dłuższy czas.

# Czynniki redukcji kosztów konserwacji (1)

**1. Znajomość dziedziny problemu.** Jeżeli analitycy pracujący nad systemem dobrze znają daną dziedzinę problemu, mają mniej trudności z właściwym zebraniem wymagań oraz budową oddającego rzeczywistość modelu.

**2. Wysoka jakość modeli**, w szczególności:

ich spójność, stopień powiązania składowych oraz przejrzystość.

**3. Wysoka jakość dokumentacji technicznej.**

Dokumentacja powinna:

- w pełni odpowiadać systemowi
- być wystarczająco szczegółowa
- być zgodna z przyjętymi w firmie standardami.

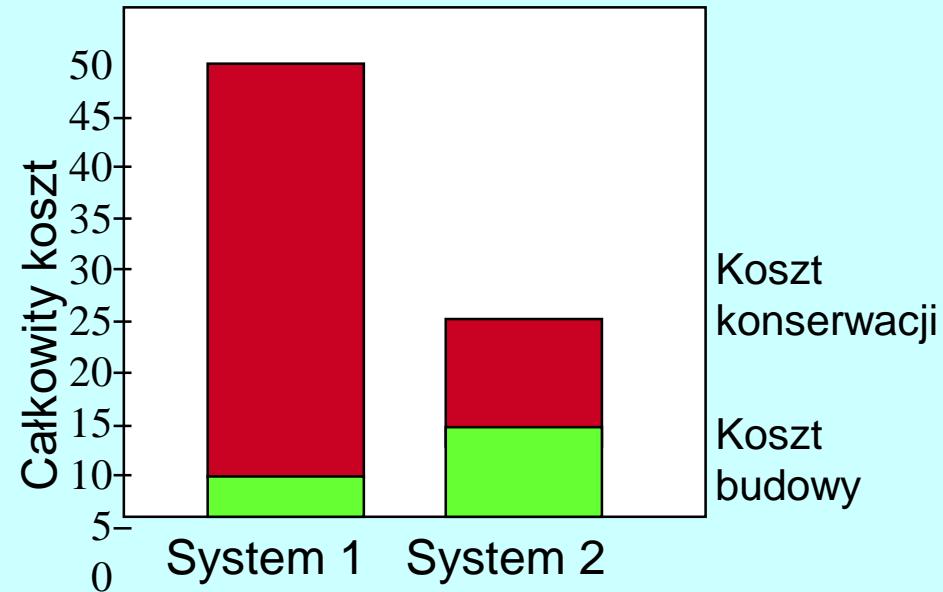
**4. Stabilność personelu.** Niezależnie od jakości dokumentacji, pewne aspekty systemu są znane tylko osobom bezpośrednio uczestniczącym w realizacji. Niekoniecznie muszą one same dokonywać modyfikacji, ale mogą istotnie wspomagać konsultacjami.

# Czynniki redukcji kosztów konserwacji (2)

5. **Środowisko implementacji.** Zaawansowane środowisko implementacji sprzyja skróceniu czasu niezbędnego na wprowadzenie modyfikacji.
6. **Niezawodność oprogramowania.** Wysoka niezawodność oprogramowania przekazanego klientowi zmniejsza liczbę modyfikacji.
7. **Inżynieria odwrotna.** Pod tym pojęciem rozumie się odtwarzanie dokumentacji technicznej na podstawie istniejącego oprogramowania.
8. **Zarządzanie wersjami.**

**GENERALNIE** to

9. „**Rzetelne wytworzenie**”  
→ a to kosztuje



# Eksplotacja oprogramowania

## Pytania:

1. Ile osób pracuje na Uniwersytecie Wr przy eksploatacji systemu USOS?
2. Ile specjalistów zapewniających eksploatację IT pracuje w Instytucie ?
3. Ile osób pracuje w VOLVO IT w dziale maintenance i service ?
4. Ile kosztuje eksploatacja oprogramowania ?

Jednym z elementów eksploatacji oprogramowania jest omówiona powyżej

### KONSERWACJA

Polega na wprowadzaniu zmian.

## Pytanie

Dlaczego trzeba zarządzać eksploatacją ?

# Eksplotacja oprogramowania - ITIL

Od lat 80 XX w zapewnienie korzystania ze sprzętu i oprogramowania jest traktowane jako dostarczanie/świadczenie usług informatycznych.

Dostarczanie/świadczenie usług może być zapewnione przez dział informatyki lub zewnętrzna firma. Na Uniwersytecie jest DUI – dział usług informatycznych.

W grupie VOLVO jest specjalna spółka – VOLVO IT.

**ITIL** (ang. *Information Technology Infrastructure Library*) - kodeks postępowania dla działów informatyki. Zbiór zaleceń, jak efektywnie i skutecznie oferować usługi informatyczne. Prace nad pierwszą wersją rozpoczęto w połowie lat 80 w Wielkiej Brytanii lat 80 na zlecenie administracji rządowej.

Pierwsza książka "Help desk" ukazała się w 1989 roku.

W roku 2001 opublikowano drugą wersję biblioteki, opisaną w dwóch głównych publikacjach a w roku 2007 opublikowano wersję trzecią.

Źródło: [Wikipedia](#)

ITIL obejmuje m. in. takie działy jak:

- zarządzanie incydentami,
- zarządzanie problemami,
- zarządzanie zmianami.

# Zarządzanie problemami - zgłoszanie problemów

Użytkownicy dokumentują problemy powstałe podczas działania systemu w specjalnym dokumencie **Zgłoszenie Problemu z Oprogramowaniem (ZPO)**.

Nie powinny to być problemy związane z brakiem wiedzy u użytkownika.

**Każde ZPO powinno dotyczyć dokładnie jednego problemu.**

**ZPO powinno zawierać:**

- Nazwę elementu konfiguracji oprogramowania.
- Wersję lub wydanie tego elementu.
- Priorytet problemu w stosunku do innych problemów (priorytet ma dwa wymiary: **krytyczność** - na ile problem jest istotny dla funkcjonowania systemu, oraz **pilność** - maksymalny czas usunięcia problemu).
- Opis problemu.
- Opis środowiska operacyjnego.
- Zalecane rozwiązanie problemu (o ile użytkownik jest je w stanie określić).
  
- Problemy mogą wynikać z wielu przyczyn: błędy w oprogramowaniu, brak funkcji, które okazały się istotne, ograniczenia, których nie uwzględniono lub które się pojawiły, zmiany w środowisku systemu.

Stąd każdy ZPO powinien być oceniony odnośnie odpowiedzialności za problem (kto ma ponosić koszt).

# Eksplotacja - zlecanie zmiany

ZPO jest analizowane przez producenta oprogramowania.

Może ono być odrzucone, mogą być podjęte negocjacje z klientem celem ustalenia warunków (np. finansowych, czasowych) wprowadzenia zmian, lub zgłoszenie może być zaakceptowane.

Po tych ustaleniach tworzony jest dokument **Zlecenie Zmiany w Oprogramowaniu (ZZO)** ang. crequest , który zawiera:

- Nazwę elementu konfiguracji oprogramowania.
- Wersję lub wydanie tego elementu.
- Wymagane zmiany.
- Priorytet zlecenia (krytyczność, pilność).
- Personel odpowiedzialny.
- Szacunkową datę początku, datę końca i pracochłonność w osobo-dniach.

**Jeżeli istnieje dokumentacja to ZZO powinien zawierać sekcje dotyczące:**

- Zmian w dokumentach wymagań (użytkownika, na oprogramowanie)
- Zmian w dokumentach projektowych (sekcja dla każdego dokumentu)
- Zmian w dokumentach dotyczących zarządzania, testowania, zapewniania jakości.

# Eksplotacja - ocena efektu zmian

ZZO jest realizowane przez wyznaczony personel. Nanoszone są zmiany do kodu i odpowiednich dokumentów opisujących oprogramowanie.

Dokument **Raport z Modyfikacji Oprogramowania (RMO)** określa wszystkie zmiany w kodzie i w dokumentach. Zmiany są oceniane; ocenie podlegają następujące aspekty:

- **Wydajność** (szybkość) oprogramowania
- **Zużycie zasobów** (pamięci dyskowej, czasu procesora, pamięci operacyjnej).
- **Stopień powiązania** elementów systemu (kohezji).
- **Niezależność zmienianego elementu** (od pozostałych elementów programowania).
- **Złożoność** (na ile została zwiększena).
- **Spójność** (odstępstwa od reguły spójności, np. interfejsów użytkownika).
- **Przenaszalność** (czy oprogramowanie będzie działać na innej platformie).
- **Niezawodność** (czy zmiana mogła spowodować jej obniżenie).
- **Podatność na konserwację** (inaczej pielęgnacyjność - czy nie została obniżona).  
Oceniane są wszelkie ewentualne odstępstwa od przyjętych standardów.)
- **Bezpieczeństwo** (czy zmiana nie tworzy zagrożenia dla biznesu klienta)
- **Ochrona** (czy zmiana nie powoduje wyłomów w ochronie systemu).

# Kluczowe czynniki sukcesu

- Wysoka jakość definicji wymagań, modelu i projektu
- Dobra znajomość środowiska implementacji
- Właściwa motywacja pracowników
- Właściwe oszacowanie kosztów konserwacji

## **Podstawowy rezultat:**

poprawiony kod, projekt, model i specyfikacja wymagań.

# **INŻYNIERIA OPROGRAMOWANIA**

**Dziękuję za uwagę**

# **PODSTAWY INŻYNIERII OPROGRAMOWANIA**

**wykład 6:**

**ZARZĄDZANIE PROJEKTEM PROGRAMISTYCZNYM**

**dr inż. Leszek Grocholski**

# ZARZĄDZANIE - definicja

**Zarządzanie - wyszukiwarka poda linki do ponad 26,5 miliona artykułów.**

Zarządzanie jako działanie praktyczne istnieje od dawien dawna !

Szczególnie rozwijało się podczas i po wojnach światowych.

## Definicje

1. **Zestaw działań** (obejmujący planowanie i podejmowane decyzji, organizowanie, przewodzenie, tj. kierowanie ludźmi, i kontrolowanie), skierowanych na zasoby organizacji (ludzkie, finansowe, rzeczowe i informacyjne) i **wykonywanych z zamiarem osiągnięcia celów organizacji w sposób sprawny i skuteczny.**

(**Griffin R. W., Podstawy zarządzania organizacjami**, Wydawnictwo Naukowe PWN, Warszawa 2005, s. 6)

2. **Działanie polegające na dysponowaniu zasobami**; ponieważ do zasobów najważniejszych należą ludzie, zasobami są pieniądze, a przez nie oddziałuje się na ludzi. Zarządzanie wiąże się z kierowaniem i bardzo często używa się łącznie terminów „organizacja i zarządzanie”, „kierowanie i zarządzanie”.

(**Pszczółkowski T., Mała encyklopedia prakseologii i teorii organizacji**, Ossolineum, Wrocław - Warszawa - Kraków - Gdańsk 1978, s. 288)

3. **Jest to działanie zmierzające do spowodowania funkcjonowania rzeczy, organizacji lub osób podległych, zgodnie z celami zarządzającego.**

(**Gliński B., Mała encyklopedia ekonomiczna**, Warszawa 1974, s. 929; za T. Pszczołowskim)

źródło: <https://mfiles.pl/pl/index.php/Zarządzanie>

# PROJEKT - definicja

**Projekt** jest **złożonym** działaniem o **charakterze jednorazowym**, które jest podejmowane dla osiągnięcia z góry określonych celów.

**Złożoność** wynika z szeregu działań, jakie należy wykonać w określonej kolejności, aby można było osiągnąć cele. Działania często wymagają złożonych zasobów.

**Jednorazowość** projektu jest jednym z jego kluczowych atrybutów, nie możemy bowiem nazwać projektem działania powtarzalnego.

W przypadku działań powtarzalnych mówimy raczej o operacjach lub procesach.

**Do podstawowych atrybutów projektu należy zaliczyć:**

- zdefiniowanie w czasie
- niepowtarzalność (jednorazowość)
- złożoność
- celowość

**!! Każdy projekt musi mieć szczegółowo określone TERMINY. Muszą one dotyczyć nie tylko projektu jako całości, ale każdego zadania w nim wykonywanego. Służy temu plan projektu !!**

**Projektem nie można nazwać pojedynczego zadania lub też zestawu niepowiązanych zadań.**

Wszystkie zadania wykonywane w projekcie muszą przyczyniać się do realizacji celów.

Nie wolno w projekcie realizować zadań nie przyczyniających się do osiągnięcia celów, ponieważ zmniejsza to efektywność projektu.

Źródło: <https://mfiles.pl/pl/index.php/Projekt>

# KIEROWNIK PROJEKTU - definicja

**Kierownik projektu** (*PM – Project Manager*) – specjalista w dziedzinie zarządzania projektami. Jest odpowiedzialny za planowanie, realizację i zamykanie projektu. Podstawowym zadaniem kierownika projektu jest zapewnienie osiągnięcia założonych celów projektu, wytworzenie produktu spełniającego określone wymagania jakościowe. PM jest odpowiedzialny za efekt końcowy realizowanego projektu i musi być aktywny podczas wszystkich etapów projektu.

Kierownik projektu może kierować m.in. projektami w budownictwie, projektami informatycznymi, projektami telekomunikacyjnymi, projektami finansowymi.

Kierownik projektu, aby sprawnie zarządzać projektem i posiadanymi zasobami ludzkimi, często pełni wiele funkcji jednocześnie.

## **Podstawowe funkcje osoby zarządzającej projektem:**

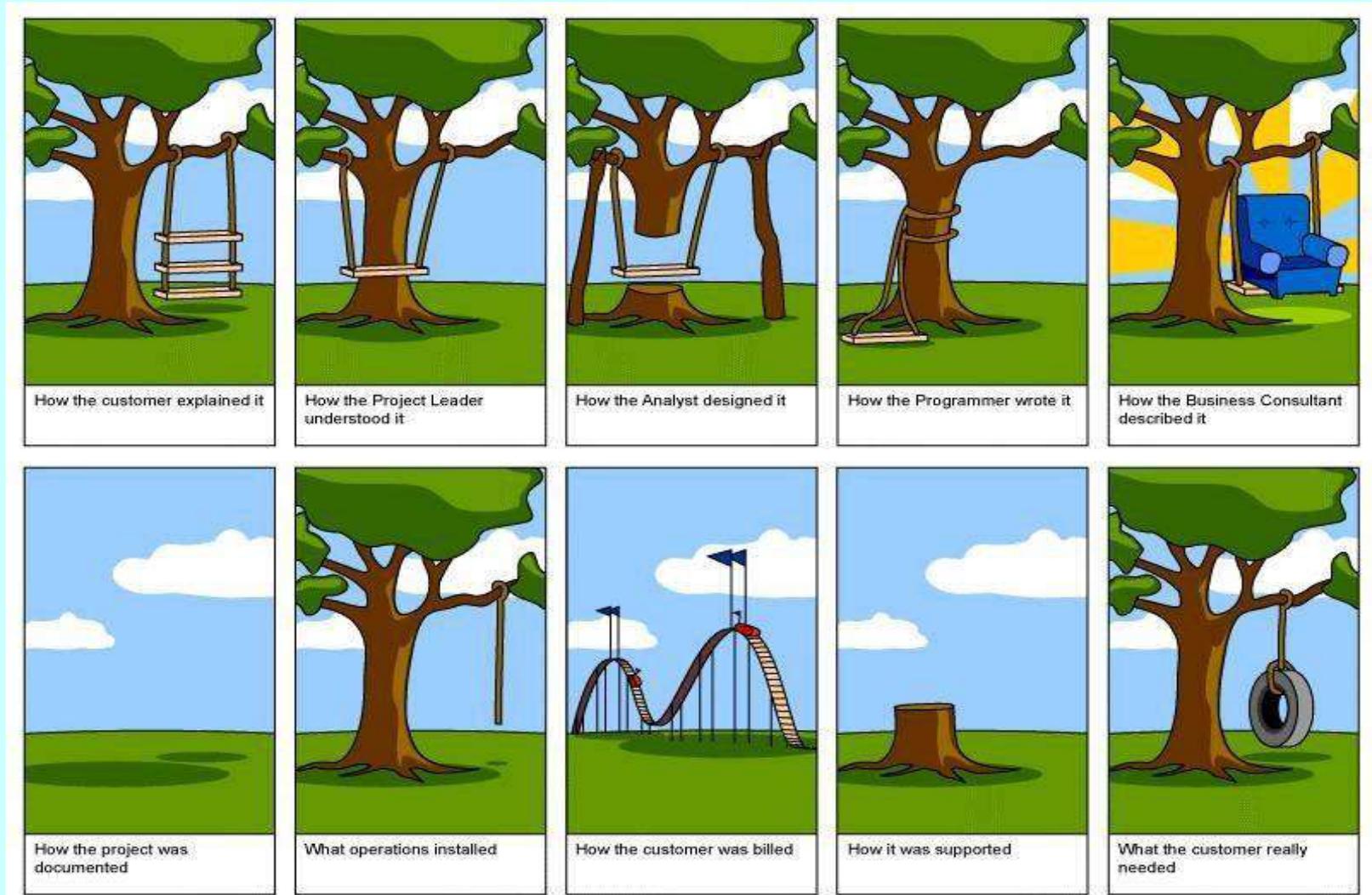
- planowanie
- organizowanie
- motywowanie
- kontrolowanie
- komunikowanie

Źródło: [https://pl.wikipedia.org/wiki/Kierownik\\_projektu](https://pl.wikipedia.org/wiki/Kierownik_projektu)

# PRAWDA o projekcie IT ☺

- Źródło: znany od ponad 30 lat komiks autorstwa Scott'a Yang'a.

Osoba potrzebująca opisała produkt następująco: służące do huśtania urządzenie zawieszane na drzewie .  
Zamawiała osoba z działu zamówień.



# SPECYFIKA PROJEKTU PROGRAMISTYCZNEGO

Pytanie: kiedy projekt (programistyczny) kończy się sukcesem ?

**Trzy proste prawdy:**

1. Nie jest możliwe zebranie i opisanie wszystkich wymagań na początku projektu !
2. Jakiekolwiek wymagania zostaną zebrane, na pewno się zmienią !
3. Zawsze będzie więcej do zrobienia, niż pozwalają czas i pieniądze !

Ale kierownictwo klienta potrzebuje i szuka w szacowaniu tego czego tam nie ma:

- Dokładnych przewidywań co do przyszłości: budżet i czas.

Również w firmie wytwarzającej trzeba zaplanować: ilu ludzi, czasu, pieniędzy i innych zasobów będzie potrzebnych.

**Zauważmy, że należy przygotować dokładne oszacowania w sytuacji niepewności !:**

- jeszcze niedokładnie opisanego systemu
- często używając nowej albo nieprecyzyjnie określonej technologii
- z zespołem różnych, często nieokreślonych ludzi
- w nieokreślonym nieznanym środowisku biznesowym
- dla projektu, który rozpocznie się w przyszłości - za 3, 6 miesięcy albo za rok

# **WIEDZA dot. ZARZĄDZANIA PROJEKTAMI**

**Wiedza jest ogromna.**

**Zarządzanie jest kierunkiem studiów**, np.:

- Wydział Zarządzania AGH
- Wydział Zarządzania Uniwersytetu Łódzkiego

Na Politechnice Wrocławskiej jest Wydział Informatyki i Zarządzania.

**Zarządzanie projektem jest kierunkiem studiów podyplomowych**, np.:

- Uniwersytet Ekonomiczny we Wrocławiu,  
w programie, np.: - Kompetencje Project Managera  
(Microsoft MS Project Certificate)
- WSB (Wyższa Szkoła Bankowa) we Wrocławiu
- WSB we Wrocławiu, też: *Zarządzanie projektem informatycznym*

Dostępnych jest dużo kursów przygotowujących do egzaminów dot. określonego certyfikatu zarządzania

# **STANDARDY ZARZĄDZANIA PROJEKTEM**

## **PRINCE2**

- Metodyka zarządzania projektami oparta na produktach (dokumentach)
- Opublikowany w 1996 r. (oparty o metodykę opracowaną w latach 70 XX w.)
- standard w Wielkiej Brytanii

## **PMI/PMBOK Guide – ISO 21500**

- PMI (Project Management Institute) powstało w 1969 roku w USA.

PMBOK pisuje 9 obszarów zarządzania:

- zarządzanie integralnością projektu
- ..... zakresem
- ..... czasem
- ..... kosztami
- ..... jakością
- ..... zasobami ludzkimi
- ..... komunikacją
- ..... ryzykiem
- ..... zaopatrzeniem

# **STANDARDY ZARZĄDZANIA PROJEKTEM ..**

Standardy zarządzania - projekty programistyczne

## **SCRUM**

- iteracyjna i inkrementalna ( = przyrostowa) metodyka prowadzenia projektów
- kolejne "sprinty" dostarczają coraz bardziej dopracowanych wyników
- zespół sam się organizuje
- codzienne Scrum-meeting'i

**KANBAN** metoda oparta wizualizacji zadań oczekujących na wykonanie i trakcie wykonania.

## **Programowanie ekstremalne (XP)**

- wydajne tworzenie małych i średnich "projektów wysokiego ryzyka", czyli takich, w których nie wiadomo do końca, co się tak naprawdę robi i jak to prawidłowo zrobić.

Inne mniej popularne, np:

- FDD (Feature Driven Development),
- DDD (Domain Driven Design)

# **4 DEMONY ZRZĄDZANIA PROJEKTEM**

**4 demony to:**

1. CZAS
2. BUDŻET
3. ZAKRES (wymagania funkcjonalne - co system na robić)
4. JAKOŚĆ (wymagania niefunkcjonalne - jak system ma działać)

**Uwaga:**

Każdym z powyższych czterech demonów trzeba zrządzić.

**Przypomnienie:**

**Zawsze będzie więcej do zrobienia, niż pozwalają czas i pieniądze !**

**Główny problem:**

Jak w powyższej sytuacji postępować ?

**Nie okłamywać klienta i siebie – ograniczyć zakres – zrobić to co klientowi jest w rzeczywistości potrzebne w MINIMALNEJ WERSJI.**

# Zarządzanie interesariuszami

**Kierownik projektu ze strony klienta NIE JEST JEDYNĄ osobą u klienta, do której obowiązku należy co najmniej interesowanie się przebiegiem i wynikami projektu.**

Społeczność związana z projektem jest zawsze większa, niż nam się wydaje!

**Jeżeli będziemy współpracować jedynie z kierownikiem projektu ze strony klienta to może nas czekać coś dziwnego.**

Ludzie i grupy, których nigdy nie spotkaliśmy ani nawet widzieliśmy zaczynają się pojawiać znikąd i stawiać nam różne, nieznane dotąd wymagania, np:

- Jedna grupa będzie chciała sprawdzać naszą architekturę.
- Inna będzie chciała upewnić się, że spełniamy korporacyjne reguły bezpieczeństwa.
- Jeszcze inna będzie chciała zrobić przegląd dokumentacji.

Każda osobę u klienta, która interesuje się przebiegiem lub wynikami projektu nazywamy interesariuszem.

**Interesariuszy należy zidentyfikować i zarządzać nimi !**

# Zarządzanie interesariuszami ..

**Na początku projektu, należy zrobić przegląd społeczności związanej z projektem, namierzyć wszystkich i rozpocząć relacje, zanim wykonawcy ( czyli my) będą ich potrzebować.**

W ten sposób, gdy przyjdzie ten moment, kiedy oprogramowanie trzeba będzie dostosować do reguł firmy ( zasad korporacyjnych) oraz ominąć pułapki organizacyjne interesariusze nie będą obcy i będą w dużo lepszej pozycji aby pomóc.

Podstawą każdej odnoszącej sukcesy organizacji jest zaufanie.  
Jednym z elementów zaufania jest poznanie się – interesariuszy trzeba poznać i zapoczątkować dobre stosunki.

**Pytanie:**

Kto w Państwa projekcie jest interesariuszem ?  
Za co on w firmie odpowiada?  
Jaki wpływ na projekt może mieć dany interesariusz ?

# Zarządzanie interesariuszami..

## Przykłady interesariuszy:

- zarząd
- kierownictwo działu, którego dotyczy system
- audytorzy bezpieczeństwa
- architekci
- dział zarządzania ryzykiem
- specjaliści ds. rekrutacji
- zarządzający zmianą ( biznesową i/lub IT)
- administratorzy sieci, serwerów, aplikacji, baz danych
- organizatorzy szkoleń
- dział prawny ( sporządzanie/negocjowanie umowy)
- dział kontroli wewnętrznej
- dział pomocy użytkownikom (Help Desk)
- organizatorzy wdrożeń
- specjaliści ds. dokumentacji

Zwłaszcza w dużych organizacjach, korporacjach (banki, koncerny energetyczne, .. ) jest wielu interesariuszy. Np. w banku Santander Bank występują powyżsi interesariusze. Architektów jest 8 w 4 lokalizacjach.

# Zarządzanie personelem

**PODSTAWOWY ZAKRES OBSZARÓW WIEDZY ( KOMPETENCJI) dot. wytwarzania oprogramowania (inżynierii oprogramowania), który powinien posiadać zespół:**

1. Zarządzanie wymaganiami dot. oprogramowania (inżynieria wymagań)
  2. Analiza i projektowanie oprogramowania
  3. Wytwarzanie oprogramowania ( w tym programowanie )
  4. Testowanie oprogramowania
  5. Wdrożenie i eksploatacja ( utrzymywanie ) oprogramowania
- 
6. Zarządzanie zmianami i konfiguracjami
  7. Zarządzanie wytwarzaniem oprogramowania
  8. Procesy w życiu oprogramowania
  9. Metody i narzędzia inżynierii oprogramowania
  10. Jakość oprogramowania

W związku z powyższym w skład zespołu wytwarzającego oprogramowanie muszą wchodzić ludzie posiadający kompetencje dot. wyżej wymienionych zakresów wiedzy.

# Zarządzanie personelem ...

**Wymieniony powyżej zakres obszarów wiedzy to tylko podstawa !**

Nie zwiera on wielu innych obszarów wiedzy, np.

- zarządzania zmianą,
- zarządzania ryzykiem,
- zarządzania kryzysem, konfliktami
- zarządzania wiedzą
- zarządzania personelem

**DOBRY ZESPÓŁ to taki, który może obsłużyć swojego klienta od początku do końca.**

Oznacza to posiadanie niezbędnych umiejętności i doświadczenia w zespole, aby być w stanie określić każdą funkcjonalność, jaką może potrzebować klient zespołu i być w stanie w pełni ją zrealizować.

**Rekrutując ludzi do zespołu należy szukać takich, którzy czują się komfortowo wykonując wiele różnych zadań.** W przypadku programistów oznacza to wyszukiwanie ludzi, którzy mają opanowaną całą technologie ( a nie tylko interfejsy czy bazę danych) oraz potrafią zaprojektować architekturę (strukturę) aplikacji oraz potrafią pisać testy jednostkowe oraz dokonywać refaktoryzacji.

# Samoorganizacja – zarządzanie personelem

## Zasada zwinności – gdzie powstają najlepsze rozwiązania?

Najlepsze architektury, wymagania i projekty ( wzory wykonania) pojawiają się w samoorganizujących się zespołach.

źródło zasad zwinności: <http://agilemanifesto.org>

### Samoorganizacja polega na ustaleniu przez zespół:

- Celu jaki należy osiągnąć ( to potrzebna jest aprobatka klienta)
  - Sposobu w jaki cel zostanie osiągnięty
- a następnie:
- Determinacji w realizacji działań, które zaplanowano jako prowadzące do osiągnięcia celu.

### Zasada zwinności - kogo rekrutować ?

Angażuj do projektów zmotywowanych indywidualistów. Stwórz im dobre środowisko, zaspokaj ich potrzeby i uwierz im, że wykonują swoje zadanie.

Zmotywowani indywidualiści nie mogą być zamknięci – sukces zespołu zależy od tego jak będą się komunikować i jakie relacje (zaufanie !) budować.

# Zarządzanie wiedzą

**Zasada zwinności – efektywne uczenie się, komunikowanie się.**

Pracownicy klienta i twórcy oprogramowania muszą pracować razem każdego dnia w trakcie trwania projektu.

- 1. Zespół twórca uczy się od przedstawicieli klienta co należy zrobić ( poprzez dokładne poznanie wymagań funkcjonalnych i niefunkcjonalnych).**
- 2. Członkowie zespołu uczą się wzajemnie jak spełnić wymagania klienta.**

**Zasada zwinności – dostarczanie informacji**

**Najbardziej wydajnym i efektywnym sposobem: 1. dostarczania informacji do zespołu, 2 wymiany informacji wewnętrz zespołu JEST ROZMOWA.**

Do najbardziej oczywistych sposobów uczenia się należy wykonanie zaplanowanych działań sprawdzania czy nie popełniono błędów i wyciągnięcie wniosków z popełnionych błędów.

Etapy cyklu uczenia się:

1. Wyznaczenie celu
2. Podjęcie działań na rzecz realizacji tego celu
3. Porównanie wyników tych działań z oryginalnymi założeniami
4. Modyfikacja działań na podstawie dokonanych obserwacji i powrót do kroku 2.

# Zarządzanie wiedzą ..

**Jak uczyć się szybciej? To proste - w krótkich cyklach i komunikując się.**

Powinniśmy popełniać (mniejsze) błędy szybciej, a właściwie szybciej je odkrywać. Natychmiastowe wykorzystywanie zdobytej wiedzy i doświadczeń jest kluczem do skojarzenia nauki z dążeniem do właściwego celu.

Dzięki niezwłocznemu stosowaniu tego czego nauczyliśmy w poprzednim cyklu, możemy najefektywniej korzystać ze skutków tej nauki.

**Drugim ( oprócz cykli) wymaganym elementem jest komunikacja.** Potęguje ona efekt uczenia się przez angażowanie w ten proces wszystkich osób.

**Praktyki zwanego wytwarzania oprogramowania koncentrujące się na komunikacji:**

- samoorganizujący się zespół
- zespół pracujący w jednym miejscu
- zespół między funkcjonalny
- programowanie w parach
- radiatory informacji – ogólnie dostępne wykresy, rysunki
- dokumentacja wywołująca dyskusje
- poranne spotkania
- Częste spotkania z przedstawicielem klienta i reprezentacją klienta

# **POZIOMY ZARZĄDZANIA**

## **Zarządzanie operacyjne**

**Zarządzanie operacyjne to nic innego jak codzienne planowanie i realizacja zadań krótkokresowych.**

Charakteryzuje się dużą szczegółowością i krótkim okresem odniesienia  
Zapewnia warunki realizacji założeń strategicznych i umożliwia podejmowanie decyzji wykonawczych na różnych szczeblach organizacyjnych .

## **Zarządzanie taktyczne**

**Koncentrują na ogólnym sposobie realizacji (operacyjnalizacji) działań niezbędnych do osiągnięcia celów strategicznych.**

Dot. m.in. rozwiązywania problemów: doskonalenie przepływu informacji, rozwiązywanie konfliktów, zmian logistycznych.

Ustalenia dot. średniego horyzontu czasowego na średnim poziomie i przez kierowników średniego szczebla.

## **Zarządzanie strategiczne**

Określa główne cele i metody działania w długim horyzoncie czasowym

# ZARZĄDZANIE STRATEGICZNE

**Zarządzanie strategiczne oznacza, zależnie od kondycji organizacji następujące działania lub ich kombinację:**

- Ustalenie długookresowych celów i sposobów ich osiągania
- Zarządzanie alokacją środków na uprzednio właściwie określone cele
- Wskazanie osób decyzyjnych oraz nakreślenie struktury organizacyjnej
- Nakierowanie na analizy, techniki planowania strategicznego, dalszy rozwój oraz innowacyjność mające zagwarantować ustaloną pozycję na rynku
- Określenie ogólnych zasad postępowania i funkcjonowania przedsiębiorstwa
- Stworzenie mechanizmów identyfikacji nowych wyzwań

źródło: <http://ahprofit.pl/zarzadzanie-przedsiebiorstwem/>

# Zarządzanie strategiczne

## Przykłady decyzji strategicznych:

- Uczestnictwo w danym przetargu
- Zgoda na prace zdalne ( np. z mieszkania)
- Rozpoczęcie produkcji oprogramowania określonego rodzaju
- Kupno innej firmy
- Sprzedaż działu/firmy
- Otwarcie działalności do innego miasta
- Decyzja o utworzeniu offShore Center
- Decyzja o rozpoczęciu produkcji w Indiach, Pakistanie, Chinach

# Zarządzanie operacyjne

**Zarządzanie operacyjne** – wg osoby odpowiedzialnej za założenie we Wrocławiu

Capgemini OffShore Center

Zadawanie pytań (odmiana pojęć przez przypadki) dot. osiągnięcia celu projektu i  
znajdowanie na nie odpowiedzi. Ważne również w samoorganizacji.

**Ćwiczenie:**

**Dla każdego z przypadku zadaj pytanie dot. projektu. Odpowiedz na nie.**

1. mianownik – kto?, co?
2. dopełniacz – kogo?, czego?
3. celownik - komu?, czemu?
4. biernik – kogo?, co?
5. narzędzinik – z kim?, z czym?
6. miejscownik – o kim?, o czym?, gdzie?, kiedy?
7. wołacz – o !

# Zarządzanie taktyczne

*Problemy i sposoby ich rozwiązywania zaczerpnięto z książki*

*Amr Elssamadisy - AGILE wzorce wdrażania praktyk zwinnych, wydawnictwo Helion 2010.*

## Problemy BIZNESOWE

**Mają bezpośredni związek z odczuciami klienta odkrywającego, że zakupione oprogramowanie nie spełnia jego oczekiwania.**

Typowe problemy biznesowe

1. Jakość produktu przekazanego klientowi jest nie do przyjęcia
2. Dostarczanie klientowi nowych funkcji trwa zbyt długo
3. **Zaimplementowane funkcje nie są wykorzystywane przez klienta**
4. Oprogramowanie okazało się nieprzydatne dla klienta
5. Budowa oprogramowania jest zbyt droga
6. My kontra oni
7. Klient żąda od nas zbyt dużo

# Zarządzanie taktyczne

Problemy BIZNESOWE cd ...

**Analiza problemu 3 - zaimplementowane funkcje nie są wykorzystywane.**

## Prawdopodobne powody wystąpienia

1. W czasie formułowania wymagań klienci nie wiedzieli czego potrzebują.
2. W role klienta wcielił się dział marketingu wytwórcy.
3. Niektóre funkcje wykorzystywane są dużo rzadziej niż przewidywano.  
    Niezgodność priorytetów wytwórcy i klienta.
4. Programiści dostarczają funkcje, które wg nich są potrzebne.
5. Wymagania są zmieniane

**Rozwiążanie problemu – zwiększenie użyteczności !**

**W jaki sposób ( najbardziej skuteczne praktyki )**

**W tym przypadku - włączenie klienta do zespołu (customer part of team),**

A następnie:

- nadanie przez reprezentanta klienta priorytetów wymagań
- prezentacja wersji demonstracyjnej przedstawicielom klienta
- testowanie funkcjonalne wykonywane przez reprezentanta klienta

# Zarządzanie taktyczne

**Problemy związane z PROCESAMI należy traktować jako symptomy występowania wewnętrznych trudności w ramach zespołu wytwórcy**

**Typowe problemy związane z PROCESAMI WYTWÓRCZYMI oprogramowania**

1. Wiara w bezpośrednie i regularne sugestie klienta jest nieuzasadniona
2. Zarząd jest zaskoczony – nie widzi co dzieje się w projekcie
3. Niewystarczające zasoby – specjalisci są dzieleni między zespołami
4. „Ruchome” tzn przekraczające terminy projekty
5. Setki (lub tysiące ) błędów
6. Potrzeba fazy „hartowania” na końcu cyklu wytwarzania
7. Integracja ma miejsce zbyt rzadko ( ponieważ jest kłopotliwa)

Występowanie problemów dotyczących procesów skłania do ich doskonalenia. Rozwiązywanie tych problemów jest jednak mniej ważne jak podnoszenie walorów biznesowych poprzez rozwiązywanie wymienionych poprzednio problemów biznesowych.

# Zarządzanie taktyczne

**Problemy związane z procesami**

**Klient? Jaki klient ?**

**Wiara w bezpośrednie i regularne sugestie klienta jest nieuzasadniona !**

**Powody - scenariusze**

1. Firma nie ma kontaktu z rzeczywistymi klientami. Dział marketingu pełni role pseudoklienta. Marketingowcy pracują niezależnie od zespołu tworzącego oprogramowanie.
2. Klientami są członkowie zarządu danej firmy. Oni nie mają dużo czasu. Spotykają się tylko chwile. Z tych spotkań sporządzane są notatki.

W obu tych scenariuszach ilość informacji docierających do twórcy od klienta jest niewielka.

Problem związany z procesem być powodem kilku problemów biznesowych:

1. Istnienie nieużywanych funkcji,
2. Nieprzydatność oprogramowania dla klienta,
3. trudności (użyteczność) pracy z produktem.

# Zarządzanie taktyczne

Rozwiążanie problemu związanego z procesami

**Klient? Jaki klient ?**

**!! Wiara w bezpośrednie i regularne sugestie klienta jest nieuzasadniona !!**

Bezpośrednie i regularne, wystarczająco długie spotkania z klientem są niemożliwe.  
W efekcie niemożliwe jest uzyskiwanie od klienta jego opinii i sugestii

Ograniczenie negatywnych skutków stanu rzeczy – zmniejszenie błędów komunikacyjnych.

- Sporządzanie przez klienta (lub przez dział marketingu wytwórcy) dokumentu opisującego wymagania. Zatwierdzanie projektów (ekranów, raportów) (wada: duża pracochłonność !).
- Praca przyrostowa z iteracjami kończącymi się sporządzeniem wersji demonstracyjnej. Prowadzenie listy zaległych zadań ( product backlog).

Aby uniknąć niepotrzebnego zaskakiwania kierownictwa organizacji wytwórcy/zamawiającego powinno się robić 2 rzeczy:

- budować aplikacje przyrostowo, wysyłanie do akceptacji,
- informować kierownictwo o rzeczywistych postępach prac.

# **INŻYNIERIA OPROGRAMOWANIA**

**Dziękuję za uwagę**

**inżynieria oprogramowania**

# **SCRUM / KANBAN**

**Leszek Grocholski**

# SCRUM / KANBAN

Dobrze pracujący zespół SCRUM świadczy o bardzo dobrym stosowaniu praktycznej wiedzy - inżynierii oprogramowania.

Dzisiejszy wykład stanowi w pewnym sensie podsumowanie zdobytej do tej pory w ramach wykładu i ćwiczeń wiedzy.

## Literatura:

- <http://www.poddrzewem.pl>
- <http://www.scrumvival.com>
- <http://www.scrum.org>
  
- Scrum Guide. Przewodnik po Scrumie. Ken Schwaber, Jeff Sutherland, 2017  
-> [źródło niniejszego wykładu o Scrum](#)
- Scrum Shortcuts Without Cutting Corners: Agile Tactics, Tools & Tips. Ilan Goldstein
- Delight Their Customers, And Leave Competitors In the Dust. Ken Schwaber, Jeff Sutherland, Wiley 2012
- Succeeding with Agile: Software Development Using Scrum. Mike Cohn, Addison–Wesley 2009
- The Mythical Man–Month: Essays on Software Engineering. Frederick P. Brooks, Jr., Pearson, 1975–1995
- The New New Product Development Game. Hirotaka Takeuchi, Ikujiro Nonaka, Harvard Business Review, Jan-Feb 1986

# **Plan wykładu**

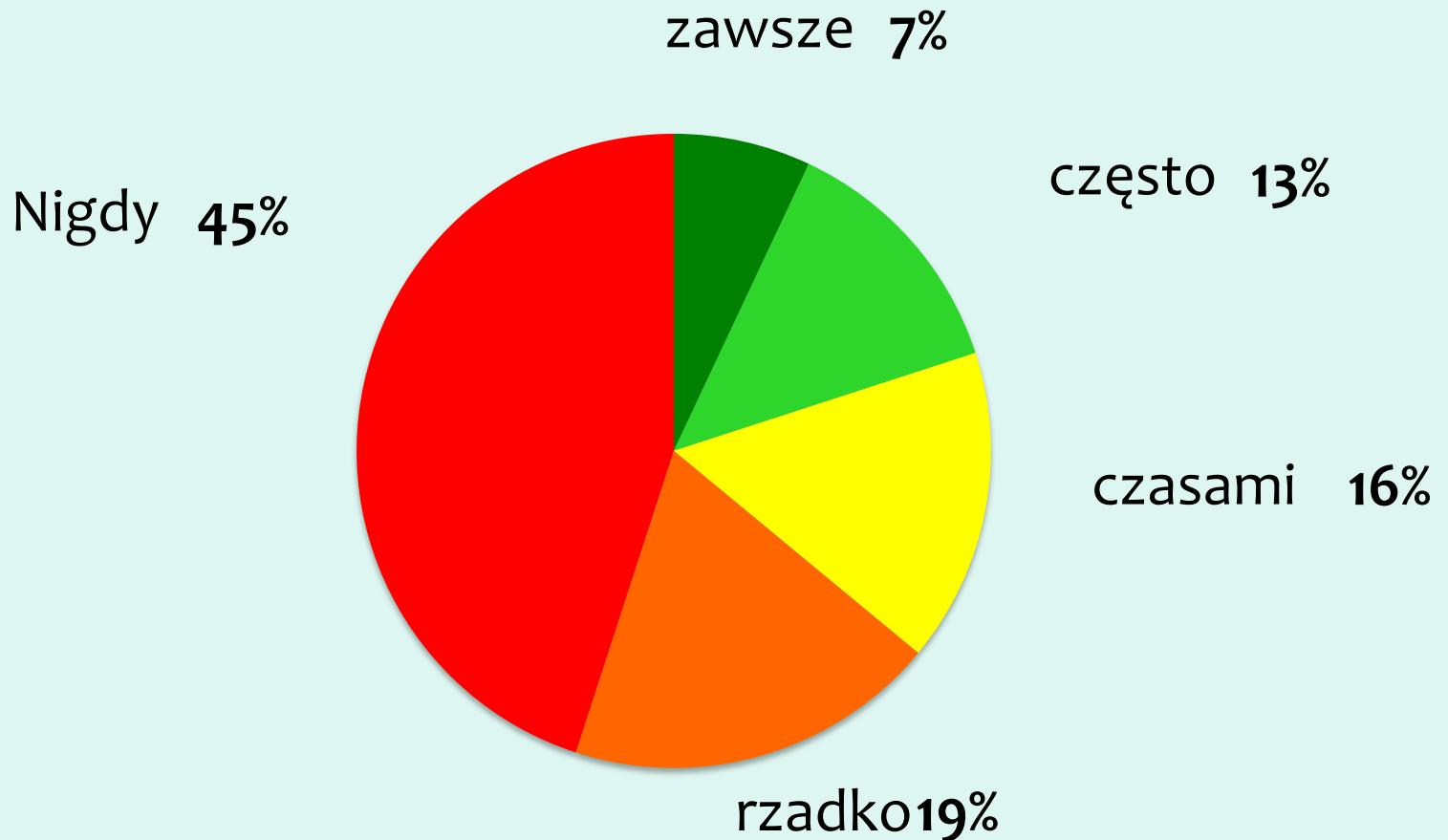
Pochodzenie

Scrum

Kanban

Podsumowanie

# Chaos Report: Feature Use



The Standish Group International, 2004

# Pochodzenie

## Prekursorzy

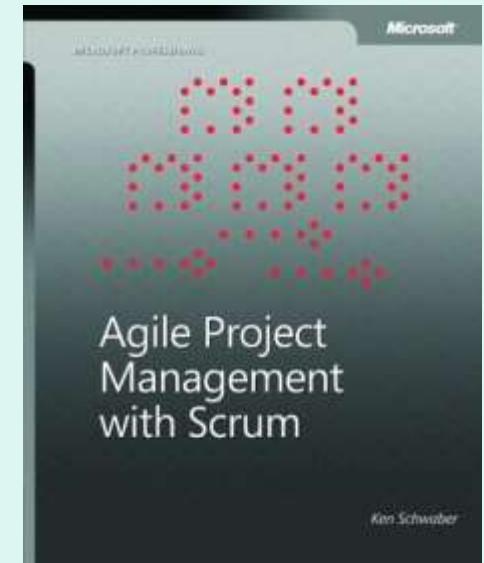
Hirotaka Takeuchi, Ikujiro Nonaka

artykuł w Harvard Business Review

The New Product Development Game (1986)

Nowe podejście do rozwoju produktu komercyjnego: samochodu, kserokopiarki

Scrum – młyn w rugby. Scrum (młyn) wznowia grę.



## Autorzy

- Jeff Sutherland
  - Patient Keeper
  - Initial Scrums at Easel Corp in 1993
  - IDX and nearly 600 people doing Scrum
- Ken Schwaber
  - Advanced Development Methods (ADM)
  - Initial definitions of Scrum at OOPSLA 96 with Jeff Sutherland



## **Uogólniona Definicja Scruma**

„Scrum (rzeczownik): ramy postępowania (ang. framework), dzięki którym ludzie mogą z powodzeniem rozwiązywać złożone problemy adaptacyjne, by w sposób produktywny i kreatywny wytwarzać produkty o najwyższej możliwej wartości.”

**Scrum jest metodyką czyli ustandaryzowanym podejściem do rozwiązywania problemów.**

Scrum jest:

- lekki,
- łatwy do zrozumienia,
- trudny do opanowania.

## **Teoria Scruma**

Scrum został osadzony w teorii empirycznego sterowania procesem, lub — krócej — w teorii empiryzmu. Empiryzm reprezentuje pogląd, iż wiedza wynika z *doświadczania i podejmowania decyzji* w oparciu o to, co poznane. Scrum wykorzystuje podejście iteracyjne i przyrostowe w celu zwiększenia przewidywalności i lepszej kontroli ryzyka.

Każda realizacja empirycznego sterowania procesem opiera się na **trzech filarach: przejrzystości, inspekcji i adaptacji.**

## Wartości Scrum

- **Commitment.** Odpowiedzialne podejmowanie zobowiązania.  
Umiejętność podejmowania zobowiązania rozumiana jako silne poczucie związku z wyrażanymi opiniami i podejmowanymi działaniami oraz konsekwencja w ich realizacji.
- **Focus.** Koncentracja i uważność, bez nich nie jest możliwe ograniczanie się do najważniejszych tematów i dogłębne ich rozumienie. Bez tego przygotowywane rozwiązania charakteryzują się przeciętnością i bylejakością.
- **Openess.** Otwartość (szczerość) wskazywana jako gotowość do dzielenia się prawdziwą informacją ze wszystkimi, bez względu na jej charakter – zgodny lub nie – z oczekiwaniami odbiorcy.
- **Respect.** Poszanowanie oznaczające pozytywne odczucia względem innych i umiejętność budowania synergii przy uwzględnieniu różnic w doświadczeniu, wykształceniu i kulturze oraz cechach charakteru.
- **Courage.** Odwaga (co do wprowadzania zmian) jest elementem niezbędnym do ciągłego doskonalenia się. Scrum domaga się zmiany dotychczasowego porządku, przełamywania zwyczajów i sposobów wykonywania pracy. Odwaga potrzebna jest do postępowania zgodnie z nowym modelem pracy.

# Cykl wytwarzania oprogramowania wg Scrum



## Role

- 1 Product Owner
- 2 Scrum Master
- 3 Członek zespołu Scrum
- 4 Udziałowiec (interesariusz)

## Artefakty

- 1 Product Backlog
- 2 Sprint Backlog
- 3 Product Increment

## Ceremonie

- 1 Sprint Planning
- 2 Daily Scrum
- 3 Sprint Review

Sprint Retrospective

# Role w Scrum

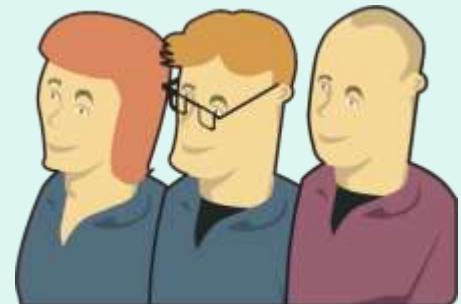
Scrum Master pomaga organizacji i zespołowi wykorzystać Scrum do osiągnięcia celów.



Product Owner odpowiedzialny za maksymalizację wartości produktu i pracy Zespołu.



Scrum Team tworzy oprogramowanie i organizuje sobie pracę w ramach Scrum.



# Scrum Master



- **Odpowiedzialny za:**
  - Sukces Scrum
  - Wprowadzenie i postępowanie zgodne z praktykami Scrum opisanymi w przewodniku
  - Monitorowanie prac
  - Usuwanie przeszkód
  - Organizacje prac, która zachęca do pracy zespołowej, samoorganizacji i odpowiedzialności

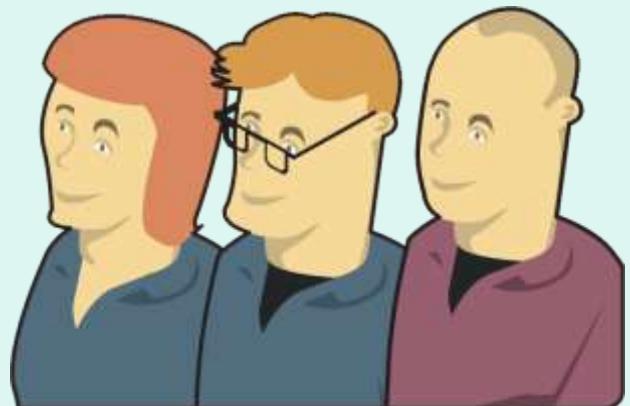
# Product Owner



- **Ustala harmonogram produkcji poprzez priorytetowe listy zadań do wykonania (product backlog ).**
- **Jest jedną osobą która zapewnia, że tylko jeden zestaw zadań jest realizowany w danym Sprincie.**
- **Eliminuje zamieszanie wywoływanie przez wielu szefów, różnych opinie i inne zakłócenia.**
- **Jest jedna osobą, która w wyniku kontaktów z interesariuszami klienta i producenta, określa priorytety zadań.**
- **Stanowi podstawową siłą napędową realizacji wizji produktu i zapewnienia osiągnięcia celu.**

# Scrum Team

- **6 +/- 3 osoby**
- **Cel działania: przyrostowe wytwarzanie oprogramowania**
- **Samoorganizująca się grupa**
- **Brak specjalizacji ( wszyscy są twórcami - developerami )**
- **Możliwość wszechstronnego rozwoju każdej osoby**
- **(Samo) Zobowiązani do wykonywania zadań**
- **Mają prawo ( i władzę), robić wszystko co potrzeba aby wykonać zadania**

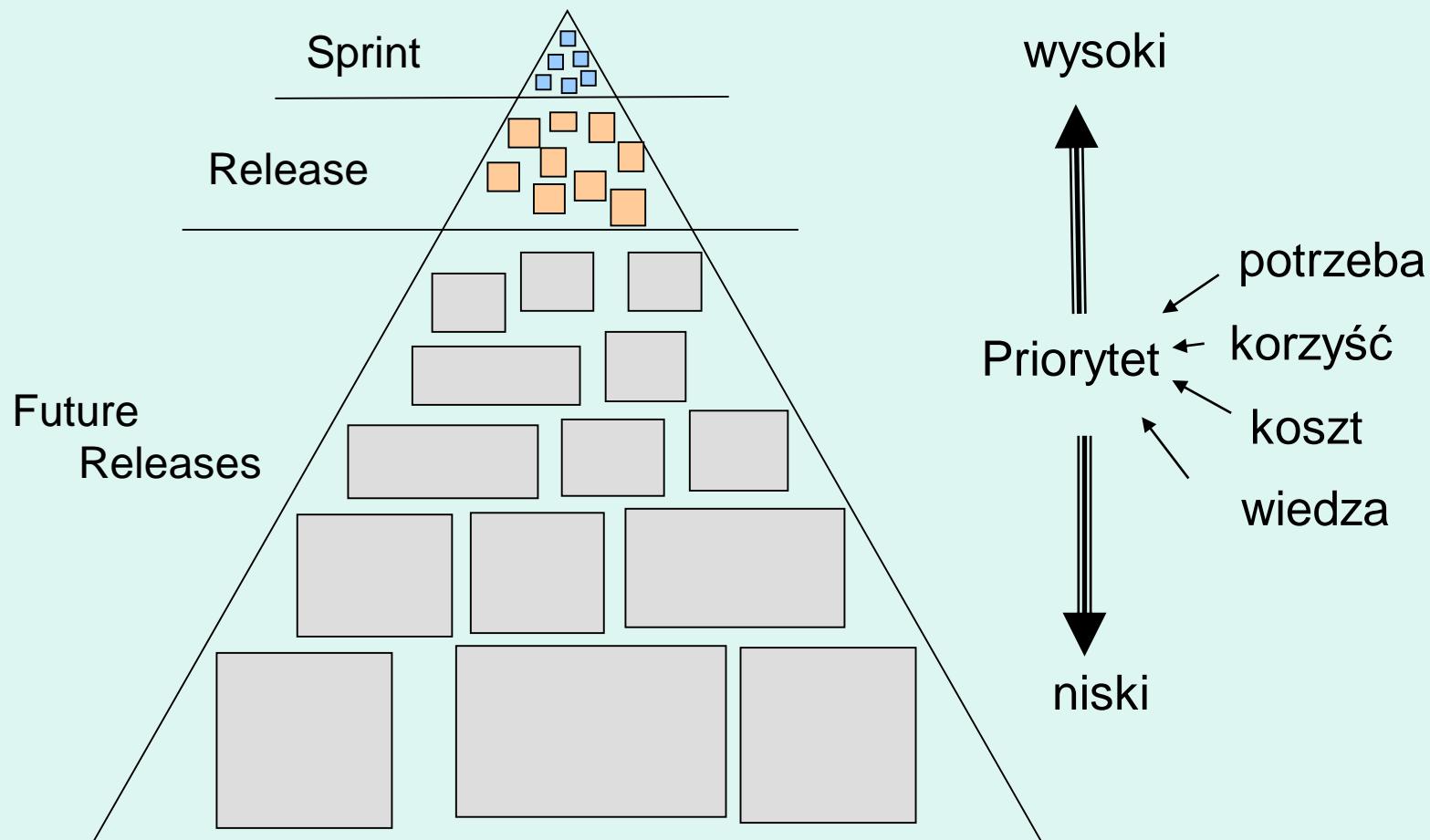


# Product Backlog

- **Lista wszystkich wymaganych zadań (prac) dotyczących produktu – np:**
  - dostarczenie funkcjonalności (“let user search and replace”)
  - modyfikowanie funkcjonalności
  - wykonanie zadania (“upgrade to Oracle 11”)
  - wykonanie testów i usuwanie błędów
- **Lista jest porządkowana przez Product Ownera wg zadań/prac najbardziej potrzebnych w danej chwili**
- **Pozycje listy ciągle ulegają zmianie ( dodawanie, usuwanie, modyfikowanie )**
  - Pozycje o wyższej pozycji są bardziej szczegółowo opisane
- **Jedna lista dla wielu zespołów**

# Góra lodowa Product Backlog

## (The Product Backlog Iceberg )



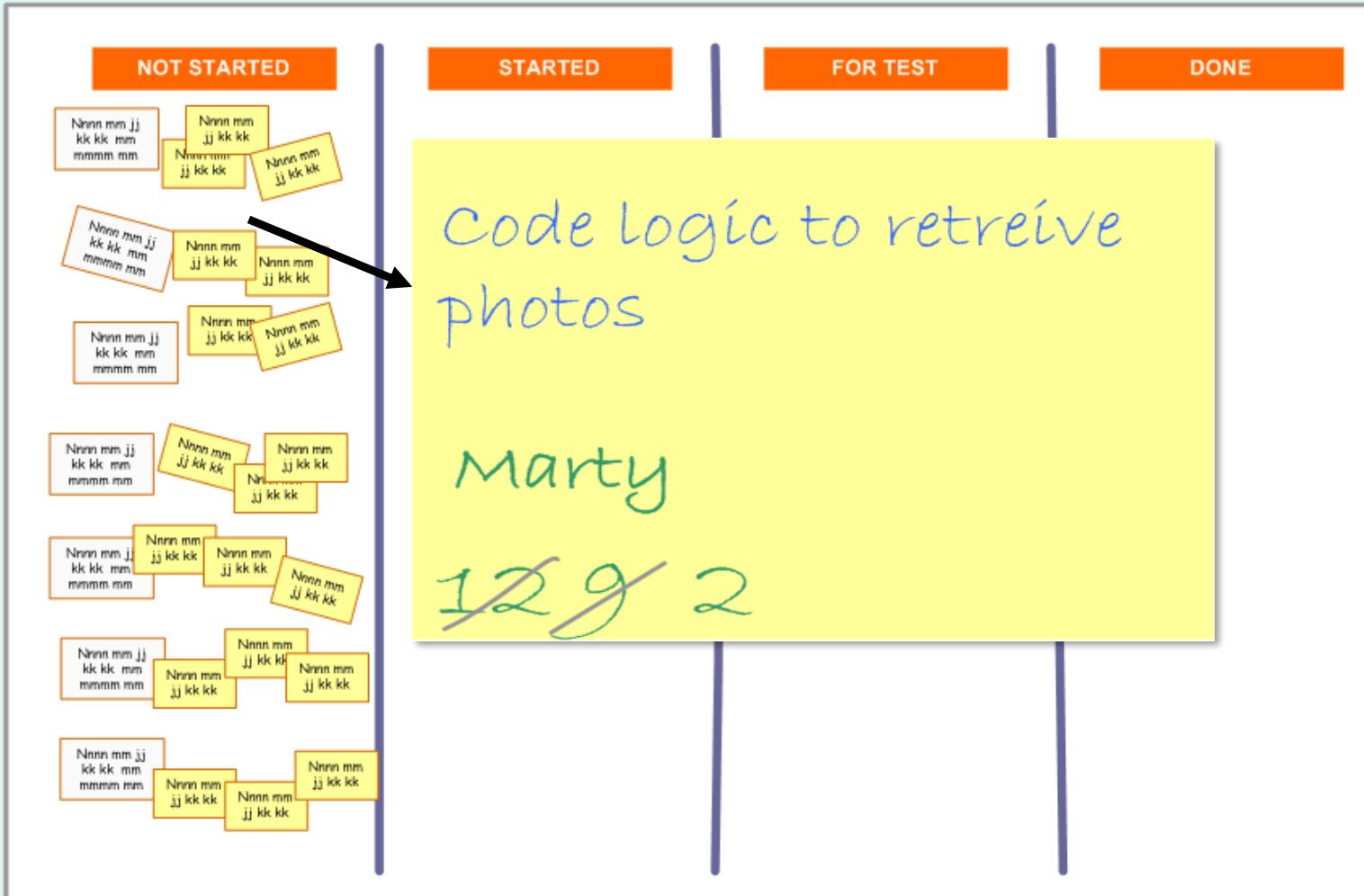
# Pozycje Sprint Backlog

## Sprint Backlog Items (= TASKS)

- **Zadania, które umożliwiają transformacje pozycji Backlog w produkt (kod, testy, dokumentacja, zmiany konfiguracji, itd. ...)**
- **Pracochłonność 4-16 godzin. Jeżeli większa to podział zadań na mniejsze pracochłonne.**
- **Sprint Backlog jest tworzony w zespole i jego właścicielem jest Scrum Team.**
- **Lista zadań pozostałych do wykonania jest codziennie aktualniana.**
- **Tylko zespół może dodać elementy do Sprint Backlog.**
- **Członkowie zespołu przypisują się do zadań zaraz po rozpoczęciu sprintu.**

# Tablica zadań

## (Task board)



# WYKRES WYPALANIA

## (Sprint Burndown Chart )



# Planowanie Sprintu (Sprint Planning)

Uczestnicy:



Cel:

OKREŚLENIE CO i W JAKI SPOSÓB

- przydzielenie pozycji z product backlog i zdefiniowanie celu sprintu
- utworzenie sprint backlog
- osiągnięcie porozumienia co do celu sprintu i przydziału zadań

Przebieg:

- Product Owner prezentuje pozycje z produkt Backlog do wykonania
- Product Owner i Scrum team definiują cel sprintu
- Produkt Owner i Scrum Team:
  - //- zbierają informacje niezbędne do wykonania zadań
    - > Definition of Ready
  - //- określają kryteria (testy) uznania zadania za zakończone
    - > Definition of Done
  - //- określają pracochłonność zadań sprint backlog
  - //- określają harmonogram prac ( ludzie, zdania, czas )
- Członkowie zespołu wzajemnie zobowiązują się wykonać zadania

# CODZIENNY SCRUM

## (Daily Scrum)

Uczestnicy:



Cel:

- synchronizacja pracy zespołu
- przy okazji zapewnienie przejrzystości

Przebieg:

Czas trwania, maks. 15 minut, na stojąco!

Każdy członek zespołu odpowiada na następujące 3 pytania

- które zadania skończył od ostatniego spotkania ?
- które zadania planuje skończyć przed następnym spotkaniem?
- które zadanie aktualnie wykonuje
- oraz zgłasza ew. problemy

Uaktualnienie tablicy zdań, wykresu wypalania

Identyfikacja problemów

**Kluczowym wynikiem jest synchronizacja.**

**Określenie stanu projektu jest uzyskiwane przy okazji.**

# Przegląd Sprintu (Sprint Review)

Uczestnicy:

udziałowcy +



Cel:

przegląd wszystkich zadań wykonanych w danym sprintie  
sprawdzenie czy cel sprintu został osiągnięty

Przebieg:

Zespół prezentuje wyniki pracy  
uwaga: przedstawiane jest tylko działające oprogramowanie

Zespół demonstruje nowe/zmodyfikowane funkcjonalności  
Product owner + reprezentanci klienta oceniają wyniki

Czasami reprezentanci klienta wykonują testy funkcjonalne.

# Ocena Sprintu (Sprint Retrospective)

Uczestnicy:



Cel:

Umożliwienie oceny co było dobre a co złe w sprincie  
Uczenie się na podstawie doświadczenia  
Osiągnięcie porozumienia zespołu co do priorytetu i kolejnych działań naprawczych

Przebieg:

Omówienie tego jak postępowano  
Znalezienie odpowiedź na 3 pytania  
- jakie praktyki powinniśmy **kontynuować**?  
- co musimy **zaprzestać** robić ?  
- co powinniśmy **zacząć** robić?  
Zbudowanie planu działań naprawczych w przyszłym sprincie

# Scrum



**Łatwy do zrozumienia  
ale  
trudny do opanowania ...**

# Co jest trudne w SCRUMie?

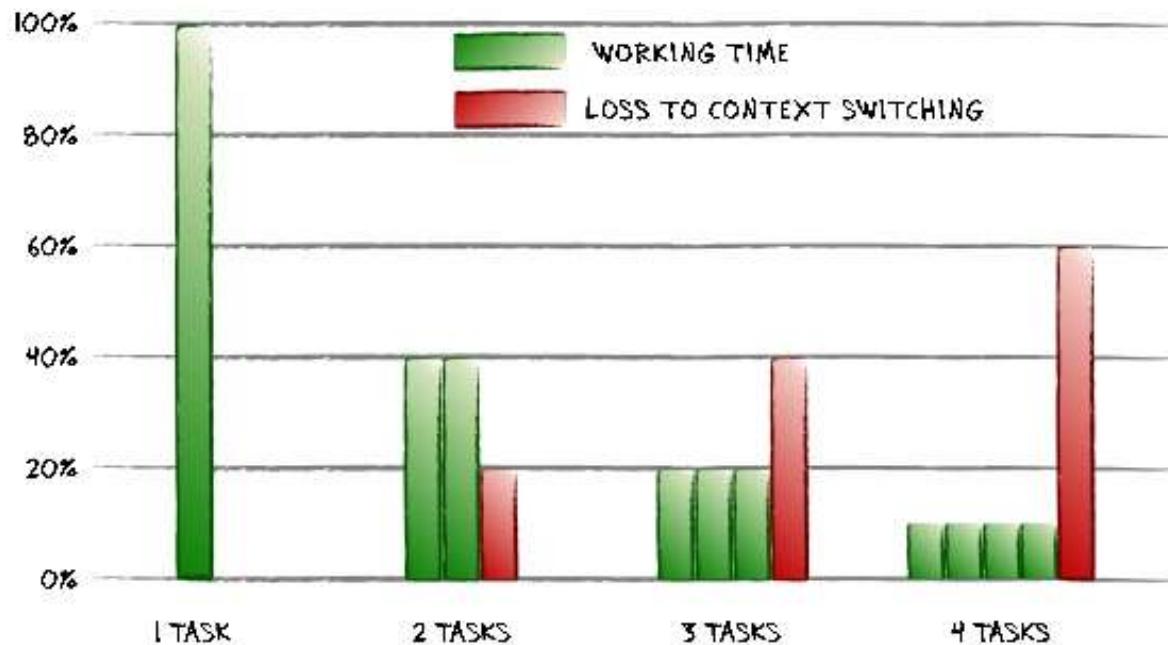
- Zapewnienie ciągłej dostępności przedstawicieli klienta
- Zbudowanie samo-organizującego się, składającego się z członków o wszechstronnych umiejętnościach zespołu.  
→ Scrum nie da się wdrożyć, jeżeli nie ma woli pracy zespołowej.
- Zapewnienie udziału naprawdę zaangażowanego Product Ownera  
→ Scrum nie da się wdrożyć jeżeli klient/Product Owner nie rozumie specyfiki pracy w Scrum i/lub nie zapewni odpowiednich zasobów w czasie przedsięwzięcia
- Wiarygodne szacowanie pracochłonności zadań i ponownych oszacowań.
- Synchronizacja pracy zespołów międzynarodowych  
→ Trzeba zapewnić ten sam sposób podejścia do pracy, i stosowanie tych samych narzędzi.

# (dowcipna ?) zagadka

- Co programistom najbardziej przeszkadza w pracy?

# Koszt wielozadaniowości

## Cost of multitasking



Source: Gerald Weinberg, Quality Software Management: Vol. 1 System Thinking

**lunar logic**

看 kanbanery



# 看板 – Kanban Background

- 看板 – W pierwotnym znaczeniu  
– w języku japońskim – oznacza:  
„szyld”, „tabliczkę z napisem  
informującym”, „billboard”.
- Charakterystyczną cechą tej metody jest  
praktyczna likwidacja magazynów  
przedprodukcyjnych (cały zapas znajduje  
się na stanowisku roboczym),  
międzyoperacyjnych i wyrobów gotowych.
- **Metoda KANBAN opiera się na  
poszczególnych kartach wyrobów, ich  
cyrkulacji i analizie.**
- Kanban jest przejrzystym, ograniczającym  
prace i zapasy, zaciągającym prace  
system organizacji pracy.



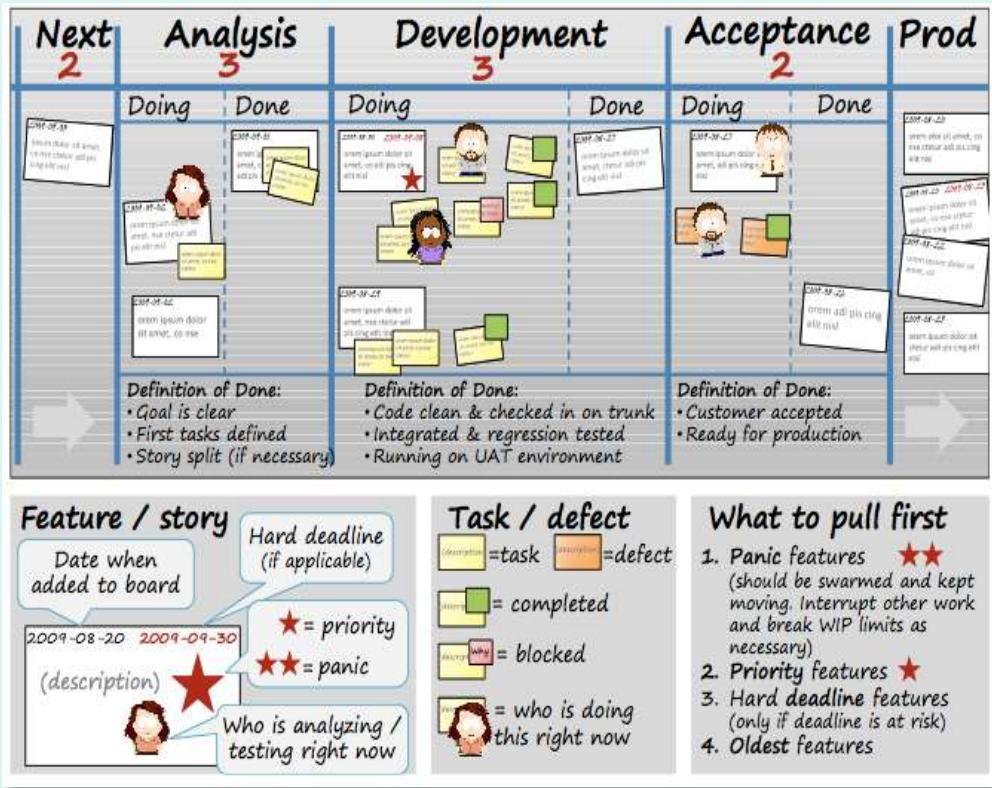
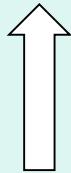
# Kanban w pigułce

„głębokie  
wdrożenie

## Podstawowe praktyki Kanban :

- wizualizacja pracy
- ograniczanie Work in Progress (WIP)  
tzn pracy w toku
- śledzenie przepływu pracy
- oczywista, prosta organizacja pracy
- stymulacja pracy zespołowej

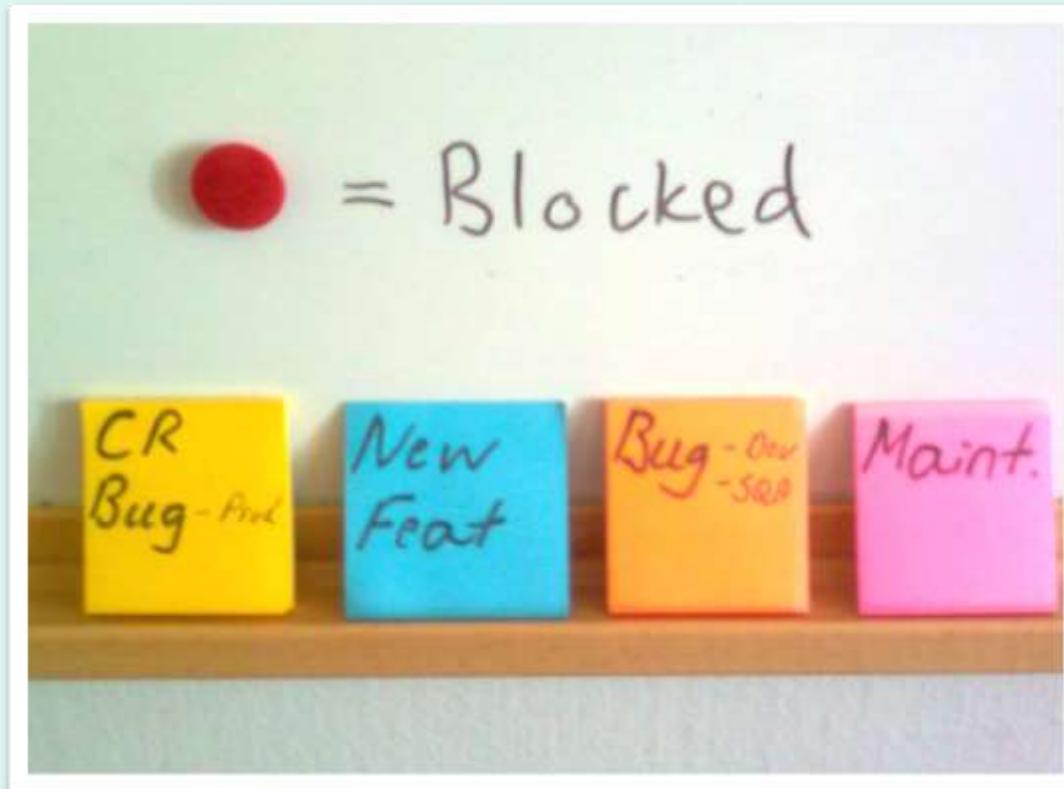
„głębokie  
wdrożenie



5 podstawowych praktyk zaobserwowanych  
w udanych wdrożeniach Kanban

# Karty

Różne nazwy: karty, żetony, żelki, bilety, tickety



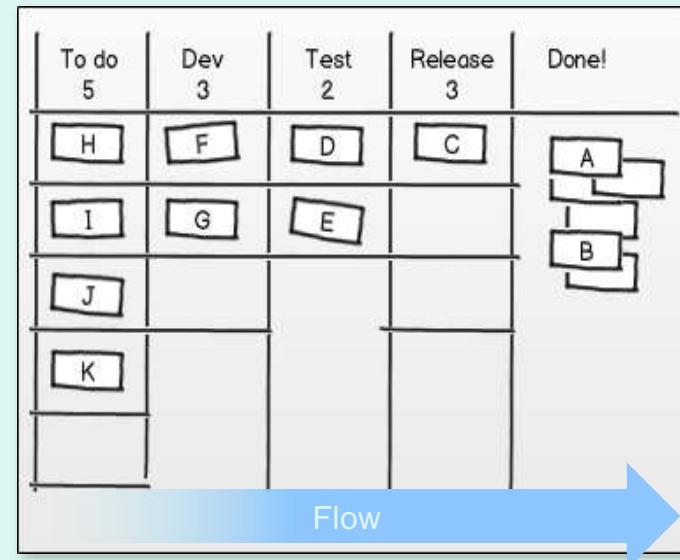
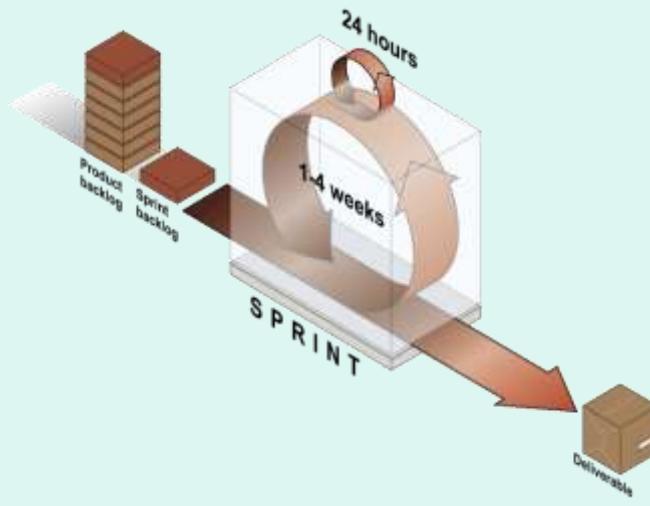
# Przykład tablicy Kanban - White Board



# Scrum vs. Kanban

## podobieństwa

- Praca jest zaciągana (pull) przez osoby.
- Ogranicza prace w toku (WIP = work in progress).
- Wizualizuje planowaną, wykonywaną, wykonaną pracę.
- Stymuluje doskonalenie procesu wytwarzania.
- Zorientowany na dostarczanie wydań oprogramowania szybko i często.
- Wymaga podziału prac na „małe” zadania.



# Scrum vs. Kanban

## różnice

Scrum	Kanban
<b>Praca zorganizowana w iteracje (sprinty).</b>	<b>Interakcja nie jest w standardzie.</b>
<b>Wymaga porozumienie zespołu co do wspólnego wykonania w iteracji określonych prac.</b>	<b>Zespół niezależnych pracowników. Porozumienie zespołu co do wspólnego wykonywania prac nie jest w standardzie.</b>
Stosuje <b>pracochłonność</b> jako domyślną metrykę planowania oraz doskonałości procesu.	Stosuje <b>faktyczny czas realizacji</b> jako domyślną metrykę planowania i doskonalenia procesu
<b>Zalecane wszechstronne</b> zespoły.	Wszechstronne zespoły mogą wystąpić. <b>Standard mówi o zespołach specjalistów indywidualistów.</b>
<b>Nakazuje 3 role</b> (PO, SM, Development Team).	<b>Żadne role nie są nakazane.</b>
<b>Nakazuje oszacowania pracochłonności.</b>	<b>Oszacowania pracochłonności mogą wystąpić.</b>
<b>Sprint backlog is własnością jednego zespołu</b>	<b>Tablica Kanban może być wspólna</b> dla wielu zespołów indywidualistów

# Scrum i Kanban są tylko ramami

**Wytwarzając KAŻDE oprogramowanie, też zgodnie z Scrum czy Kanban  
TRZEBA - NIE MA WYJŚCIA ! przejść przez następujące etapy:**

- zbudowanie wizji,
- analiza,
- projektowanie,
- kodowanie,
- testowanie,
- wdrożenie.

**Na pytanie w jaki sposób powyższe etapy realizować, Scrum i Kanban jedynie podpowiadają:**

- **w sposób zwinny, tzn elastycznie i skutecznie, wykonując jedynie niezbędne prace zapewniające wytworzenie potrzebnego klientowi oprogramowania.**

Istnieje wiele ZWINNYCH PRAKTYK dot. wytwarzania oprogramowania.

Na ostatnich slajdach omówiono:

- zwinne szacowanie pracochłonności,
- zwinne zbudowanie wizji.

Uwaga: w/w PRAKTYKI nie są składowymi standardów Scrum, Kanban.

# **SZACOWANIE PRACOCHŁONNOŚCI**

**W zwinnym wytwarzaniu oprogramowania zauważa się, że niemożliwe jest precyzyjne ( w godzinach, osobodniach ) oszacowanie pracochłonności.**

„Szacowanie na wysokim poziomie to zgadywanki  
(i to zazwyczaj bardzo nieudane, a przy tym zbyt optymistyczne )”.

Bywa, że dokładne oszacowanie jest niemożliwe i musimy przestać udawać, że jest inaczej.

Jedynym pytaniem na które początkowe oszacowania mogą próbować odpowiedzieć na pytania, jest:      **Czy ten projekt DA SIĘ W OGÓLE WYKONAĆ !?**  
                                 **(w takim czasie i takimi środkami, jakimi dysponujemy)**

Ale to do zbudowania planu wytwarzania szacowanie pracochłonności (w tym też planu współpracy z klientem) jest bardzo ważne.

**To czego potrzeba w zwinnym wytwarzaniu to sposób szacowania, który:**

- umożliwia planowanie przyszłości;
- przypomina nam, że nasze szacunki to tylko zgadywanie;
- bierze pod uwagę wewnętrzną złożoność tworzenia oprogramowania.

# **SZACOWANIE PRACOCHŁONNOŚCI ...**

**!! 1 dzień przeliczeniowy <> 1 dzień kalendarzowy !!**

Badania psychologiczne) pokazały, że ludzie są dobrzy w szacowaniu względnym. Grupowe szacowanie względne prowadzi do lepszych wyników.

Szacowanie zwinne potrzebuje dwóch rzeczy:

- wzorcowych zadań, których rozmiary można porównywać z innymi zadaniami
- systemu punktowego szacowania pracochłonności i śledzenia postępów.

W metodach zwinnych zaleca się zamknięcie oszacowań w prostym, łatwym do wykorzystania systemie punktowym i unikanie wiązania ich z czasem kalendarзовym.

Skala punktowa 1, 3, 5 punktów. Nie więcej. Duże zadania dzielimy na mniejsze.

**Planistyczny poker** to gra, w której zespół programistyczny szacuje najpierw zadania indywidualnie (1,3, 5 pkt) a następnie wspólnie porównuje wyniki.

**Rzeczywista szybkości pracy osoby i zespołu mierzmy w punktach.**  
**Te dane używamy do planowania przyszłości.**

# WIZJA SYSTEMU

**Dzięki wizji członkowie zespołu rozumieją kto, co i dlaczego potrzebuje.**

Dlatego mogą myśleć samodzielnie:

- podejmować lepsze, bardziej świadome decyzje,
- w lepszy sposób godzić przeciwnieństwa i wypracowywać kompromisy,

**I w efekcie tworzyć lepsze, bardziej innowacyjne rozwiązania !**

**Tablica koncepcyjna – krótkie, najważniejsze informacje o przedsięwzięciu.**

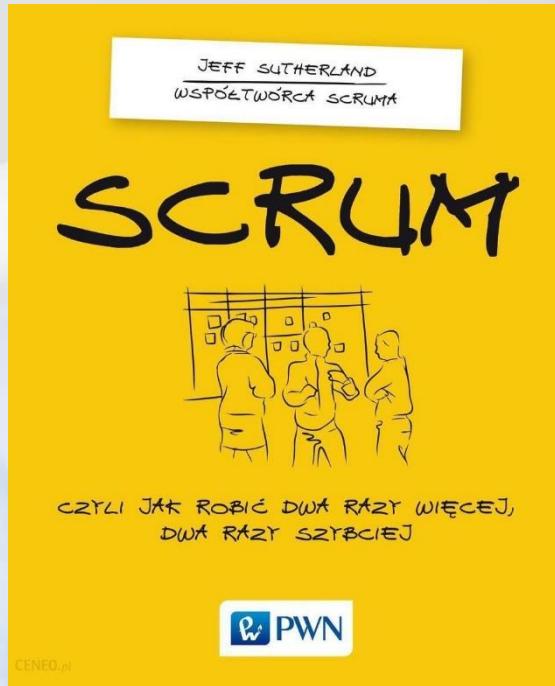
Tablica zbudowana po to aby zespół i interesariusze wspólnie wiedzieli:

1. Co zespół buduje i po co?
2. Co jest ważne w projekcie ( jakich korzyści spodziewa się klient, dlaczego kupi)?
3. Co jest w zakresie przedsięwzięcia a co poza nim ?
4. Jakie są wymagane najważniejsze funkcjonalności ?
5. Kto jest w sąsiedztwie ( interesariusze – osoby u klienta zainteresowane wynikiem) ?
6. Jak wygląda rozwiązanie od strony technicznej ( architektura) ?
7. Czego należy się obawiać ( ryzyka) ?
8. Jak duży jest projekt ( zwymiarowanie wszystkiego) ?
9. Kiedy i gdzie trzeba będzie się elastycznie dostosować ( co trzeba odrzucić)?
10. Ile w przybliżeniu to pochłonie ( pracowników, czasu i pieniędzy ) ?

# Survey & Questions?



# SCRUM jako filozofia życia i pracy, czyli robienia wszystkiego



Na podstawie książki Jeffa Sutherlanda pt.  
„Scrum, czyli jak robić dwa razy więcej, dwa  
razy szybciej”

Prezentacja „soft” dla informatyków i nie tylko

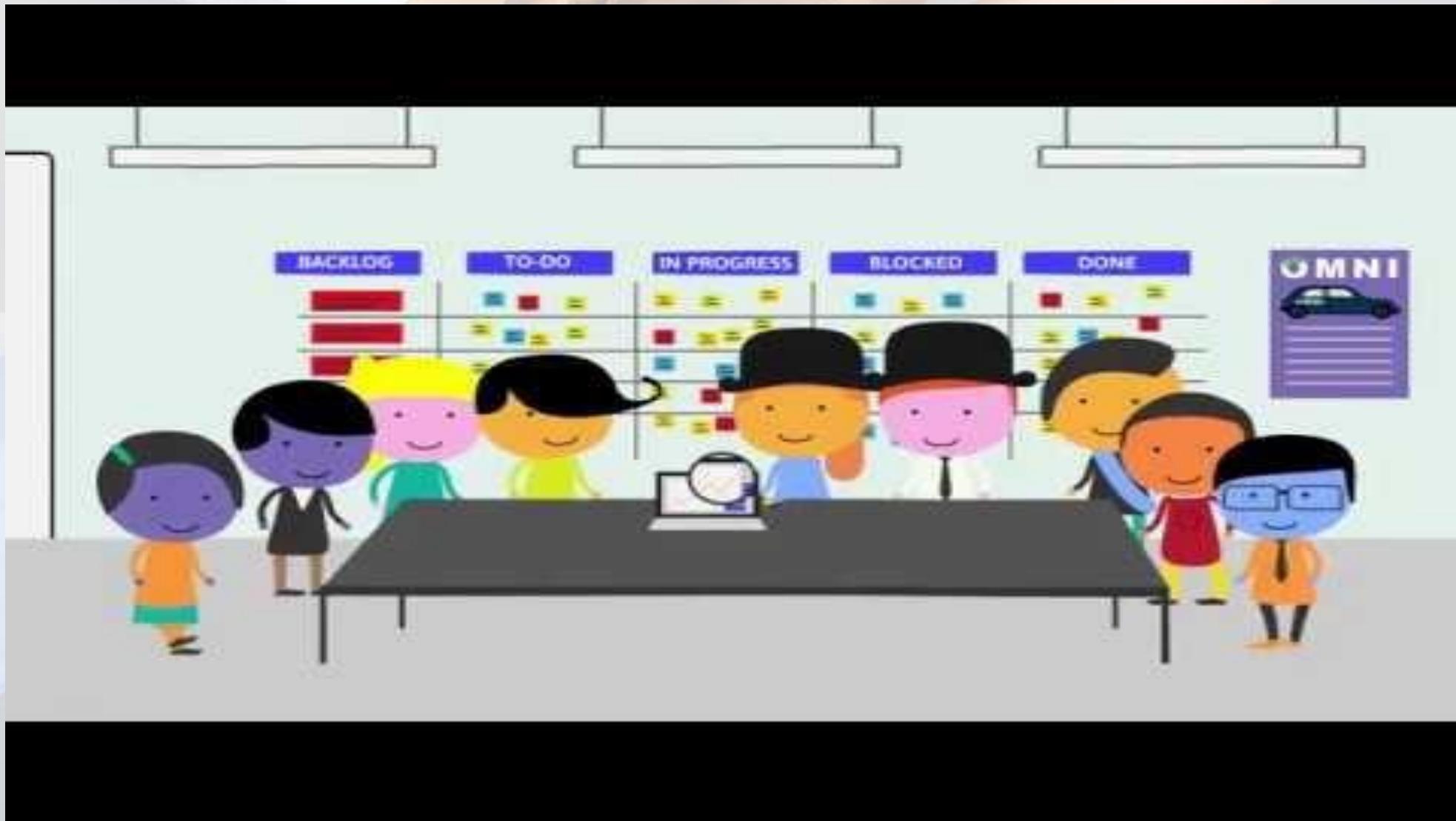
# Czym właściwie jest ten Scrum?

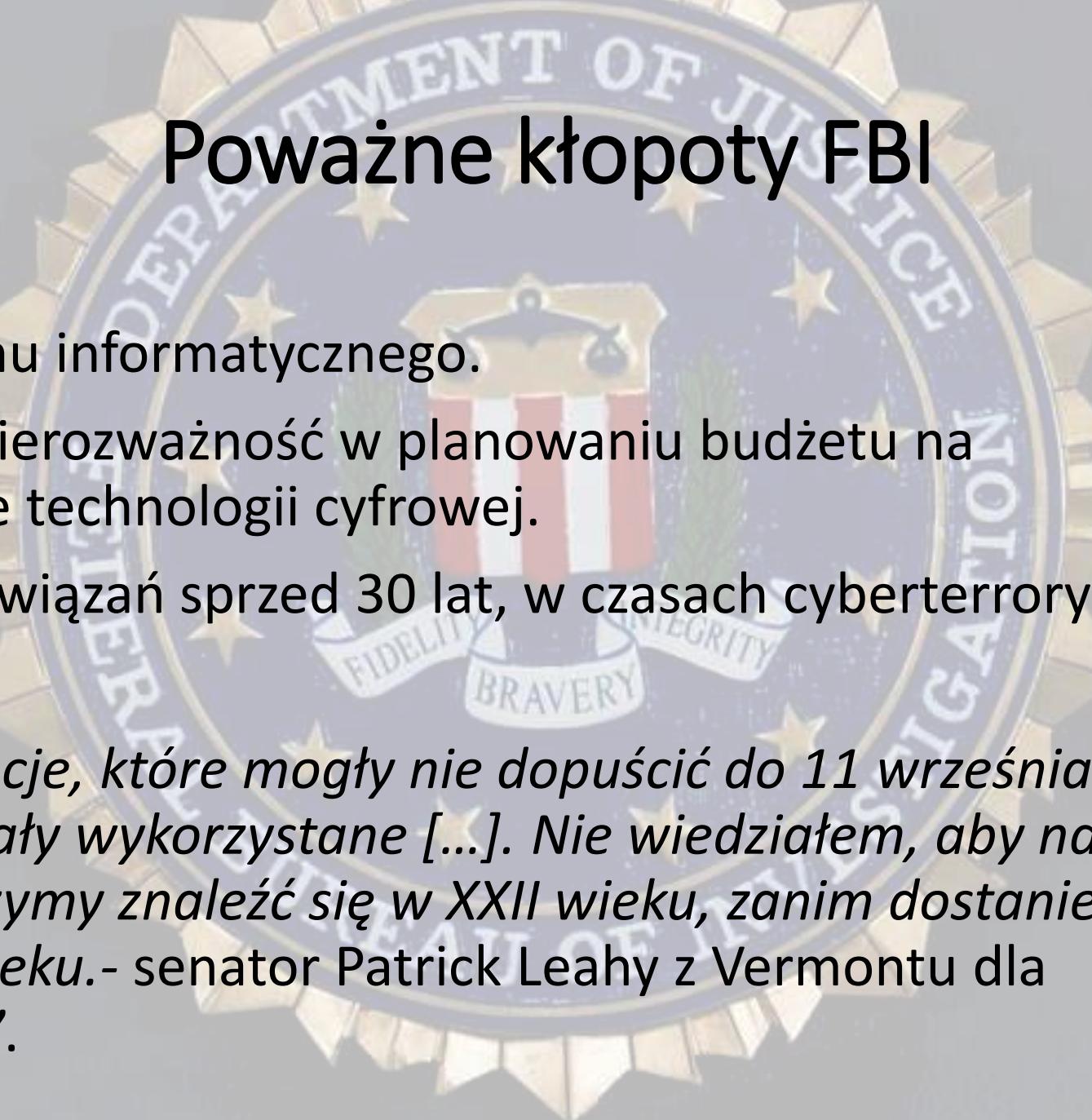
- Metoda opracowana prawie 30 lat temu (1993r.).
- Szybszy, bardziej niezawodny i efektywny sposób wytwarzania oprogramowania w branży technicznej.
- Wcześniej (nawet do 2005 roku) większość projektów programistycznych była tworzona metodą kaskadową (Waterfall), w której projekt realizowany był w odrębnych etapach i posuwał się krok po kroku w kierunku ostatecznej wersji.
- Metoda Scrum zrewolucjonizowała pracę w wielu projektach/branżach, od firm informatycznych, po zespoły lotnictwa bojowego w USA.

# Założenia Scrumu

- Oparte na iteracyjnych cyklach zwanych sprintami, trwającymi od tygodnia do maksymalnie miesiąca, z czego każdy z nich jest skupiony na jakimś założonym celu/funkcjonalności.
- Istnieje kluczowy podział na role Scrum Master, Product Owner, Development Team (3-9 osób) [wyjaśnienie ról w materiale video].
- Przed każdym sprintem przygotowuje się Sprint Backlog, czyli rejestr zadań wraz z pracochłonnością/ryzykiem/czasochłonnością każdego z nich.
- Codziennie odbywają się maksymalnie 15-minutowe Daily Scrumy, czyli spotkania, mające na celu podsumowania zadań i ewentualne uwagi, pomoce.

# Materiał przybliżający odrobinę metodę Scrum





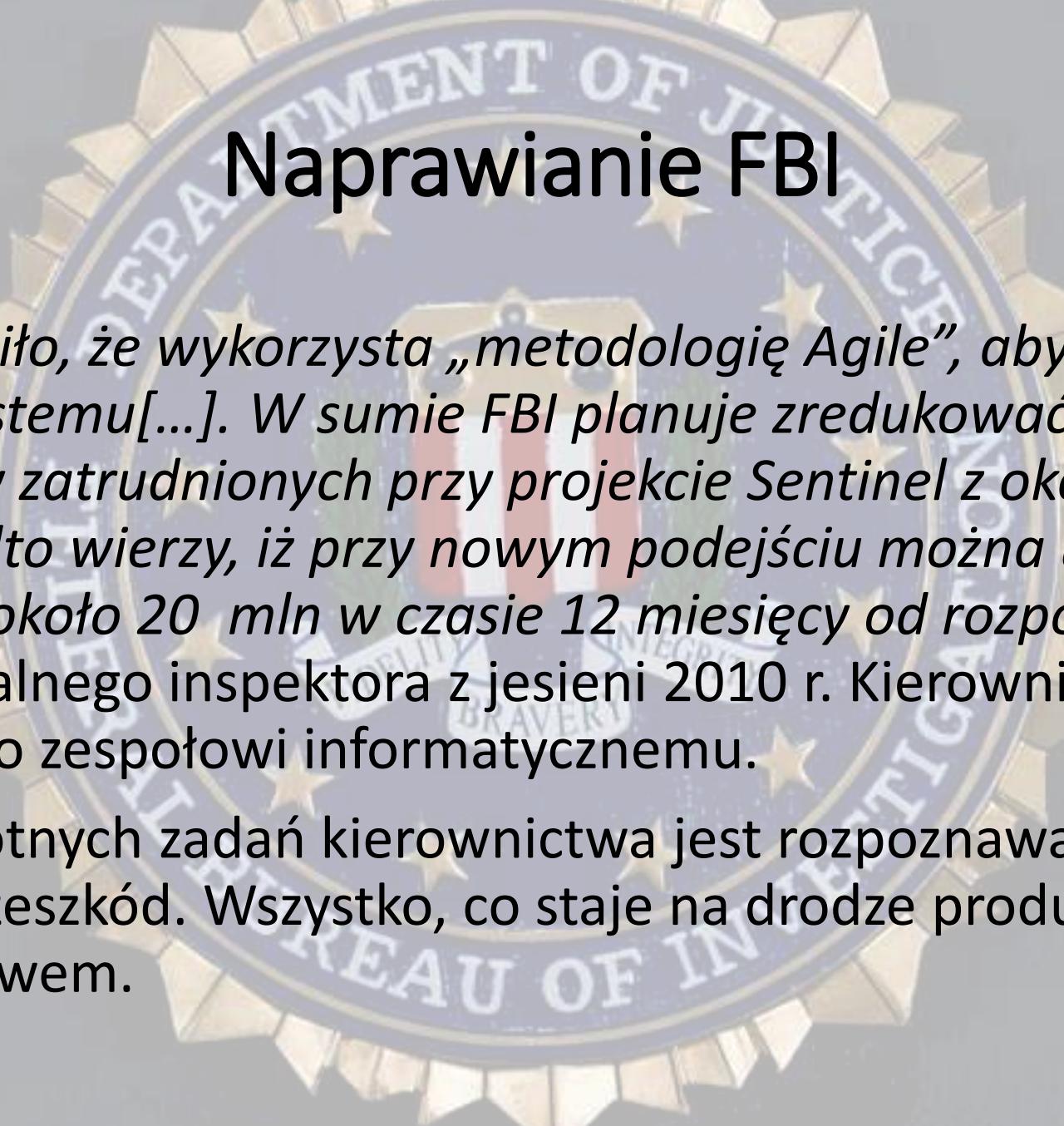
# Poważne kłopoty FBI

- Zacofanie systemu informatycznego.
- Chaotyczność i nierozważność w planowaniu budżetu na unowocześnienie technologii cyfrowej.
- Korzystanie z rozwiązań sprzed 30 lat, w czasach cyberterroryzmu.

*„Mieliśmy informacje, które mogły nie dopuścić do 11 września. One tam były i nie zostały wykorzystane [...]. Nie wiedziałem, aby naprawiali problemy [...]. Zdążyliśmy znaleźć się w XXII wieku, zanim dostaniemy technologię XXI wieku.- senator Patrick Leahy z Vermontu dla „Washington Post”.*

# Nowy sposób myślenia

- Oparta na analizowaniu tego jak ludzie **pracują naprawdę**, a nie tego **co mówią** o swojej pracy.
- Scrum (młyn z rugby) – staranne ustawienie w linii, jasny cel i zjednoczenie w dążeniu do niego.
- Cykl „sprawdzania i dostosowywania” (Inspect and Adapt) – na samym początku projektu sprawdzamy regularnie, czy działamy właściwym kierunku i czy robimy to czego ludzie naprawdę chcą.



# Naprawianie FBI

- „*FBI stwierdziło, że wykorzysta „metodologię Agile”, aby dokończyć tworzenie systemu[...]. W sumie FBI planuje zredukować liczbę pracowników zatrudnionych przy projekcie Sentinel z około 220 do 40.[...] Ponadto wierzy, iż przy nowym podejściu można ukończyć Sentinela za około 20 mln w czasie 12 miesięcy od rozpoczęcia.*” - raport generalnego inspektora z jesieni 2010 r. Kierownictwo niedowierzało zespołowi informatycznemu.
- Jednym z istotnych zadań kierownictwa jest rozpoznawanie i usuwanie przeszkód. Wszystko, co staje na drodze produkcji, jest marnotrawstwem.

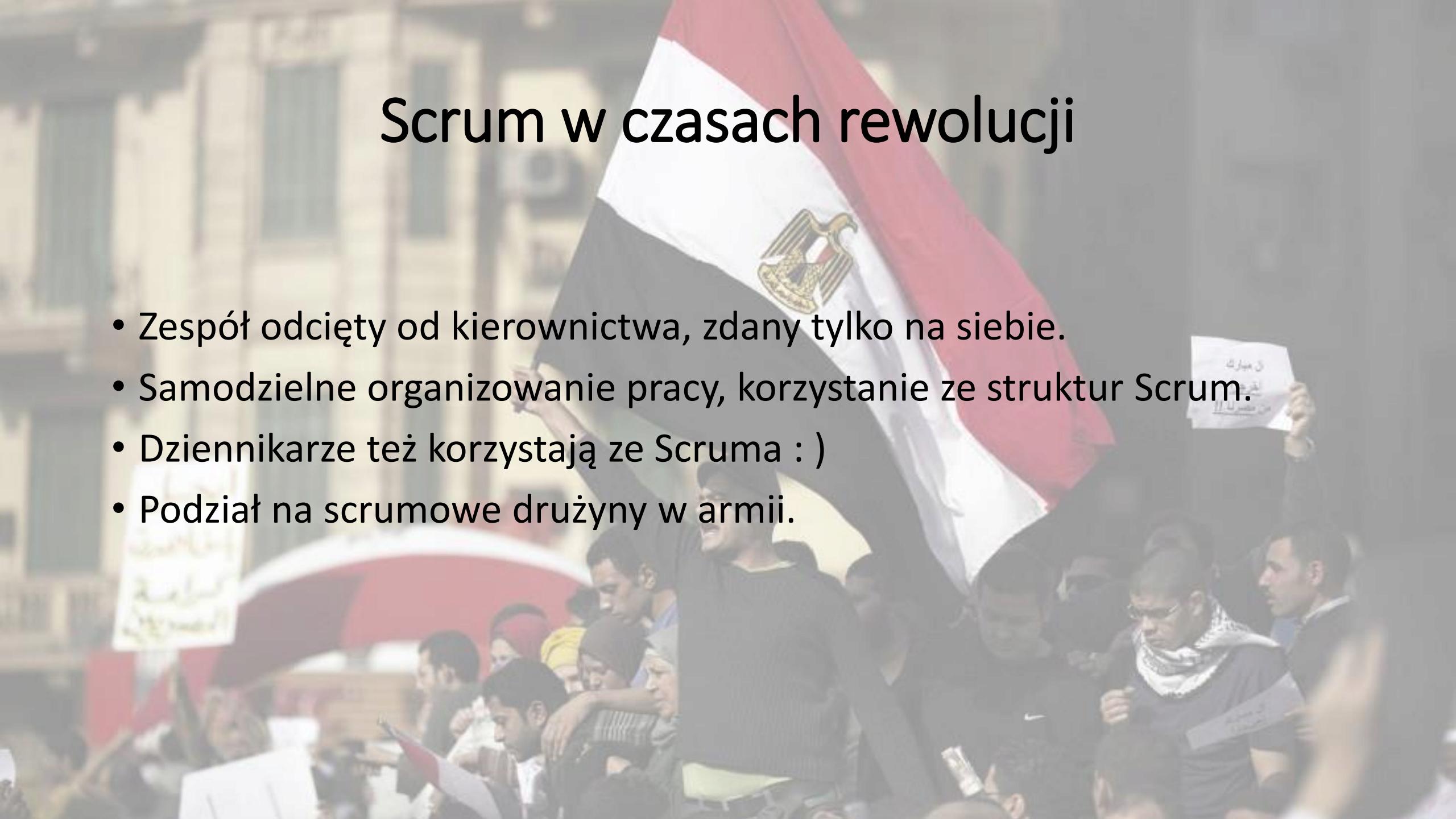
# Przestań podążać za metodą kaskadową

- Obserwuj, nie trać orientacji, decyduj i działaj.
- Hirotaka Takeuchi i Ikujiro Nonaki – japońscy profesorowie, wpadli na podobną metodę w Kraju Kwitnącej Wiśni.
- 1995 rok – opublikowanie artykułu „SCRUM Development Process”, autorzy Ken Schwabery i Jeff Sutherland
- „*Zmieńcie się albo zginiecie*” – firmy, które nie chcą się dostosować do nowszych trendów niestety skazane są na upadek.

# Zespoły

- Zespoły napędzają świat, na nich opiera się metoda Scrum.
- Zmień wydajność zespołu. Ma ona **znacznie** większe znaczenie, niż wydajność osobista
- Transcendentność, autonomia i funkcjonalność.
- Mniejsze zwycięża. Niewielki zespoły pracują szybciej i wydajniej niż duże. Zasada siedmioosobowa, plus minus dwóch członków.
- „*Gdy zespół znajduje się w dużym pokoju, informacje posiadane przez innych stają się bez wysiłku twoje. Zaczynasz myśleć w kategoriach, co jest najlepsze dla grupy jako całości, a nie tylko dla twojego otoczenia*”.

# Scrum w czasach rewolucji

A photograph showing a large crowd of people during a revolution. In the center, a person holds a large Egyptian flag. To the right, another person holds up a white piece of paper with Arabic writing on it. The scene is filled with smoke and dust, suggesting a protest or riot.

- Zespół odcięty od kierownictwa, zdany tylko na siebie.
- Samodzielne organizowanie pracy, korzystanie ze struktur Scrum.
- Dziennikarze też korzystają ze Scruma : )
- Podział na scrumowe drużyny w armii.

# Rozmiar ma znaczenie

- Najefektywniej pracują zespoły 7 +- 2 osobowe.
- Dla członków zespołów Scrum kluczowa jest wiedza, czym zajmują się pozostali towarzysze.
- Małe zespoły => duża efektywność.
- Scrum Master – ktoś pomiędzy kapitanem, a coachem.

# Maoryska haka



# Przestań nienawidzić zawodnika, znienawidź grę

- W sytuacjach problemowych, zamiast szukać rozwiązań, szukamy winowajcy.
- Każdy z nas jest stworzony przez system.
- Scrum akceptuje to, jednak dopuszcza modyfikację wadliwie działającego systemu.
- Zmiana systemu jest najczęściej bardziej znacząca od zmiany jednostek.

# Czas, czas i jeszcze raz czas

- Czas jest ograniczony. Podziel pracę na fragmenty, które możesz wykonać w regularnym, ustalonym, krótkim okresie (1-4 tygodni)- sprint
- Wersja demonstracyjna lub śmierć. Na końcu każdego sprintu musi znaleźć się coś co jest **gotowe**.
- Wyrzuć swoje wizytówki, tytuły to specjalne oznaczenia statusu. Bądź znany z tego **co robisz**, a nie z tego, jaki tytuł nosisz.
- Każdy ma wiedzieć wszystko. Intensywność komunikacji przyspiesza pracę.
- Jedno spotkanie dziennie. Ludzie zbierają się każdego dnia na kwadrans, aby zobaczyć, co można zrobić, aby przyspieszyć.

# Marnotrawstwo jest przestępstwem

- Wzorce pozytywne i negatywne,
- Rozwiążanie problemu => Scrum,
- Czy warto marnować 85% włożonej w projekt pracy?



**Taiichi Ohno – „Toyota Production System”:**

1. „muri” – marnotrawstwo w wyniku braku rozsądku,
2. „muda” - marnotrawstwo ze względu na wyniki,
3. „mura” – marnotrawstwo w wyniku niekonsekwencji.

# Rób jedną rzecz naraz

- Żonglowanie zadaniami a ich jakość.
- Ile masz IQ?

Dane mamy 3 projekty: projekt A, złożony z części: A1, A2 i A3, projekt B, o składowych: B1, B2 i B3, oraz C o częściach: C1,C2 i C3. W jakiej kolejności należy realizować zadania z zespołem, aby wykonać je jak najszybciej?

## Ustalanie priorytetu projektów

Produkt A



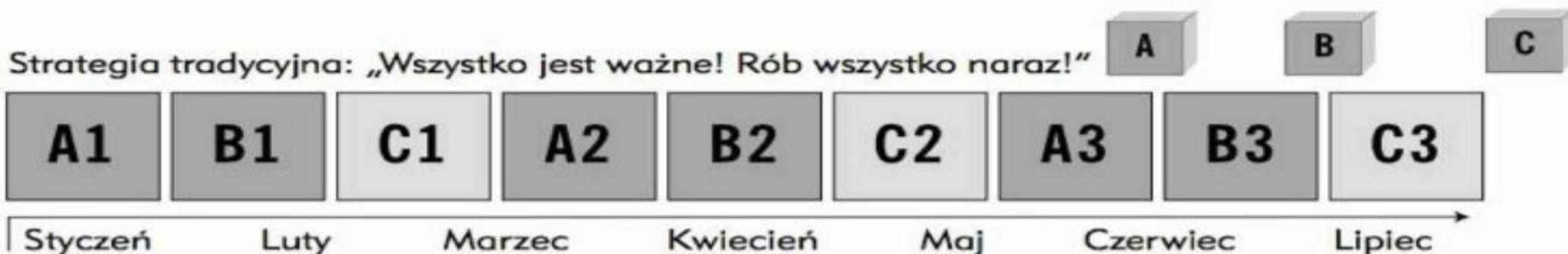
Produkt B



Produkt C



Strategia tradycyjna: „Wszystko jest ważne! Rób wszystko naraz!”



Strategia Agile: „Cele i kierunek!”

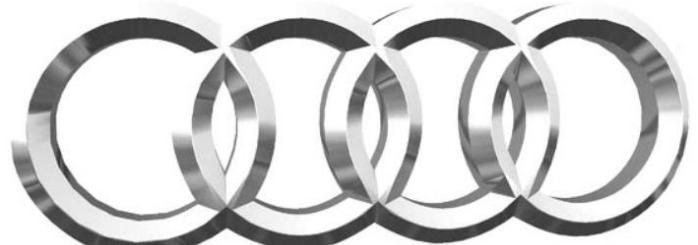


# Wykonanie połowy pracy to praca niezrobiona

- „Powtarzająca się praca”



Znajdź  
różnicę.

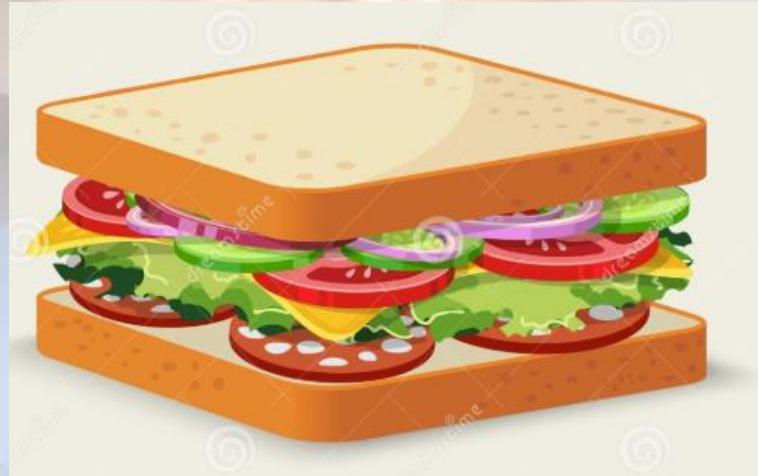


# Człowiek to nie fabryka

- Ludzki umysł ma swoje ograniczenia,
  - Ile trwa proces myślowy?
  - Czy błędy prowadzą nas na dno?

# Zbyt ciężka praca sprawia, że trzeba pracować więcej

- Czy, aby wykonać wszystkie zadania powierzone nam w pracy, wystarczą nam 3 dni z tygodnia?
- Czy kanapki mają wpływ na podejmowane decyzje?
- Czy podejmowanie decyzji ma wpływ na naszą samokontrolę?
- Co daje nam Scrum?



# Marnotrawstwo typu „muri”

MARNOTRAWSTWO  
EMOCJONALNE

ABSURD

NIEROZSĄDNE  
OCZEKIWANIA

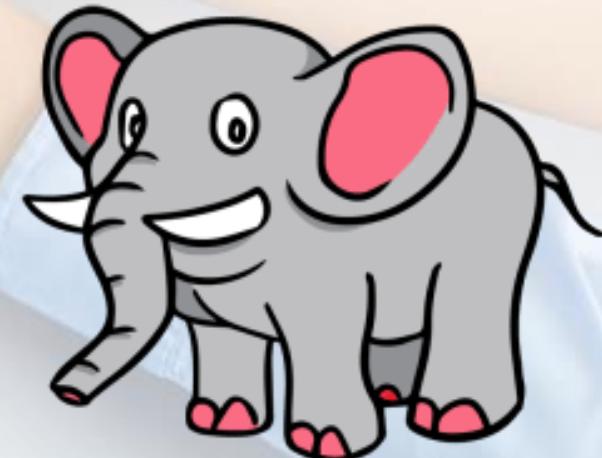
PRZECIĄŻENIE

# Przepływ

- Wielozadaniowość ogłupia i spowalnia, a efekty pracy tracą na jakości,
- Zrobione w połowie to nie zrobione wcale. Praca w toku kosztuje,
- Zrób to dobrze od razu, lub od razu napraw błędy,
- Zbyt ciężka praca to więcej pracy,
- Bądź rozsądny. Cele, które są wyzwaniem stanowią motywacje, te które są nieosiągalne, powodują depresję,
- Bez heroizmu,
- Żadnych dupków, praca bez chaosu emocjonalnego.

# Planuj rzeczywistość, a nie fantazję

- Firma Medco i jej „świetne” pomysły,
- Akt planowania jest bardzo kuszący,
- Ładne wykresy wszystkim się podobają,
- Czy półtora metra wysokości dokumentacji jest potrzebne?
- „Jak zjeść słonia? Po kawałku!”



# Planowanie ślubu

- Panna młoda, pan młody, kwiaty, zaproszenia, kościół, sala, jedzenie, ksiądz, suknia, obrączki, muzyka,
- Mamy już uszeregowaną listę według priorytetów, kolejnym zadaniem jest określenie ile wysiłku, czasu i pieniędzy zajmie ten projekt,

**Wielkość względna** – porównujemy jeden rozmiar do drugiego  
np. dla rozmiarów koszulek.





Labrador – 5 pkt



Dog Niemiecki - 13 pkt



Terier Irlandzki – 5 pkt



Setter Irlandzki – 8  
pkt



Pudel – 3 pkt



Jamnik – 1 pkt



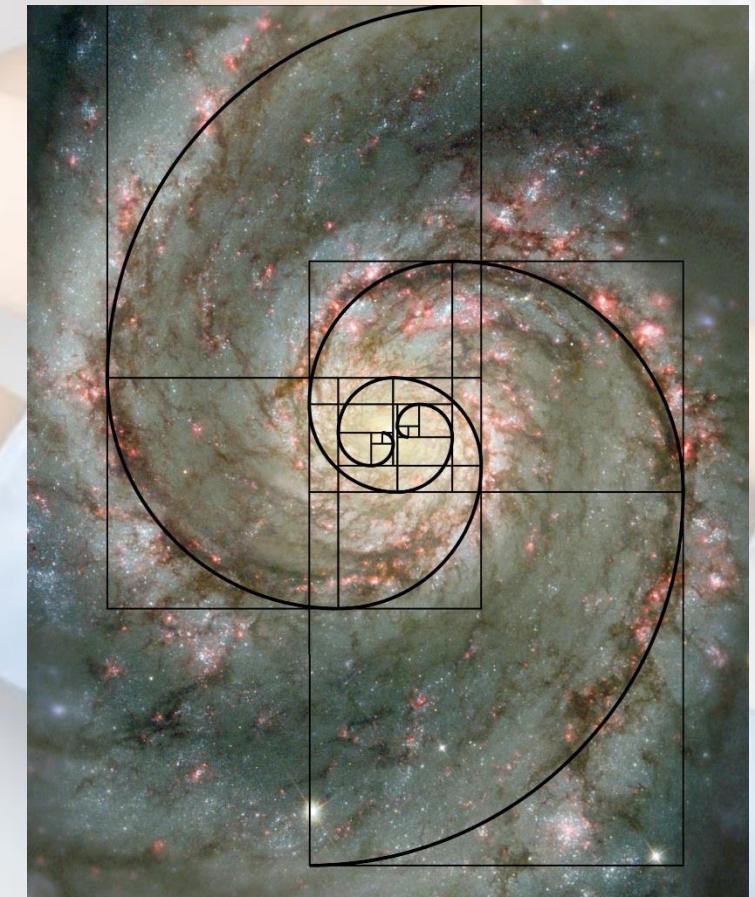
Owczarek Niemiecki –  
8 pkt



Buldog – 3 pkt

# Liczby Fibonacciego

- Psy tworzą szereg: 1, 3, 5, 8, 13 -> Liczby Fibonacciego, to wzorzec uporządkowania w naturze.
- Dlaczego liczby Fibonacciego są takie ciekawe?
- Złoty podział w architekturze.



# Wyrocznia delficka

- Syndrom grupowego myślenia,
- Ludzie zakładają, że inni stosują właściwe oceny, nawet jeśli te oceny są sprzeczne z ich własnymi,
- „Efekt halo”,
- Jeśli ktoś jest przystojny, ludzie zakładają, że jest też mądry i godny zaufania,
- Wojna nuklearna – ile bomb potrzeba Rosji?

# Planning Poker

- W jaki sposób pytać zespoły na ile punktów wyceniają dane zadanie?



# Nie ma zadań. Są tylko historyjki.

- Czy mamy wystarczająco dużo informacji?
- Tworzymy bajki => analiza zadania, która prowadzi do właściwych historyjek
- Cechy dobrych historyjek. Co to EPIK?
- Kiedy historyjka jest gotowa?

## INVEST

Independent, Negotiable, Valuable, Estimable, Small, Testable.

# Planowanie Sprintu

- Jak szybko produkt będzie gotowy?
  - Dlaczego pan prezes krzyczy?
    - Jak pracować szybciej?
  - Szybkość x Czas = Dostawa.

Podsumowanie;

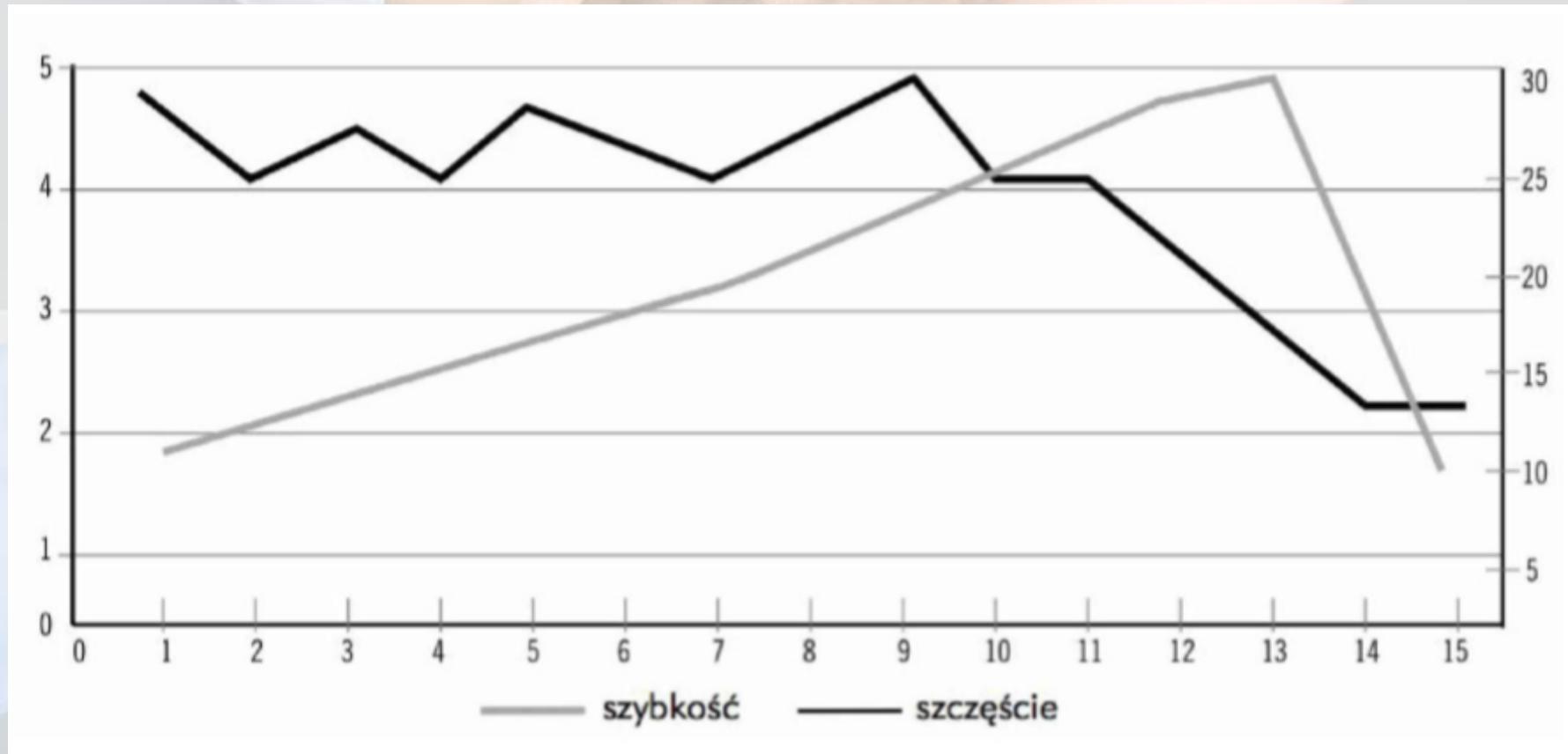


## 😊 Szczęście 😊

- Sposób na szczęście w XXI wieku,
- Czym jest prawdziwe szczęście?
- Jak zmaksymalizować swoje szczęście pracując jednocześnie?
  - Szczęście = sukces.
- Czy jeśli jesteś szczęśliwy, jesteś też przystojniejszy?
- Jak zmierzyć poziom szczęścia pracowników? Jak sprawić, żeby byli szczęśliwi, a co za tym idzie zwiększyli wydajność?



# Mierzyć szczęście: autonomia, mistrzostwo i cel



# Przekluj bańkę

- Scrum polega na ciągłym doskonaleniu siebie i całego procesu. W momencie, gdy team wpada w bańkę samozachwytu i dumy z jakiegoś osiągnięcia, np. Zwiększenia wydajności o 200% po prostu przestaje się rozwijać. Postępy należy monitorować i omawiać, żeby wiedzieć, czy dalej mają miejsce.



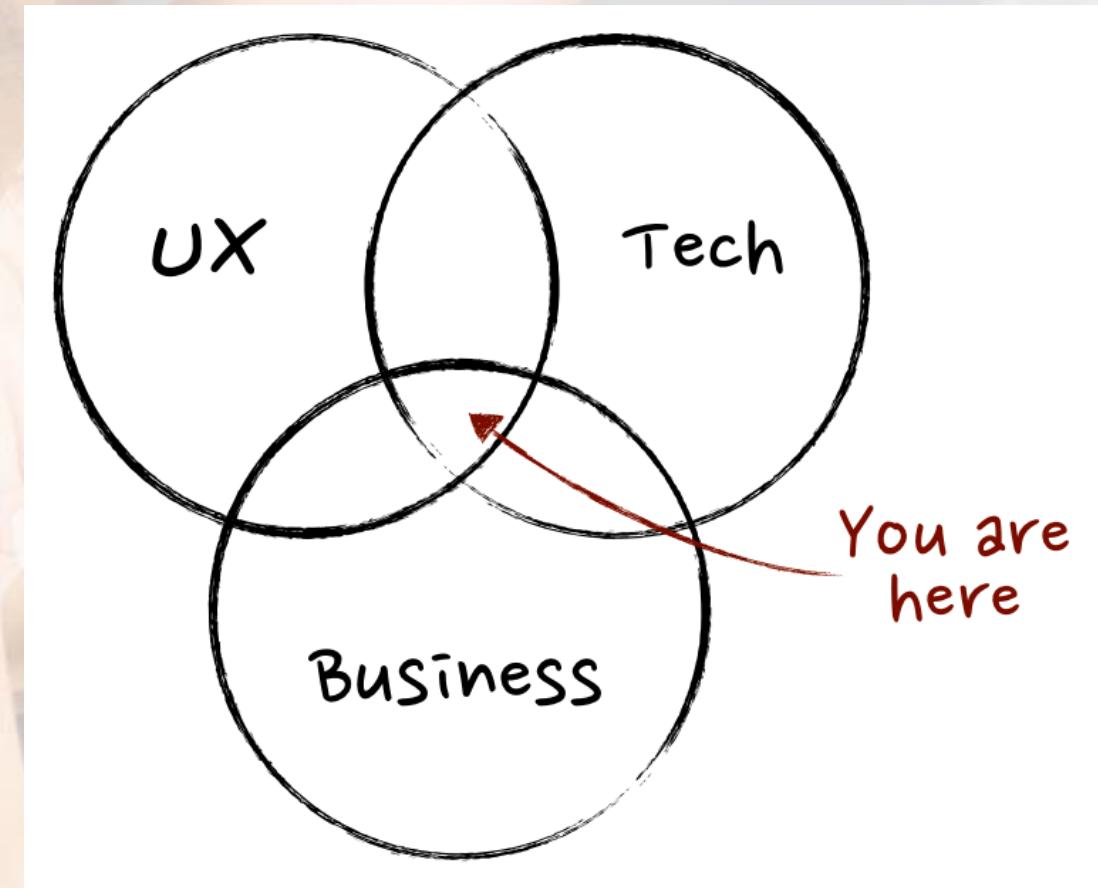
# Priorytety

- Zrób listę. Sprawdź ją dwukrotnie. - Utwórz listę wszystkiego co jest do zrobienia w projekcie i uporządkuj ją według priorytetu
  - Zastosowania:
    - planowanie nauki
    - planowanie wyjazdu
    - planowanie zakupów

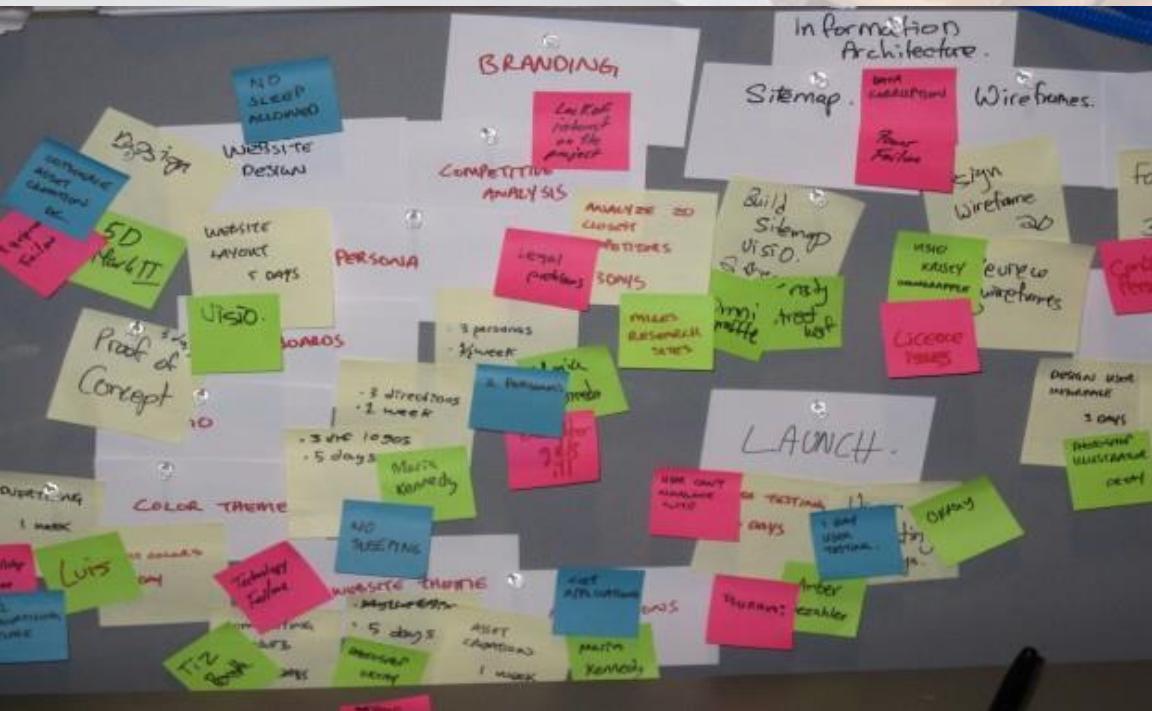
# to do list

# Product Owner

- Właściciel produktu - przekłada wizję na rejestr produktu. Musi rozumieć klienta, przypadki biznesowe oraz rynek. Ma wiedzę w danej dziedzinie i może podejmować decyzje ostateczne. Musi być dostępny, aby odpowiadać na pytania. Jest odpowiedzialny za dostarczenie wartości. Właściciel produktu określa, co powinno być zrobione i dlaczego. Jak zespół to osiąga i kto to robi zależy od zespołu.
- Rejestr produktu – spis rzeczy, które należy wykonać, aby stworzyć produkt.



# Scrum w szkole



no scrum@school



scrum@school



# OODA - observe, orient, decide, act

- Patrz strategicznie na całość, ale działaj taktycznie i szybko.



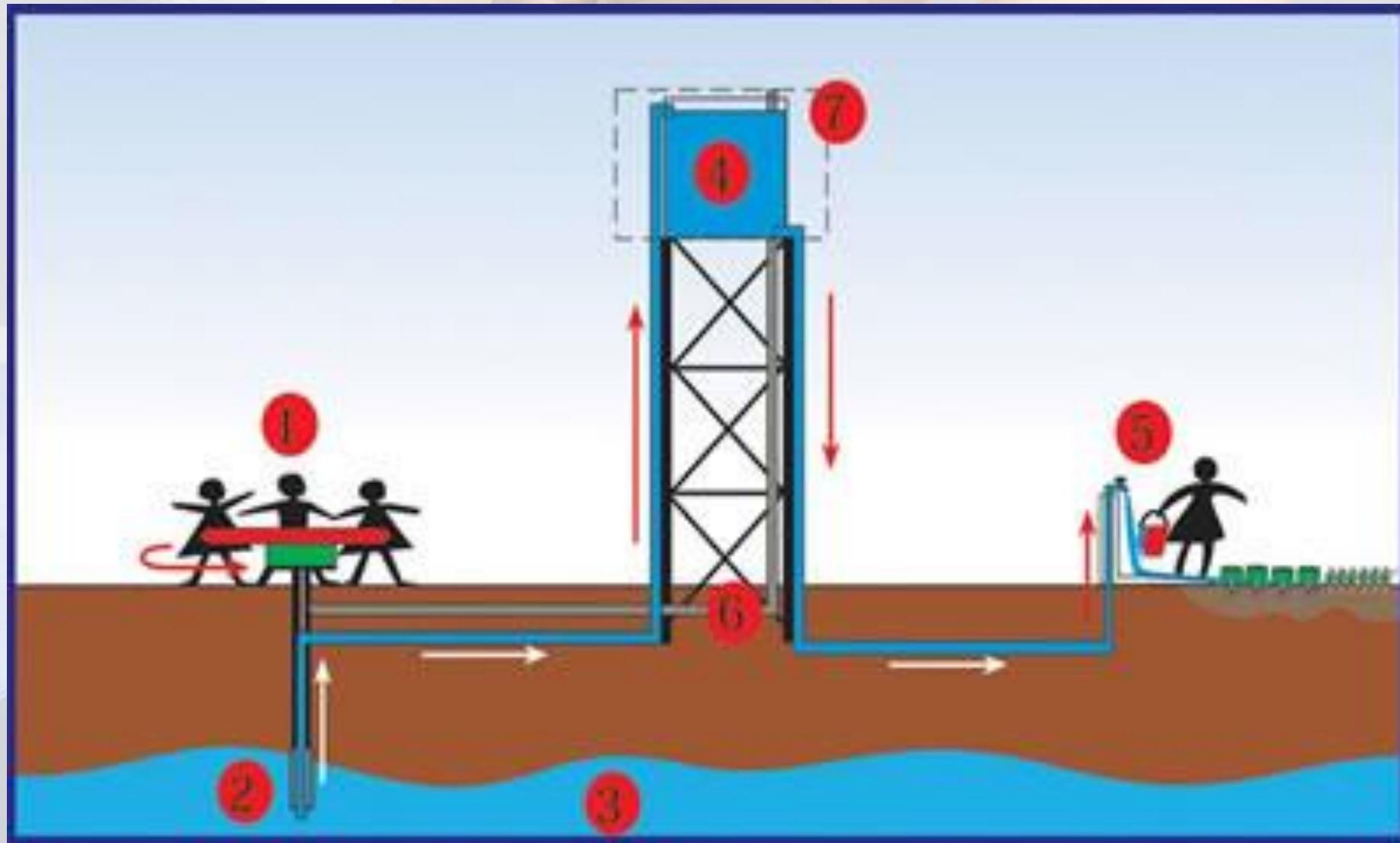
# Scrum w rodzinie

- Ciągła adaptacja – eksperymentuj z nowymi pomysłami i zachowaj otwarty umysł.
- Daj dzieciom możliwość podejmowania decyzji - pozwól im planować własne cele, oceniać swoją pracę, odnosić sukces na swoich zasadach a czasem też ponosić porażkę i uczyć się na błędach.
- Opowiadaj swoją historię - zdefiniuj podstawowe wartości w rodzinie i pozwól dzieciom poczuć się elementem czegoś większego.



[https://www.ted.com/talks/bruce\\_feiler\\_agile\\_programming\\_for\\_your\\_family?language=pl](https://www.ted.com/talks/bruce_feiler_agile_programming_for_your_family?language=pl)

# Roundabout PlayPump – co poszło nie tak?

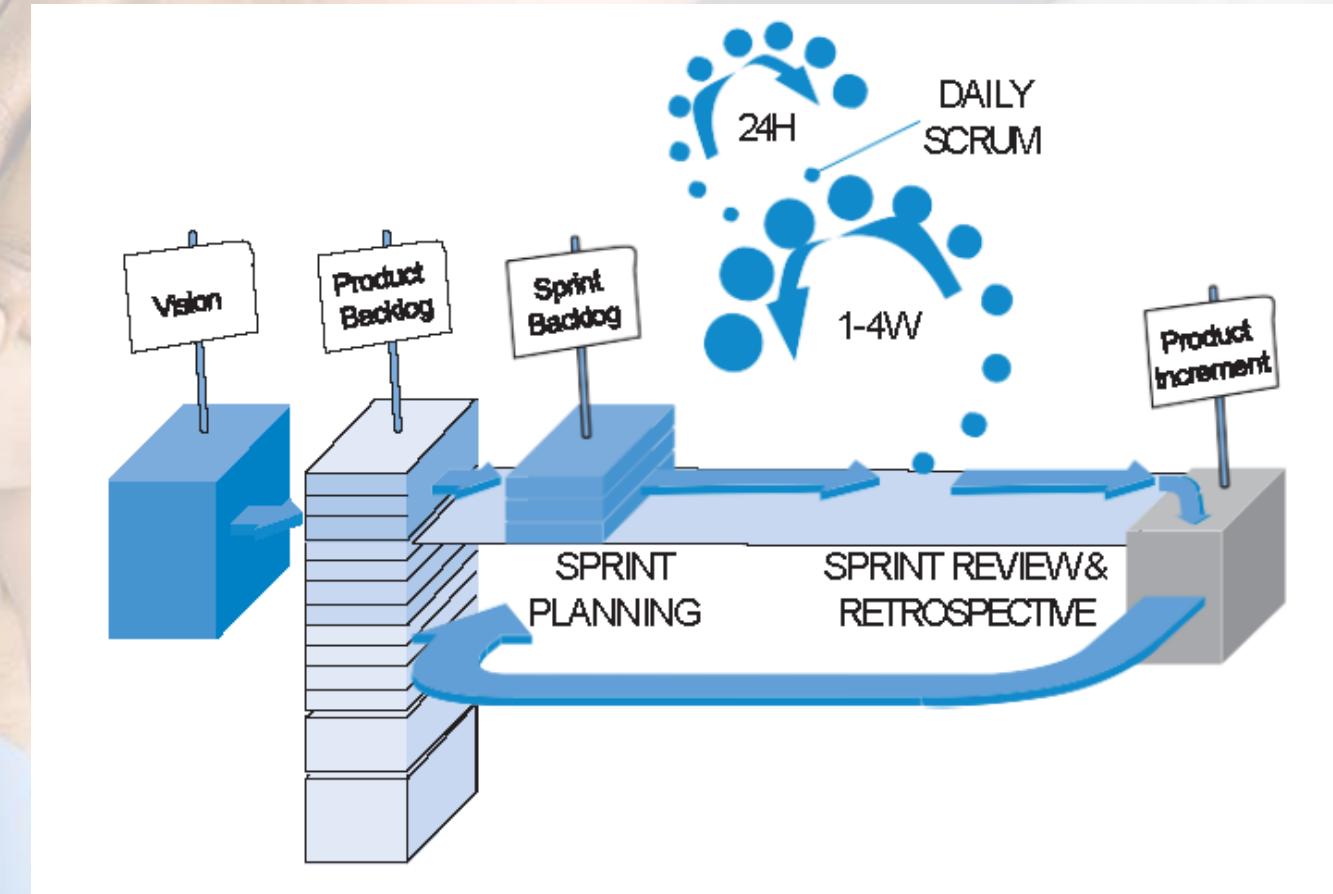


# Organizacja własnego czasu

- Tworzenie list rzeczy do zrobienia dzisiaj/w tym tygodniu
- Wyznaczanie sobie celów dotyczących czasu
- Rozeznanie w tym, jak będzie wyglądać nauka/praca w ciągu kilku kolejnych miesięcy
- Minusy (w porównaniu do Scrum 'a w grupach):
  - subiektywny feedback od samego siebie
  - brak teamu, brak możliwości otrzymywania wsparcia i udzielania pomocy
  - podejmowanie wszystkich decyzji samodzielnie (brak eksperta który może decydować i uzasadniać potrzebę wykonania pewnych rzeczy)

# Najważniejsze elementy adaptacji Scruma w życiu codziennym

- Adaptacja
- Pragmatyzm
- Planowanie
- Komunikacja
- Jasne określenie celu
- Podsumowywanie osiągnięć
- Analiza feedbacku
- Wyznaczanie łatwych do zrealizowania, krótkoterminowych celów



A photograph showing a group of diverse hands stacked together in a circle. The hands belong to people of various skin tones, including white, black, and tan. They are wearing different types of clothing, such as a white shirt cuff, a grey sleeve, and a blue denim cuff. The hands are positioned in the center of the frame, with some hands overlapping others. The background is a plain, light color.

Dziękuję za uwagę :)



# Zarządzanie Cyklem Testów

Wrocław

21.12.2017

# Agnieszka Grodzka

agnieszka.grodzka@volvo.com

- 2004 – 2009 - Tester oprogramowania w firmie zajmującej się programem prawniczym i translatorami
- 2009 – 2011 - Tester oprogramowania w firmie zajmującej się produktami ERP
- 2012 – 2014.X - Quality Assurance & Test Engineer w Volvo IT (pracownik zewnętrzny)
- Od 2014. - Quality Assurance & Test Engineer w Volvo IT (pracownik wewnętrzny)

# Andrzej Zdebik

andrzej.zdebik@volvo.com

- 2007 - 2011
  - Tester oprogramowania / Lider testów w BZ WBK
- 2012
  - Quality Assurance & Test Engineer w Volvo IT  
rola: Tester oprogramowania (pracownik zewnętrzny)
- Od 2013.
  - Quality Assurance & Test Engineer w Volvo IT  
rola: Test Manager (pracownik wewnętrzny)

# Organizacja Grupy Volvo



Group Trucks Sales



Group Trucks Operations



Group Trucks Technology



Construction Equipment



Business Areas



Volvo Financial Services

# Grupa Volvo

## Nasze marki



# Volvo Group IT

## Globalna obecność

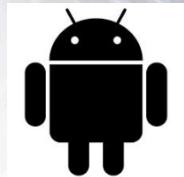


# Volvo Group IT

- Dostarcza rozwiązania i usługi IT dla całej grupy Volvo
- Wspiera wszystkie gałęzie biznesu Volvo (Trucks, Buses, Construction Equipment, Penta, Financial Services, HR, IT)
- Ponad 8000 pracowników globalnie, 800 w Polsce
- Szeroka gama technologii oraz gotowych produktów i aplikacji



# Volvo Group IT



# Grupa Volvo

## Volvo Polska Sp z.o.o

### Wrocław ul. Mydlana 2



# Tytuł Top Employer dla Volvo Polska





# Zarządzanie Cyklem Testów

Wrocław

21.12.2017

# Agenda

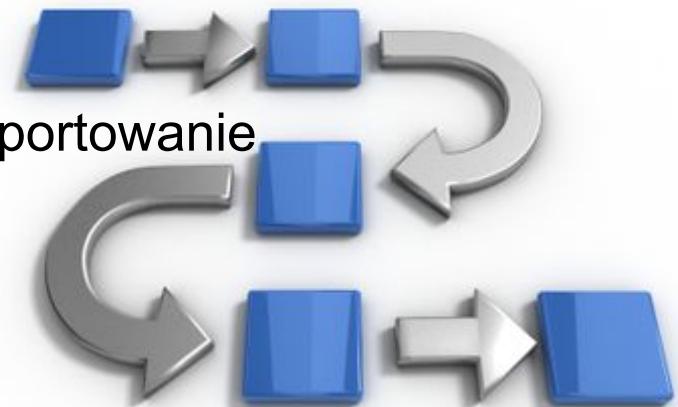
- Volvo Group i Volvo IT
- **Proces i cykle testowe**
- Techniki testowe
- Strategia i struktura testów
- Planowanie i organizacja
- Metryki i narzędzia testowe



# Proces testowy

**Podstawowy proces testowy składa się z następujących etapów:**

- Planowanie i nadzór
- Analiza i projektowanie testów
- Implementacja i wykonanie testów
- Ocena spełnienia kryteriów zakończenia i raportowanie
- Czynności zamykające testy.



# Planowanie i nadzór

- Definiowanie celów testowania i czynności testowych
- Monitorowanie testów – porównywanie uzyskiwanych rezultatów z oczekiwanyimi
- Raportowanie statusu testów.



# Analiza i projektowanie testów

- **Identyfikacja warunków testowych**

*Warunek testowy – element lub zdarzenie modułu lub systemu, który może być zweryfikowany przez jeden lub więcej przypadków testowych, np. funkcja, transakcja, cecha, atrybut jakości lub element struktury.*

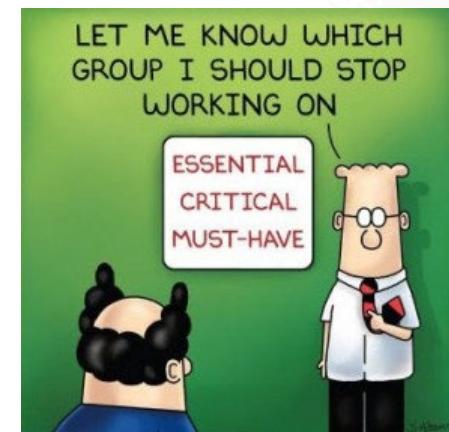
- **Utworzenie przypadków testowych**

*Przypadek testowy – zbiór danych wejściowych, wstępnych warunków wykonania, oczekiwanych rezultatów i końcowych warunków wykonania opracowany w określonym celu lub dla warunku testowego*



# Implementacja i wykonanie testów

- Utworzenie procedur testowych z przypadków testowych
- Weryfikacja środowiska testowego i niezbędnych danych testowych
- Wykonanie procedur testowych zgodnie z ustalonymi priorytetami i odznaczanie rezultatów
- Rejestrowanie / retesty incydentów.



# Ocena spełnienia kryteriów zakończenia i raportowanie

- Sprawdzanie, czy kryteria zakończenia zostały spełnione
- Sporządzenie raportu podsumowującego testy.

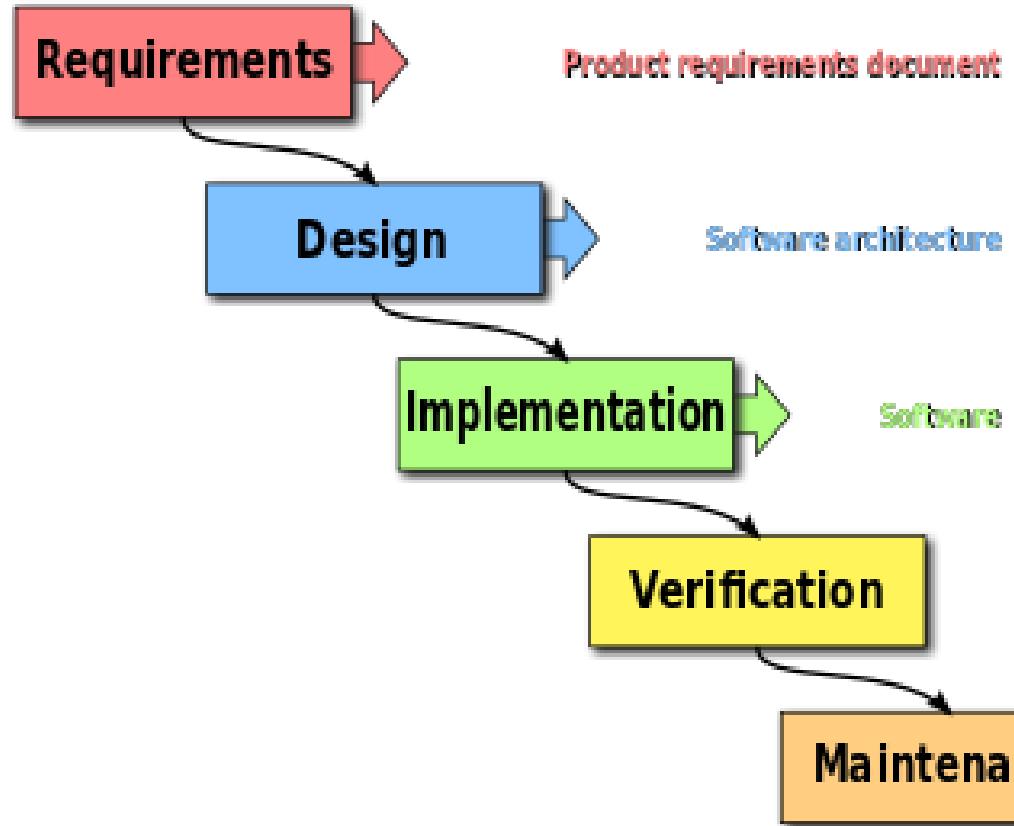


# Czynności zamykające testy

- Upewnienie się co do ukończenia testów
- Zamykanie incydentów lub tworzenie na ich bazie nowych wymagań
- Archiwizowanie danych, które mogą być przydatne w przyszłości
- Przekazanie testów i środowiska testowego do wsparcia
- Wyciąganie wniosków na przyszłość.

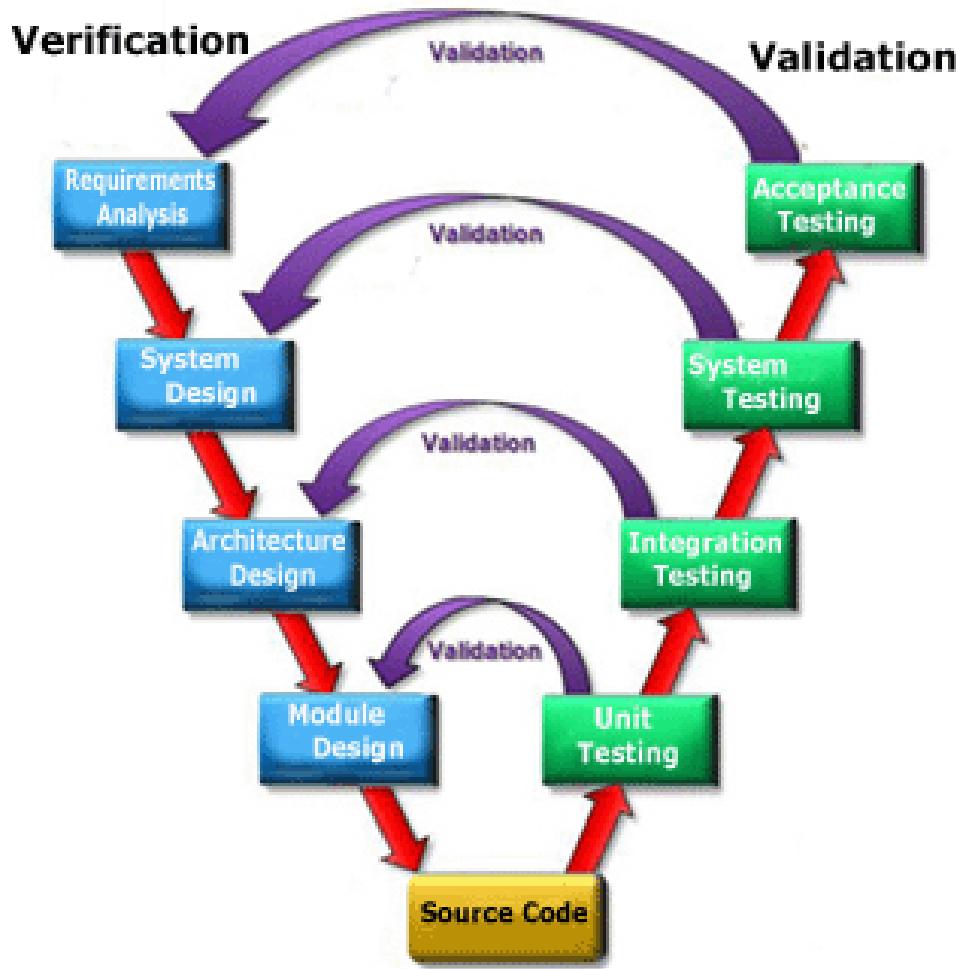


# Cykle testowe – Model Kaskadowy



- Każda faza musi zostać ukończona, zanim kolejna faza się rozpoczęcie

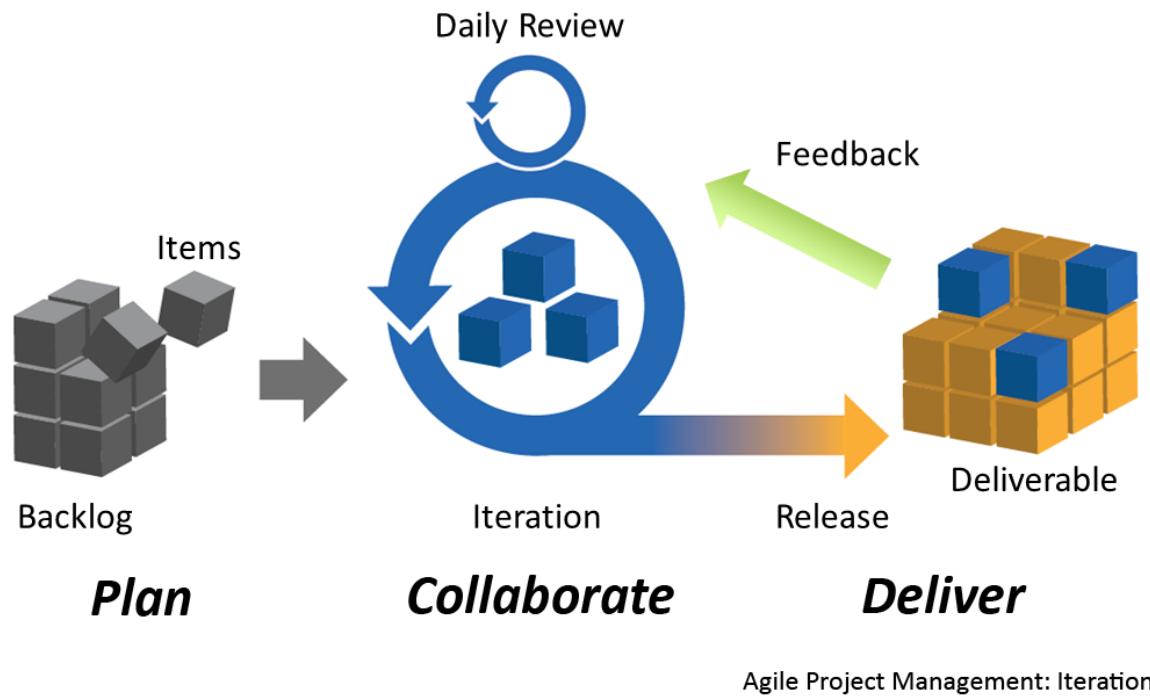
# Model V (Model Sekwencyjny)



- Testowanie przebiega równolegle z odpowiednią fazą dewelopmentu

# Modele Iteracyjno-Przyrostowe

- Proces definiowania wymagań, projektowania, implementowania i testowania systemu podzielony na serię krótkich cykli



# Jira - Backlog

- Backlog jest priorytetyzowany przez klienta biznesowego

The screenshot shows a Jira Agile board titled "Copy of VOSPCQ Product Backlog". The board has columns for "Backlog", "Active sprints", "Reports", and "Board". The backlog contains 315 issues, which are listed below:

- VOSPCQ-5793 Removal of old optimization logic (3.42.0 - VOSP - Open)
- VOSPCQ-5722 VOSPCQ schedule to be in sync (3.42.0 - CQVOSP, VOSPCQ - Analysis - On-Going)
- VOSPCQ-5825 Annual service for RT (3.42.0 - VOSP - Open)
- VOSPCQ-5197 Replan - Show the latest Mileage value from LDE and compare towards last reg mileage from VSR (3.42.0 - VOSP - Analyzed)
- VOSPCQ-4944 Warrenty Service operation should be calculated starting from Delivery date (3.42.0 - VOSP - Analyzed)
- VOSPCQ-4286 Startdate in Schedule be equal to or come before Startdate in Contract (3.42.0 - VOSPCQ - Analyzed)

The right side of the board shows a sprint backlog for "3.42.0" with tasks assigned to "A" (1w), "D" (1w), "D" (2h), "D" (2h), "D" (1d 4h), "D" (2d), "D" (3d), "D" (3d), and "D" (2d 2h). The board also includes sections for "Sprints", "Epic", and "Issues".

# Jira – board dla bieżącego sprintu

SPRINT: Sprint 2 R3.42 ▾

QUICK FILTERS: 3.42.0 3.41.3 New vosp GUI Ready for Test IT Registered Yestarday Registered Last 2 Weeks Fixed Yestarday Tested Yestarday Automated Tests Component VospCq ... Show more

To Do	In Progress	For IT Test	For Test
<p>VOSPCQ-5945 ↑ [GVW/GCW] and [Max speed] displayed in view mode</p> <p>None Open VOSP</p>	<p>VOSPCQ-5947 ↑ Operations deselected after editing sched</p> <p>None In Progress Internal bugs</p>	<p>VOSPCQ-3181 ↓ Current mileage disappeared after e calculation</p> <p>None Ready For Test IT CQ</p>	<p>VOSPCQ-5348 ↑ TAGS - for SOPS - Incl, Excl, In Contract, In Vosp,</p> <p>4 hours Ready For Test CQ</p>
<p>VOSPCQ-5951 ↑ Wrong order of operations on tab</p> <p>None Open VOSP</p>	<p>VOSPCQ-5005 ↑ Connection between VSS EDU and CQ E available</p> <p>0 minutes In Progress CQ</p>	<p>VOSPCQ-5056 ↓ Null pointer exception while printing</p> <p>None Ready For Test IT CQ</p>	<p>VOSPCQ-4360 ↑ New field in Customer for contract invoice email address + Tag</p> <p>4 hours Ready For Test CQVOSP</p>
<p>VOSPCQ-5952 ↑ Incorrect width of service schedule</p> <p>None Open VOSP</p>	<p>VOSPCQ-5681 ↑ ISE occurs during trying to get the list of a</p> <p>None In Progress Internal bugs</p>	<p>VOSPCQ-5745 ↑ Increasing duration in VOSP does not update schedule.</p> <p>None Ready For Test IT VOSP</p>	<p>VOSPCQ-5321 ↓ Add payment frequency to Admin/Contract Type/Defaults</p> <p>0 minutes Ready For Test CQ</p>
<p>Tags in CQ - CQ documents</p> <p>4 hours Analyzed CQ</p>	<p>Turn on Uptime when chassisid is added or changed</p> <p>Uptime in VOSP 2 weeks In Progress VOSP</p>	<p>Impossible to Import multiple vehicle calculation to VOSP</p> <p>0 minutes Ready For Test IT CQVOSP</p>	<p>VOSPCQ-2557 ↑ Validate max frozen price deviation when changing from non-frozen to frozen</p> <p>3 days Ready For Test CQ</p>

# Agenda

- Volvo Group i Volvo IT
- Proces i cykle testowe
- **Techniki testowe**
- Strategia i struktura testów
- Planowanie i organizacja
- Metryki i narzędzia testowe



# Techniki testowe

STATYCZNE	DYNAMICZNE
<b>Nie wymaga</b> uruchomienia kodu	<b>Wymaga</b> uruchomienia kody
Wspiera <b>weryfikacje</b>	Wspiera <b>validacje</b>
Znajduje <b>błędy</b>	Wykrywa <b>awarie</b>
Wymaga <b>checklisty</b> lub procesu	Wymaga <b>test casów</b>
<b>Niższe koszty</b> znalezienia i naprawy błędu	<b>Wyższe koszty</b> znalezienia i naprawy błędu

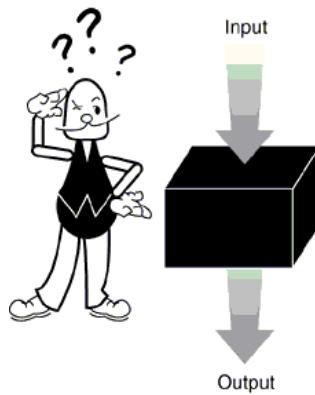
# Techniki statyczne

- Przegląd
  - Formalny
  - Nieformalny
  - Przegląd techniczny
  - Inspekcja
- Analiza statyczna

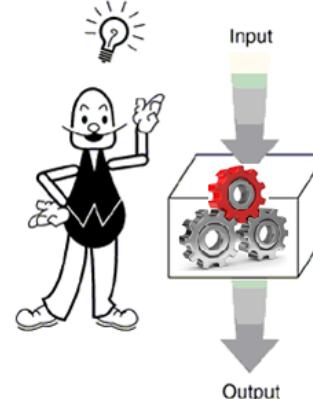


# Techniki projektowania testów

BLACK-BOX



WHITE-BOX



# Techniki Black-box

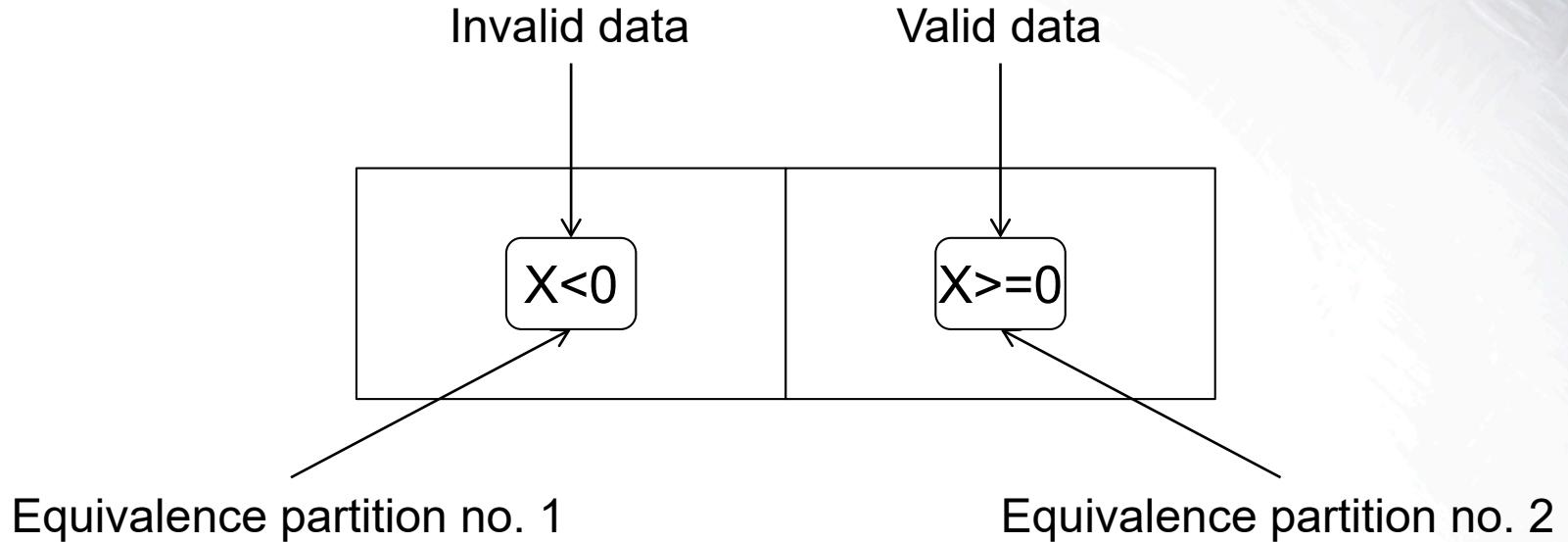
## Partycje równoważne

- Wejścia systemu są dzielone na grupy, które powodują podobne zachowanie oprogramowania
- Wyznaczane są klasy równoważności dla danych poprawnych oraz niepoprawnych
- Stosowana na każdym poziomie testów



# Techniki Black-box

## Partycje równoważne



**Test case no. 1:**  $x = -3$

**Test case no. 2:**  $x = 10$

# Techniki Black-box

## Analiza wartości brzegowych

- Rozszerzenie techniki równoważnych partycji
- Projektowanie testów dla minimum i maksimum klasy równoważności – duże prawdopodobieństwo wykrycia błędu
- Stosowana na każdym poziomie testowania



# Techniki Black-box

## Analiza wartości brzegowych

Invalid partition

Valid partition

Invalid partition

$$x \geq 0, x < 100$$



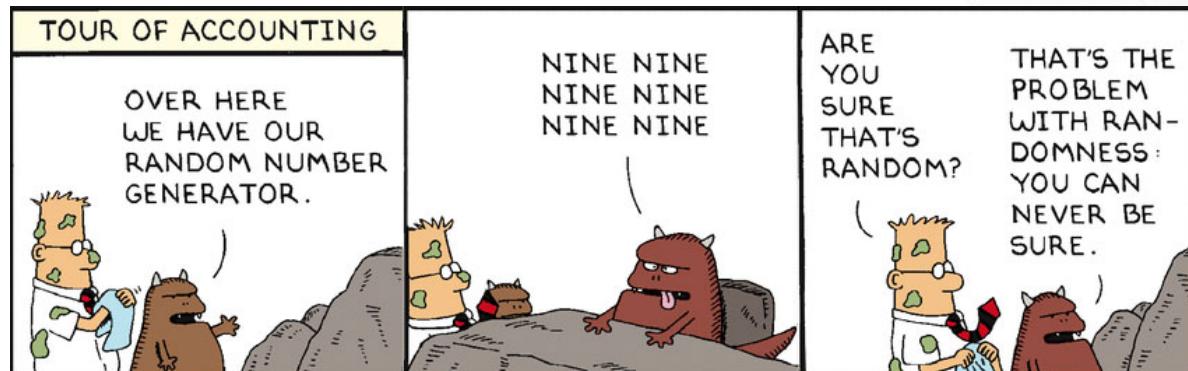
**Test case no. 1:**  $x = -1$   
**Test case no. 2:**  $x = 0$   
**Test case no. 3:**  $x = 1$

**Test case no. 4:**  $x = 98$   
**Test case no. 5:**  $x = 99$   
**Test case no. 6:**  $x = 100$

# Techniki Black-box

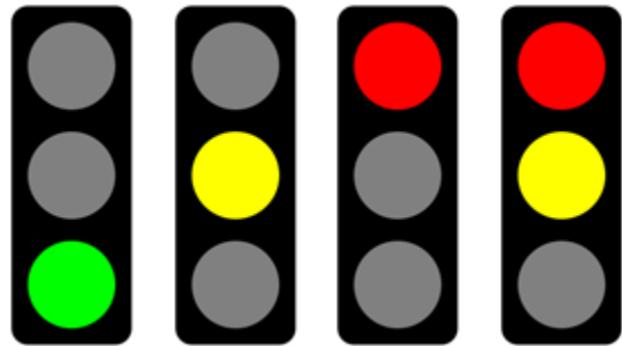
## Tablice decyzyjne

- Warunki wejściowe oraz zachowanie systemu zapisane jako prawda lub fałsz
- Tabela decyzyjna zawiera warunki uruchamiające
- Każda kolumna tabeli odpowiada jednej regule biznesowej



# Techniki Black-box

## Tablice decyzyjne



CONDITIONS		RULES			
	Red light	False	False	True	True
	Yellow light	False	True	False	True
	Green light	True	False	False	False
ACTIONS					
	You can drive	X			
	Stop			X	
	Be ready to drive				X
	Stop immediately		X		

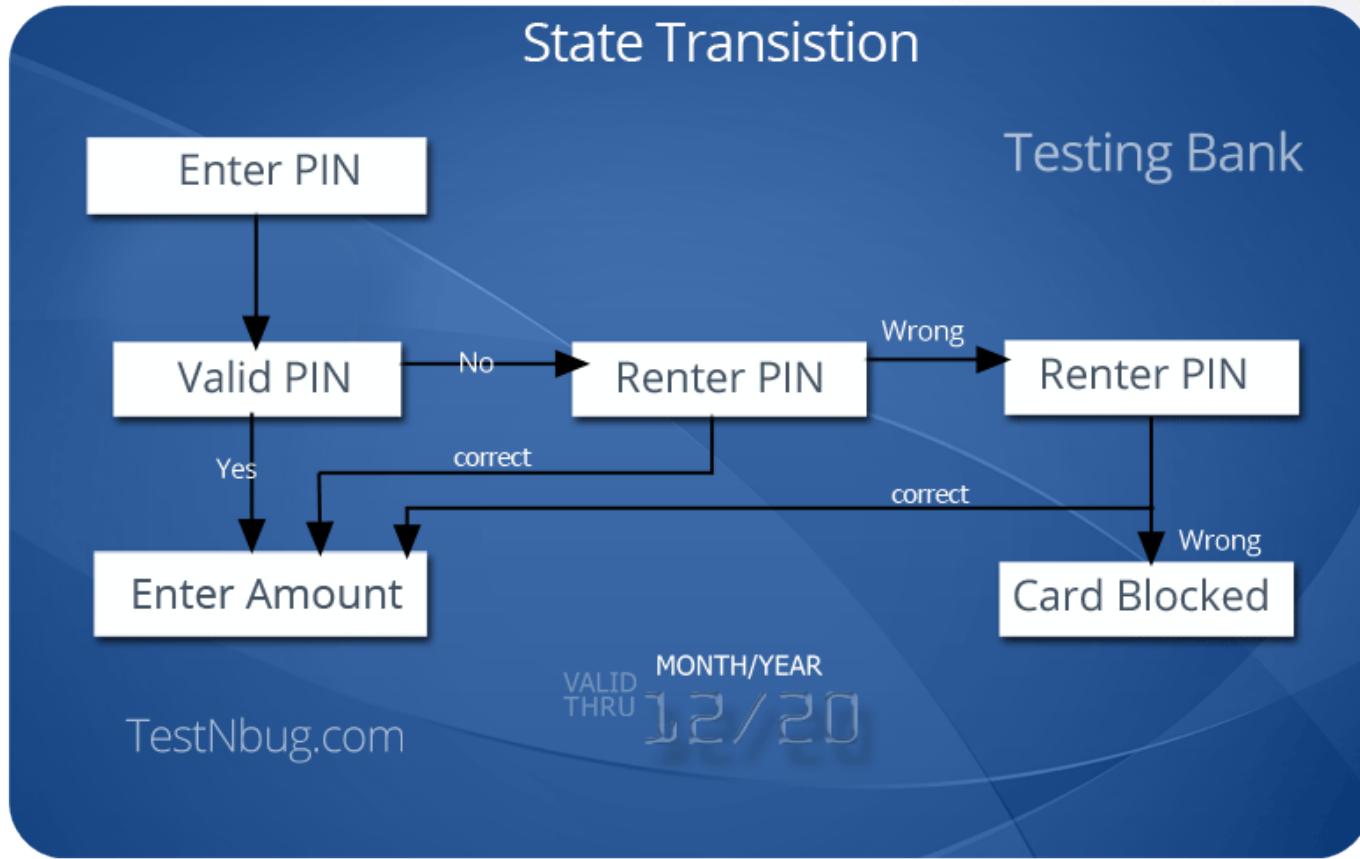
# Techniki Black-box

## Przejścia między stanami

- Zachowanie systemu można opisać diagramem przejść stanów
- Pokrycie testami:
  - typowe sekwencje stanów
  - każdy stan
  - każde przejście
  - konkretny ciąg przejść

# Techniki Black-box

## Przejścia między stanami



# Techniki Black-box

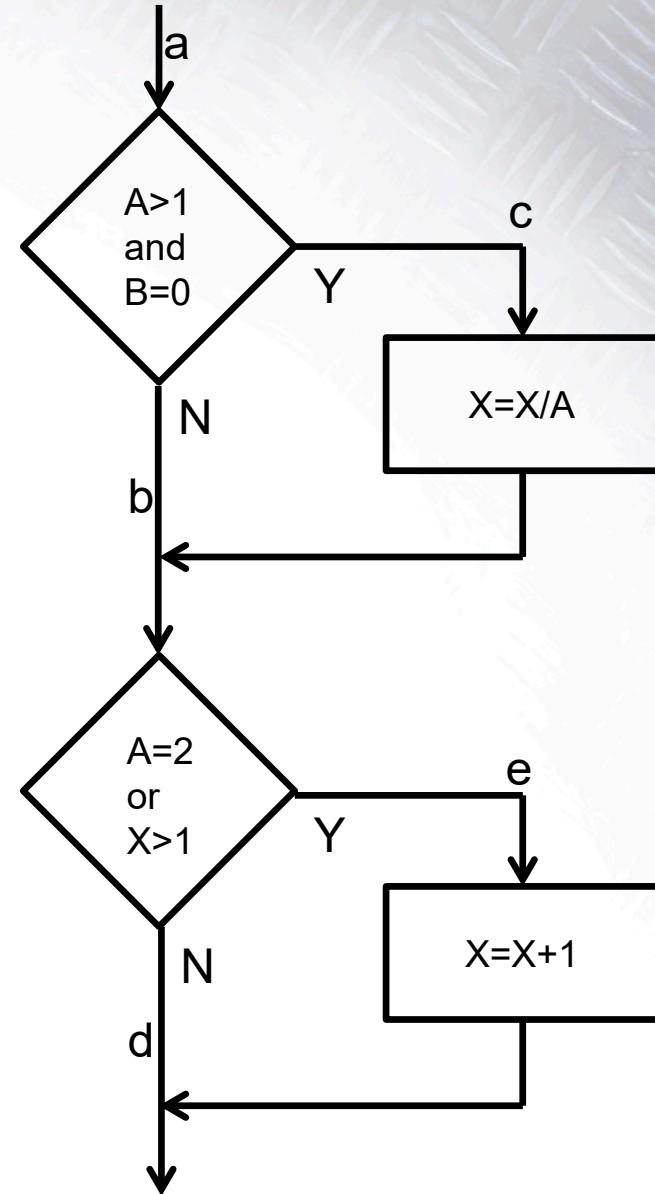
## Przypadki użycia

- Opisuje interakcje pomiędzy aktorami, które powodują powstanie wyniku wartościowego z punktu widzenia użytkownika
- Przypadki użycia zwykle posiadają scenariusz główny oraz scenariusze poboczne
- Przypadki użycia opisują najbardziej prawdopodobne przypadki

# Techniki White-box

## Pokrycie instrukcji

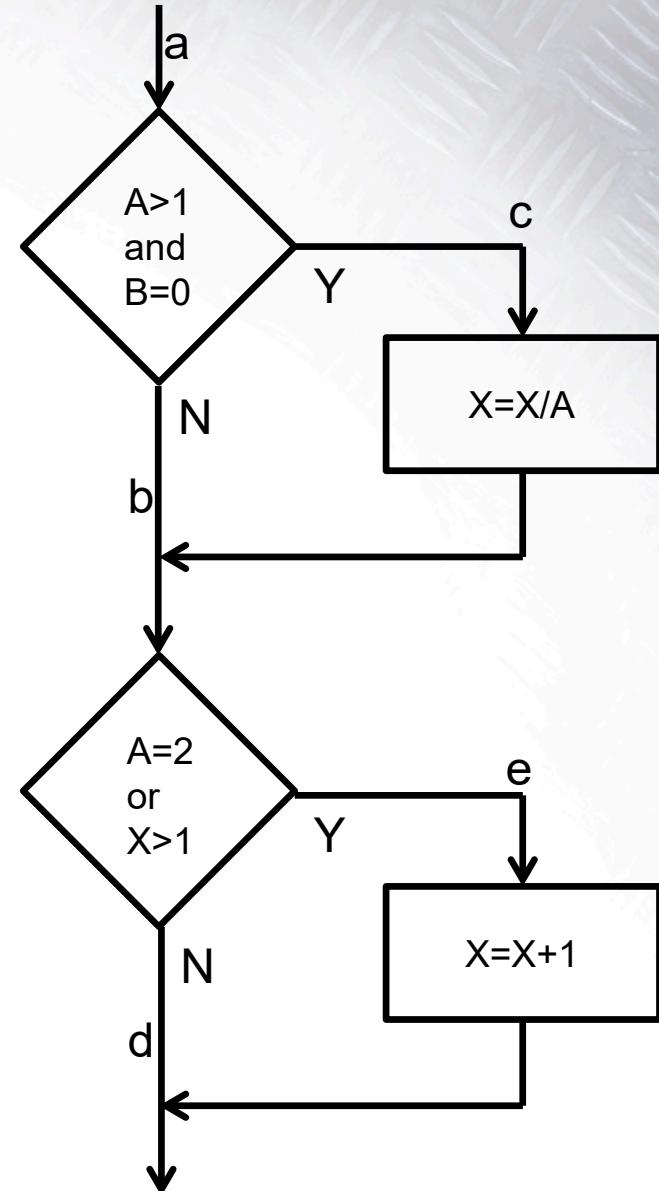
- Pokrycie wszystkich instrukcji przynajmniej raz
- Test cases:
  - $A = 2$  and  $B = 0$  (ace)



# Techniki White-box

## Pokrycie decyzji

- Pokrycie wszystkich decyzji „prawdy” i „fałszu” przynajmniej raz
- Test cases:
  - $A = 2$  and  $B = 0$  (ace)
  - $A = 1$  and  $X = 1$  (abd)



# Agenda

- Volvo Group i Volvo IT
- Proces i cykle testowe
- Techniki testowe
- **Strategia i struktura testów**
- Planowanie i organizacja
- Metryki i narzędzia testowe



# Strategia testowa

- Opisuje ogólne podejście organizacji do testów
- Zawiera informacje na temat zarządzania ryzykiem, podział na poziomy testów i czynności związane z testowaniem
- Powinna zawierać kryteria rozpoczęcia i zakończenia testów.



# Typy strategii testowych

## Strategie analityczne

- Zespół testowy opiera przypadki testowe na przykład o wymagania

The screenshot shows a JIRA issue page for VOSPCQ-5067. The page has a header with 'JIRA' and navigation links like 'Dashboards', 'Projects', 'Issues', 'Agile', and 'Create'. Below the header is a breadcrumb trail 'VOSPCQ / VOSPCQ-5067' and the title 'Offer validity settings & alert'. The main content area has tabs for 'Edit', 'Comment', 'Assign', 'More', 'Test OK, Close', 'Reject', and 'Test Failed'. On the right, there's a 'People' section with fields for 'Assignee' (Johan Strikwerda), 'Reporter' (Kacper Potrykus), 'Votes', and 'Watchers'. A sidebar on the right shows a timeline with tasks: '1w 2d', '0m', and '1w 2d 1h 30m'. The central part of the page displays 'Details' for the issue, including Type (Acceptance Test), Priority (Major), Affects Version/s (None), Component/s (CQ), Labels (None), Status (READY FOR TEST), Resolution (Unresolved), Fix Version/s (3.42.0), and an 'Expected Result' section with several bullet points.

**Details**

Type:	Acceptance Test	Status:	READY FOR TEST
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	3.42.0
Component/s:	CQ		
Labels:	None		

**People**

Assignee:	Johan Strikwerda
Reporter:	Kacper Potrykus
Votes:	Vote for this issue
Watchers:	Start watching this issue

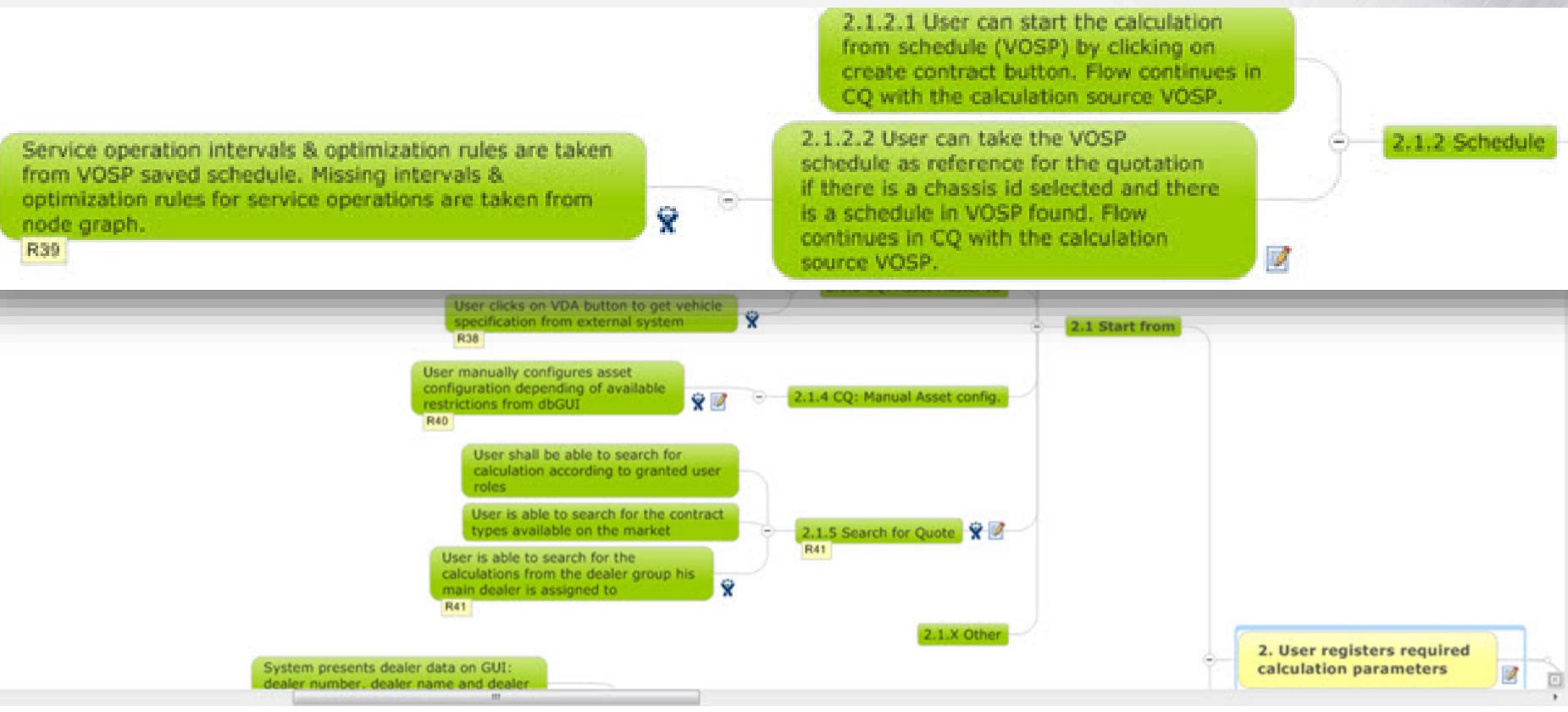
**Expected Result:**

- ✓ A. Administration/General settings per each brand (see GUI prototype)
  - 1. New setting available to set offer validity in days: numeric, no decimal, min 1 max 100
  - 2. New setting available to set the trigger for the alerts in days: numeric, no decimal, min 1 max 100
  - 3. Setting change when there are existing alerts calculated will have impact on recalculation of overview content.
- ✓ B. Calculation wizard (see GUI prototype)
  - 1. Time stamp  
When printing the offer calculation has to log the time stamp of that action (data base log). This date shall be used to calculate the number of valid dates for the offer and the trigger for alerts (A1, A2).
  - 2. Alert rules  
Example:  
Offer has been printed for the first time at: 15.11.2015 (X)

# Typy strategii testowych

## Strategie analityczne

- Mapa wymagań źródłem przypadków testowych



# Typy strategii testowych

## Strategie metodyczne

- Zespół testowy używa predefiniowanych warunków testowych – na przykład listy warunków testowych do przeprowadzenia smoke testu

B	C	D	E	F	G	H	I	J	K	L
1			Checklist after 3.40 Production deploy	Date:	2015-10-11					
12	8	AGR	CQ	<b>Contract calculations</b>		OK				
13	9	AGR	CQ	<b>Contract calculations - Search</b>		OK				
14	10	AGR	CQ	Check language		OK				
15	11	AGR	CQ	Search, change search criteria, change dates		OK				
16	12	AGR	CQ	Contract details		OK				
17	13	AGR	CQ	Offer		OK				
18	14	AGR	CQ	Contract		OK				
19	15	AGR	CQ	General terms		OK				
20	16	AGR	CQ	Risk and reward letter		OK				
21	17	AGR	CQ	Copy to new -> save		OK				
22	18	AGR	CQ	TQ		OK				
23	19	AGR	CQ	CA (only for Contract Printed status)		OK				

# Typy strategii testowych

## Inne typy strategii

- Strategie na podstawie modelu
- Strategie zgodne z procesem lub standardem
- Strategie konsultatywne
- Strategie testów regresywnych
- **Zwykle używanych jest kilka strategii testowych**



# Struktura testów – typy testów

## Testy Funkcjonalne

- Testowanie „co” robi system (testy czarnoskrzynkowe)

## Testy Niefunkcjonalne

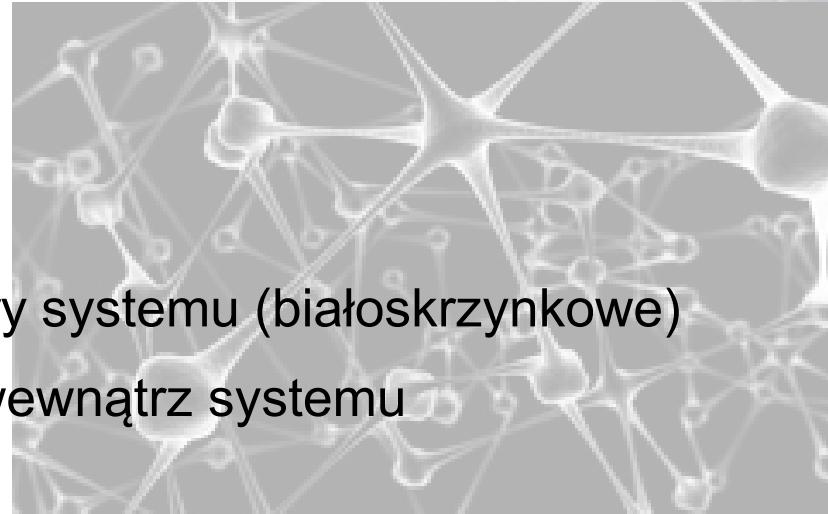
- Testowanie „jak” działa system
- Usability, Portability, Security, Load, Performance Testing



# Struktura testów – typy testów

## Testy Strukturalne

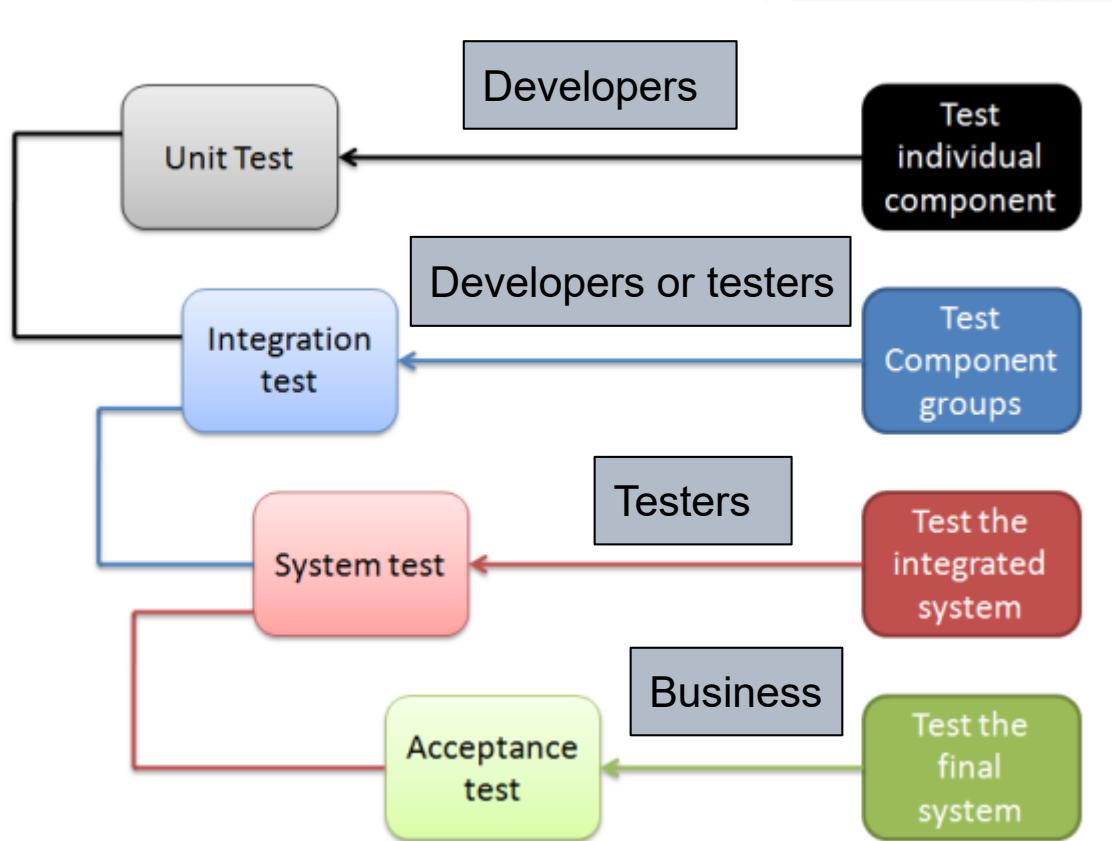
- Testowanie architektury/struktury systemu (białoskrzynkowe)
- Testowanie tego, co dzieje się wewnątrz systemu



## Testy związane ze zmianami (Testowanie Potwierdzające)

- Retestowanie – testowanie po wykryciu i naprawie defektu
- Testowanie regresyjne – weryfikacja, czy wprowadzone modyfikacje nie mają skutków ubocznych

# Struktura testów – poziomy testów



# Agenda

- Volvo Group i Volvo IT
- Proces i cykle testowe
- Techniki testowe
- Strategia i struktura testów
- **Planowanie i organizacja**
- Metryki i narzędzia testowe



# Planowanie i organizacja

## Master test plan / test plan



- Strategia testów / zakres testów
  - **Model wytwarzania oprogramowania**
  - Podział testów
- Kryteria wejściowe / wyjściowe
- Estymacje / **harmonogram**
- **Cykl Życia błędu**
- Metryki / lokalizacja dokumentacji oraz rezultatów testów
- Ryzyka
- Narzędzia wspomagające testowanie
- Środowiska testowe
- **Zasoby**



# Planowanie i organizacja

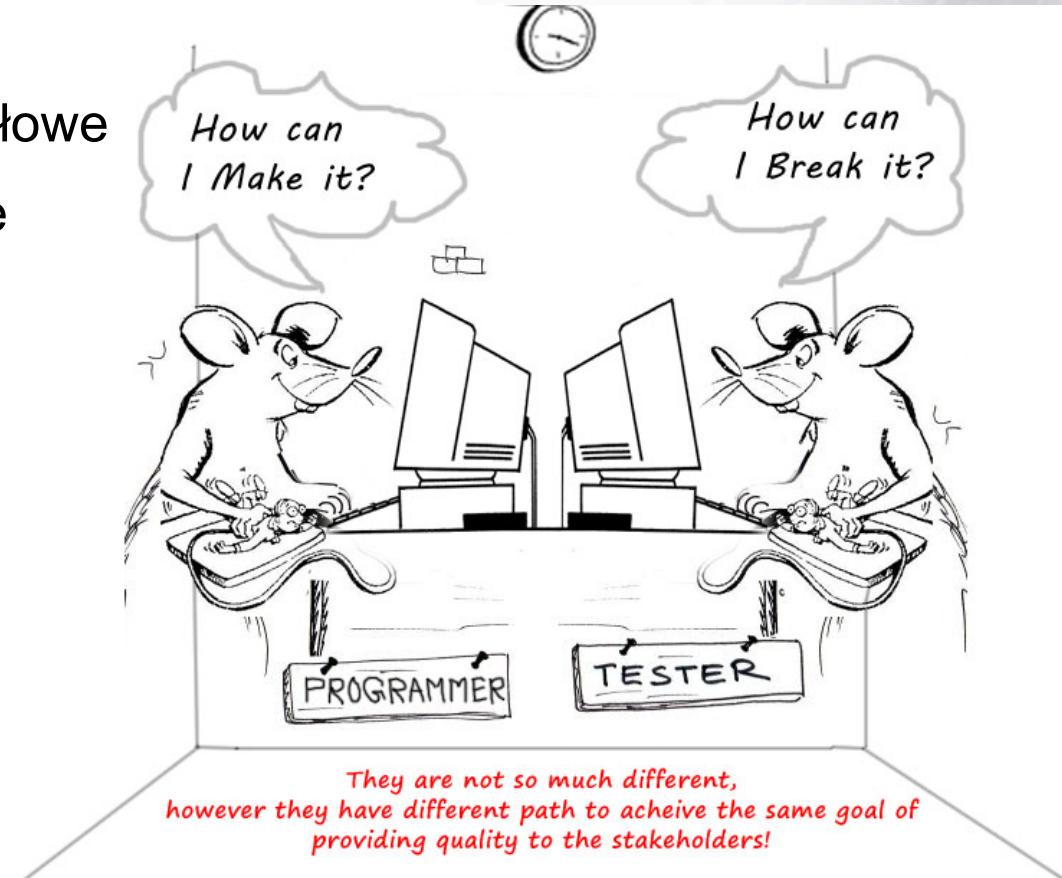
## Podział testów w zespole

- **Programiści:**

- Testy jednostkowe/modułowe
- Testy „małe” integracyjne

- **Testerzy:**

- Testy systemowe
- Tesy „duże” integracyjne
- Testy akceptacyjne



# Planowanie i organizacja

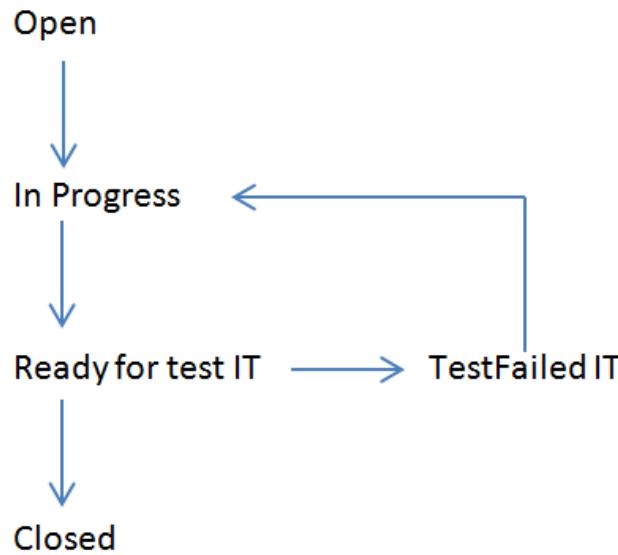
## Harmonogram testowania – Gantt chart

		man hours of tester [h]	man hours of test manager [h]	Avg Estimation [man-hours]	Confirmed Estimations [man-hours]	Remaining time [man-hours]	W4	1	2	3	4	5	6	7	W5	1	2	3	4	5	6	7	W6	1	2	3	4	5	6	7	W7	1	2	3	4	5	6	7	W			
SCRUM	Workshops															x	x			x	x			x	x			x	x			x	x			x	x					
	Sprint planning															x	x			x	x			x	x			x	x			x	x			x	x					
	Demo															x	x			x	x			x	x			x	x			x	x			x	x					
	Code Freeze															x	x			x	x			x	x			x	x			x	x			x	x					
Impact 4.01.20																																										
	Test Planning					0										x	x			x	x			x	x			x	x			x	x			x	x					
	Test Design					0										x	x			x	x			x	x			x	x			x	x			x	x					
	Test Execution															x	x			x	x			x	x			x	x			x	x			x	x					
	Regression tests on 4.00.51 (4.00.50 + Full changes) - online only	done		80		0										x	x			x	x			x	x			x	x			x	x			x	x					
	correction + Retests DVD bugs			60												x	x			x	x			x	x			x	x			x	x			x	x					
	specification analysis			70												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4193			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4193			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4194			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4194			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4195			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4195			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4196			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4196			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4197			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4197			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	Design TC - IMP-4198			8												x	x			x	x			x	x			x	x			x	x			x	x					
	system tests - IMP-4198			10		4										x	x			x	x			x	x			x	x			x	x			x	x					
	corrections + retests			4												x	x			x	x			x	x			x	x			x	x			x	x					
	performance tests			16		16										x	x			x	x			x	x			x	x			x	x			x	x					
	regression tests on online			80		24										x	x			x	x			x	x			x	x			x	x			x	x					
	regression tests on DVD	CANCELED		24		24										x	x			x	x			x	x			x	x			x	x			x	x					
	regression tests on TT + online and DVD	CANCELED		32		32										x	x			x	x			x	x			x	x			x	x			x	x					
	acceptance tests															x	x			x	x			x	x			x	x			x	x			x	x					
	DEPLOY TO PRODUCTION															x	x			x	x			x	x			x	x			x	x			x	x					
	TEST ESTIMATION			494		129										x	x			x	x			x	x			x	x			x	x			x	x					

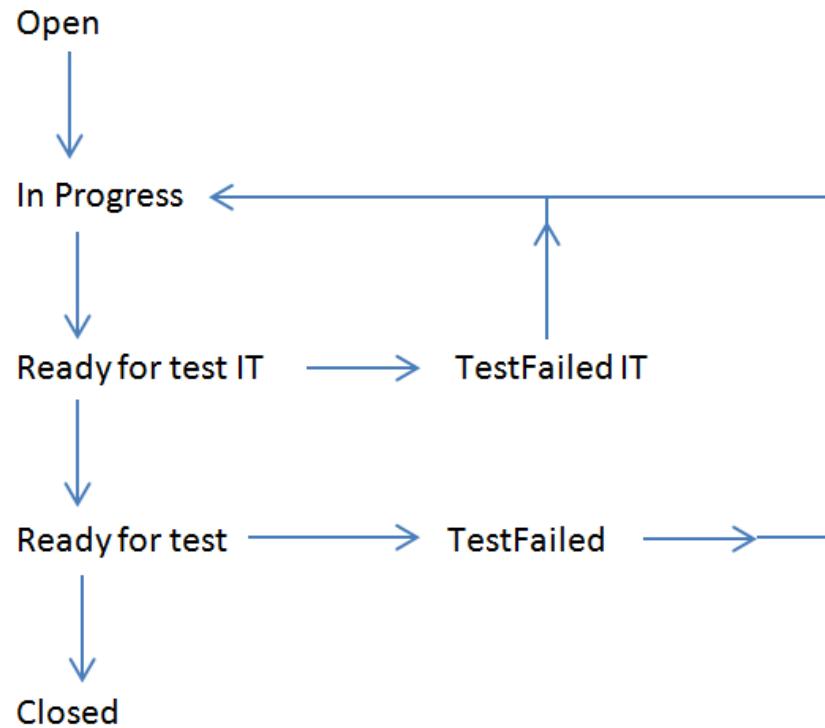
# Planowanie i organizacja

## Cykl życia błędu

### IT Bug Report



### Bug Report



# Planowanie i organizacja

## Zgłoszenie błędu

The screenshot shows a bug tracking system interface with several fields and their corresponding labels in red:

- Unikalny identyfikator**: Points to the URL "Impact / IMP-4814 Edit Favorite list / IMP-6216".
- Nazwa błędu**: Points to the error message "Favorite list is not updated after removal".
- Autor**: Points to the status "READY FOR IT TEST".
- Data zgłoszenia**: Points to the creation date "01/Dec/15 1:51 PM".
- Opis środowiska**: Points to the environment details.
- Wersja oprogramowania**: Points to the software version "4.02.00".
- Wynik testów**: Points to the test results section.
- Priorytet naprawy**: Points to the fix priority section.
- Severity**: Points to the severity level.

**Details** section (left side):

- Type: Sub-Bug
- Priority: Major
- Affects Version/s: 4.02.00
- Component/s: None
- Labels: None
- Environment: env: http://impacttest.got.volvo.net/impact3/application
- WBS: one item
- Failed in release: 4.02.00.28 Build 2015-11-30T10:10:07Z
- Corrected in release: 4.02.00.40
- severity: Severe
- Sprint: IMPACT 01

**Description** section:

Preconditions:  
You have a few favorites in Parts tab (you may used the file attached to import these)

Repro:

0. Clean browser's cache
1. Login to Impact
2. Go to Menu -> Favorites -> Edit -> Parts
3. Quickly remove all lists EXCEPT the last one
4. Close the modal window by clicking 'X' in top right corner
5. Go to Menu -> Favorites -> Edit -> Parts

Actual result: the last list appears in the window instead of the first one

Expected result: only one list appears in the window

**Attachments** section:

- err2.avi

**Activity** section:

- All
- Comments
- Work Log
- History
- Activity
- Transitions
- Subversion

# Planowanie i organizacja

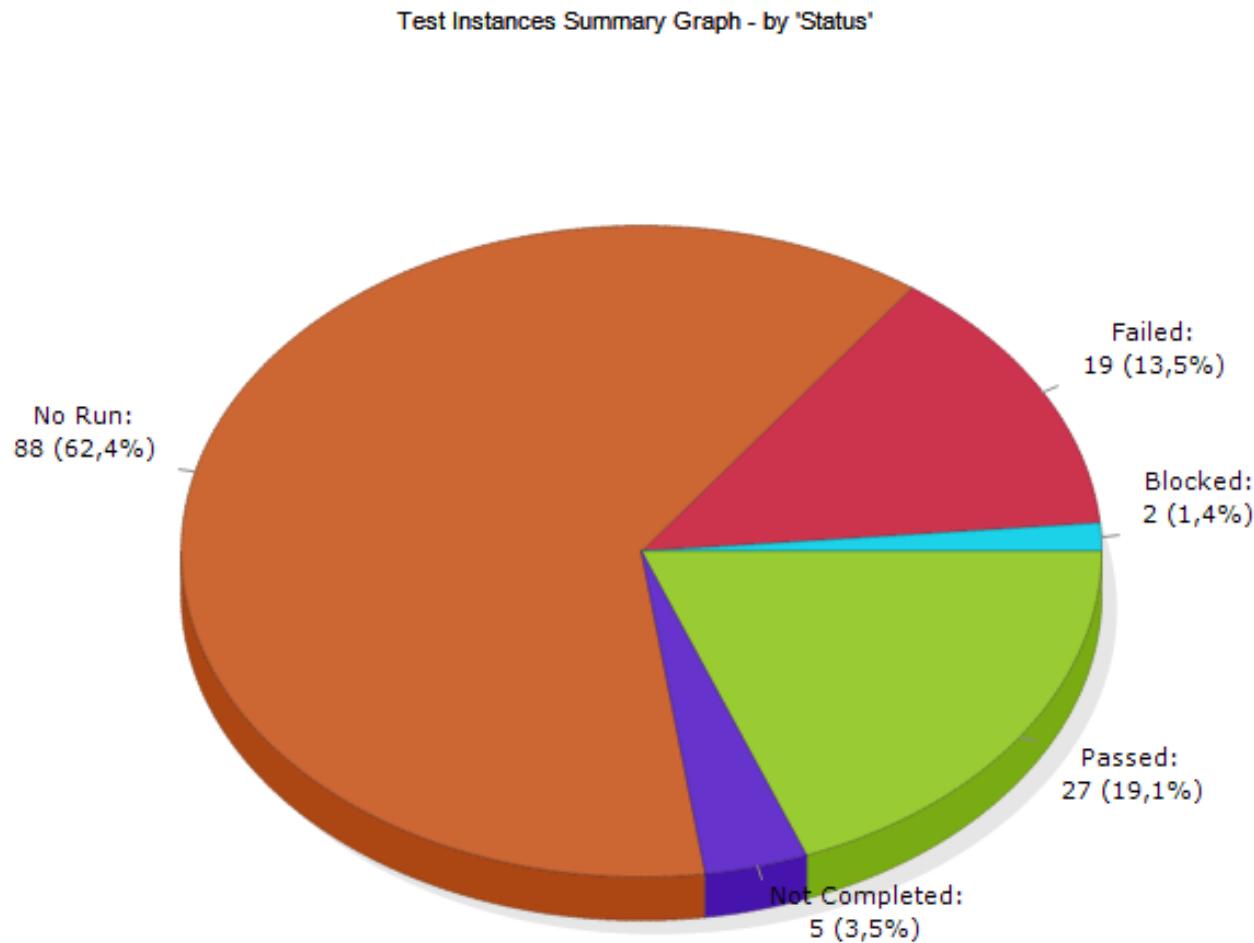
## Monitorowanie i nadzorowanie testów

- Monitorowanie jako część zaplanowanych przypadków testowych została:
  - Przygotowana
  - Wykonana
- Monitorowanie przebiegu wykonania testów:
  - Liczba wykonanych i niewykonanych przypadków testowych
  - Liczba przypadków testowych które „przeszły” i „nie przeszły”
- Statystyki o błędach
  - Liczba znalezionych i naprawionych defektów
  - Czas „Życia błędu”



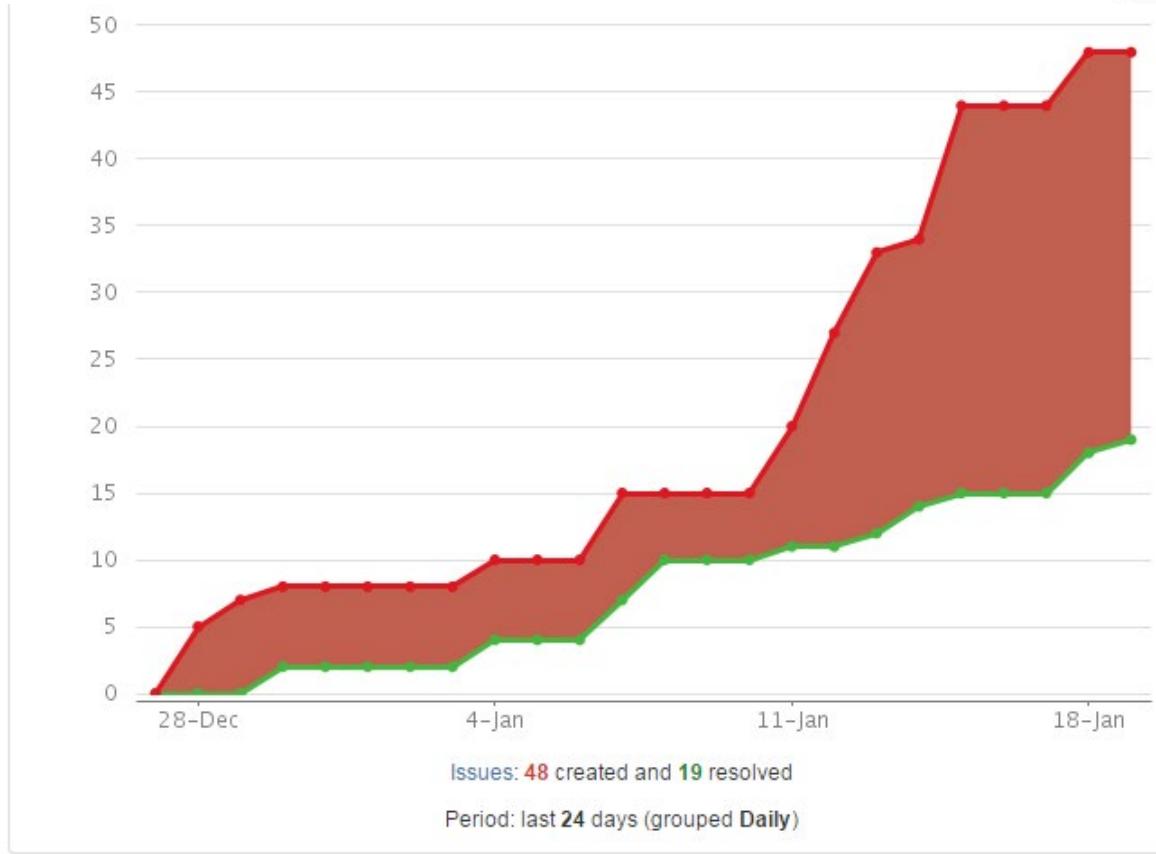
# Planowanie i organizacja

## Przebieg wykonania testów



# Planowanie i organizacja

## Liczba znalezionych, naprawionych błędów



# Planowanie i organizacja

## Skład zespołu testowego

- **Lider testów**
- **Testerzy:**
  - Analitycy testowi
  - Programiści testów automatycznych
  - Wykonujący testy
- **Inne osoby:**
  - Programista – testy modułowe/integracyjne
  - Specjalista biznesowy – testy akceptacyjne
  - Zewnętrzne zasoby – testy wydajnościowe



# Planowanie i organizacja

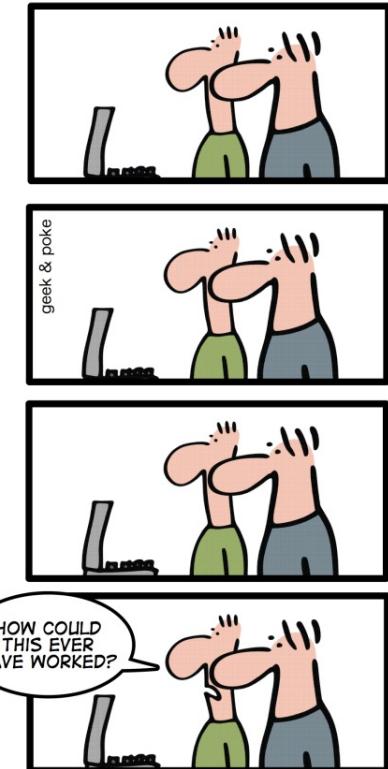
## Raport z testów

- **Lista błędów:**
  - Według poziomu krytyczności
  - Według wykonywania przez testerów
- **Raporty czasowe**



GEEK&POKE LOOKS BEHIND THE SCENES  
OF CODERS

TODAY: ANALYSIS OF A PRODUCTION BUG



# Agenda

- Volvo Group i Volvo IT
- Proces i cykle testowe
- Techniki testowe
- Strategia i struktura testów
- Planowanie i organizacja
- **Metryki i narzędzia testowe**



# Metryki testowe



Typy metryk	Co mierzymy	Przykład
Metryki projektowe	Progres w odniesieniu do kryteriów końcowych	Liczba zaplanowanych i wykonanych warunków testowych
Metryki produktowe	Atrybuty produktu	Gęstość defektów
Metryki procesowe	Możliwości procesu testowego lub deweloperskiego	Procentowa ilość defektów wykrytych w procesie testowania
Metryki ludzkie	Zdolności jednostek lub grup	Ilość utworzonych przypadków testowych w danym czasie

# Narzędzia testowe

- **Przykłady narzędzi testowych wspierających testowanie**

- Jira



- HP Quality Center / HP ALM



# Jira – narzędzie do śledzenia defektów

VOSPCQ / VOSPCQ-5919 3 o Return

Possible to save [Periodic stop times] value when start or end date is empty.

Edit Comment Assign More ▾ Fixed Test Failed IT Close

**Details**

Type:	IT Bug Report	Status:	READY FOR TEST IT
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	3.42.0	Fix Version/s:	3.42.0
Component/s:	Internal bugs		
Labels:	new_vosp_gui		
Environment:	3.42.0 (2015-12-28 10:20) on : TEST, Report Bug, DB: VOSPCQ_TEST_R42, Market: FI, Dealer: 1000, Language: GB, Last calculation: null User: A205690		

**People**

Assignee:	Arkadiusz Zieba
Reporter:	Arkadiusz Zieba
Votes:	0 Vote for this issue
Watchers:	2 Start watching this issue

**Dates**

Not Specified 0m 3h 45m 1/Dec/15

**Description**

It is possible to add periodic stop times without start date.

**Steps to reproduce:**

1. Go to VOSP -> New VOSP and click Edit button in order to create a new [Periodic stop times]
1. Add new periodic stop times.
2. Click Edit once again and remove Start date. Click Save button

**Result:**

Both fields: start and end date are mandatory, but there is possible to create periodic stop time without Start or End date.

**Attachments**



# Jira – filtrowanie defektów

R42 Ready for test IT Save as Details ★

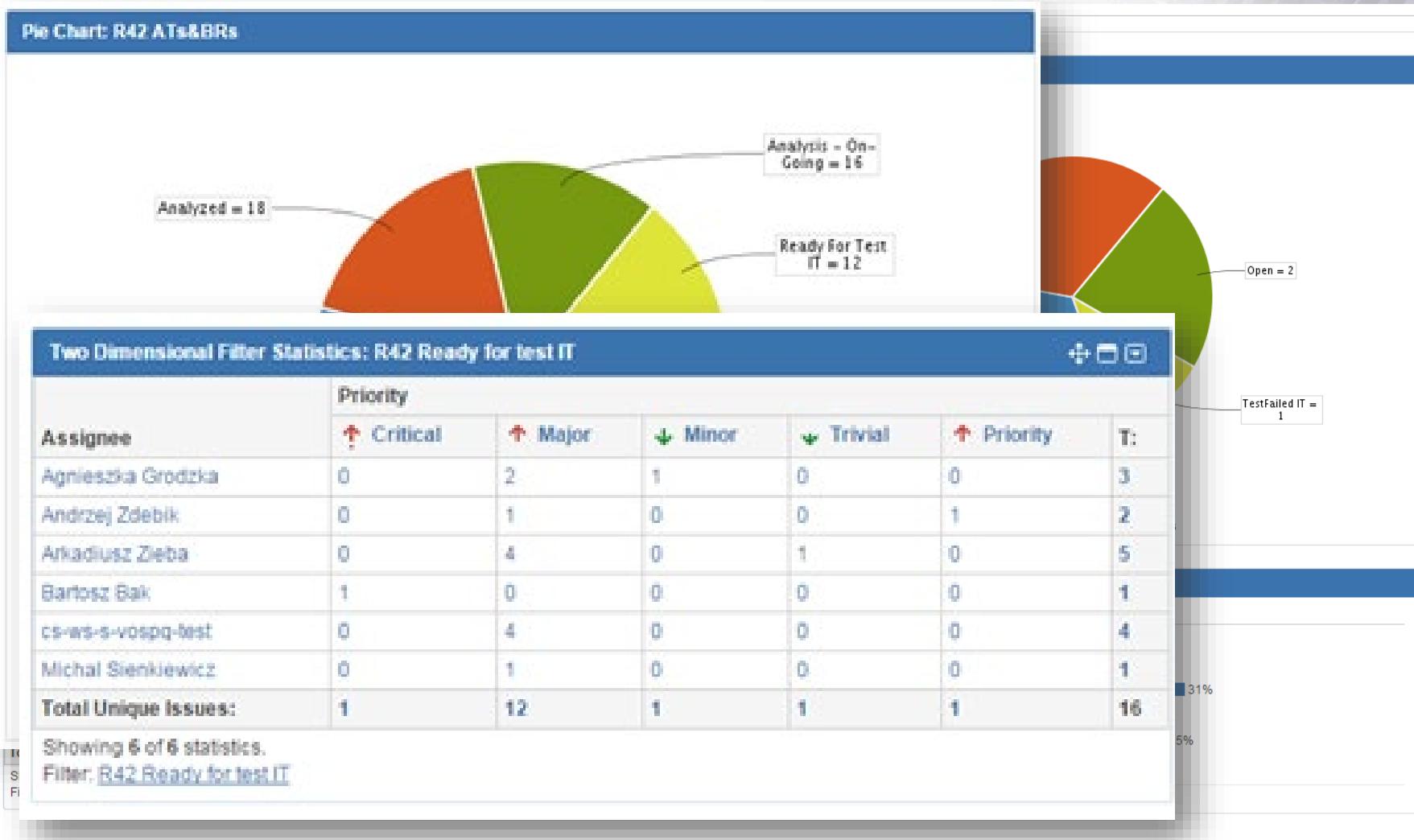
Share Export Bas

project = VOSPCQ AND status = "Ready For Test IT" AND fixVersion = 3.42.0

T	P	Key	Summary	Reporter	Status	Fix Version/s	Re	
<input type="checkbox"/>	↑	VOSPCQ-5929	Error while preparing data for DPP	VOSP	Arkadiusz Zieba	Adam Czechowski	READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-5928	ISE occurs when url is incorrect	Technical	Arkadiusz Zieba	Arkadiusz Zieba	READY FOR TEST IT	3.42.0
<input type="checkbox"/>	T	VOSPCQ-5929	Error while preparing data for DPP				READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-5928	ISE occurs when url is incorrect				READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-5919	Possible to save [Periodic stop times] value when start or end date is empty.				READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-5909	Optimization gap for rolling				READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-5904	Messages are saved but not processed				READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-3459	VDA button for MV is hidden when creating calculation from schedule	CQ	Michał Sienkiewicz	Witold Bieniek	READY FOR TEST IT	3.42.0
<input type="checkbox"/>	↑	VOSPCQ-2557	Validate max frozen price deviation when changing from non-frozen to frozen	CQ	Agnieszka Grodzka	Janis Upmanis	READY FOR TEST IT	3.42.0

1-13 of 13

# Jira – statystyki testowe



# HP Quality Center - baza przypadków testowych

The screenshot shows the HP Quality Center web interface. The left sidebar has a tree navigation with 'Integration Tests W51' expanded, showing 'AMT', 'MTS', and 'Belley'. 'Belley' is selected and expanded, listing items from 'BEL-1' to 'BEL-120'. The main content area displays a table for 'BEL-1'. The table has columns for 'Description', 'Expected Result', and 'Module'. The 'Description' row contains 'Update class with VCE\_MTS'. The 'Expected Result' row contains 'std PDM work moved to preparations'. The 'Module' row contains 'FM'. Below the table is a large, empty text area. The bottom right corner shows 'Server Time: 2015-12-22 3:45'.

Description	Expected Result	Module
Update class with VCE_MTS	std PDM work moved to preparations	FM

# HP Quality Center - śledzenie rezultatów testów

Application Lifecycle Management

Execution Grid

Name	Test: Test Name	Type	Status
HBEL-1	BEL-1	MANUAL	Passed
HBEL-10	BEL-10	MANUAL	Passed
HBEL-11	BEL-11	MANUAL	Passed
HBEL-12	BEL-12	MANUAL	Passed
HBEL-13	BEL-13	MANUAL	Passed
HBEL-14	BEL-14	MANUAL	Passed
HBEL-15	BEL-15	MANUAL	Passed
HBEL-16	BEL-16	MANUAL	Passed
HBEL-17	BEL-17	MANUAL	Failed
HBEL-18	BEL-18	MANUAL	Failed
HBEL-19	BEL-19	MANUAL	Passed
HBEL-2	BEL-2	MANUAL	Passed
HBEL-20	BEL-20	MANUAL	Passed
HBEL-21	BEL-21	MANUAL	Passed
HBEL-22	BEL-22	MANUAL	No Run
HBEL-23	BEL-23	MANUAL	No Run
HBEL-24	BEL-24	MANUAL	No Run
HBEL-25	BEL-25	MANUAL	Passed
HBEL-26	BEL-26	MANUAL	No Run
HBEL-27	BEL-27	MANUAL	Passed
HBEL-28	BEL-28	MANUAL	Passed
HBEL-29	BEL-29	MANUAL	Passed
HBDI-1	BDI-1	MANUAL	Passed

Logout

Pinned items

Test Sets Edit View Tests Favorites Analysis

Dashboard Analysis View Dashboard View Management Requirements Testing Test Resources Business Com... Test Plan Test Lab Test Runs Defects

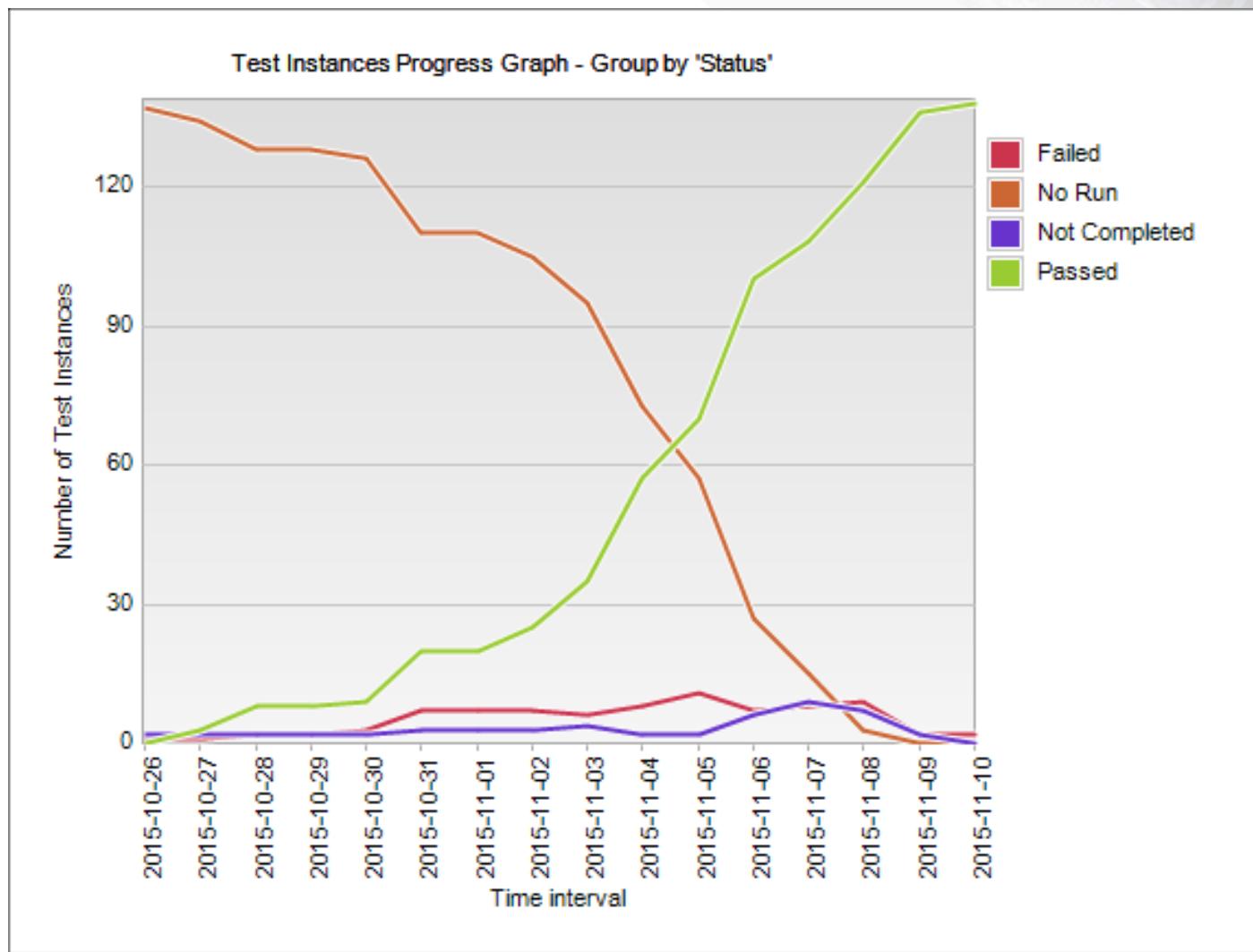
No Filter Defined

- Root
  - Unattached
  - 1.IT Integration Test
  - 2.E2E flow for MTS
  - 3.Regression Test
  - 4.UAT
  - AMT UAT test script\_V0.5
  - AMT UAT test script\_V1.0
  - Integration Tests W51
    - AMT
    - MTS
      - Belley
      - Konz
      - Shippensburg
  - Test
  - TEST-AZ

Help ?

2015-12-22 3:44

# HP Quality Center - progres testów



# HP Qu



## Defect report by sta

Closed

Duplicated

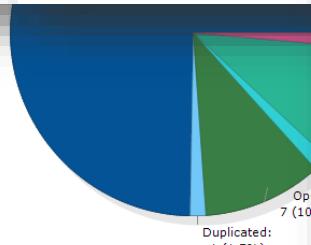
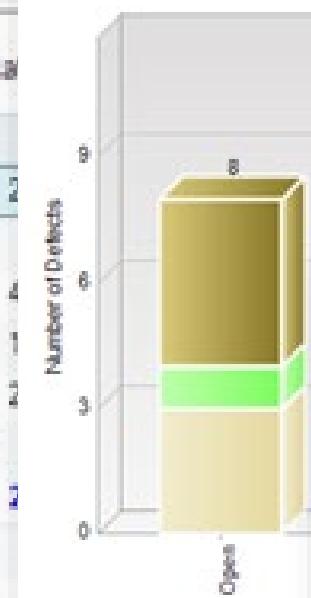
Open

Pending VCE

Ready to test by...

Ready to test by...

<Total>

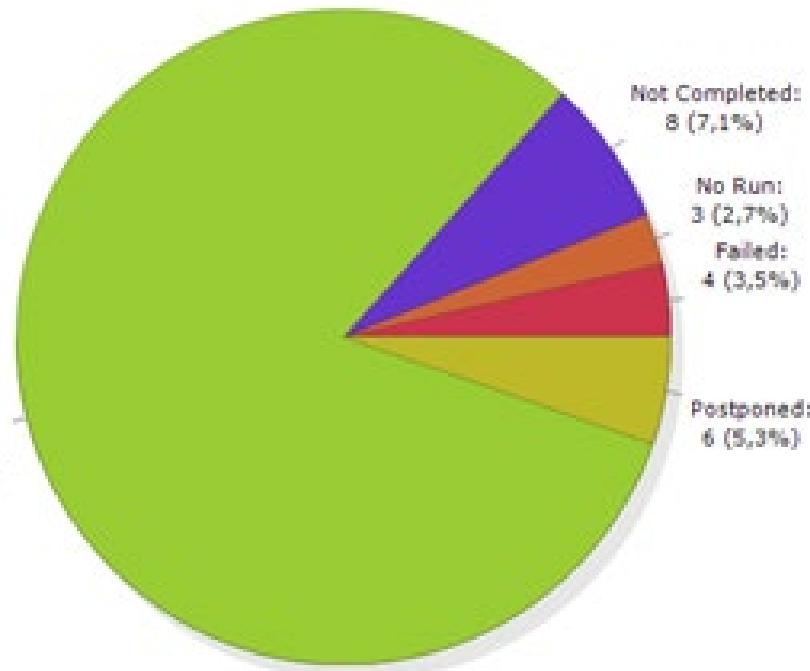


Last Generated (Local Time): 2015-12-22 15:41:55

Defect report by status (not closed)



General status AMT





# PROGRAM PRAKTYK LETNICH

lipiec – wrzesień 2018



Praktyki są płatne i skierowane do studentów III, IV i V roku kierunków informatycznych, ekonomicznych lub pokrewnych, zainteresowanych obszarem IT, znających dobrze język angielski.

Obszary praktyk:

- Microsoft .NET oraz SharePoint
- Java
- IBM iSeries
- SAP
- Business Intelligence
- Aplikacje mobilne (iOS, Android)
- Testowanie
- Wsparcie klienta

Rekrutacja do programu zaczyna się w kwietniu pod:

**[volvogroup.pl/kariera](http://volvogroup.pl/kariera)**



Pytania?



# Dziękujemy!

[agnieszka.grodzka@volvo.com](mailto:agnieszka.grodzka@volvo.com)  
[andrzej.zdebik@volvo.com](mailto:andrzej.zdebik@volvo.com)  
[www.volvo.com](http://www.volvo.com)