# Wprowadzenie do systemów informacyjnych

Projektowanie metodami „zwinnymi”

© UEK w Krakowie Ryszard Tadeusiewicz



Metody kaskadowe i spiralne używane do projektowania systemów bywają czasem nadmiernie sformalizowane, dlatego w ostatnich latach zaczęto lansować bardziej swobodną metodykę projektowania systemów informatycznych, nazywaną *„agile software development methods”*

W polskiej literaturze odpowiednie metody zwykło się określać jako „**zwinne**”.

**Zwinne** metodyki prowadzenia projektu informatycznego różnią się od tradycyjnych metod generalną ideą.

Podczas gry tradycyjne metody prowadzenia projektu w zamyśle sprzedają produkt - oprogramowanie, nowe techniki zarządzania sprzedają usługę – opracowanie oprogramowania.

## Generalne własności metod „zwinnych”:

- Wszystkie opisywane techniki zakładają ścisłą współpracę z użytkownikiem czy odbiorcą. Właściwie postuluje się włączenie użytkownika w proces projektowania oprogramowania (eXtreme Programming).
- Sprzedawana jest usługa tworzenia oprogramowania a nie gotowy produkt -oprogramowanie, tak więc użytkownik jest tym, kto podejmuje decyzje co i w jakiej kolejności będzie w projekcie wykonywane.
- Istotną wagę przywiązuje się do poprawnego szacowania kosztów prac, tak by inwestor/użytkownik mógł świadomie planować swe wydatki na rozwój oprogramowania.

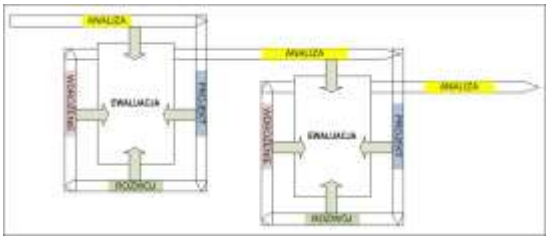
## Własności – ciąg dalszy

- Zobowiązuje się wytwórcę oprogramowania do tego, że każdym swym działaniem powinien udowadniać inwestorowi efektywne wykorzystanie czasu i powierzonych mu środków.
- Sprzedając usługę programowania rezygnuje się z zysków z ponownego użycia kodu i modeli analitycznych, bo prace odniesione są do niepowtarzalnego projektu. Przy takim założeniu rozległa dokumentacja projektowa staje się zbędnym kosztem obciążającym użytkownika i unika się jej.
- Uproszczenia dokumentacyjne wymuszają specyficzny sposób porozumiewania się z użytkownikiem. W trakcie tworzenia oprogramowania często (na bieżąco) zadaje się mu pytania i prośby o potwierdzenie dotyczące niewielkiego zakresu funkcjonalności. Stąd wynikają inkrementalny sposób dostarczania oprogramowania oraz krótkie iteracje implementacyjne (XP, Scrum).

## Własności - koniec

- Nie specyfikuje się formalnych punktów kontrolnych w projekcie - nie są one potrzebne, gdyż zakończenie każdej iteracji jest punktem kontrolnym samym w sobie. Z drugiej strony wprowadzenie sformalizowanych punktów kontrolnych nie we wszystkich technikach jest możliwe.
- Sekwencyjna realizacja wymagań użytkownika powoduje częste zmiany w architekturze systemu i konieczność przebudowy kodu. W nowych metodykach zadanie przebudowy kodu postrzega się nie jako wyjątek, lecz jako regułę.

## Zasadnicza idea metod zwinnych




## Najważniejsze kierunki innowacji wprowadzanych w systemach informacyjnych oparte są o wymagania:

- **integracji** systemów, danych i procesów,
- **unifikacji** funkcji cząstkowych systemów,
- zwiększania **dostępności** do bazy danych dla wszystkich komórek organizacyjnych,
- upowszechniania nowoczesnych sposobów **prezentacji** danych (wizualizacji) dla celów wspomagania ich analizy,
- doskonalenia procesów podejmowania **decyzji** i ich przekazywania,
- zmierzania do budowy **modułowej** i otwartości całego systemu,

## Dalsze wymagania dotyczące projektowanego systemu są następujące:

- zapewnienia **kompleksowego** charakteru funkcjonalnego całego systemu,
- stałego podnoszenia **zaawansowania** merytorycznego i technologicznego,
- zmierzania do **elastyczności** funkcjonalnej i strukturalnej,
- zapewnienia stałej **zgodności** ze zmieniającymi się elementami otoczenia systemowego, a zwłaszcza z aktualnym stanem prawnym, ewoluującym zgodnie z przyjętymi procedurami legislacyjnymi.

Ekonomiczne systemy informacyjne są projektowane i realizowane w taki sposób, aby dane przetwarzane przez ów system były bezpieczne i na każdym jego etapie chronione.

Dlatego też w jak największym stopniu musi być zapewniona poufność i integralność wszelkich danych posiadanych przez system, a dostępność do danych zawartych w systemie powinna być zgodna z przyjętą hierarchią haseł i przywilejów dostępu.

# Agile

## Deklaracja Agile:

***Manifesto for Agile Software Development***

*We are uncovering better ways of developing software by doing it and helping others do it.*
*Through this work we have come to value:*
***Individuals and interactions*** *over processes and tools*
***Working software*** *over comprehensive documentation*
***Customer collaboration*** *over contract negotiation*
***Responding to change*** *over following a plan*
*That is, while there is value in the items on the right, we value the items on the left more.*

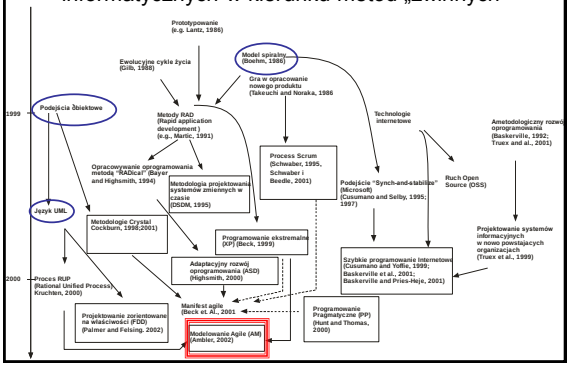
## Podstawowe składniki „manifestu” zwolenników „zwinnych” metod projektowania są dosyć oczywiste i łatwe do zaakceptowania

- ludzie, ich kontakty, zdolność do rozwiązywania problemów są ważniejsze niż sztywne procedury i narzędzia zarządzania,
- wynikiem projektu jest pracujące oprogramowanie a nie dokumentacja,
- z użytkownikiem się współpracuje a nie negocjuje kontrakt,
- ważniejsza jest umiejętność reagowania na zmieniające się warunki otoczenia niż podążanie za opracowanym na wstępie planem.

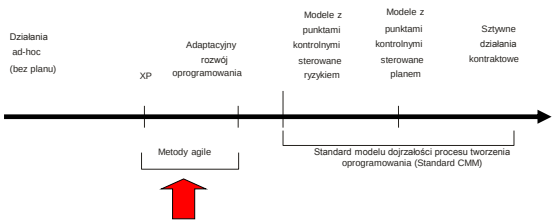
W myśl manifestowanych zasad metody „agile” nie są w istocie nowymi praktykami postępowania w projekcie, bo te są tradycyjne.

Nowością jest traktowanie człowieka jako nadrzędnego czynnika sukcesu.

## Rozwój metodyki projektowania systemów informatycznych w kierunku metod „zwinnych”




## Skala dojrzałości modeli tworzenia systemów informatycznych pokazuje, że metody „zwinne” można stosować głównie dla niezbyt dużych systemów

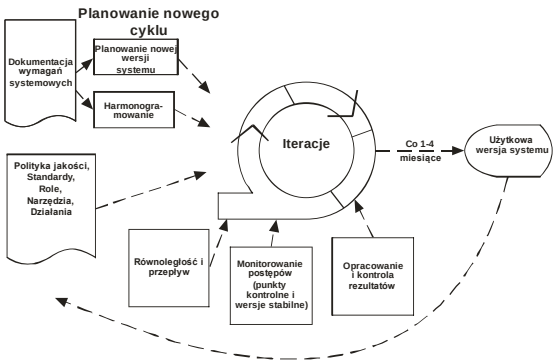



Wśród metod zwinnych obecnie można wymienić

- Metodyka Crystal (Crystal family),
- Projektowanie zorientowane na właściwości FDD (*Feature Driven Development*),
- Modelowanie zwinne (Agile Modeling),
- Programowanie ekstremalne (Extreme Programming),
- Adaptacyjny rozwój oprogramowania (Adaptive Software Development),
- Metodyka SCRUM (Scrum development),
- Prototypowanie (Prototyping methodology),
- Szybkie programowanie internetowe (Internet Speed Development),
- Pragmatyczne programowanie (Pragmatic Programming).

# Cristal

Pierwsza ze „zwinnych” metodologii, nazywana *Crystal*

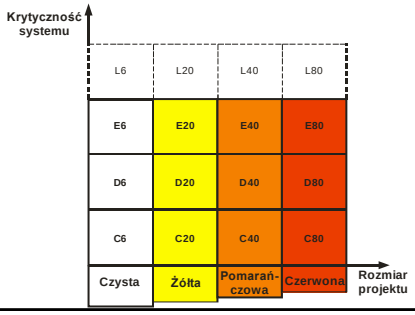



Metodyka *Cristal* ma odmiany w zależności od stopnia **krytyczności** projektu (kategoryzowanego poprzez litery **C, D, E, L** – objaśnione dalej) i w zależności od jego **rozmiaru**, mierzonego liczbą projektantów zaangażowanych w tworzenie projektu.

## Kategorie krytyczności projektowanego systemu:

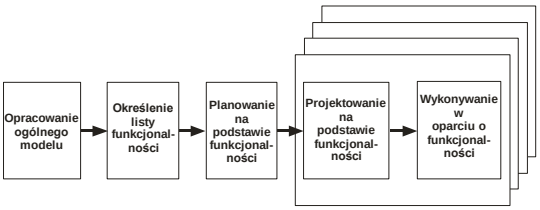
- **C** - **K**omfortowe (Comfort),
- **D** - **Z**arządzające **F**inansami (Discretionary Money),
- **E** - **F**inansowo **i**stotne (Essential Money)
- **L** - **K**rytyczne dla **Ż**ycia (Life Critical)

Na tej podstawie proponowana jest cała rodzina metod typu *Cristal*. Ich mapa podana jest niżej.

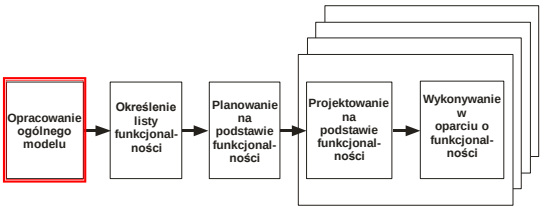



# FDD
# Projektowanie zorientowane na właściwości

Projektowanie zorientowane na właściwości (FDD-*Feature Driven Development*)




Projekt w myśl FDD składa się z pięciu sekwencyjnie następujących etapów, które będą kolejno opisane.



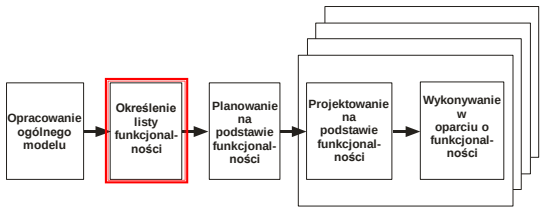

Założeniem leżącym u podstaw FDD jest **inkrementacyjne** opracowywanie poszczególnych funkcjonalności systemu (*features*).

Prace rozpoczynają się od określenia „ogólnego modelu systemu".

Zespół projektantów korzysta z opracowanych wcześniej wymagań systemowych i przypadków użycia.

Określana jest domena projektu i iteracyjnie dzielona na coraz to mniejsze znaczeniowo obszary.

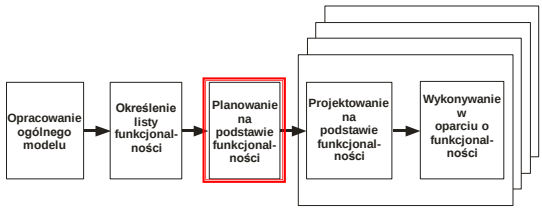
Każdy niepodzielny obszar znaczeniowy jest opracowywany przez przypisaną do niego grupę projektantów, w wyniku czego powstaje model szczegółowy, będący składnikiem całościowego modelu systemu.




W następnym etapie na podstawie specyfikacji wymagań systemowych oraz opracowanego modelu/modeli opracowywane są listy funkcjonalności.

Listy te mają charakter hierarchiczny i zawierają funkcjonalności główne, które rozpadają się na zestawy funkcjonalności w kolejnych hierarchiach.

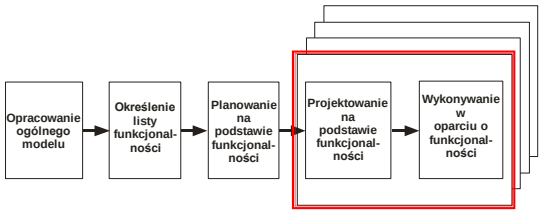
Listy te są przeglądane przez użytkowników i inwestorów w celu kontroli poprawności i kompletności.




Planowanie nakłada na kierownictwo projektu obowiązek opracowania długookresowego planu prac powiązanego z funkcjonalnościami, ich zależnościami i priorytetami.

Poszczególne elementy listy funkcjonalności przydzielane są zespołom a w ich ramach konkretnym programistom – opiekunom klas.

Klasa jest tu rozpatrywana w rozumieniu programowania obiektowego.




Kolejne fazy:

Projektowanie funkcjonalności
i
wykonanie funkcjonalności

wykonuje się iteracyjnie
i jeśli to możliwe równolegle.

Każda funkcjonalność znajduje swą realizację wewnątrz obiektu (klasy), do każdej klasy przyporządkowany jest programista dbający o jej spójność, poprawność i efektywność kodu.

Jest on **zarządzający obiektem** (klasą).

Zarządzający obiektami są formowani w małe zespoły (*feature teams*).

Sposób organizacji zespołów w przybliżeniu odzwierciedla hierarchię funkcjonalności.

Istotną postacią w zespole projektowym jest **zarządca konfiguracji** zajmujący się zagadnieniami kontroli wersji i identyfikacją każdej „historycznej” wersji kodu źródłowego.

# DSDM

Metodyka Projektowania Systemów Zmiennych w Czasie

Inną ważną metodologię projektowania zaproponowało brytyjskie konsorcjum DSDM.

Biorąc pod uwagę fakt, że zadania związane z projektowanym systemem **zawsze** podlegają zmianom opracowano Metodykę Projektowania Systemów Zmiennych w Czasie **DSDM** (*Dynamic Systems Development Methods*)

## Proces DSDM składa się z:

- inspekcji zastosowalności,
- badań biznesowych,
- iteracyjnego opracowania modelu funkcjonalnego (Functional model iteration),
- iteracyjnego projektowania i implementacji (Design and build iteration),
- wdrożenia.

**Inspekcja zastosowalności** wykonywana jest jednokrotnie na początku projektu po to aby potwierdzić zasadność stosowania metody DSDM.

W trakcie tej fazy wstępnie określa się ryzykowne punkty w projekcie, jeśli to niezbędne buduje się prototypy.

Rozległość prac w tej fazie jest ograniczona do kilku tygodni.

**Badania biznesowe** mają na celu wstępne rozpoznanie zagadnienia i identyfikację osób po stronie odbiorcy, które są kluczowe dla projektu lub jego części. Osoby te zostaną w późniejszym okresie włączone w proces opracowywania systemu.

Wyniki tej fazy to także wysokopoziomowy opis systemu (*Businnes Area Definition*), specyfikacja zakresu systemu, zarys architektury systemu (*System Arhitecture Definiction*) oraz Plan prototypowania (*Outline Prototyping Plan*).

B**udowa modelu funkcjonalnego** jest działaniem składającym się naprzemiennie z procesów analizy i budowy **prototypów**.

Wyniki prototypowania służą do poprawienia i uszczegółowienia modeli analitycznych.

Jeśli to możliwe prototypy są udoskonalane w taki sposób, by można było je włączyć do produktu końcowego.

Wynikiem tej fazy jest model funkcjonalny oraz kod prototypów.

Prototypowanie jest często traktowane jako forma testowania modelu funkcjonalnego.

## W każdej pętli iteracji tworzone są następujące dokumenty:

- lista funkcjonalności do opracowania wraz z ich priorytetami,
- uwagi i komentarze użytkowników na temat prac w aktualnym cyklu (*Functional prototyping review documents*),
- wymagania niefunkcjonalne,
- analiza ryzyka pod kątem dalszych prac.

**Projektowanie i implementacja**
to właściwa faza budowania systemu.

Wyniki prac nad modelem funkcjonalnym są przetwarzane w kod źródłowy właściwego produktu.

Prototypy powstałe w trakcie prac nad modelem funkcjonalnym mogą być w tej fazie adaptowane do kodu aplikacji.

Wynikiem tej fazy jest przetestowany produkt zawierający uzgodniony wcześniej zestaw funkcjonalności.

Zaletą metodyki **DSDM** jest to, że na każdym etapie projektowania i budowy systemu produkt jest oceniany przez twórców i użytkowników, a uwagi wynikające z oceny opracowywane są w ramach kolejnych iteracji.

DSDM literalnie wprowadza następujące praktyki:

- Aktywny udział użytkownika w procesie tworzenia oprogramowania;
- Zespoły DSDM muszą być uprawnione do podejmowania decyzji;
- Naciska się na częste dostarczanie nowych wersji oprogramowania;
- Nowe wersje są oceniane pod kątem odpowiedniości dla zastosowań biznesowych;
- Stosuje się iteracyjne i inkrementalne podejście do tworzenia oprogramowania;
- Prowadzi się kontrolę wersji tak by każda zmiana była odwracalna;
- Wymagania systemowe są określane zgrubnie i są uszczegóławiane samym procesem DSDM;
- Testowanie jest integralną częścią wszystkich faz w projekcie.

# Programowanie ekstremalne

**Programowanie ekstremalne** (*eXtreme Programming*) powstało jako próba zaradzenia problemom związanym z długimi cyklami dostarczania oprogramowania i spadkiem zainteresowania inwestora zadaniem.

XP charakteryzują:
ewolucyjne podejście do projektowania i programowania oraz <u>ekstremalnie ścisła współpraca z odbiorcą</u>.

Zasadą są ekstremalnie krótkie iteracje w dostarczaniu kolejnych wersji oprogramowania prowadzące do szybkich odpowiedzi użytkownika.

Przy wytwarzaniu oprogramowania stosuje się programowanie w parach, ustawiczną przebudowę (*refactoring*) kodu źródłowego, ustawiczną integrację i testowanie połączonych modułów.

Hasła towarzyszące projektowaniu XP to:

- gra w planowanie (planning game),
- szybkie iteracje,
- porozumienie pomiędzy użytkownikiem i programistami za pomocą metafor,
- prostota kodu (brzytwa Ockhama),
- ustawiczne testowanie i integracja modułów,
- programowanie w parach,
- kolektywne posiadanie kodu,
- unikanie nadgodzin,
- komunikacja pomiędzy programistami poprzez kod źródłowy
- konstrukcja kodu źródłowego wedle zasad akceptowanych i przestrzeganych w całym zespole.

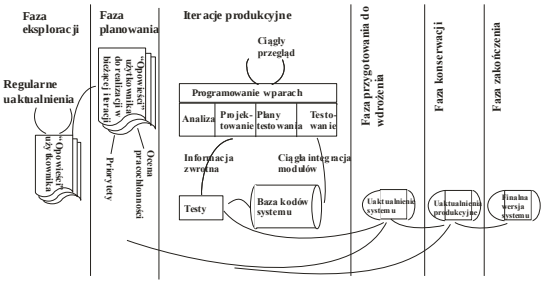
Jedna z zasad XP głosi, że nie ma uniwersalnej metody prowadzenia projektu informatycznego.

Wobec tego praktyki XP powinny być przystosowywane do aktualnych potrzeb i specyfiki projektu.

W cyklu życia projektu XP wyróżnia się fazy:

- eksploracji,
- planowania,
- iteracji wykonawczych,
- przygotowania do produkcji,
- utrzymania w ruchu,
- zakończenia projektu.

Projektowanie (i programowanie) ekstremalne




## W fazie eksploracji

zespół projektowy zapoznaje się z tematem prac i pozyskuje podstawowe informacje od użytkownika.

Użytkownik przedstawia sposób użytkowania systemu poprzez **opowiadania** („*story*”), na podstawie których budowany jest zarys architektury systemu oraz budowana jest lista funkcjonalności.

W tym czasie projektanci testują wybraną technologię tworząc niezbędne prototypy oraz zapoznają się z używanymi narzędziami.

## Faza planowania.

Opowiadania przedstawione przez użytkownika są analizowane i nadawane są im priorytety.

W porozumieniu z użytkownikiem zestawiana jest lista funkcjonalności, które mają być opracowane.

Programiści oceniają czas realizacji zadań i ustalany jest harmonogram prac i termin zakończenia prac.

## Faza iteracji wykonawczych.

Składa się z jedno lub kilkutygodniowych mini cykli implementujących kolejne właściwości systemu.

Wykonywane są działania analityczne, projektowe, kodowanie i testowanie.

Na końcu każdego mini cyklu wykonywane są testy oprogramowania zaplanowane przez użytkownika.

### Faza przygotowania do produkcji.

W tej fazie system zawierający uzgodnioną porcję funkcjonalności jest przygotowywany do wdrożenia.

Pojawiające się uwagi użytkownika są implementowane lub przeznaczane do implementacji w następnej wersji oprogramowania.

Wykonywane są dodatkowe gruntowne testy.

## Faza konserwacji.

Użytkownik jest wyposażony w działającą wersję oprogramowania, która wymaga opieki i nadzoru.

Zespół projektowy w tym samym czasie wykonuje kolejną wersję oprogramowania.

W trakcie pracy z oprogramowaniem odbiorca formułuje kolejne postulaty dla zespołu projektowego.

Wejście fazę produkcji często pociąga za sobą zmiany w zespołach projektantów i wzrost zatrudnienia.

Wysiłek poświęcany na utrzymanie w ruchu wersji produkcyjnej wpływa na zmniejszenie prędkości opracowywania nowej wersji oprogramowania.

### Zakończenie projektu.

Gdy użytkownik nie jest już zainteresowany dodawaniem funkcjonalności do oprogramowania, tempo współpracy z użytkownikiem spada, formułowane wnioski o rozszerzenie funkcjonalności mają charakter drugorzędny i często nie są wdrażane z powodów ekonomicznych.

W tej fazie zespół projektowy podejmuje decyzję o zakończeniu projektu, blokuje zmiany w architekturze systemu i kodzie źródłowym, opracowuje dokumentację systemu i projektu, ostateczne wersje instrukcji użytkownika oraz instrukcje konserwacji.

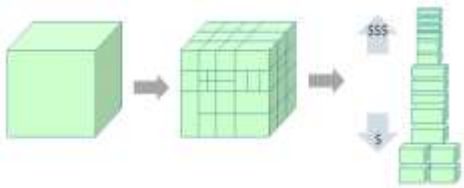
Jedną z najczęściej stosowanych technik „zwinnego” projektowania jest **Metodyka Scrum**

Istotą metody **Scrum** jest adaptacyjny, samoorganizujący się proces wytwarzania oprogramowania.

Nazwę zapożyczono ze strategii gry w rugby.

## Istotę metodyki Scrum można ująć w następujących punktach:

- Podzielenie organizacji na małe, interdyscyplinarne, samoorganizujące się zespoły.
- Podzielenie planowanej pracy na małe, dobrze zdefiniowane elementy.
- Określenie kosztów tych elementów oraz uporządkowanie ich ze względu na priorytet

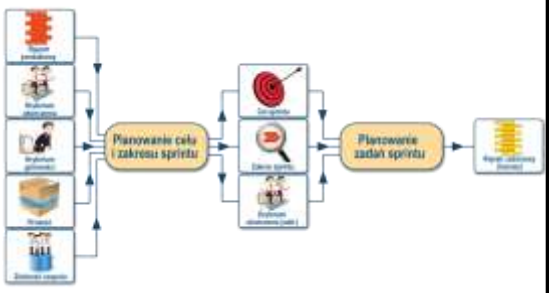



## Uniwersalne zasady Scrum

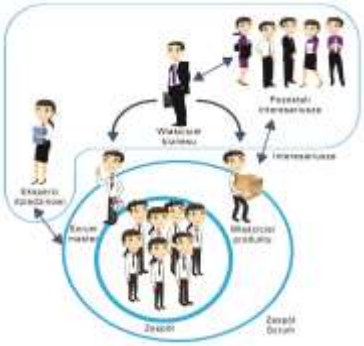
## Horyzonty planowania Scrum

Skupmy uwagę na horyzoncie jakim jest sprint

## Schemat procesów planowania sprintu




## Role w metodyce Scrum



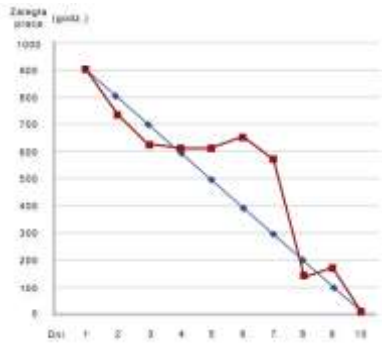
## Struktura czasowa sprintu dwutygodniowego

## Schemat przebiegu sprintu

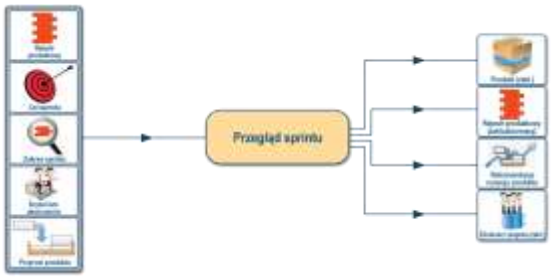
## Schemat procesu rozwoju produktu.

## Schemat procesów kontroli i adaptacji w części wykonawczej sprintu

## Wykres wypalania sprintu




## Schemat procesu przeglądu sprintu




## Schemat procesów retrospektywy sprintu i realizacji rekomendacji retrospektywy

## Szczegółowe zalecenia Scrum

Podzielenie czasu na krótkie iteracje o stałej długości, dążenie do posiadania działającego kodu na końcu każdej iteracji.

Optymalizowanie planu wytwarzania w kooperacji z klientem w oparciu o produkty wykonane w każdej iteracji.

Optymalizacja procesu podstawie doświadczeń każdej iteracji.

Scrum jest w istocie techniką zarządzania projektem informatycznym, utworzoną na podstawie doświadczeń w realnych projektach.

Koncentrując się na zadaniach zarządzania pozostawia wolny wybór w wyborze technik prowadzenia prac programistycznych.

Zakłada jednakże ewolucyjny styl tworzenia oprogramowania.

Podstawę adaptacyjności w Scrum stanowi założenie, że rozwój oprogramowania zachodzi w niestałych warunkach, na które składają się: nieprzewidywalne zmiany w wymaganiach, terminach, zasobach oraz dostępnych technologiach, wobec czego sam proces tworzenia oprogramowania jest złożony i nieprzewidywalny.

Scrum zastosowane w praktyce, jak twierdzą twórcy, jest w stanie usprawnić stosowane metody produkcji oprogramowania, bo przewiduje częste

działania zarządcze

skupiające się na identyfikowaniu problemów i przeszkód w pracach inżynieryjnych.

Proces Scrum podzielony jest na trzy główne etapy:

- rozpoczęcie gry (pregame),
- faza produkcji (development phase),
- gra na zakończenie (postgame).

**Faza rozpoczęcia** obejmuje czynności planowania i opracowania zarysu architektury systemu (*Architecture high level design*).

W trakcie tej fazy wszystkie znane wymagania są spisywane i opracowywana jest lista wymagań (*Product backlog list*).

Lista ta jest otwarta, a zadania do realizacji dopisywane są do niej w trakcie całego procesu tworzenia oprogramowania.

Źródłem wymagań są przede wszystkim użytkownicy, ale również dział marketingu i sprzedaży, dział obsługi klienta oraz sam zespół projektantów-programistów.

Wymaganiom zestawionym na liście przypisywane są priorytety.

Na podstawie listy opracowywana jest architektura systemu.

Każdorazowo, gdy do listy dopisywane są nowe wymagania, są one rozpatrywane w ramach specjalnego spotkania (*Design Review Meeting*).

Rozpatrywane są również zmiany w architekturze systemu wynikłe z wprowadzenia nowych wymagań.

Finalnie, w ramach oddzielnego spotkania, tworzony jest podzbiór listy wymagań.

Zawarte tam wymagania przeznaczone są do realizacji w ramach aktualnej iteracji (*sprint backlog list*).

Każdy cykl to w istocie podprojekt kaskadowy składający się z opracowania wymagań, analizy, projektowania, kodowania i wdrożenia trwający nie dłużej niż 30 dni.

**Czynności zarządcze w fazie produkcji zasadzają się na spotkaniach organizacyjnych.**

- Rozpoczęcie prac związane jest ze Spotkaniem Planowania Cyklu (Sprint planning meeting),
- Zakończenie prac z nieformalnym Spotkaniem Przeglądowym (Scrum Review meeting).
- Są również Codzienne Spotkania Zespołu projektantów i programistów (Daily Scrum meeting).

**Spotkanie Planowania Cyklu**

(*Sprint planning meeting*) organizowane jest przez zarządcę procesu dwukrotnie.

W pierwszym spotkaniu biorą udział użytkownicy, nabywcy, zarząd i zespół projektantów. Ustala się cele i priorytety właśnie rozpoczynającej się iteracji. Wymagania wpisuje się we wspomnianą wyżej listę (*Sprint product backlog*).

W drugim spotkaniu biorą udział jedynie wykonawcy i Zarządca Scrum, którzy ustalają sposób przeprowadzenia prac przy implementacji wymagań.

**Codzienne Spotkania Scrum**

(*Daily Scrum meeting*) są krótkie, najwyżej 15 minut, mają na celu motywowanie personelu oraz śledzenie postępów prac.

W trakcie spotkania omawiane są problemy oraz planowane są posunięcia z nich wynikające.

W spotkaniach uczestniczy zespół projektantów i programistów oraz zarządca Scrum.

**Spotkanie podsumowujące**

(*Scrum Review Meeting*) odbywa się w ostatni dzień trwania iteracji produkcyjnej (iteracja nie trwa więcej niż 30 dni).

Omawiane są na nim postępy prac oraz formułowane są wnioski, nowe wpisy do listy wymagań lub postulowane są generalne zmiany w architekturze systemu.

## Faza zakończenia projektu

rozpoczyna się wraz z ustaleniem pomiędzy użytkownikiem a projektantami, że wymagania są zrealizowane (lista wymagań jest pusta).

System jest przygotowany do instalacji.

W tej fazie wykonywana jest ostateczna integracja modułów i testowanie, a także przygotowuje się dokumentację.

**Scrum** wprowadza interesujące narzędzie zarządcze jest nim omawiana już lista wymagań (produkt backlog list).

Opisuje ona wszystko to, co powinno się znaleźć w ostatecznej wersji oprogramowania (wedle aktualnej wiedzy).

W ten sposób lista wymagań opisuje wszystko, co należy zrobić w projekcie.

Lista zwykle zawiera właściwości, funkcje, usterki, defekty, żądania rozszerzeń i żądania uaktualnień technologicznych.

Do zarządzania listą przeznaczony jest pracownik - **Zarządca Projektu**.

On trzyma pieczę nad dodawaniem nowych pozycji do listy, jak i usuwaniem pozycji gdy są zrealizowane.

W pragmatyce rozwoju oprogramowania „open source” taka lista nosi nazwę „to do list”.

Na diagramie przebiegu projektu nie przedstawiono jednego istotnego procesu biegnącego niezależnie od procesów wytwórczych.

Jest nim estymacja pracochłonności.

W trakcie całego projektu równolegle z pracami projektowymi i implementacyjnymi trwa proces oceny pracochłonności postulatów zawartych w liście wymagań.

W początkowych fazach projektu oceny te są zgrubne, lecz w miarę gromadzenia informacji z postępu prac implementacyjnych stają się coraz bardziej dokładne.

Proces estymacji pracochłonności polega na gromadzeniu informacji statystycznych o przebiegu projektu i wyznaczaniu kosztu prac na ich podstawie.

Estymacja pracochłonności nie bierze pod uwagę dużych zmian w architekturze systemu lub użytkowanej technologii.

W przypadku projektu prowadzonego metodą Scrum od początku zaleca się by użytkownik wraz z zespołem projektantów spędził kilkanaście dni nad opracowaniem listy wymagań.

Muszą się w niej znaleźć zapisy dotyczące zarówno wymagań biznesowych jak i technologicznych.

Celem nadrzędnym pierwszej iteracji produkcyjnej jest pokazanie użytkownikowi jakiegoś fragmentu funkcjonalności systemu zaimplementowanego w ramach wybranej technologii.

Należy przewidzieć dużą ilość pracy przy pierwszej iteracji, bo dochodzą tu prace nad opracowaniem szkieletu systemu, do którego będą dodawane funkcjonalności w ramach kolejnych iteracji.

Pierwsza iteracja produkcyjna różni się od kolejnych również z tego powodu, że na jej listę wymagań wpisane są takie zadania jak: zapoznanie się z technikami Scrum, organizacje zespołów Scrum, rozdział ról w projekcie.

Dalsze iteracje są prostsze i szybsze.

# Analiza i projektowanie systemów informacyjnych

Projektowanie metodami „zwinnymi”

© UEK w Krakowie Ryszard Tadeusiewicz