

Dr Katarzyna Grzesiak-Kopeć

Inżynieria oprogramowania



6. Metodyki zwinne



Plan wykładu

- Tworzenie oprogramowania
- Najlepsze praktyki IO
- Inżynieria wymagań
- Technologia obiektowa i język UML
- Techniki IO
- Metodyki zwinne
- Refaktoryzacja
- Mierzenie oprogramowania
- Jakość oprogramowania
- Programowanie strukturalne
- Modelowanie analityczne
- Wprowadzenie do testowania

Agile Software Development

- Agile
 - Zwinny, zręczny, zwrotny, ruchliwy
- Filozofia tworzenia oprogramowania
- Wiele metod
 - Extreme Programming, Scrum, Lean Development, ...
- *Metodyki agilne*
- <http://agilealliance.org>



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.

Manifest ⇒ Zasady

- Najważniejsza satysfakcja klienta
 - Wczesne i ciągłe dostarczanie oprogramowania
- Witamy zmieniające się wymagania ☺
 - Nawet w końcowych fazach projektu
- Częste dostarczanie działającego oprogramowania
- Menadżerowie i deweloperzy pracują codziennie razem w trakcie całego projektu

Manifest ⇒ Zasady

- Motywacja pracowników
 - Odpowiednie warunki, wsparcie, zaufanie
- Rozmowa twarzą w twarz jest najlepszą formą komunikacji
- Działający program jest podstawową miarą postępu
- Samoorganizujące się zespoły
- I inne...

- Metodologie sterowane planem
 - Praktyki inżynierskie (mechanika, inż. lądowa)
 - Zaplanuj zanim zbudujesz
 - Szczegółowy plan konstrukcji/budowy
 - Ustalenie harmonogramu i budżetu dla budowy
 - Etap budowy raczej manualny niż intelektualny

Dlaczego AGILE?

- Metodologie sterowane planem
 - ...
 - Potrzebny szczegółowy harmonogram budowy dla ludzi z mniejszymi umiejętnościami
 - Należy rozdzielić projektowanie od wytwarzania/budowy oprogramowania
 - Płynne przejście od projektu do budowy – UML

Dlaczego AGILE?

- Metodologie sterowane planem
 - ...
 - Jak trudno zbudować projekt w UMLu gotowy do przekazania programistom?
 - Jaki jest koszt projektowania?
 - Budowa mostu: **10%** koszt projektu
 - Software: 15% kosztów to kodowanie + testowanie unitów; **50%** koszt projektu (bez testowania) !!!
 - Rapid Development (Paperback) by Steve McConnell

Dlaczego AGILE?

- Metodologie sterowane planem
 - ...
 - Budowa oprogramowania – zautomatyzowane (kompilator)
 - Rational Unified Process (RUP) ???
 - Biurokracja

Dlaczego AGILE?

- Oprogramowanie
 - Budowa tak tania, że praktycznie darmowa
 - Cały wysiłek to projektowanie i dlatego wymaga kreatywnych i utalentowanych ludzi
- Procesy kreatywne nie dają się zaplanować, a więc nie mogą być przewidywalne
- Metody tradycyjnej inżynierii zawodzą przy wytwarzaniu oprogramowania

Nieprzewidywalność wymagań

- Zdefiniowanie wymagań na początku
- Zdefiniowanie kosztu realizacji wymagania
- Tworzenie oprogramowania
 - To projektowanie (trudno zaplanować i wycenić)
 - Zależy od ludzi realizujących zadanie (jak przewidzieć i zmierzyć?)
 - Był logiczny (jak ocenić wartość cechy przed zobaczeniem jej w akcji?)
 - Wartość podlega szybkim zmianom rynkowym

Nieprzewidywalność wymagań

- Dziwne jeśli ktoś się dziwi, że wymagania się zmieniają ☺
- Zmiany to norma, a problem: jak sobie z nimi radzić?
- Cały proces tworzenia oprogramowania bazuje na wymaganiach
 - Brak niezmiennych wymagań ⇒ brak przewidywalnego planu

Da się przewidzieć?

- Generalnie nie
- Da się np. w NASA
 - Pełno ceregieli
 - Mnóstwo czasu
 - Duży zespół
 - Stałe wymagania

Nie bój się!

- Przewidywalność jest bardzo pożądana ...
- ... ale nie ludź się, że możesz przewidzieć coś, czego się nie da
- Najtrudniejsze jest przyznanie, że istnieje problem
- Rezygnacja z przewidywalności nie oznacza powrotu do chaosu!
- Potrzebny proces, który pozwoli kontrolować nieprzewidywalność ☺

ADAPTACJA

Kontrola zmian

- Iteracje

- Odpowiedź na to, gdzie jesteśmy
- Krótkoterminowe planowanie
- Jak długa powinna być iteracja?
 - XP: 1-2 tygodnie
 - SCRUM: 1 miesiąc
 - ...
 - Tak krótka, jak się da ☺

Kontrola zmian

- Ustalona cena wymaga ustalonych wymagań, a więc przewidywalnego procesu
- Adaptujący się klient
 - Można ustalić budżet
 - Nie można ustalić czasu, ceny i zakresu projektu
- Podejście ZWINNE
 - Ustalenie czasu i ceny
 - Zakres zmienia się w sposób kontrolowany

Klient

- Aktywnie uczestniczy w procesie tworzenia oprogramowania
- Otrzymuje produkt lepiej spełniający jego wymagania
- Lepsza kontrola ryzyka

Ocena projektu

- Ocena zaplanowanego z góry projektu
 - Jak mieści się w planie?
 - Na czas i w kosztach – SUKCES
- Podejście ZWINNE
 - Istotna jest wartość biznesowa
 - Czy klient otrzymał produkt dla niego bardziej wartościowy niż poniesione koszty?

Czynnik ludzki

- Ludzie plug-and-play
 - Nie są istotne osoby, tylko role (analityk, programista, tester itd.)
 - Czy ludzie są wymiennymi komponentami?
 - Brak motywacji
- Podejście ZWINNE
 - Ludzie to niewymienne zasoby
 - Ludzie to jednostki, od których zależy sukces projektu

Odpowiedzialni i profesjonalni

- Robotnicy w fabryce z XIX wieku
 - Nie są w stanie ocenić jak najlepiej wykonać swoją pracę
 - Niezbyt inteligentni i kreatywni
- Tworzenie oprogramowania
 - Bystrzy i zdolni, sława i pieniądze
 - Trzeba dbać o specjalistę
 - Odpowiedzialni profesjonaliści decydujący jak wykonać zadania techniczne

Proces zorientowany na ludzi

- Wymaga poświęcenia i zaangażowania jednostek
- Deweloperzy :
 - Dzielą odpowiedzialności z kierownictwem
 - Podejmują samodzielnie wszystkie techniczne decyzje
 - Oceniają czas realizacji zadań
 - Mają dostęp do eksperta
- Szybka dewaluacja technicznej wiedzy
 - Dzielenie pracy i zaufanie w zespole
- Samoadaptujący się proces

Dla kogo podejście ZWINNE?

- Zespół musi chcieć tak pracować
- Klient musi chcieć tak pracować
- Łatwiej zacząć od prostszego procesu
- Zaczni od prostszego projektu
- Dobry mentor wart swojej wagi w złocie
- Słuchaj mentora ☺

Kiedy złe jest podejście ZWINNE?

- Zależy od nastawienia ludzi
- Granice stosowania
 - Zbyt młode techniki
 - Co oznacza sukces, a co porażkę przy tworzeniu oprogramowania?

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.

- Jednostki i komunikacja zamiast procesów i narzędzi

Rozmawianie z ludźmi pozwala nam robić to, co chcemy, w taki sposób jak chcemy. Oczywiście zakładamy, że wiemy czego chcesz, nawet jeśli ty tego nie wiesz.

- Działające oprogramowanie zamiast pełnej dokumentacji

Cały czas chcemy kodować. Prawdziwi programiści nie piszą dokumentacji.

Krytycznym okiem

- Współpraca klienta zamiast negocjacji kontraktowych

Nie marnujmy czasu na targowaniu się o szczegóły, to tylko uniemożliwia spędzenie całego czasu na kodowaniu. Rozwiąkłamy wątpliwości, jak coś dostarczymy.

- Adaptacja zmian zamiast przestrzegania planu

Przestrzeganie planu wymagałoby od nas przemyślenia problemu i sposobów jego rozwiązania. Po co mielibyśmy to robić, skoro możemy kodować?

Metodyki ZWINNE

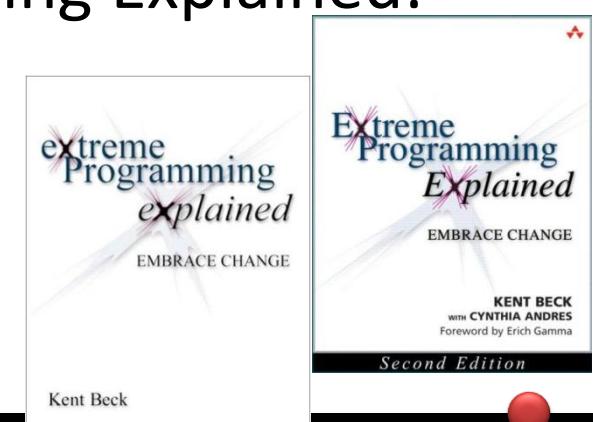
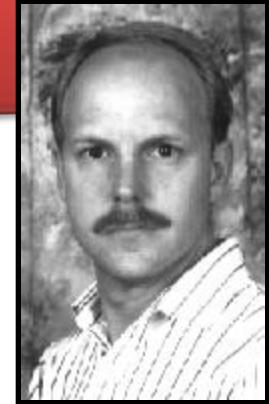
- eXtreme Programming (XP)
- SCRUM
- Crystal
- Lean Development
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Feature Driven Development (FDD)

1. eXtreme Programming



Podstawa ☺

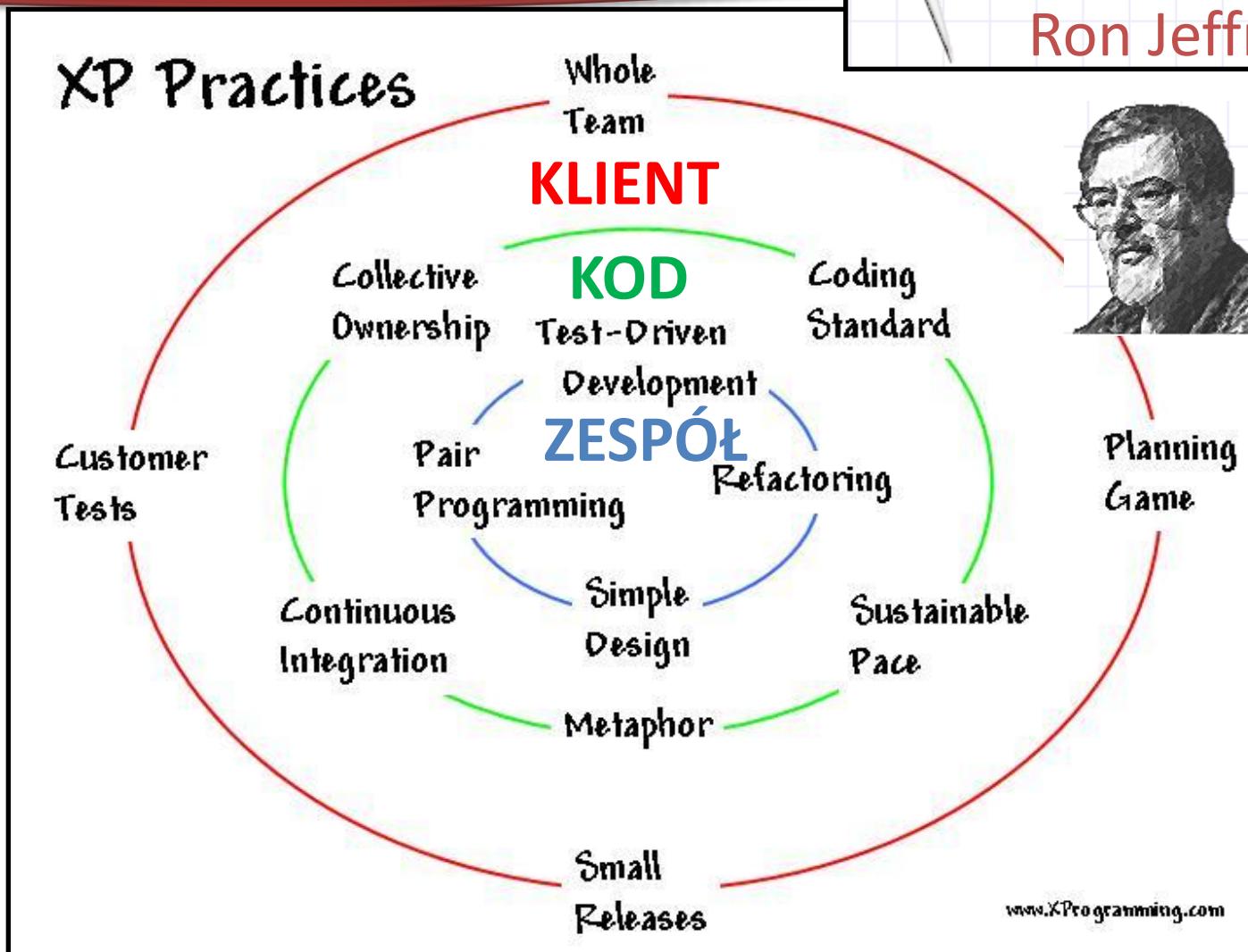
- Kent Beck, Chrysler C3 project, 1997
- Podstawa licznych opracowań
 - Kent Beck, Extreme Programming Explained: Embrace Change
- 2004 Przeforumułowanie podejścia
 - Kent Beck, Extreme Programming Explained: Embrace Change (2nd Edition)



4 wartości

- Prostota (simplicity)
- Komunikacja (communication)
- Informacja zwrotna (feedback)
- Gotowość do zmian (courage)

Ron Jeffries



Podział praktyk (Klient)

- Whole Team
 - Każdy uczestniczący w projekcie jest członkiem zespołu
 - Zespół tworzony wokół Klienta
- Planning Game, Small Releases, Customer Tests
 - Proste planowanie i śledzenie projektu
 - Wartość biznesowa przede wszystkim
 - Małe w pełni zintegrowane wdrażanie oprogramowania, które przeszło testy Klienta

Podział praktyk (Zespół)

- Simple Design, Pair Programming, Test-Driven Development, Refactoring
 - Praca grupowa i w parach
 - Prosty projekt
 - Obsesyjne testowanie
 - Ciągłe udoskonalanie projektu by spełniał bieżące wymagania

Podział praktyk (Kodowanie)

- Continuous Integration, Collective Code Ownership, Coding Standard
 - Stale działający i zintegrowany system
 - Kodowanie w parach
 - Ciągła współpraca
 - Ustandardyzowane kodowanie – czytelny kod, można udoskonalać

Podział praktyk (Kodowanie)

- Metaphor, Sustainable Pace
 - Wspólny prosty obraz systemu
 - Każdy pracuje ze stałą nigdy nie słabnącą wydajnością (szybkością)



The Rules and Practices of Extreme Programming.

*Lessons
Learned*

Planning

- User stories are written.
- Release planning creates the schedule.
- Make frequent small releases.
- The Project Velocity is measured.
- The project is divided into iterations.
- Iteration planning starts each iteration.
- Move people around.
- A stand-up meeting starts each day.
- Fix XP when it breaks.

Coding

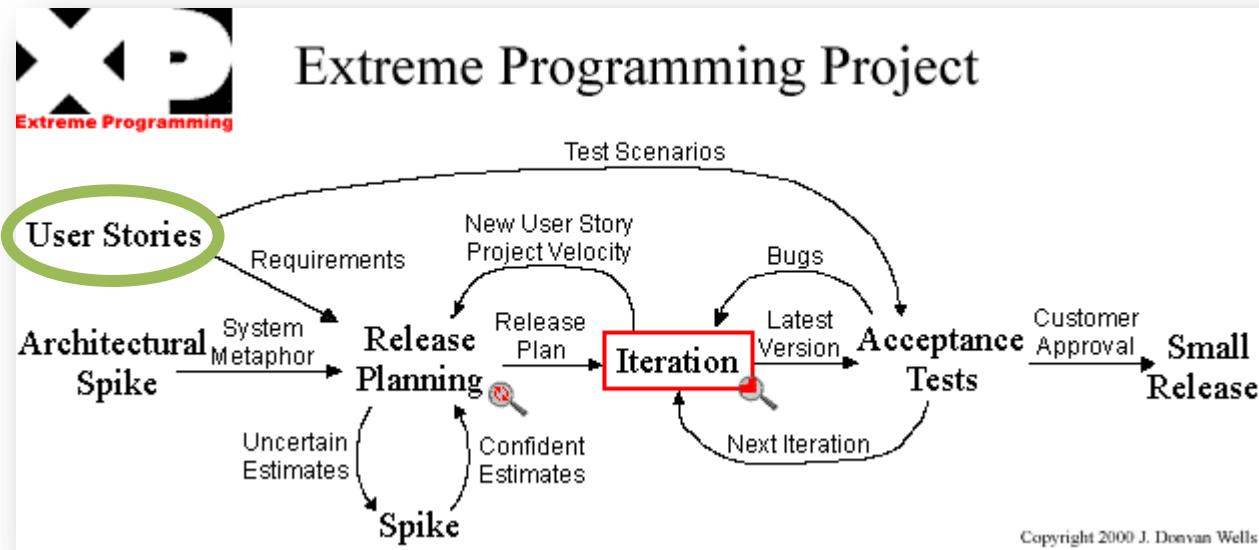
- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Use collective code ownership.
- Leave optimization till last.
- No overtime.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.



Zwinna specyfikacja wymagań

- Karty wymagań (User Stories)
 - Opowiadania użytkownika
 - Historyjki użytkownika ☺
- Szkice ekranów
- Testy akceptacyjne

Madeyski L., Kubasiak M.: Zwinna specyfikacja wymagań

Karta wymagań

Jako [osoba odgrywająca daną rolę] chciałbym móc [wykonać jakąś czynność] aby [osiągnąć jakiś cel].

Karta wymagań

Jako [osoba odgrywająca daną rolę] chciałbym móc [wykonać jakąś czynność] aby [osiągnąć jakiś cel].

- Jako klient chciałbym móc złożyć zamówienie.
- Jako użytkownik chciałbym móc się zalogować.

Karty wymagań

UTWÓRZ KONTO

OPIS

Jako administrator chciałbym móc utworzyć konto pracownikowi dydaktycznemu, aby umożliwić mu dostęp do systemu.

Po wypełnieniu wymaganych pól, system automatycznie generuje hasło oraz wysyła nowemu użytkownikowi wiadomość e-mail z danymi dostępowymi.

PRIORYTET

Krytyczne

OSZACOWANIE (godz.)

6

Szkic ekranu

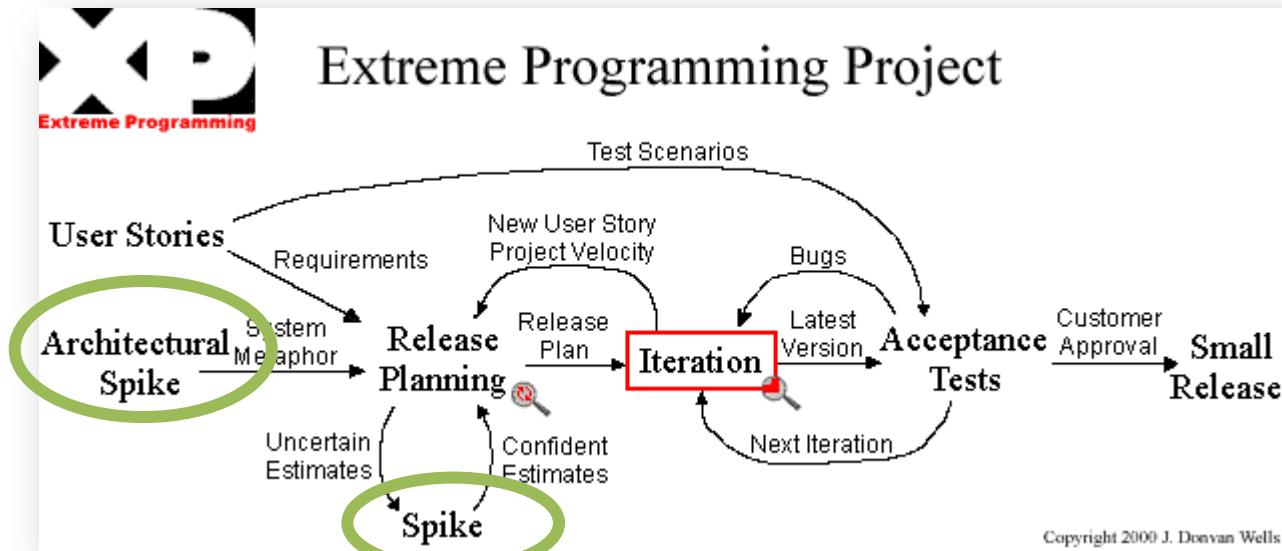
Testy akceptacyjne

UTWÓRZ KONTO testy akceptacyjne

- 1 Administrator wypełnił prawidłowo wszystkie wymagane pola, wprowadził również unikalną wartość identyfikatora użytkownika – w nowo otwartym oknie system informuje o pomyślnym utworzeniu konta i wysłaniu wiadomości e-mail.
- 2 Administrator wypełnił wszystkie wymagane pola, jednak w systemie figuruje już użytkownik o takim samym identyfikatorze – system wyświetla na formularzu komunikat ostrzegawczy o tym, że pole „login” musi zawierać unikalną wartość.
- 3 Administrator nie wypełnił wymaganych pól (oznaczonych *) – system wyświetla na formularzu informację o niekompletnych danych wraz z prośbą o ich uzupełnienie.

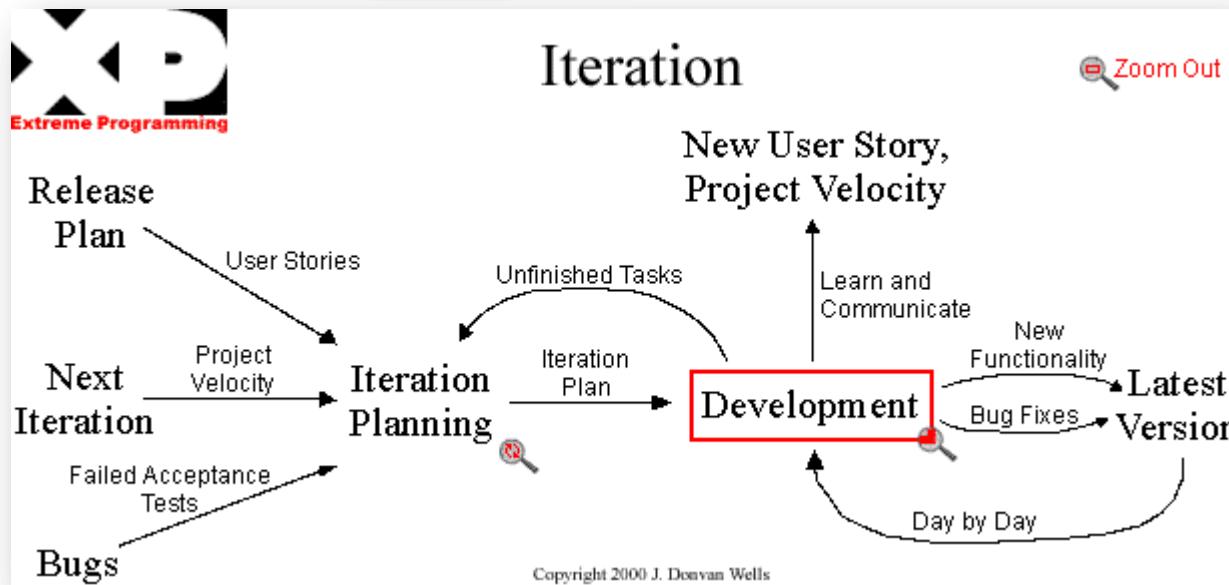
Karty wymagań

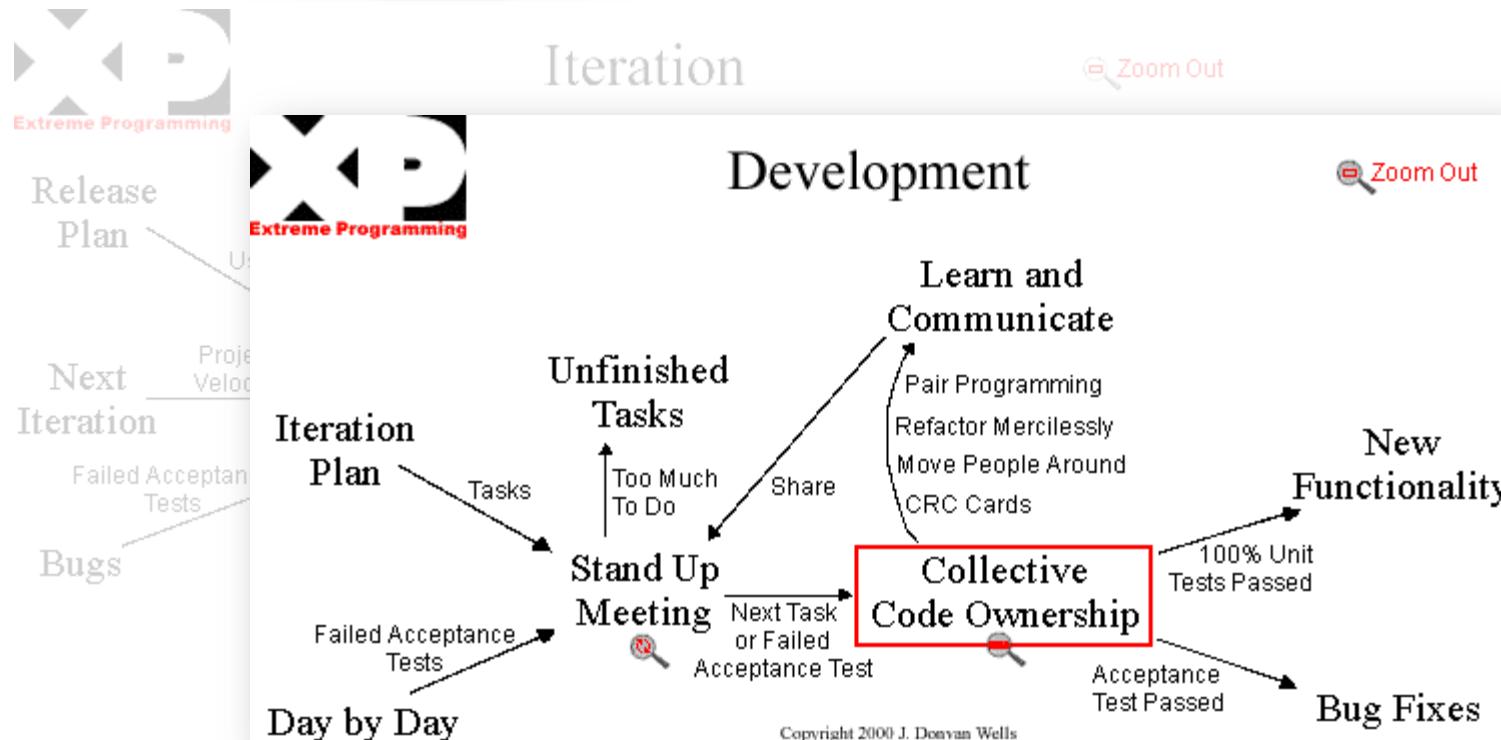
- Bardzo ogólne do oszacowania czasu implementacji
 - Deweloper idzie do klienta i dostaje szczegóły
- Każdy scenariusz jest szacowany przez dewelopera na 1, 2 albo 3 tygodnie „idealnego programowania”
 - Dłużej niż 3 tygodnie ⇒ trzeba rozbić scenariusz
 - Mniej niż tydzień ⇒ zbyt niski poziom szczegółowości
- 80 ± 20 US= idealna liczba by stworzyć Release Plan

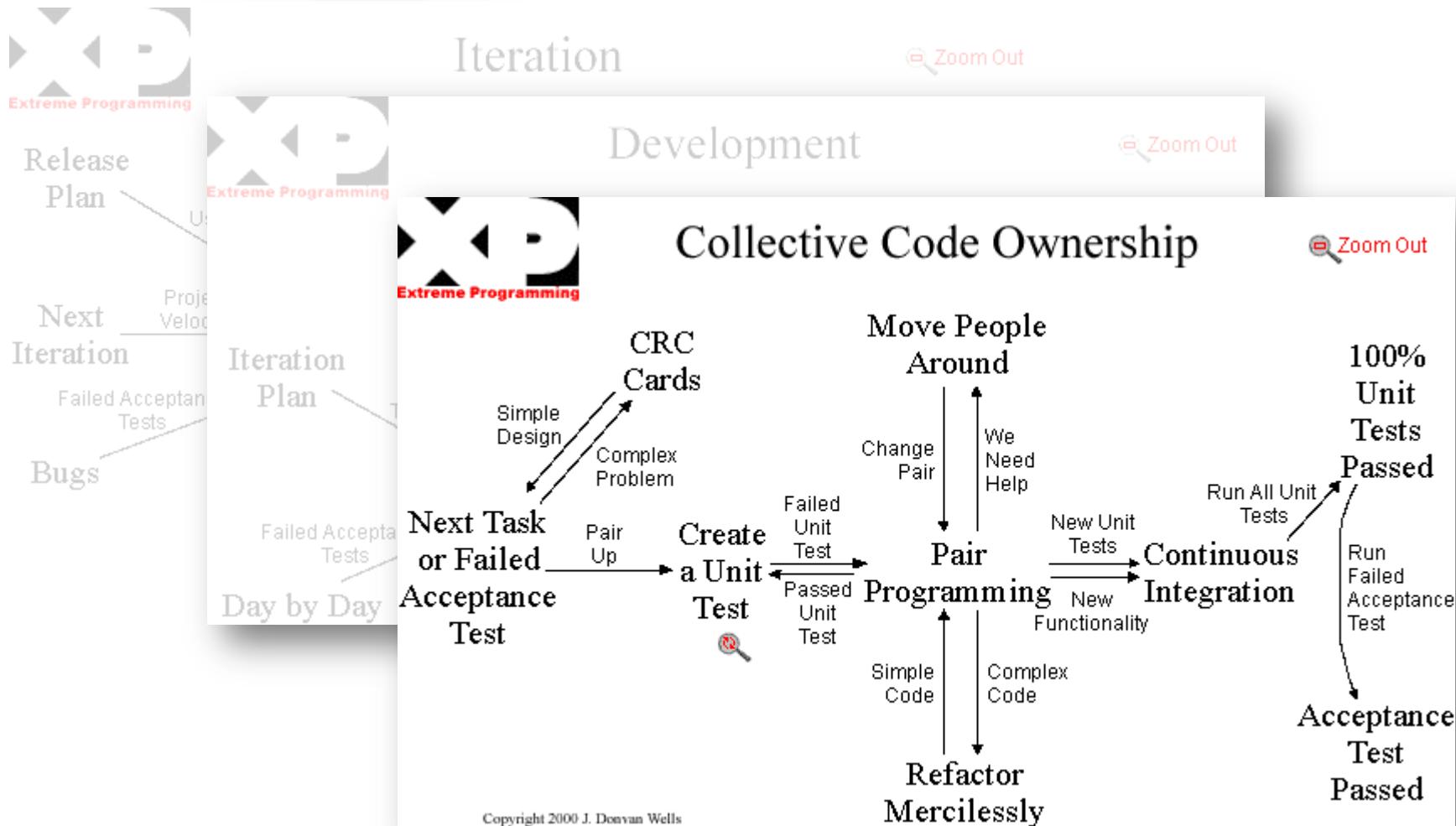


Spike Solution

- Tworzone by rozwiązać problemy techniczne bądź projektowe
- Jest to bardzo prosty program do zbadania potencjalnych rozwiązań
- Tworzone by:
 - zredukować ryzyko rozwiązań technicznych
 - zwiększyć wiarygodność oszacowania User Story
- Throw-away prototyping ☺







CRC cards

- Class-Responsibility-Collaboration card
- Burza mózgów – projektowanie zorientowane obiektowo (Ward Cunningham)
- Pierwsza definicja klas i ich współpracy
- Kartka papieru
 - Nazwa klasy
 - Nazwa pakietu (jeśli jest)
 - Obowiązki klasy
 - Nazwy klas współpracujących

2. Scrum



Podstawa ☺

- Ken Schwaber
 - <http://www.controlchaos.com>
- Polska grupa Scrum
 - <http://www.scrum.org.pl/>
- Przewodnik po metodyce Scrum
 - <http://www.scrum.org/scrumguides/>



Scrum

- Zaprezentowany OMG w 1995 przez Advanced Development Methods (ADM) and VMARK Software (VMARK)
 - Niewydajne prowadzenie projektów zorientowanych obiektowo

Scrum?



MŁYN

<http://www.jirifabian.net/wordpress/?p=135>

Scrum?

- To nie akronim!
- AGRESYWNE usuwanie przeciwności!
- Metodyka zarządzania projektem
 - Często stosowana z XP

Scrum

- Scrum jest wzorcem!!!
- Małe zespoły (< 10 people)
- Serie sprintów
- Rozwój inkrementacyjny
- Time-boxing
- Koncentracja na zarządzaniu projektem
- Mały nacisk na praktyki
 - Łączenie z praktykami XP

Role w zespole

- Właściciel produktu
 - Decyduje o funkcjonalności i priorytetach realizacji
 - Prowadzi rejestr produktowy (wymagania funkcjonalne)
- Mistrz
 - Odpowiada za przestrzeganie reguł Scruma
 - Usuwa przeszkody w pracy zespołu
- Zespół
 - Interdyscyplinarny, samoorganizujący się i samowystarczalny

Proces tworzenia oprogramowania

- Zbiór luźno powiązanych czynności łączących znane i działające techniki oraz narzędzia, pozwalający zbudować system
- Dwie fazy
 - Planowanie
 - Sprint

Sprint

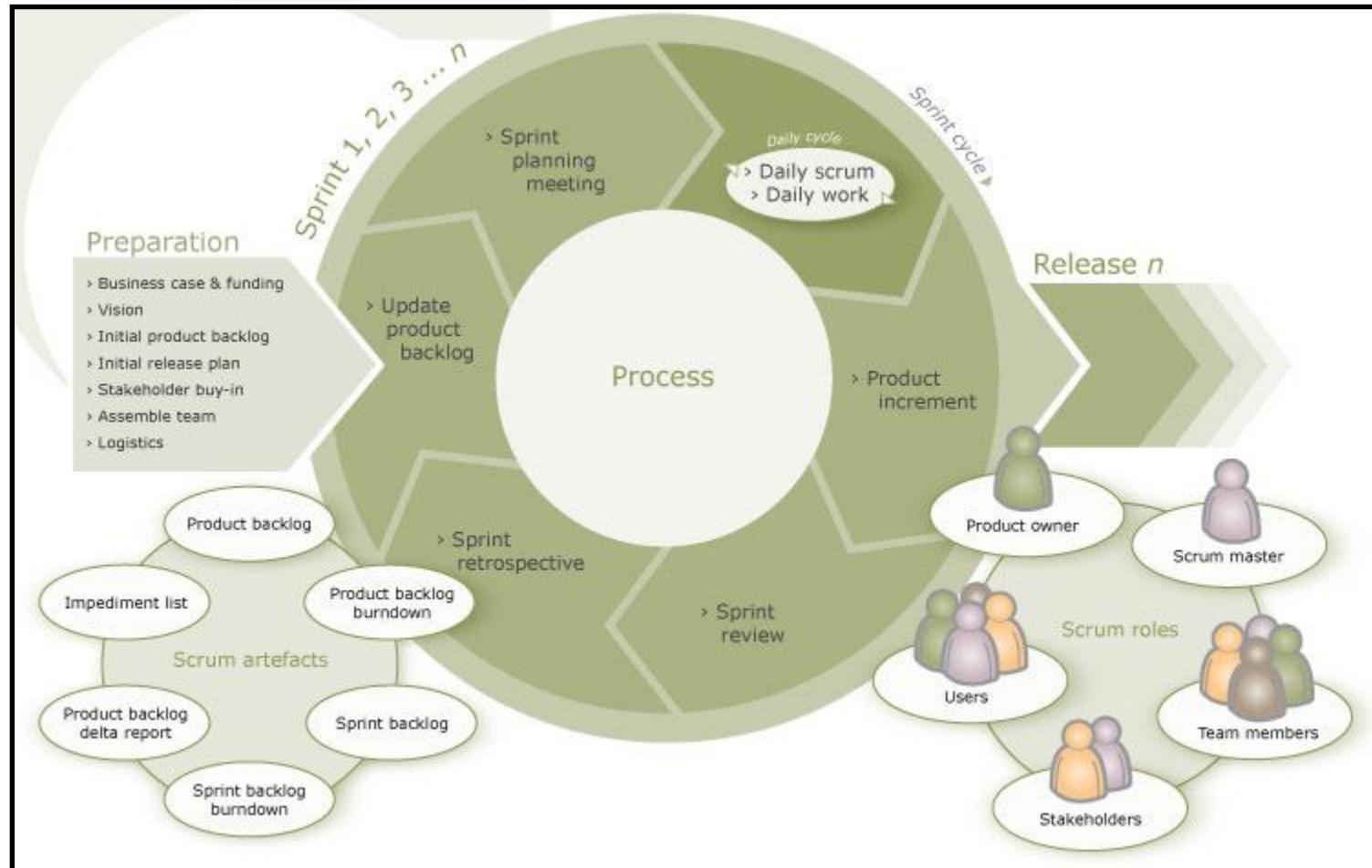
- Iteracja (2-4 tygodni)
- Zadania przypisane członkom zespołu
- Koncentracja na celu
 - Żadnego rozpraszania uwagi ☺ ScrumMaster!!!
 - Zero przerwań/zmian z zewnątrz
 - Nowe zadania mogą zostać zidentyfikowane przez zespół

Sprint

- Częste, krótkie spotkania (Scrum Meetings)
- Każdy zespół dostarcza widocznego, działającego elementu
- Kolejne iteracje są inkrementacyjne
- Jasno zdefiniowane obowiązki oraz wyniki iteracji
- Wszyscy członkowie zespołu zaangażowani w realizację zadania

Scrum Meeting

- 15-30 minutowe, częste (codzienne) spotkania organizowane przez Mistrza
- Cały zespół bierze udział
- Do każdego zadania 3 pytania:
 - Co zrobiłeś od ostatniego spotkania?
 - Jakie napotkałeś trudności?
 - Co zrobisz do następnego spotkania?



Koniec sprintu

- Spotkania:
 - Przegląd sprintu
 - Retrospektywa sprintu
 - Spotkanie planistyczne następnej iteracji
- Dostarczenie działających części oprogramowania
- Zebranie niespodzianek ☺
- WSZYSTKO może zostać zmienione, zakres prac, priorytety
- Nowe oszacowania i przydzielenie prac na następny Sprint
- Projekt może zostać przerwany

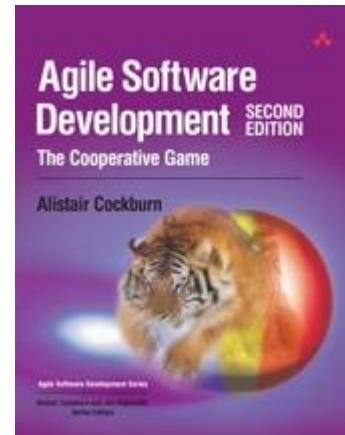
3. Crystal Family



Podstawa ☺



- Alistair Cockburn
 - Agile Software Development: The Cooperative Game (2nd Edition)
 - alistair.cockburn.us



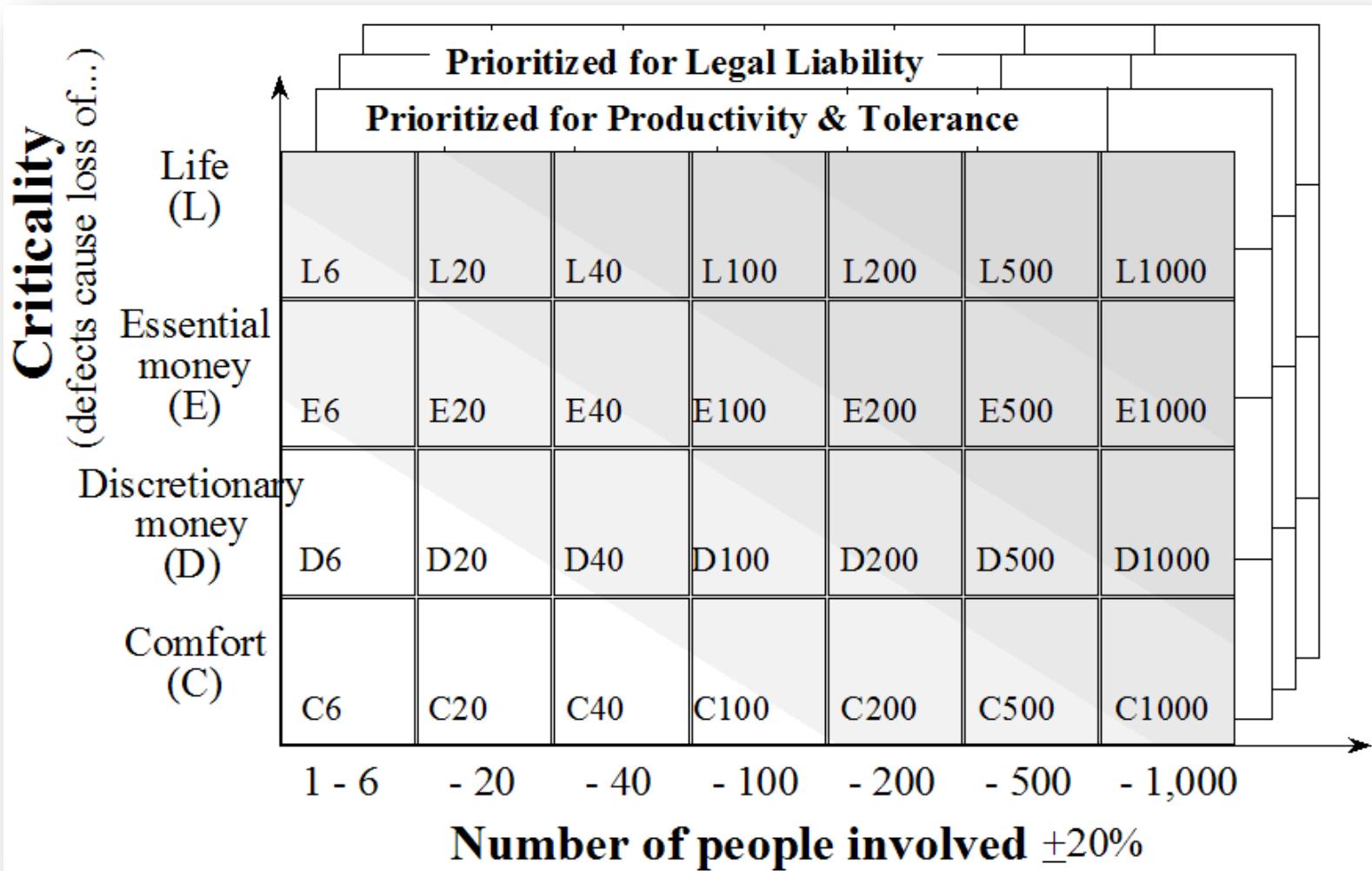
Tak się zaczęło...

- Alistair Cockburn początek lat 90tych
 - Opracowanie metodologii dla the IBM Consulting Group
 - 1991 – 1999, Hong Kong – obie Ameryki, Norwegia i Ameryka Południowa
 - COBOL, Smalltalk, Java, VB, Sapiens, Synon

Wnioski

- Rozmowy twarzą w twarz zamiast dokumentów
- Im częściej dostarczane są małe, działające części systemu, tym pewniejsza pomyślna realizacja systemu
- Różne projekty mają różne potrzeby

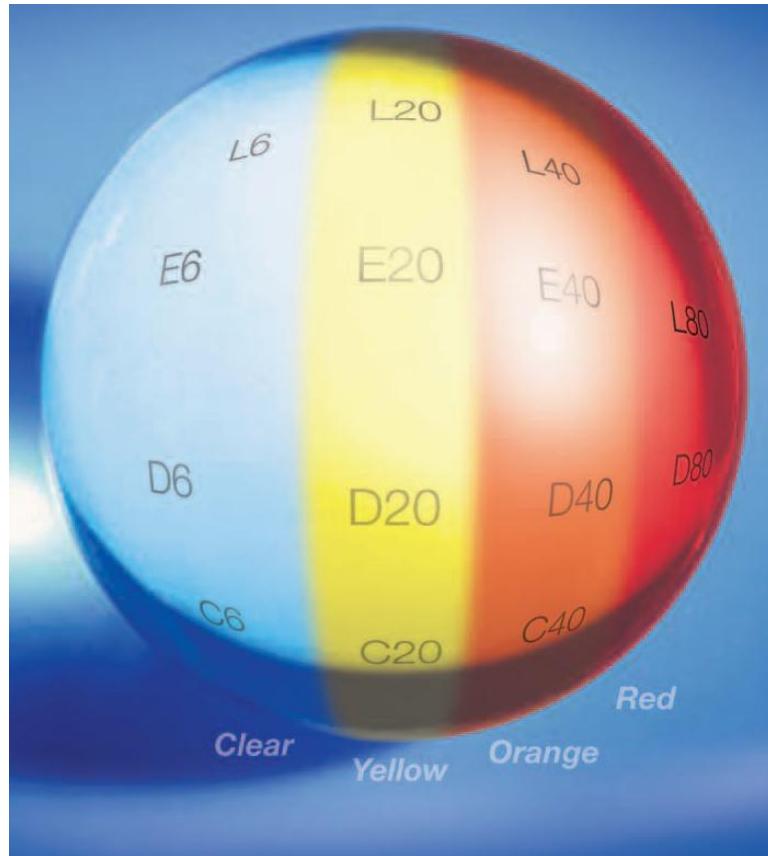
Wnioski



The Crystal Light family

- Crystal Clear: 2-6 osób
- Crystal Yellow: 6-20 osób
- Crystal Orange: 20-40 osób
- Crystal Red: 40-100 osób
- Crystal Magenta, Crystal Blue, etc.
- Przesunięcie wzdłuż osi Y ⇒ przejście z wersji „light” w kierunku „hard”

The Crystal Light family



E6	E20	E40	E80
D6	D20	D40	D80
C6	C20	C40	C80

©Alistair Cockburn 2003-2005

Niespodzianki

- Jak mało potrzeba procesu i kontroli by zespół dobrze funkcjonował
- Krytyczne dla zarządzania projektem jest zrozumienie i informowanie o współzależnościach
 - Często nie potrzebne są narzędzia
 - Zaufanie i komunikacja!!!!!!
- Crystal Light wygląda ciężko przy XP
 - XP – dyscyplina

Manifest

Software development is a (series of) cooperative game(s)... The endpoint of the game is an operating software system... The next game is the alteration or replacement of the system, or creation of a neighboring system .

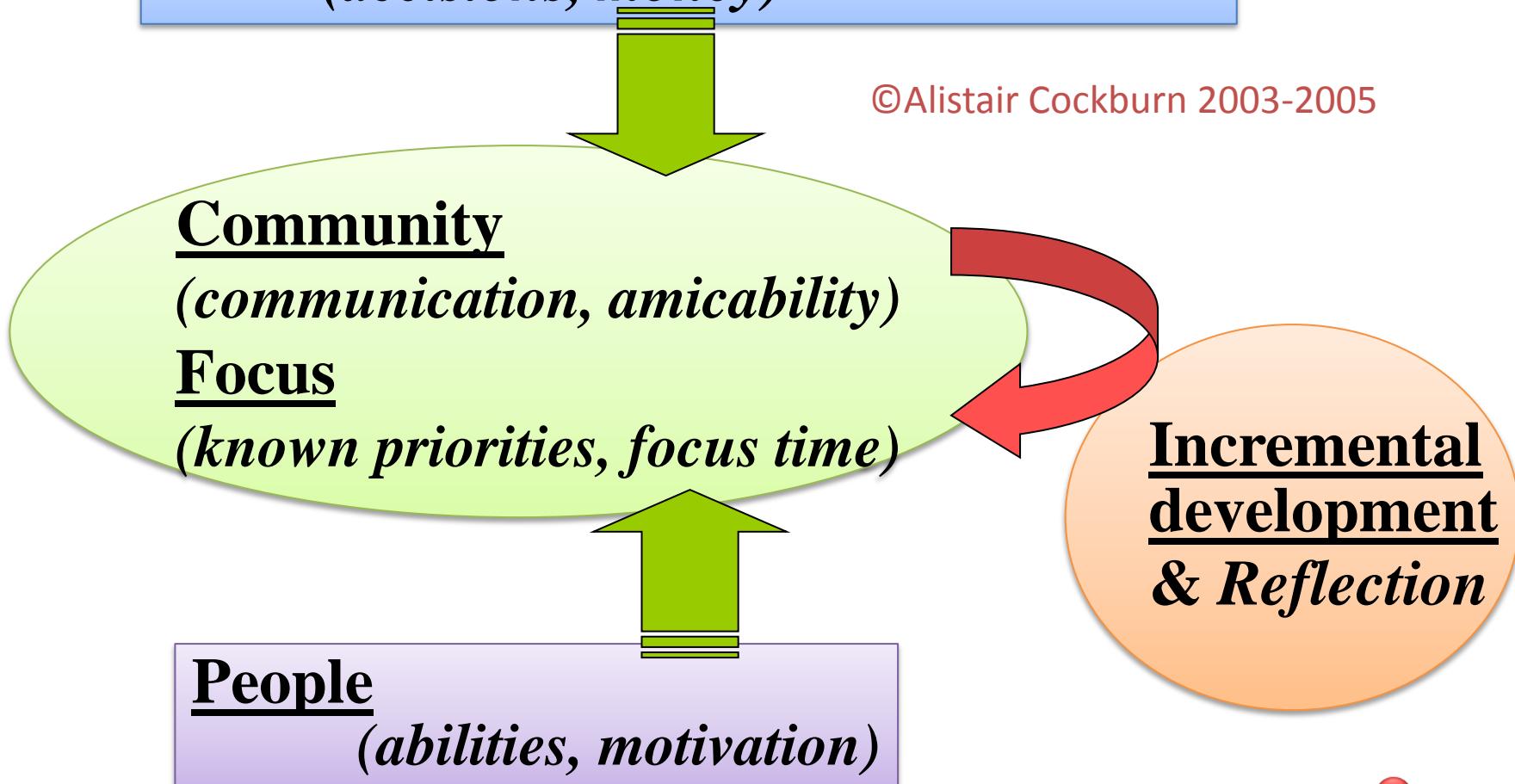
Alistair Cockburn

By zrozumieć GRĘ

- 10 krytycznych czynników sukcesu
- 7 własności projektów zakończonych sukcesem

10 czynników sukcesu

Nourishment from Executive Sponsors *(decisions, money)*



7 praktyk sukcesu

1. Częste iteracje (time-boxing; 2x0.5 roku)
2. Przemyślane udoskonalanie
 - Krytykuj i naprawiaj
3. Osmotyczna komunikacja i zdobywanie wiedzy
 - Odpowiednia organizacja
 - Kanały komunikacji
4. Osobiste bezpieczeństwo
 - Można być szczerym
 - Można się nie zgadzać

7 praktyk sukcesu

5. Koncentracja

- Jasne zadania i priorytety
- Bez nadmiernego obciążania pracą

6. Eksperci w zasięgu ręki

7. Środowisko zwinne ☺

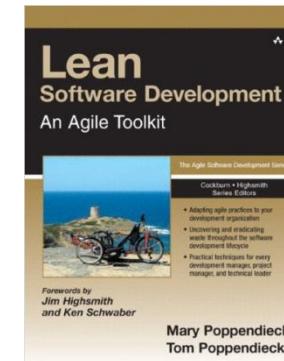
- Automatyczne testowanie
- Zarządzanie konfiguracją
- Częste integracje

4. Lean Development



Podstawa ☺

- Robert Charette, początek lat 90tych, ITABHI Corporation
 - Lean manufacturing, Taiichi Ohno, the Toyota Production System
 - "Just-In-Time"
 - "Jidoka"
 - W. Edwards Deming, Total Quality Management (TQM), lata 80te
- Mary Poppendieck
 - <http://www.poppendieck.com/>



14 punktów TQM

- Stale udoskonalaj system
- Likwiduj bariery pomiędzy departamentami
- =====
- Pozbądź się strachu
- Wyeliminuj normy, numeryczne cele i oceny osiągnięć
- Nie oceniaj biznesu wyłącznie na podstawie kosztów
- Minimalizuj całkowite koszty

Lean Development

- Nie dla każdego
- Podejście strategiczne, sterowane zasadami
 - Większość zwinnych jest taktyczna, zorientowana na zespół

12 zasad

1. Najwyższym priorytetem jest satysfakcja klienta
2. Zawsze dostarczaj najlepszą jakość za rozsądную cenę
3. Sukces zależy od aktywnego udziału klienta
4. Dewelopment to praca zespołowa
5. Wszystko jest zmienne
6. Ogólne, nie specjalistyczne oprogramowanie
7. Buduj z komponentów

12 zasad

8. 80% rozwiązań dzisiaj zamiast 100% jutro
9. Kluczowy jest minimalizm
10. Potrzeby definiują technologię
11. Rozwój produktu to rozwój funkcjonalności, nie rozmiaru
12. Nie stosuj LD tam, gdzie można użyć innych metod z większym zyskiem

13ta niepisana zasada

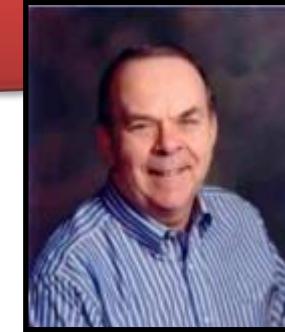
- 75% sukcesu to czynnik ludzki ⇒ potrzeba zaangażowanych i zadowolonych pracowników

5. Adaptive Software Development



Podstawa ☺

- Jim Highsmith
 - RAD
 - Adaptive Software Development: A Collaborative Approach to Managing Complex Systems
 - <http://www.adaptivesd.com/>
 - Połowa lat 90tych
 - Complex Adaptive Systems (CAS) theory [Holland1995]



CAS theory

- Świat
 - Agenci
 - Środowisko
 - Emergencja
- Deweloperzy tworzą środowisko
- Deweloperzy są agentami
- Oprogramowanie – emergentny rezultat współpracy i współzawodnictwa agentów

Zarządzanie

- Adaptacja
 - Szybka odpowiedź na zmiany
- the Adaptive Leadership-Collaboration model
 - Strategia:
 - Koncentracja na produkcie, nie na procesie
 - Duża skala: dostarczenie narzędzi i technik umożliwiających samoorganizację wirtualnym zespołom

Speculate-Collaborate-Learn

- Cykl życia zorientowany na
 - Zmianę
 - Ciągłe uczenie się
 - Patrzenie w niepewną przyszłość
 - Intensywną współpracę deweloperów, kierownictwa i klienta
- Każda iteracja to 3 fazy
 - Spekulacja
 - Współpraca
 - Nauka

Spekulacja 5 kroków

- Rozpoczęcie projektu – ustalenie misji
- Ustalenie time-boxu dla całego projektu
- Ustalenie liczby iteracji i przypisanie im time-boxów
- Wybranie tematu bądź celu dla każdej iteracji
- Deweloperzy i klienci przypisują rezultaty każdej iteracji

Współpraca

- Zespół techniczny dostarcza działającego oprogramowania
- Kierownictwo ułatwia współpracę i współbieżne działania
- Dobra współpraca
 - Umiejętność pracy w grupie
 - Dzielenie wiedzy
 - Podejmowanie decyzji

Nauka po iteracji

- Ocena jakości z punktu widzenia klienta
- Ocena jakości pod kontem technicznym
- Ocena funkcjonowania zespołu wdrożeń i praktyk stosowanych przez członków zespołu
- Ocena status projektu

6 podstaw cyklu ASD

1. Koncentracja na misji
2. Nastawienie na rezultat (nie zadanie)
3. Iteracje
4. Time-boxing
5. Sterowanie ryzykiem
6. Adaptacja zmian

Strategia nie technika

- Żadne pojedyncze działanie (czy nawet ich zestaw) nie są konieczne
- Są wskazówki/oczekiwania Highsmitha ☺
- Każda technika ma określony zakres stosowalności
- Zawsze można znaleźć sytuację, że nie zadziała

6. Dynamic Systems Development Method



Podstawa ☺

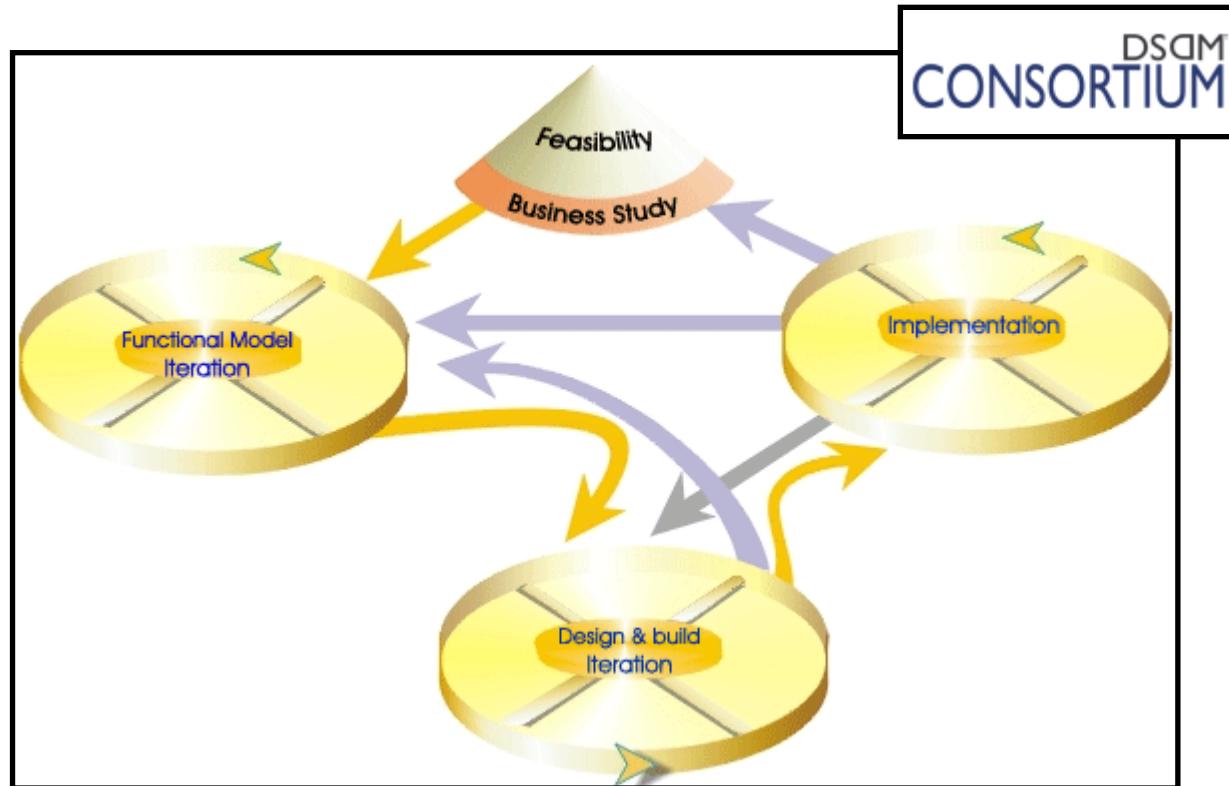
- DSDM consortium
 - <http://www.dsdm.org/>
- Framework nie metoda
 - Szkielet procesu i opisy artefaktów
 - Trzeba dostosować do projektu czy organizacji



7 faz DSDM

1. Pre-Project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and Build Iteration
6. Implementation
7. Post-Project

5 faz



1. Pre-Project

- Rozpoczęto właściwe projekty
- Odpowiednio je zorganizowano (np. fundusze)
- Wstępne planowanie kolejnej fazy

2. The Feasibility Study

- Określenie czy DSDM jest właściwe dla danego projektu
- Definicja problemu
- Oszacowanie kosztów
- Oszacowanie technicznej wykonalności oprogramowania rozwiązującego zadany problem biznesowy

3. The Business Study

- Analiza procesów biznesowych i związanych z nimi danych
- Warsztaty, by szybko ustalić priorytety wymagań

4. Functional Model Iteration

- Szczegółowa analiza biznesowych aspektów systemu komputerowego
- Artefakty
 - Modele analityczne
 - Komponenty oprogramowania
- Prototyp funkcjonalny

5. Design and Build Iteration

- Etap wytwarzania oprogramowania

6. Implementation

- Przejście ze środowiska deweloperskiego do operacyjnego – wdrażanie
- Szkolenie użytkowników
 - Dokumentacja użytkownika

7. Post-Project

- Utrzymanie i pielęgnacja oprogramowania

Podstawy DSDM

- Niezbędne jest zaangażowanie klienta/użytkownika
- Zespół musi być upoważniony do podejmowania decyzji
- Nastawienie na częste krokowe wdrażanie
- Podstawą akceptacji rozwiązania jest wartość biznesowa
- Iteracyjny i inkrementacyjny rozwój projektu

Podstawy DSDM

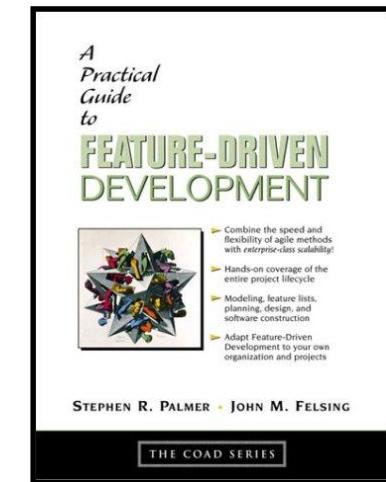
- Wszystkie zmiany w trakcie iteracji są odwracalne
- Kluczowe wymagania są stałe
- Zintegrowane testowanie
- Kluczowa jest współpraca wszystkich członków zespołu

7. Feature Driven Development



Podstawa ☺

- Jeff De Luca & Peter Coad (contributor)
 - www.nebulon.com
 - www.coad.com
- www.featuredrivendevolution.com

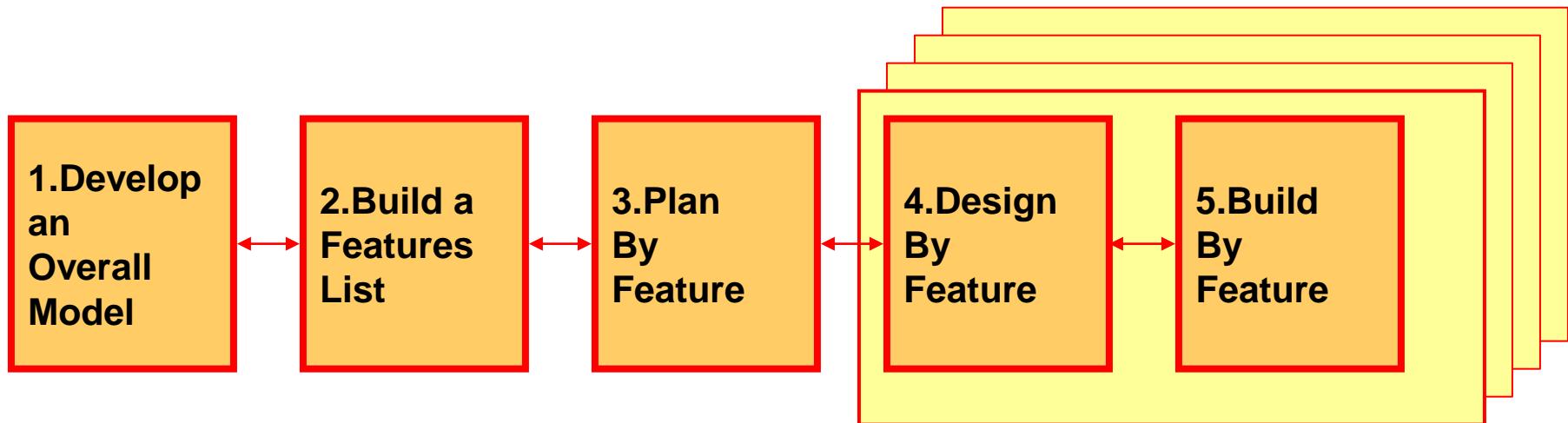


- <http://www.step-10.com/process/APracticalGuideToFDD.html>

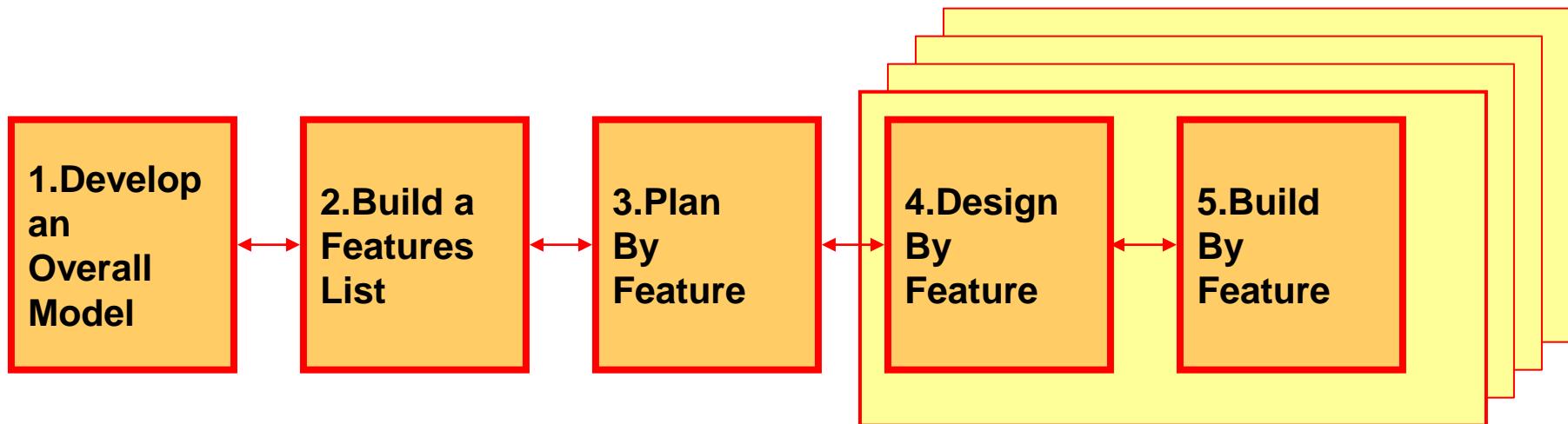
FDD

- Dobry dla większych zespołów (do 500)
- Nacisk na tworzenie oprogramowania dobrej jakości
- Częste dostarczanie działających elementów
- Włożenie na początku odpowiedniego wysiłku przed rozpoczęciem iteracji
- Jasny i oczywisty postęp oraz status projektu

5 procesów



5 procesów w czasie



3

1

1

20

6-msc, w tygodniach(~)

Charakterystyka

- Proces nigdy nie zastąpi dobrych ludzi
- Features list
 - Funkcjonalne
 - Małe
 - Wartościowe dla klienta

Charakterystyka

- Planowanie zorientowane na wymagania
 - Klasy są przypisane do osób
 - Wymagania dobrze uchwycone i zrozumiałe
- Dobre dla stabilnych systemów o przewidywalnym kierunku ewolucji

KONIEC

