

Proces Inżynierii Oprogramowania



- Ustrukturalizowany zbiór czynności wymaganych do opracowania oprogramowania.
- W użyciu jest wiele różnych procesów, ale wszystkie uwzględniają czynności:
 - **Specyfikacja** – definiowanie tego co system powinien realizować;
 - **Rozwój** – definiowanie organizacji systemu oraz implementacja systemu;
 - **Zatwierdzenie** – sprawdzanie czy system wykonuje to czego oczekuje klient;
 - **Ewolucja** – modyfikowanie systemu w odpowiedzi na zmiany potrzeb klienta.
- Model procesu Inżynierii Oprogramowania
 - abstrakcyjna reprezentacja procesu.
 - Opis procesu z określonej perspektywy.

Procesy sterowanie planem i procesy zwinne



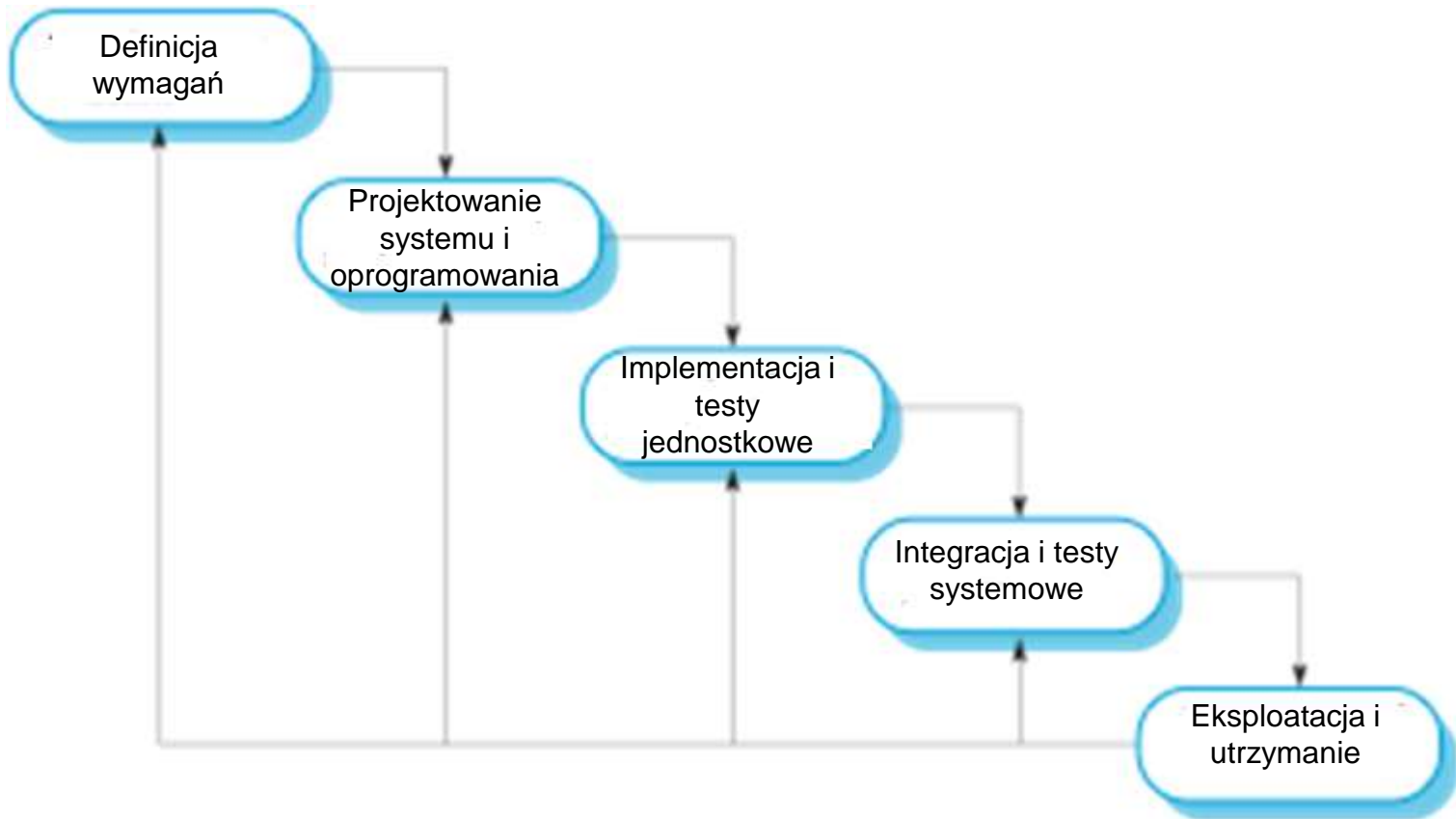
- Procesy sterowane planem to procesy, gdzie wszystkie czynności procesu są zaplanowane. Pomiar postępu prac odnoszone są do tego planu.
- W procesach zwinnych planowanie jest przyrostowe, co powoduje, że łatwiej jest zmienić proces w odpowiedzi na zmianę wymagań.
- W praktyce większość procesów zawiera elementy obu podejść
- Nie ma dobrych lub złych procesów Inżynierii Oprogramowania.

Modele (paradygmaty) procesu Inżynierii Oprogramowania



1. Model kaskadowy (The waterfall model)
 - Model sterowany planem. Oddzielne fazy specyfikacji i rozwoju.
 2. Realizacja przyrostowa (Incremental development)
 - Specyfikacja, wytwarzanie i zatwierdzanie są przeplatane. Może być sterowana planem lub zwinna.
 3. Inżynieria zorientowana na wielokrotne użycie (Reuse-oriented software engineering)
 - System jest składany z istniejących komponentów. Może być sterowana planem lub zwinna.
- W praktyce większość systemów wytwarza się z wykorzystaniem procesów, które wykorzystują elementy wszystkich tych modeli.

Model kaskadowy



Fazy modelu kaskadowego



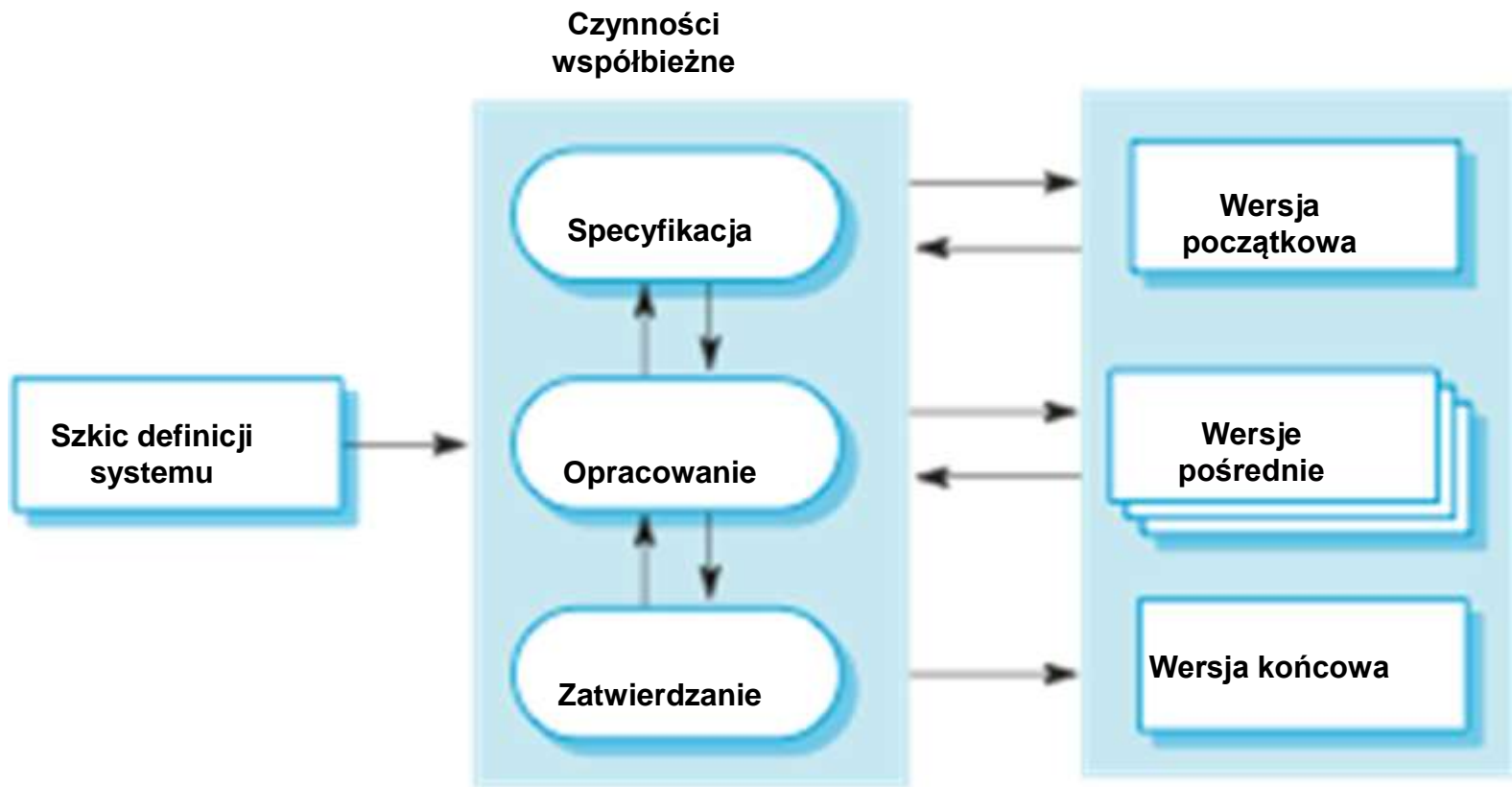
- Model kaskadowy składa się z wydzielonych faz:
 - Analiza i definiowanie wymagań
 - Projektowanie systemu i oprogramowani
 - Implementacja i testy jednostkowe
 - Integracja i testy systemowe
 - Eksploatacja i utrzymanie
- Podstawową wadą modelu jest trudność w adaptacji zmian pojawiających się w czasie realizacji procesu. Z zasady, faza musi być zakończona zanim rozpocznie się kolejna.

Model kaskadowy - dyskusja



- Brak elastyczności poprzez podział projektu na oddzielne etapy powoduje trudności w sytuacji, gdy pojawią zmiany wymagań.
 - Z tego powodu model ten jest odpowiedni w przypadkach gdy wymagania są dobrze zrozumiane a zmiany w trakcie procesu są mocno ograniczone.
 - Niewiele systemów biznesowych posiada stabilne wymagania.
- Model kaskadowy jest wykorzystywany dla realizacji dużych systemów, gdzie prace odbywają się w kilku miejscach.
 - Sterowana planem, natura modelu kaskadowego, pomaga w koordynacji prac.

Realizacja przyrostowa



Zalety realizacji przyrostowej



- Redukcja kosztów adaptacji zmian w wymaganiach.
 - Zakres analizy i dokumentacji, która musi zostać powtórnie wykonana jest zdecydowanie mniejsza w porównaniu z modelem kaskadowym.
- Łatwiej jest uzyskać informację zwrotną na temat dotychczas wykonanych prac.
 - Klienci mogą wypowiedzieć się na temat oprogramowania na podstawie prezentacji dotychczas zrealizowanego zakresu implementacji.
- Możliwe jest szybsze dostarczenie i wdrożenie użytecznego oprogramowania.
 - Klienci mogą wykorzystać oprogramowanie wcześniej w stosunku do tego co oferuje model kaskadowy.

Problemy realizacji przyrostowej



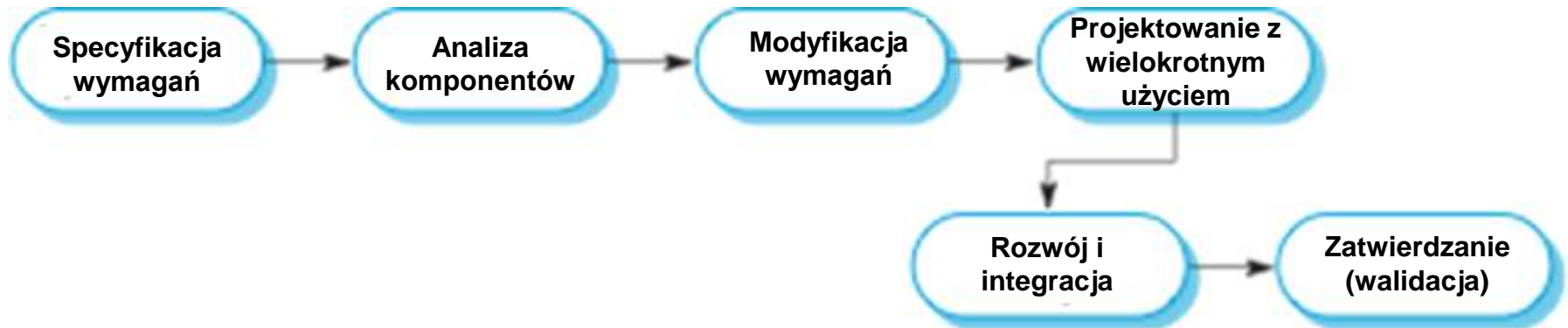
- Niejawność procesu
 - Klasyczne zarządzanie projektem wymaga regularnych rezultatów, pozwalających na pomiar postępów. Jeżeli system jest wytwarzany szybko nie opłaca się produkować dokumentacji odzwierciedlającej każdą wersję systemu.
- Struktura systemu ma tendencję do degradacji wraz z dodawaniem nowych przyrostów.
 - Jeżeli nie zostaną poświęcone dodatkowe środki oraz czas na restrukturyzację oprogramowania, regularne zmiany ostatecznie doprowadzą do zniszczenia struktury oprogramowania. Wprowadzanie kolejnych zmian stanie coraz bardziej kosztowne i skomplikowane.

Inżynieria Oprogramowania zorientowana na wielokrotne użycie

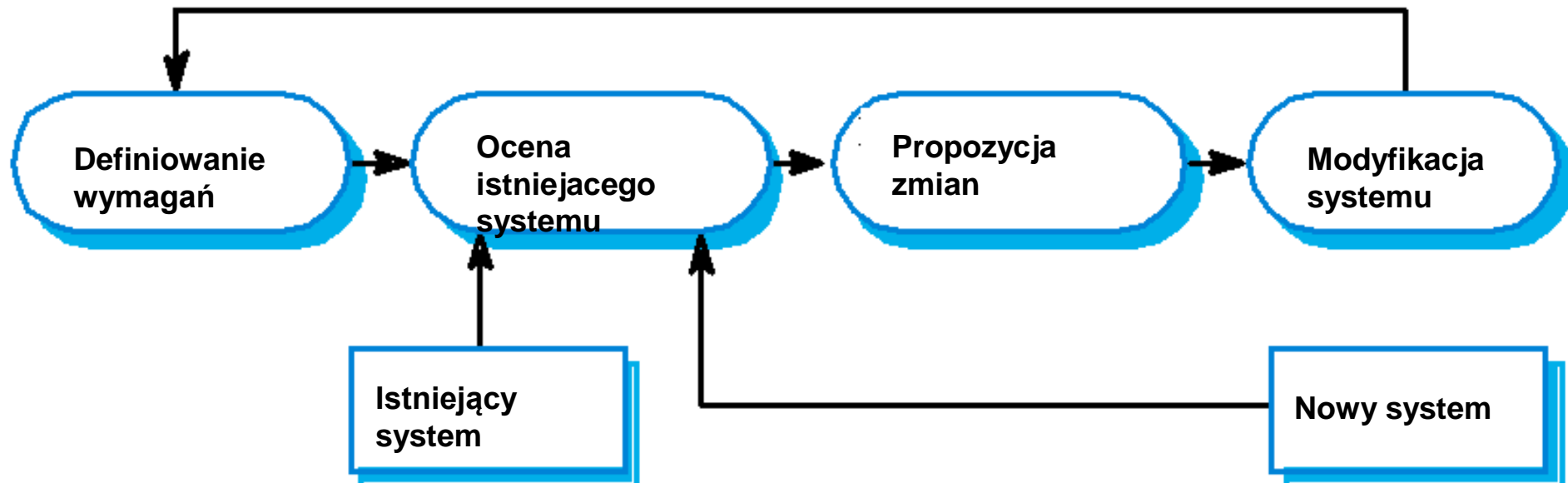


- Opiera się na systematycznym podejściu do wykorzystania istniejącego oprogramowania. System integrowany jest z istniejącymi komponentów systemów COTS (Commercial-off-the-shelf) lub usług.
- Fazy procesu:
 - Analiza komponentów/usług;
 - Modyfikacja wymagań;
 - Projektowanie z wielokrotnym użyciem;
 - Rozwój i integracja.
- Wielokrotne użycie jest obecnie standardowym podejściem do budowania wielu typów systemów biznesowych.

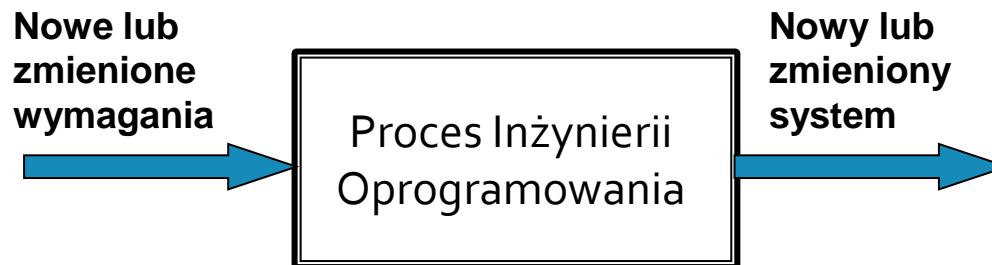
Inżynieria Oprogramowania zorientowana na wielokrotne użycie



Wytwarzanie przez ewolucję



Czyli po prostu...



Radzenie sobie ze zmianą



- Zmiany są nieuniknione we wszystkich dużych projektach oprogramowania.
 - Zmiany biznesowe powodują powstawianie nowych i zmianę istniejących wymagań
 - Owe technologie dają możliwość udoskonalania implementacji
 - Zmieniające się platformy wymuszają zmiany w aplikacjach
- Zmiany prowadzą do ponownego wykonania pewnych prac (np. ponowna analiza istniejących wymagań) oraz zaimplementowania nowych funkcjonalności. Wszystko to składa się na koszty zmian.

Redukcja kosztów modyfikacji



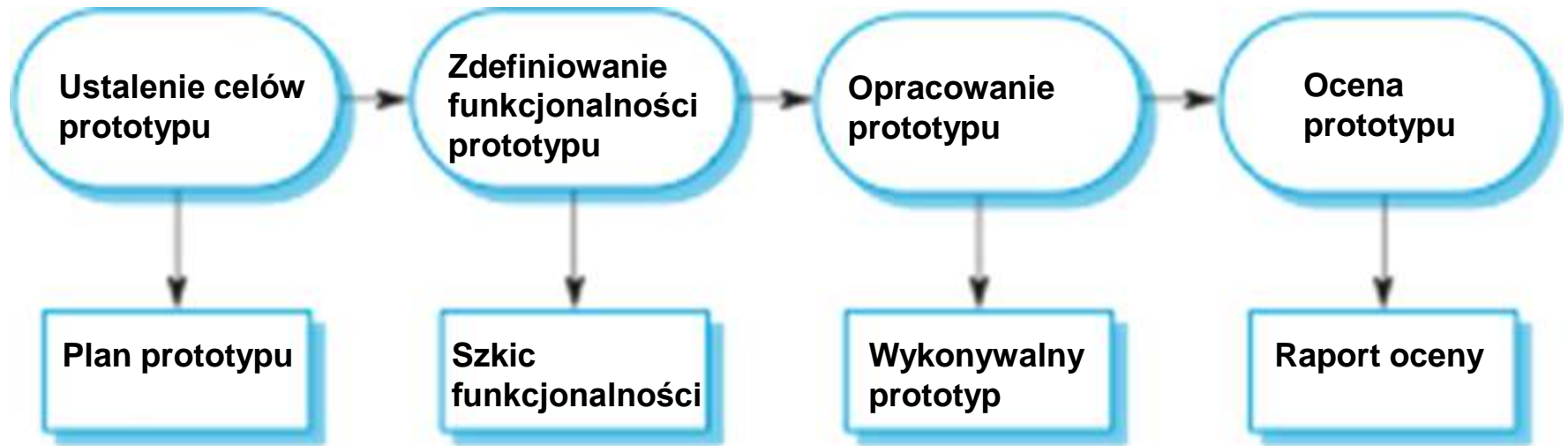
- **Unikanie zmian** - proces IO może obejmować działania, których celem jest przewidzenie ewentualnych zmian zanim spowodują one potrzebę znaczących przeróbek.
 - Na przykład można opracować prototyp systemu w celu zaprezentowania klientom kluczowych cech systemu.
- **Tolerowanie zmian** – proces IO jest konstruowany w taki sposób aby zmiany mogły być wprowadzone przy relatywnie niskim koszcie realizacji.
 - Takie podejście wymaga zazwyczaj jakiejś formy wytwarzania przyrostowego - iteracyjności. Proponowane zmiany mogą być wprowadzanie w ramach przyrostów, które nie zostały jeszcze zrealizowane. Jeżeli jest to niemożliwe wówczas zmiana może być uwzględniona poprzez modyfikację pojedynczego przyrostu (małego fragmentu systemu).

Prototypowanie



- Prototyp to początkowa wersja systemu, której celem jest zademonstrowanie jego koncepcji oraz wypróbowanie wariantów projektowych.
- Prototyp może być wykorzystany w:
 - Procesie inżynierii wymagań jako pomoc w pozyskiwaniu oraz zatwierdzaniu wymagań
 - Procesie projektowym w celu rozpatrzenia możliwych rozwiązań oraz rozwoju projektu interfejsu użytkownika.

Model procesu rozwoju prototypu



Rozwój prototypu



- Może wykorzystywać dedykowane języki i narzędzia
- Jest z założenia niekompletny
 - Prototyp powinien koncentrować się na nie do końca zrozumianych obszarach produktu,
 - Prototyp może pomijać kontrolę błędów.
 - Kwestie niezawodności czy bezpieczeństwa nie są zazwyczaj brane pod uwagę.

Prototyp a dalszy rozwój systemu



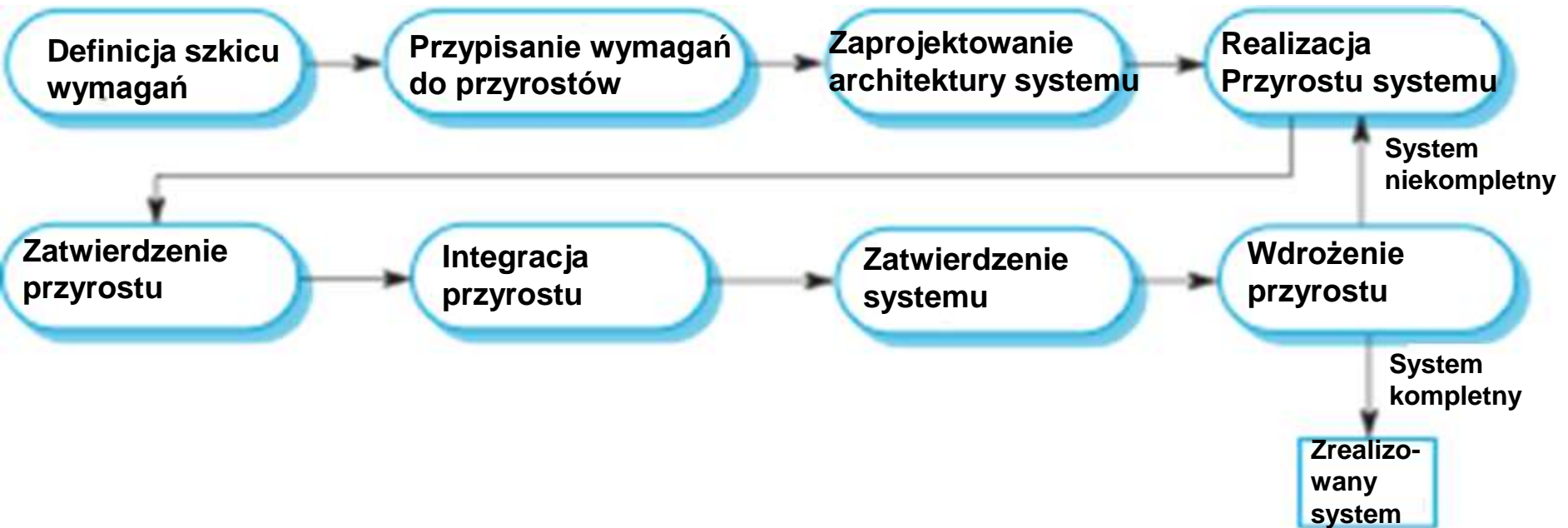
- Prototypy powinny być porzucane po opracowaniu ponieważ nie stanowią dobrej bazy do produkcji systemu:
 - Może być niemożliwe dostrojenie prototypu do spełnienia wymagań нефunkcjonalnych;
 - Prototypy są nieudokumentowane;
 - Z powodu szybkich zmian prototyp ma zazwyczaj mocno zdegradowaną strukturę;
 - Podczas produkcji prototypu omija się standardy jakości, które wykorzystuje organizacja.

Dostarczanie przyrostowe



- Zamiast dostarczać system w jednym kroku, rozwój i dostawy podzielone są na przyrosty. Każdy przyrost dostarcza część wymaganej funkcjonalności.
- Wymagania użytkowników podlegają priorytetyzacji – te, którym przypisano najwyższe priorytety włączane są do wczesnych przyrostów.
- W momencie rozpoczęcia realizacji przyrostu (iteracji) – wybrane wymagania są zamrażane. Natomiast wymagania przeznaczone do kolejnych przyrostów mogą cały czas podlegać ewolucji.

Dostarczanie przyrostowe



Realizacja przyrostowa a dostarczanie przyrostowe



- Realizacja przyrostowa
 - Iteracyjne Budowanie systemu (przyrosty). Ocena każdego przyrostu przed przejściem do kolejnego etapu.
 - Podejście wykorzystywane metodach zwinnych (agile).
 - Ocena wykonywana przez pełnomocnika użytkownika/klienta.
- Dostarczanie przyrostowe
 - Iteracyjne wdrażanie systemu – dostarczenie przyrostu do wykorzystania przez użytkowników.
 - Bardziej realistyczna ocena bazująca na praktycznym wykorzystaniu oprogramowania

Zalety dostarczania przyrostowego



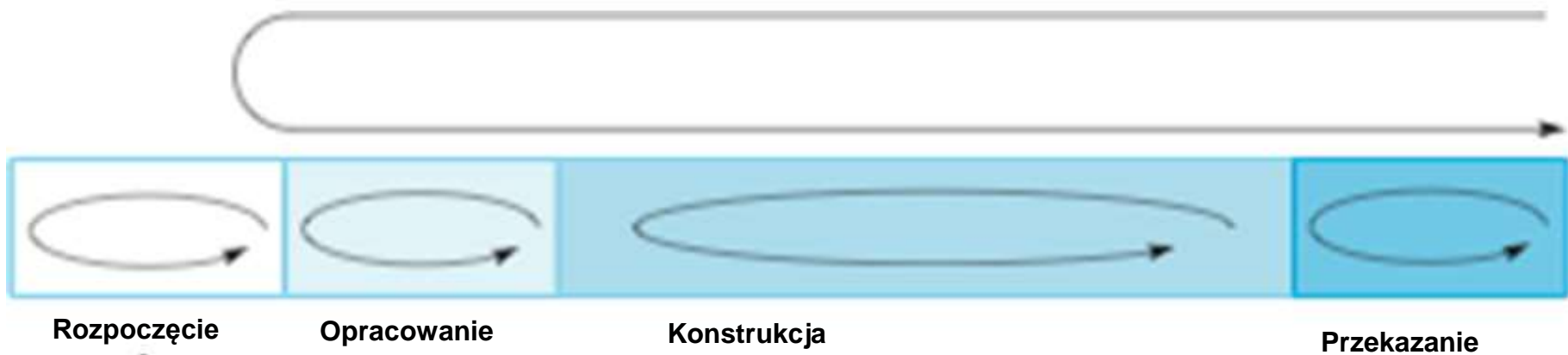
- Każdy przyrost wprowadza elementy wartościowe dla klienta – wybrana funkcjonalność jest dostarczana wcześniej.
- Wczesne przyrosty działają jak prototypu – pomagają pozyskać wymagania dla kolejnych etapów.
- Zmniejszenie ryzyka porażki projektu.
- Usługi systemu o wyższym priorytecie są dłużej testowane.

Problemy dostarczania przyrostowego



- Większość systemów wymaga zestawu podstawowych funkcjonalności, które są wykorzystywane przez pozostałe części systemu.
 - Ponieważ wymagania nie są definiowane szczegółowo do momentu rozpoczęcia przyrostu, który ma je zrealizować, trudno jest dokonać identyfikacji wspólnych udogodnień, które będą wykorzystywane we wszystkich przyrostach..
- Istotą procesu iteracyjnego jest to, że specyfikacja jest opracowywana w połączeniu z oprogramowaniem.
 - W wielu organizacjach, pełna specyfikacja systemu jest częścią umowy dotyczącej jego rozwoju.

Iteracje w procesie RUP





Zwinne wytwarzanie oprogramowania

Agile Software Development



© Scott Adams, Inc./Dist. by UFS, Inc.



Copyright © 2003 United Feature Syndicate, Inc.



**Być może problemy z
wytwarzaniem
oprogramowania mają
podłoże bardziej
socjologiczne niż
technologiczne**

Kilka sentencji na początek



The most important factor in software work is not the tools and techniques used by the programmers, but rather the quality of the programmers themselves.

The working environment has a profound impact on productivity and product quality.

Robert L. Glass, Facts and Fallacies of Software Engineering

You seldom need to take Draconian measures to keep your people working; most of them love their work. You may even have to take steps sometimes to make them work less, and thus get more meaningful work done

Tom DeMarco & Timothy Lister, Peopleware

The only real failure is the failure to learn from failure

Henrik Kniberg & Mattias Skarin, Kanban and Scrum - making the most of both

Błędy też się dla mnie liczą. Nie wykreślam ich ani z życia, ani z pamięci. I nigdy nie winię za nie innych.

Geralt z Rivii

Outline



- Metody zwinne
- Wytwarzanie sterowane planem a wytwarzanie zwinne
- Praktyki inżynierskie - Extreme programming
- Zwinne zarządzanie projektem - Scrum
- Skalowalność zwinnych metod

Szybkie wytwarzanie oprogramowania

Rapid software development



- Szybkość wytworzenia i dostarczenia gotowego produktu jest obecnie jednym z najważniejszych wymagań oprogramowania
 - Środowisko biznesowe podlega ciągłym zmianom – wypracowanie stabilnych wymagań jest praktycznie niemożliwe
 - W odpowiedzi na zmieniające się potrzeby biznesu - oprogramowanie musi szybko ewoluować.
- Szybkie wytwarzanie oprogramowania
 - Specyfikacja, projektowanie i implementacja są połączone.
 - System jest rozwijany w serii wersji, klient jest zaangażowany w ewaluację wersji
 - Interfejsy użytkownika są budowane przy wykorzystaniu IDE i narzędzi graficznych.

Metody zwinne



- **Niezadowolenie z kosztów** związanych z istniejącymi metodami wytwarzania oprogramowania w latach 80-tych i 90-tych ubiegłego wieku doprowadziły do powstania metod zwinnych, które:
 - Skupiają się na bardziej implementacji niż na projektowaniu
 - Bazują na podejściu iteracyjnym
 - Ich celem jest szybkie dostarczenie działającego oprogramowania oraz jego równie szybka ewolucja umożliwiająca spełnienie zmieniających się wymagań.
- **Głównym celem** metod zwinnych jest **redukcja kosztów** procesu Inżynierii Oprogramowania oraz zdolność do szybkiej reakcji na zmiany w wymaganiach nie wymagającej nadmiernych przeróbek.

Agile vs Plan -driven



- Planowanie adaptacyjne
 - vs
 - Planowanie predykcyjne
-
- Ludzie na pierwszym planie
 - vs
 - Procesy na pierwszym planie

Agile manifesto



- Programiści i ich harmonijna współpraca jest ważniejsza od procesów i narzędzi.
- Działające oprogramowanie jest ważniejsze od wyczerpującej dokumentacji.
- Faktyczna współpraca z klientem jest ważniejsza od negocjacji zasad kontraktu.
- Reagowanie na zmiany jest ważniejsze od konsekwentnego realizowania planu.

Manifesto for Agile Software Development

Zasady metod zwinnych



Zasada		Opis
Zaangażowanie klientów		Klienci powinni być zaangażowani w cały proces wytwarzania oprogramowania. Ich rolą jest dostarczanie oraz priorytetyzacja wymagań dla nowego systemu oraz ocena jego iteracji.
Dostarczanie przyrostach	w	Oprogramowanie jest rozwijane w przyrostach we współpracy z klientem określającym wymagania dla każdego przyrostu.
Ludzie a nie proces		Umiejętności zespołu projektowego powinny być rozpoznane i odpowiednio wykorzystywane. Członkom zespołu należy dać możliwość elastycznego rozwijania własnych metod pracy (bez normatywnych procesów).
Zaakceptuj zmiany		Zmiany w wymaganiach są normą i projekt systemu musi to uwzględniać.
Utrzymanie prostoty		Skupienie się na prostocie rozwiązań tak w zakresie budowanego rozwiązania jak i w zakresie procesu jego wytwarzania. Złożoność w systemie powinna być eliminowana, kiedy tylko jest taka możliwość.

Stosowalność zwinnych metod



- Produkty niewielkie lub średniej wielkości przeznaczone na sprzedaż.
- Niestandardowy system rozwijany w ramach organizacji, gdzie łatwo jest zaangażować klienta w proces rozwoju i gdzie nie istnieje wiele zewnętrznych przepisów i regulacji, które wpływają na oprogramowanie.
- Ze względu na skupianie się na małych i silnie zintegrowanych zespołach problemem jest skalowanie metod zwinnych do większych projektów.

Problemy metod zwinnych

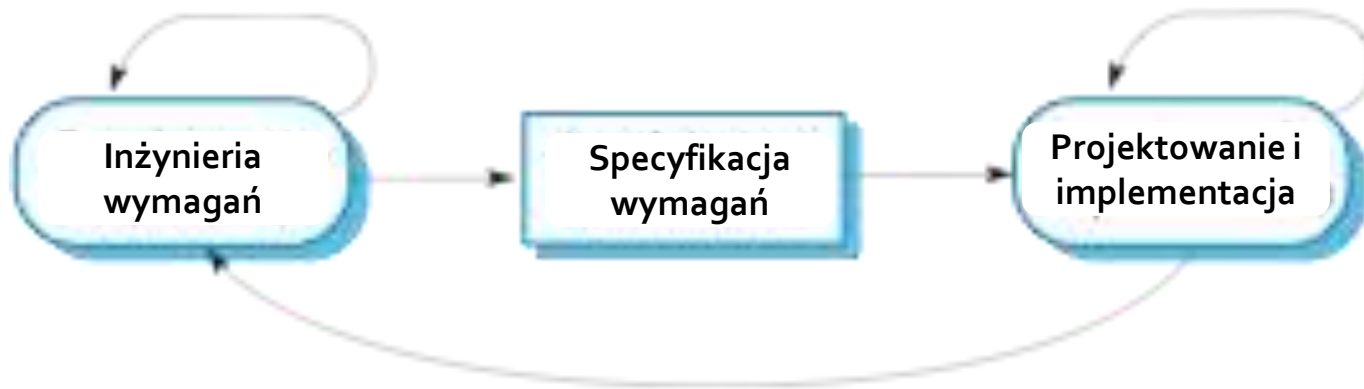


- Może nie być łatwo utrzymać odpowiedniego poziomu zainteresowania klienta zaangażowanego w projekt.
- Członkowie zespołu mogą nie być przyzwyczajeni do poziomu intensywności zaangażowania charakterystycznego dla metod zwinnych.
- Jeżeli grup zainteresowania (ang. stakeholders) jest więcej mogą wystąpić problemy z priorytetyzacją zmian.
- Utrzymanie prostoty systemu wymaga dodatkowych nakładów pracy.
- Kontrakty z klientem mogą stanowić problem – cecha podejścia iteracyjnego.
 - Specyfikacja jest częścią kontraktu. Metody zwinne zakładają przyrostową specyfikację.
 - Metody zwinne muszą polegać na kontraktach w których płaci się za czas wymagany do opracowania systemu.

Wytwarzanie sterowane planem a wytwarzanie zwinne

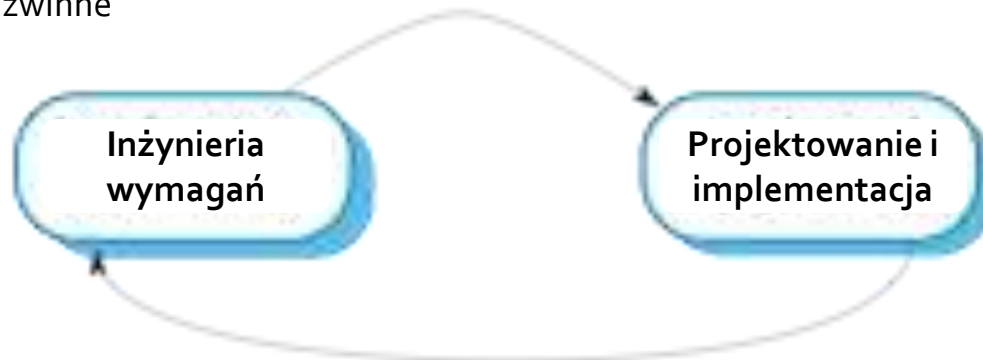


Wytwarzanie sterowane planem



Żądanie zmiany w wymaganiach

Wytwarzanie zwinne



Extreme programming – wydajne (ekstremalne) programowanie



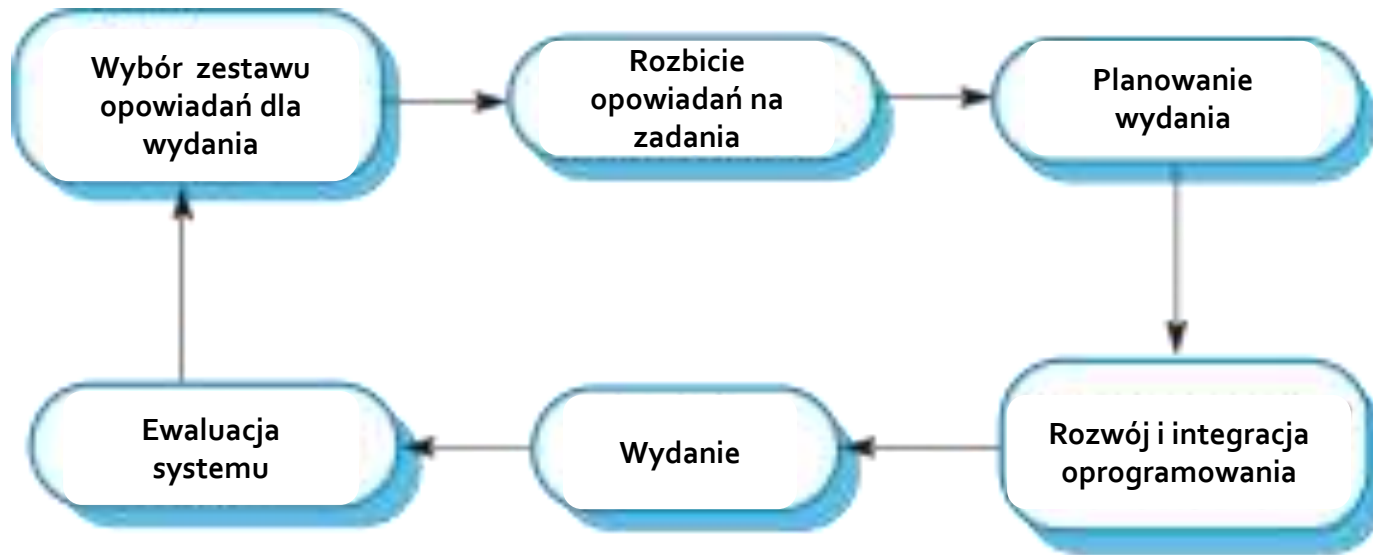
- Extreme = skrajne, radykalne, ostateczne
- Pierwsza popularna metoda zwinna (zbiór praktyk).
- Extreme Programming (XP) przyjmuje skrajne podejście do wytwarzania iteracyjnego:
 - Nowe wersje systemu mogą być budowane nawet kilka razy dziennie
 - Przyrosty są dostarczane do klienta co dwa tygodnie
 - Wszystkie testy muszą być uruchomione dla każdej zbudowanej wersji systemu – wersja jest akceptowana tylko wówczas gdy wszystkie testy zakończyły się powodzeniem.

XP i zasady metod zwinnych



1. Realizacja przyrostowa jest wspierana przez małe i częste wydania systemu.
2. Wymagana jest pełnoetatowe zaangażowanie klienta w prace zespołu (klient jest częścią zespołu)
3. Ludzie a nie proces: programowanie parami, współwłasność kodu, unikanie pracy po godzinach.
4. Zmiany realizowane poprzez regularne wydania.
5. Utrzymywanie prostoty przez ciągłą refaktoryzację.

Cykl wydania w XP



Praktyki XP (1 z 2)



Zasada lub praktyka	Opis
Planowanie przyrostowe	Wymagania są zapisywane są w postaci tzw. „user stories” Są one podstawą wydań, podlegają oszacowaniu. W ramach wydania są rozbijane na zadania do realizacji.
Małe wydania	Jako pierwszy realizowany jest minimalny, użyteczny zestaw funkcjonalności dostarczający wartości biznesowej. Kolejne wydania systemu są częste i przyrostowo dodają nowe funkcjonalności do pierwszego wydania.
Prosty projekt	Projekt obejmuje tylko te elementy, które realizują bieżące wymagania.
Rozwój sterowany testami	Testy dla nowych fragmentów funkcjonalności są tworzone z wykorzystaniem frameworków umożliwiających automatyzację testów jednostkowych i opracowywane są przed implementacją samej funkcjonalności.
Refaktoryzacja	Wszyscy programiści są zobligowani do refaktoryzacji kodu jeżeli tylko pojawi się możliwość ulepszenia. Celem jest utrzymanie prostego kodu..

Praktyki XP (2 z 2)



Programowanie w parach	Programiści pracują parami, kontrolując się wspierając wzajemnie.
Wspólna własność kodu	Pary programistów angażują się w coraz to różne obszary systemu. Zwiększane są kompetencje poszczególnych członków zespołu projektowego oraz poziom odpowiedzialności za całość kodu.
Ciągła integracja	Natychmiast po ukończeniu zadania następuje integracja jego rezultatów z całością systemu i uruchamiane są wszystkie testy (które muszą zakończyć się sukcesem).
Zrównoważone tempo	Nie akceptuje się dużych nadgodzin (praca po godzinach często powoduje obniżenie jakości kodu oraz produktywności).
Klient „w miejscu”	Reprezentant użytkowników systemu powinien być dostępny przez cały czas pracy zespołu. Oznacza to, że klient jest członkiem zespołu projektowego i jest odpowiedzialny za dostarczanie wymagań do zaimplementowania.

Scenariusze wymagań



- W XP, klient będący członkiem zespołu jest odpowiedzialny za podejmowanie decyzji dotyczących wymagań.
- Wymagania użytkowników są wyrażane w terminach scenariuszy zwanych „user stories” (historyjki/opowiadania).
- Scenariusze są zapisywane na kartach a zespół rozbija je na zadania implementacyjne. Zadania te stanowią podstawę do harmonogramowania i szacowania nakładów.
- Klient wybiera scenariusze do realizacji w ramach kolejnego wydania bazując na priorytetach i oszacowaniu nakładów.

Przykładowy scenariusz (story): 'przepisywanie leku'



Przepisywanie leku

Jako lekarz potrzebuję przepisać leki dla pacjenta w przychodni. Karta pacjenta jest już wyświetlona na komputerze i powinienem móc wybrać opcje 'Leki'. Po wybraniu powinienem mieć możliwość wyboru opcji: 'Bieżący lek', 'Nowy lek gotowy' lub 'Nowy lek recepturowy'. Jeżeli wybiorę opcję 'Bieżący lek', system prosi, by sprawdzić dawkę. Jeśli lekarz chce zmienić dawkę, wprowadzam jej nową wartość, a następnie potwierdzam receptę.

Jeśli wybiorę opcję 'Nowy lek gotowy', system zakłada, że wiem, jaki lek przepisać. Lekarz wpisuje kilka pierwszych liter nazwy leku. System wyświetla listę możliwych leków, których nazwa rozpoczyna się tym ciągiem liter. Wybieram wymagany lek a system prosi o potwierdzenie, że wybrany lek jest prawidłowy. Wprowadzam dawkę leku, a następnie potwierdzam receptę.

Jeśli wybiorę opcję 'Nowy lek recepturowy', system wyświetli okno wyszukiwania dla zatwierdzonych receptur. Mogę wyszukać wymagany lek. Wybieram lek i jestem proszony o potwierdzenie prawidłowości wyboru. Wprowadzam dawkę leku, a następnie potwierdzam receptę. System zawsze sprawdza, czy dawka leku jest w poprawnym zakresie. Jeśli nie jest, jestem proszony o zmianę dawki.

Po potwierdzeniu recepty, zostaje ona wyświetlona w celu sprawdzenia. Mogę wybrać opcję 'OK.' albo 'Zmiana'. Jeżeli wybiorę 'OK' recepta zostaje zapisana w bazie danych. Jeżeli wybiorę 'Zmiana' przejdę ponownie do procesu 'Przepisywanie leku'.

Przykłady kart z zadaniami dla scenariusza „przepisywanie leku”



Zadanie 1: Zmiana dawki przepisywanego leku

Zadanie 2: Wybór receptury

Zadanie 3: Kontrola dawki

Sprawdzenie dawki to środek ostrożności, zapobiegający przepisaniu przez lekarza niebezpiecznie małej lub dużej dawki leku. Wykorzystując identyfikator receptury dla nazwy rodzajowej leku pobierana jest maksymalna i minimalna dawka leku. Następuje sprawdzenie przepisanej dawki. Jeżeli nie mieści się ona w granicach dawek minimalnej i maksymalnej leku system informuje o tym komunikatem błędu zbyt niskiej lub zbyt wysokiej dawki. Jeżeli dawka mieści się w zakresie opcja ‘Zatwierdź’ jest uaktywniana.

XP a zmiany w oprogramowaniu



- W klasycznej Inżynierii Oprogramowania kładzie się nacisk na projektowanie dla zmian. Warto jest spędzić więcej czasu i nakładów w celu przewidzenia zmian co ma wymierny wpływ na redukcję kosztów w późniejszych fazach cyklu życia oprogramowania.
- W XP zakłada się, że nie warto tego robić ponieważ zmian nie da się miarodajnie przewidzieć.
- W zamian za to proponowane jest podejście oparte o stałe ulepszanie kodu (refaktoryzację). Zakłada się, że kod dobrej jakości ułatwi wprowadzenie zmiany w momencie zapotrzebowania na nią.

Refaktoryzacja



- Zespół programistów poszukuje możliwości udoskonalenia oprogramowania – jeżeli taka możliwość zostanie znaleziona zmiany zostają wprowadzone niezależnie od tego czy istnieje obecnie na nie zapotrzebowanie.
- Zaletą refaktoryzacji jest zwiększenie zrozumiałości kodu – co potencjalnie redukuje potrzebę istnienia dokumentacji.
- Wprowadzanie zmian jest prostsze ponieważ struktura kodu jest lepsza.
- Oczywiście pewne zmiany wymagają refaktoryzacji na poziomie architektury – koszty takie zmiany mogą być znaczne.

Przykłady refaktoryzacji



- Reorganizacja hierarchii klas w celu usunięcia powtórzeń w kodzie.
- Uporządkowanie i zmiana nazw atrybutów i metod w celu lepszego odzwierciedlenia ich semantyki.
- Zastąpienie kodu „inline” odwołaniami do metod z bibliotek.

Testowanie w XP



- Testowanie jest centralnym punktem XP. W ramach XP wypracowano podejście, w którym zakłada się cały program jest testowany po każdej zmianie.
- Przystosowanie XP do testów:
 - Rozwój sterowany testami (test-first development/test-driven development).
 - Przyrostowe testy.
 - Zaangażowanie użytkowników w opracowywanie testów oraz ich walidację.
 - Wykorzystanie narzędzi umożliwiających automatyczne uruchamianie testów w trakcie budowania nowego wydania.

Wytwarzanie sterowane testami

Test-first development



- Pisanie testów przed kodem implementującym precyzuje wymagania, które mają być zrealizowane.
- Testy są pisane w postaci programów a nie danych – mogą być uruchamiane automatycznie. Zazwyczaj opierają się na frameworkach takich jak JUnit, NUnit, TestNG.
- Wszystkie istniejące oraz nowo dodane testy są uruchamiane automatycznie w trakcie dodawania nowej funkcjonalności.

Zaangażowanie klienta w proces testowania



- Rolą klienta w procesie testowania jest pomoc w opracowywaniu testów akceptacyjnych dla scenariuszy, które będą implementowane w ramach kolejnego przyrostu i staną się elementem następnego wydania systemu.
- Przedstawiciel klienta będący członkiem zespołu pisze testy w ramach procesu implementacji.
- Problem polega na tym, że osoby pełniące role klienta mają zazwyczaj ograniczony czas i praktycznie nigdy nie mogą być zaangażowane „na pełny etat”. Często uważają, że wystarczy, iż dostarczyli wymagania i trudno jest ich przekonać do włączenia się w proces testowania.

Przypadek testowy dla zadania kontroli dawki leku



Test 4: Kontrola dawki

Wejście:

1. Wartość liczbowa (w miligramach) reprezentująca pojedynczą dawkę leku.
2. Wartość liczbowa reprezentująca dzienną liczbę pojedynczych dawek.

Testy:

1. Test wartości wejściowych w których pojedyncza dawka jest prawidłowa ale dzienna częstotliwość jest za duża.
2. Test wartości wejściowych w których pojedyncza dawka jest za duża lub za mała.
3. Test wartości wejściowych w których pojedyncza dawka x częstotliwość jest za duża lub za mała.
4. Test wartości wejściowych w których pojedyncza dawka x częstotliwość mieści się w prawidłowym zakresie.

Wyjście:

5. Zatwierdzenie lub komunikat informujący, że dawka nie mieści się w zakresie.

Automatyzacja testowania



- Testy są pisanie w postaci wykonywalnych komponentów zanim zadanie, które mają testować zostanie zaimplementowane
 - Komponenty testujące powinny być na tyle niezależne, żeby mogły być uruchamiane w izolacji. Powinny symulować dane wejściowe i sprawdzać czy dane wyjściowe są zgodne ze specyfikacją wyjść. Wykorzystanie frameworków przeznaczonych do automatyzacji testów upraszcza ten proces.
- Ponieważ testy są automatyzowane – zawsze dostępna jest pula testów, który może być szybko uruchomiona.
 - W momencie dodania nowej funkcjonalności można łatwiej wyłapać potencjalne problemy jakie nowy kod wprowadził do systemu.

Trudności związane z testowaniem w XP



- Programiści przedkładają programowanie nad testy – powoduje to „chodzenie na skróty” – pisanie niekompletnych testów, które nie sprawdzają wszystkich sytuacji, które mogą zaistnieć.
- Niektóre testy trudno jest pisać przyrostowo. Na przykład dla skomplikowanego interfejsu użytkownika trudno jest pisać testy jednostkowe, dla kodu implementującego logikę wyświetlania czy przepływ pomiędzy ekranami.
- Trudno jest ocenić kompletność zbioru testów. Możliwe jest, iż pomimo istnienia dużego zbioru testów mogą one nie pokrywać całości funkcjonalności.

Programowanie w parach (1 z 2)



- W metodzie XP, pary programistów wspólnie pracują nad kodem (siedzą przy jednej stacji roboczej i wspólnie pracują nad systemem).
- Pomaga to rozwijać poczucie wspólnej własności kodu i wiedzy na jego temat w obrębie zespołu (pary są tworzone dynamicznie).
- Technika może być postrzegana jako nieformalna metoda przeglądów – każda linia kodu jest przejrzana przez więcej niż jedną osobę.
- Takie podejście zachęca do refaktoryzacji – a na tym może skorzystać cały zespół.

Peer reviews are both technical and sociological. Paying attention to one without the other is a recipe for disaster.

Programowanie w parach (2 z 2)



- Ważnym aspektem jest współdzielenie wiedzy
 - redukcja ryzyka związanego z odejściem członka zespołu.
- Badania wykazują, że produktywność par programistów jest podobna do produktywności dwóch programistów pracujących niezależnie
 - (niektóre wykazują nawet, że para jest produktywniejsza)

Agile in the Real World

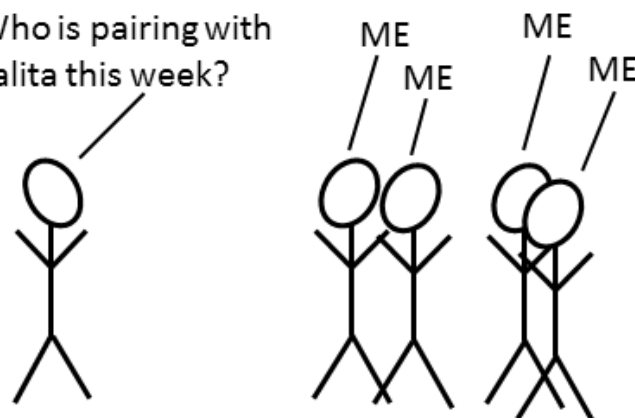
User Stories

So I met this new girl on Facebook yesterday...



Pair Programming

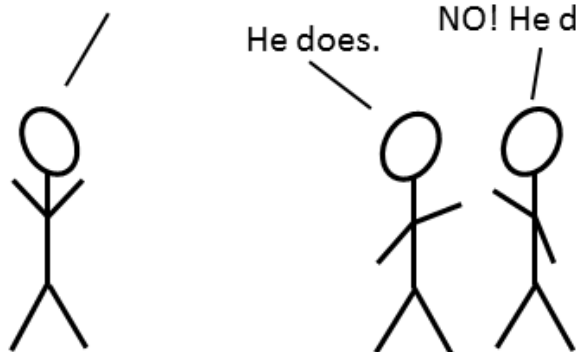
Who is pairing with Lalita this week?



Collective Ownership

We found a new bug in this module. Who owns it?

He does. NO! He does.



Short Release Cycles

Announcing Microsoft Windows 1128. In this release we have added support for Magenta as a background color for the Windows clock. Next month we will RTM Windows 1129 with support for teal and beige.



Podsumowanie



- Metody zwinne są metodami przyrostowymi ukierunkowanymi na szybkie wytwarzanie, częste wydania, zmniejszenie ogólnych kosztów projektu oraz produkcję wysokiej jakości kodu. Ich cechą jest bezpośrednie zaangażowanie klientów w proces wytwarzania.
- Decyzja o tym czy korzystać z metod zwinnych czy też opartych na planowaniu powinna być podejmowana na podstawie typu oprogramowania, możliwościach zespołu projektowego oraz kultury organizacji realizujące projekt.
- Przykładem metody zwinnej jest Extreme programming. Integruje ona praktyki takie jak częste wydania, ciągła integracja czy też bezpośrednie zaangażowanie klienta.
- Silną stroną metodyki eXtreme Programming jest opracowywanie zautomatyzowanych testów przed implementacją danej cechy systemu. Wszystkie testy muszą być wykonane poprawnie w celu zintegrowania przyrostu z istniejącym systemem.



Zwinne zarządzanie projektami

Agile Software Management

Zwinne zarządzanie projektem



- Podstawowym obowiązkiem kierownika projektu jest zarządzanie projektem, w taki sposób, że oprogramowanie jest dostarczane na czas i w ramach zaplanowanego budżetu.
- Standardowe podejście do zarządzania projektami oparte jest na podejściu sterowanym planem. Kierownicy przygotowujące plan dla projektu określając to, co powinno być dostarczone, kiedy powinno być dostarczone i kto będzie pracował nad rozwojem projektu.
- Zwinne zarządzanie projektem wymaga podejścia, które jest dostosowane do przyrostowego rozwoju i mocnych stron poszczególnych zwinnych metod.

Metodyka Scrum



- Scrum jest ogólną metodą zwinną ale skupia się przede wszystkim na zarządzaniu wytwarzaniem iteracyjnym. Mniejszy nacisk kładziony jest na specyficzne praktyki.
- Scrum składa się z trzech faz
 - Faza inicjacyjna - ustalane są ogólne cele projektu oraz projektowana jest architektura systemu. Budowany jest rejestr wymagań (product backlog).
 - Sprinty – Seria iteracji, w której każda realizuje przyrost systemu. Praca bazuje na rejestrze zadań sprintu (sprint backlog)
 - Faza końcowa projektu – zamykanie projektu, uzupełnienie dokumentacji (pomoc, dokumentacja użytkownika), ocena lekcji wyciągniętej z projektu.



Proces Scrum



Blending Scrum and Kanban to Create an End-to-End Agile Enterprise
Mike Cottmeyer

Role

- Product Owner
- Scrum Master
- The team

Sprint (1 z 2)



1. Sprints mają stałą długość (zazwyczaj 2-4 tygodnie) – są odpowiednikiem procesu opracowywania wydania systemu w metodzie XP.
2. Punktem wyjścia dla planowania jest tzw. **product backlog** czyli priorytetyzowany rejestr wymagań do wykonania w trakcie trwania projektu ustalana z klientem i którą zarządza rola właściciel produktu (**ang. product owner**).
3. Dla pojedynczego sprintu **zespół** (**ang. Team**) wraz z klientem wybiera elementy z **rejestru wymagań**, które mają zostać wykonane w jego ramach
 - Na tej podstawie tworzony jest rejestr zadań sprintu (**sprint backlog**).

Sprint (2 z 2)



4. Po ustaleniu zestawu cech do realizacji zespół organizuje się w celu wytworzenia oprogramowania. W tej fazie sprintu **zespół (ang. Team)** zostaje odizolowany od klienta oraz organizacji. Kanał komunikacyjny realizowany jest przez członka zespołu o roli **Scrum master**.
5. Odpowiedzialnością roli **Scrum master** jest ochrona zespołu przed zakłóceniami zewnętrznymi oraz mentoring i monitorowanie postępów.
6. Po zakończeniu sprintu, jego rezultaty są przeglądane i prezentowane udziałowcom. Rozpoczyna się kolejny sprint.

Praca zespołowa w Scrum



- **Scrum master** jest koordynatorem, który:
 - Organizuje codzienne spotkania, monitoruje zawartość pozostałą w **sprint backlog** (śledzi postępy w realizacji zadań),
 - dokonuje pomiarów,
 - zapisuje podjęte decyzje,
 - komunikuje się z klientami oraz kadrą zarządczą organizacji.
- Cały zespół bierze udział w codziennych krótkich spotkaniach gdzie członkowie zespołu informują się o:
 - postępach poczynionych od ostatniego spotkania,
 - problemach, które się pojawiły w trakcie prac,
 - Planach na najbliższy dzień pracy.
- Każdy członek zespołu wie co się dzieje i, jeżeli pojawią się problemy możliwa jest zmiana krótkoterminowych planów w celu skierowania sił na pokonanie przeciwności.

Zalety metody Scrum



- Produkt zostaje podzielony na zestawy możliwych do zarządzania i zrozumiałych elementów.
- Dzięki lepszemu zrozumieniu wymagań postęp prac nie jest wstrzymywany.
- Cały zespół jest informowany na bieżąco, komunikacja jest ulepszona.
- Klienci otrzymują przyrosty na czas i przekazują informacje zwrotne o działaniu systemu.
- Budowane jest zaufanie pomiędzy klientami a zespołem projektowym, budowany jest pozytywny klimat w którym każdy oczekuje sukcesu projektu.

**NOT
CHECKED OUT**

CHECKED OUT

DONE! :o)

SPRINT GOAL: BETA-READY RELEASE!

**MIGRATION
TOOL**

Impl.
migration
8d

BACKOFFICE

LOGIN

Integr.
with
JBoss
2d

Impl
GUI
1d

**BACKOFFICE
USER ADMIN**

Write
failing
test
3d

GUI
design
(CSS)
1d

Clarify
require-
ments
2d

Impl
GUI
6d

GUI
spec
2d

Tapestry
spike
1d 2d

Write
failing
test
2d

DEPOSIT

Code
cleanup
1d

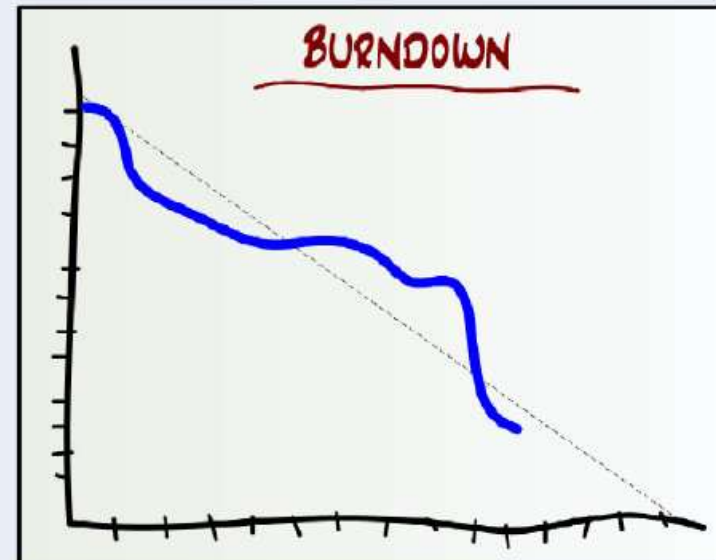
Integr
test
2d 0.5d

Write
failing
test
2d

DAO

DB
design
2d 1d

Write
failing
test
2d 3d



UNPLANNED ITEMS

NEXT

Fix memory
leak
(125)
Write
failing
test

Sales support
3d

Write
whitepaper
4d

WITHDRAW

Skalowanie metod zwinnych



- Metody zwinne udowadniają swoją przydatność w obszarze projektów małego i średniego rozmiaru, które mogą być realizowane w ramach małego, nierozproszonego zespołu.
- Twierdzi się czasem, że sukces tych metod spowodowany jest usprawnieniem komunikacji, które możliwe jest właśnie ze względu na niewielkie kolokowane zespoły.
- Skalowanie metod zwinnych w kierunku większych projektów wymaga zmiany ich w taki sposób, aby można je było wykorzystać w większych, dłużej trwających projektach, nad którymi pracuje wiele rozproszonych.

Wytwarzanie dużych systemów

(1 z 2)



1. Duże systemy są zazwyczaj kolekcją niezależnych, współpracujących systemów, z których każdy jest realizowany przez osobny zespół. Często zespoły te pracują w różnych lokacjach (i strefach czasowych).
2. Duże systemy są mocno powiązane z istniejącymi systemami (na zasadzie zawierania oraz interakcji). Wiele wymagań dotyczy właśnie tych interakcji i nie nadaje się do realizacji za pomocą elastycznego i przyrostowego procesu.
3. Kiedy kilka systemów jest integrowanych w celu zbudowania nowego systemu to duża część rozwoju związana jest z konfiguracją niż opracowaniem nowego kodu.

Wytwarzanie dużych systemów (2 z 2)



4. Duże systemy oraz proces ich rozwoju są często ograniczane przez zewnętrzne regulacje
 - Ogranicza to sposoby na jakie mogą być rozwijane.
5. Duże systemy powiązane są z długimi zamówieniami i czasem rozwoju.
 - Trudno jest utrzymać spójne zespoły bo nieuniknione jest, w długim horyzoncie czasowym, przeznaczanie zasobów ludzkich do różnych zadań i projektów.
6. Duże systemy mają najczęściej zróżnicowany zestaw stron zainteresowanych w jego realizacji. Jest praktycznie niemożliwe, aby zaangażować wszystkie te podmioty w procesie rozwoju.

Scaling out i scaling up



- Scaling up – Skalowanie wertykalne
 - Wykorzystanie metod zwinnych do rozwoju dużych systemów które nie mogą być realizowane z wykorzystaniem małych zespołów.
- Scaling out - Skalowanie horyzontalne
 - Wprowadzanie metod zwinnych w dużych organizacjach z wieloletnim doświadczeniem w rozwijaniu oprogramowania.
- Skalowanie musi zachowywać fundamenty zwinności
 - Elastyczne planowanie, częste wydania, ciągła integracja, wytwarzanie sterowane testami, dobrą komunikację w zespole.

Podsumowanie



- Metodyka Scrum udostępnia szkielet zarządzania projektem. Zorganizowana jest wokół zbioru przebiegów (sprintów) realizujących przyrost systemu w ramach ustalonego czasookresu.
-
- Skalowanie metod zwinnych do potrzeb rozwoju dużych systemów jest trudne. Duże systemy wymagają wstępnego projektowania oraz dokumentacji.