1 INTRO

OPROGRAMOWANIE JAKO PRODUKT

OPROGRAMOWANIE GENERYCZNE / OGÓLNE:

- odrębne systemy wytwarzane i sprzedawane szerokiemu zakresowi klientów
- Przykłady: programy graficzne, oprogramowanie wspierające zarządzanie projektami, systemy CAD, systemy specyficzne dla określonej dziedziny np. system obsługi apteki

OPROGRAMOWANIE DOPASOWANE:

- wyprodukowane dla konkretnego klienta zgodnie z dostarczoną specyfikacją
- Przykłady: systemy wbudowane, systemy kontroli ruchu lotniczego, systemy monitoringu i sterowania ruchem ulicznym

SPECYFIKACJA PRODUKTU

OPROGRAMOWANIE GENERYCZNE:

- specyfikacja jest własnością **wykonawcy**
- decyzje zw. ze zmianami podejmuje **wykonawca**

OPROGRAMOWANIE DOPASOWANE:

- specyfikacja jest własnością **klienta**
- decyzje zw. ze zmianami podejmuje **klient**

MODEL DOSTARCZANIA

INSTALACJA PO STRONIE KLIENTA

- System uruchomiony na infrastrukturze klienta (własnej lub dzierżawionej)
- Klient -> właściciel

UDOSTĘPNIANIE ZDALNE

- System uruchomiony na infrastrukturze dostawcy
- oprogramowanie jako usługa (Software as a Service)
- Dostawca -> właściciel, klient -> dzierżawca

ATRYBUTY DOBREGO OPROGRAMOWANIA

- Oprogramowanie powinno dostarczać użytkownikom wymaganą funkcjonalność i wydajność
- Oprogramowanie powinna cechować:
    1. **Zarządzalność:** musi ewoluować w odpowiedzi na zmieniające się wymagania
    2. **Wiarygodność i bezpieczeństwo:** można na nim polegać
    3. **Sprawność:** nie powinno marnować zasobów systemu
    4. **Użyteczność:** musi być użyteczne dla użytkowników, dla których zostało zaprojektowane

INŻYNIERIA OPROGRAMOWANIA

Inżynieria Oprogramowania to **dyscyplina inżynieryjna** dotycząca **wszystkich aspektów związanych z wytwarzaniem** oprogramowania

- poczynając od wstępnych zarysów specyfikacji
- kończąc na wspieraniu systemu w czasie jego działania

Dyscyplina inżynieryjna

- wykorzystanie adekwatnych teorii i metod do rozwiązania problemów z uwzględnieniem ograniczeń organizacyjnych i finansowych

Wszystkie aspekty związane z wytwarzaniem

- nie tylko techniczny proces produkcji, ale również zarządzanie projektem oraz rozwój narzędzi, metod itp. wspierających rozwój oprogramowania

CZYNNOŚCI PROCESU INŻYNIERII OPROGRAMOWANIA

Specyfikacja oprogramowania

- klient i inżynierowie definiują oprogramowanie, które ma zostać wyprodukowane oraz określają ograniczenia przy których ma realizować swoje zadania

Rozwój (produkcja) oprogramowania

- oprogramowanie jest projektowane i implementowane

Zatwierdzenie (walidacja) oprogramowania

- oprogramowanie jest sprawdzane pod kątem spełniania wymagań klienta

Ewolucja oprogramowania

- oprogramowanie jest modyfikowane w celu uwzględnienia zmian w wymaganiach klienta oraz środowiska

OGÓLNE PROBLEMY MAJĄCE WPŁYW NA OPROGRAMOWANIE

- **Heterogeniczność** - coraz częściej systemy muszą działać w środowiskach rozproszonych, które obejmują różne rodzaje systemów komputerowych i urządzeń mobilnych
- **Zmiany środowiska biznesowego i społecznego** - biznes i społeczeństwo podlegają szybkim zmianom (rozwój gospodarczy, nowe technologie). Powoduje to potrzebę wprowadzania nowych zmian w istniejącym oraz szybka produkcję nowego oprogramowania
- **Bezpieczeństwo i zaufanie** - ponieważ oprogramowanie splata się z niemal wszystkimi aspektami naszego życia, bardzo istotne jest to, czy możemy mu zaufać

INŻYNIERIA OPROGRAMOWANIA A INNE DYSCYPLINY

- **Informatyka** - związana jest z podstawami teoretycznymi; inżynieria oprogramowania związana jest z praktyką budowania i dostarczania użytecznego oprogramowania
- **Inżynieria systemów** - jest związana z wszelkimi aspektami budowo i ewolucji złożonych systemów, w których oprogramowanie może grać główną rolę

NIEJEDNOLITOŚĆ INŻYNIERII OPROGRAMOWANIA

Istnieje wiele różnych typów oprogramowania - nie istnieje zestaw uniwersalnych technik i narzędzi, który można by zastosować w produkcji

Metody i narzędzia wykorzystywane do produkcji oprogramowania **będą się różnić** w zależności od

- typu rozwijanej aplikacji
- wymagań klienta
- umiejętności, wiedzy i doświadczenia zespołu projektowego

TYPY APLIKACJI

Aplikacje samodzielne (*lokalne*) - aplikacje działające na lokalnym komputerze i posiadające wszystkie wymagane funkcjonalności bez potrzeby łączenia się z siecią

Interaktywne aplikacje transakcyjne - aplikacje wykonywane na zdalnym komputerze, do których użytkownicy mają dostęp za pośrednictwem własnego komputera (*aplikacje webowe*)

Wbudowane systemy kontrolne - oprogramowanie kontrolujące i zarządzające działaniem urządzeń. Ilościowo, systemów tego typu jest prawdopodobnie więcej niż wszystkich innych razem wziętych

Systemy przetwarzania wsadowego - systemy biznesowe przeznaczone do przetwarzania dużych danych ilości danych (*wsadów*). Przetwarzają duży zestaw pojedynczych danych wejściowych w celu utworzenia odpowiadającego im rezultatu.

Systemy rozrywkowe - systemy, przede wszystkim do użytku osobistego, przeznaczone do rozrywki

Systemy do modelowania i symulacji - systemy opracowywane przez naukowców i inżynierów dla celów modelowania fizycznych procesów lub sytuacji, które obejmują wiele odrębnych, ale będących ze sobą w interakcji obiektów

Systemy gromadzenia danych - zbierające za pośrednictwem czujników dane ze środowiska i przesyłające je do innych systemów w celu dalszego przetwarzania

Systemy systemów - złożone z wielu innych systemów

FUNDAMENTY INŻYNIERII OPROGRAMOWANIA

Niezależnie od typu oprogramowania i techniki jego opracowania istnieje zbiór podstawowych zasad, które należy stosować

- system powinien być wytwarzany przy użyciu zarządzalnego i zrozumiałego procesu
- niezawodność i wydajność są ważne
- zrozumienie wymagań i zarządzanie specyfikacją
- tam, gdzie jest to właściwe, zamiast budowania nowego powinno się powtórnie wykorzystać oprogramowanie, które już zostało opracowane

INŻYNIERIA OPROGRAMOWANIA A INTERNET

Internet jest obecnie platformą wykonawczą aplikacji. Organizacje coraz intensywniej angażują się w rozwój systemów bazujących na technologiach internetowych kosztem systemów lokalnych

Usługi internetowe pozwalają na dostęp do funkcjonalności za pośrednictwem internetu

Chmury obliczeniowe to podejście do dostarczania usług komputerowych w którym aplikacje wykonywane są zdalnie w "chmurze". Użytkownik nie kupuje oprogramowania tylko płaci za jego użycie

INŻYNIERIA SYSTEMÓW INFORMATYCZNYCH

Internet doprowadził do znacznej zmiany w sposobie organizacji oprogramowania biznesowego

Dominującym podejściem w konstrukcji oprogramowania webowego jest wielokrotne użycie - budując tego typu systemy skupiamy się na wymyśleniu w jaki sposób można złożyć system z istniejących komponentów oraz systemów.

Systemy webowe powinny być rozwijane i dostarczane przyrostowo - obecnie powszechnie uznaje się, że nie jest możliwe z góry określenie wszystkich wymagań dla takich systemów.

Interfejsy użytkownika są ograniczane możliwościami przeglądarek internetowych

- technologie takie jak AJAX pozwalają na realizację rozbudowanych i szybkich interfejsów w ramach przeglądarki. Na szeroką skalę wykorzystywane są formularze webowe oraz skrypty wykonywane lokalnie
- obecnie systemy internetowe zaczynają być konkurencyjne z systemami PC w zakresie interfejsów użytkownika

WYZWANIA W DUŻYCH PROJEKTACH

Zbudowanie skomplikowanego systemu jest dużym wyzwaniem

- duży nakład pracy
- wysokie koszty
- długi czas wytworzenia
- zmieniające się potrzeby użytkowników
- ryzyko - porażki projektu, brak akceptacji przez użytkowników, małej wydajności, trudnej konserwacji

(Inaczej niż w przypadku jednorazowych programów, gdzie często twórca jest użytkownikiem)

SUKCES PROJEKTU OPROGRAMOWANIA

Kiedy można uznać, że oprogramowanie jest sukcesem?

- projekt został ukończony
- oprogramowanie jest użyteczne
- oprogramowanie jest "używalne"
- oprogramowanie jest używane
- oprogramowanie jest opłacalne i zarządzalne

Przyczyny porażki

- poślizg czasowy
- przekroczenie zakładanego budżetu
- nie rozwiązywanie problemów użytkownikom
- słaba jakość
- problemy z zarządzaniem

Do tych problemów prowadzi najczęściej doraźność projektów software'owych

- brak planowania prac
- brak zdefiniowania tego, co ma zostać zrealizowane
- nie zrozumienie wymagań użytkowników

- brak kontroli i przeglądów
- niekompetencja
- nie zdawanie sobie sprawy z rzeczywistych kosztów i potrzebnego nakładu pracy (przez twórców i użytkowników)

INNE DYSCYPLINY

Wielkie projekty nie są niczym nadzwyczajnym, tak jak i ich sukcesy

- mosty, zapory, stadiony
- elektrownie
- samoloty, satelity

Zwykle rozwiązaniem jest "inżynieria"

- projektowanie i konstruowanie obiektów oraz urządzeń technicznych efektywnie spełniających wymagania z punktu widzenia kosztów, jakości, wydajności

INŻYNIERIA

Wymaga dobrze zdefiniowanego podejścia: powtarzalnego, przewidywalnego

Duże projekty wymagają zarządzania samym projektem

- zarządzanie zasobami, budżetem, terminarzem
- skala robi ogromną różnicę

Jakość jest wartością samą w sobie - ludzie chcą płacić za jakość!

DUŻE PROJEKTY

- Angażują różnych specjalistów
- Stały nadzór w celu zapewnienia jakości
- Wiele rezultatów (plan, model, diagram struktury)
- Normy, przepisy, konwencje, które powinny być przestrzegane
- Zdefiniowane kroki, etapy, wykonywane przeglądy, postęp jest widoczny

Oprogramowanie różni się od innych produktów

- Koszty koncentrują się na rozwoju
- Konserwacja to poprawianie błędów i udoskonalanie oraz dodawanie nowych funkcjonalności
- Postęp projektu trudno jest mierzyć

2 PROCES INŻYNIERII OPROGRAMOWANIA

Kiedy opisujemy proces to zazwyczaj myślimy o czynnościach tego procesu oraz porządku w jakim te czynności winny być wykonywane

Opis procesu może również zawierać

- **Produkty** - stanowiące rezultaty czynności procesu
- **Role** - odzwierciedlające odpowiedzialności osób zaangażowanych w procesie
- **Warunki wstępne i końcowe** - które muszą być prawdziwe, odpowiednio przed i po wykonaniu czynności procesu lub wytworzeniu produktu

PROCESY STEROWANIA PLANEM I PROCESY ZWINNE

Procesy sterowane planem - wszystkie czynności procesu są zaplanowane. Pomiary postępów są odnoszone do tego planu

Procesy zwinne - planowanie jest przyrostowe, co ułatwia zmianę procesu w przypadku zmiany wymagań

W praktyce większość procesów zawiera elementy obu podejść

Nie ma dobrych lub złych procesów Inżynierii Oprogramowania

ULEPSZANIE PROCESÓW

- dobre praktyki
- metody IO
- standaryzacja procesu:
    - redukcja czasu szkolenia
    - automatyzacja wsparcia procesu

MODELE PROCESU IO

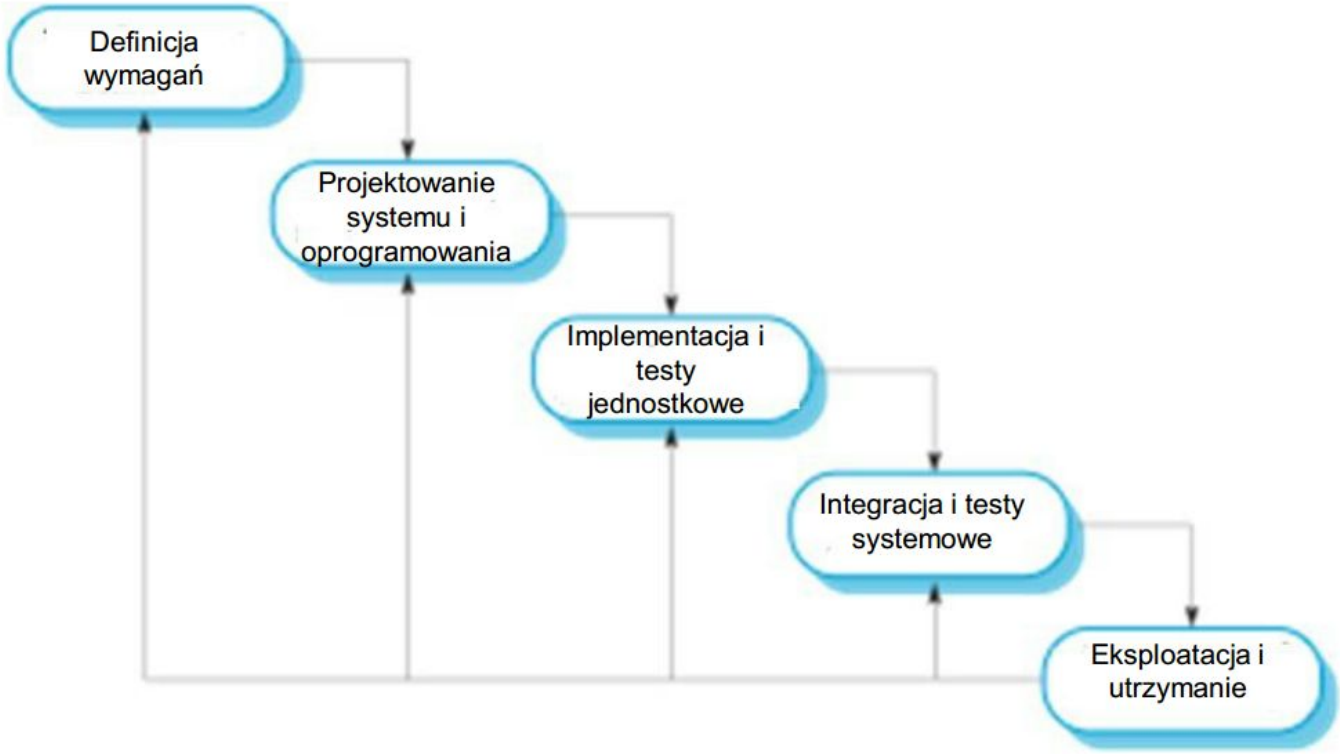
Model kaskadowy - model sterowany planem. Oddzielne fazy specyfikacji i rozwoju

Realizacja przyrostowa - specyfikacja, wytwarzanie i zatwierdzanie są przeplatane. Może być sterowana planem lub zwinna

Inżynieria zorientowana na wielokrotne użycie - 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 modeli.

MODEL KASKADOWY



Model kaskadowy składa się z wydzielonych faz:

- analiza i definiowanie wymagań
- projektowanie systemu i oprogramowania
- 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.

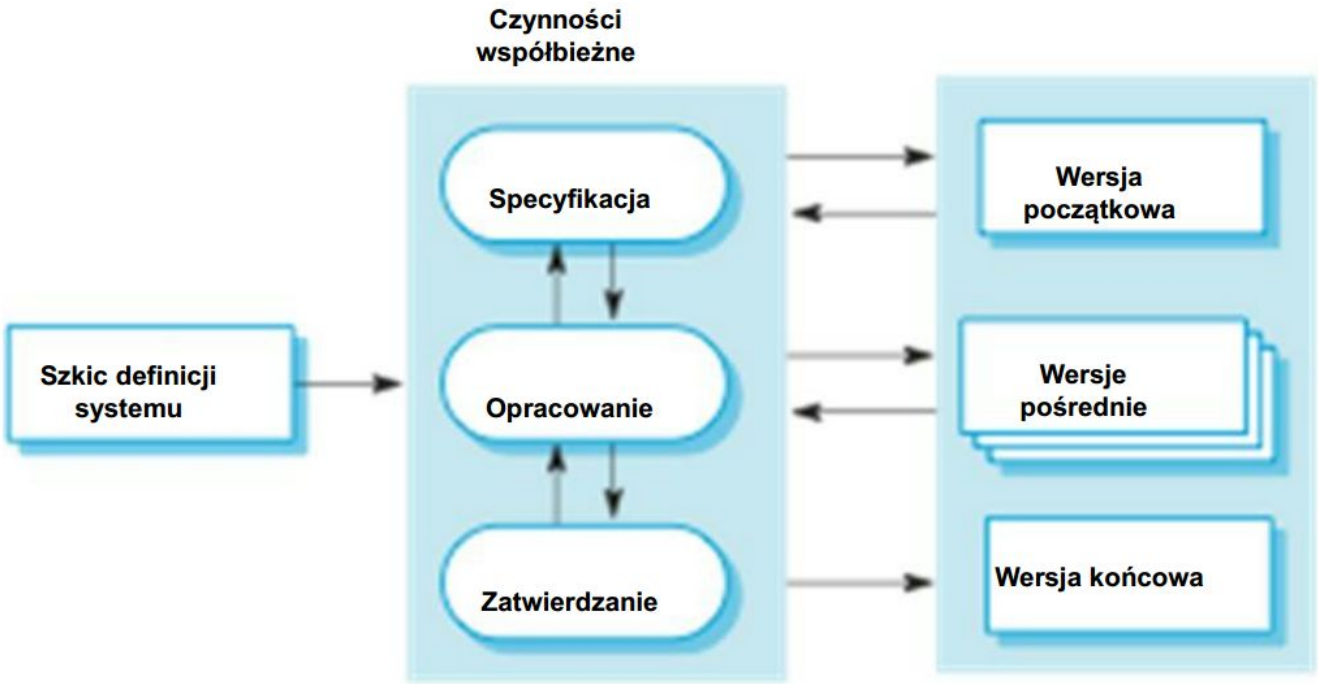
Brak elastyczności poprzez podział projektu na oddzielne etapy powoduje trudności w sytuacji, gdy pojawią się zmiany wymagań

- 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 wykonywane są w kilku miejscach

- sterowana planem natura modelu kaskadowego pomaga w koordynacji prac

REALIZACJA PRZYROSTOWA



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

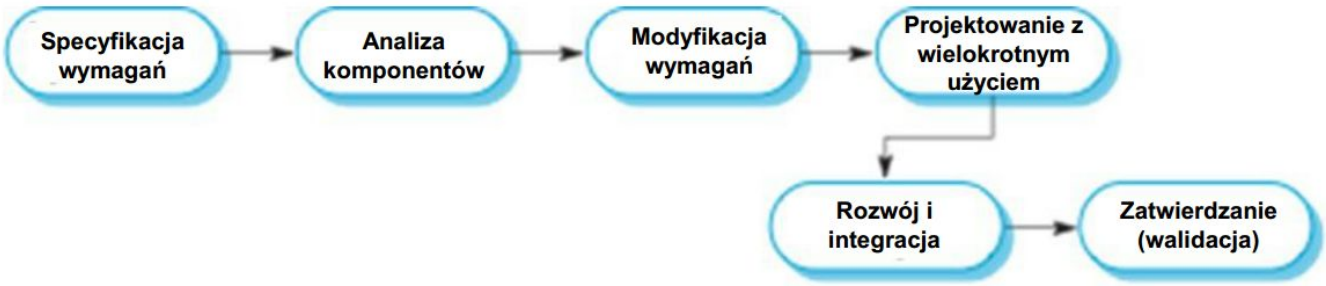
- klienci mogą wykorzystać oprogramowanie wcześnie w stosunku do tego co oferuje model kaskadowy

Problemy przy realizacji przyrostowej

Niejawność procesu - wymaga regularnych rezultatów pozwalających na pomiar postępu. Jeżeli system jest tworzony szybko, nie opłaca się produkować dokumentacji do każdej wersji

Struktura systemu ma tendencję do degradacji wraz z dodawaniem nowych przyrostów - wymaga poświęcenia dodatkowych środków i czasu na restrukturyzację oprogramowania. Bez tego wprowadzanie kolejnych zmian będzie coraz bardziej kosztowne i skomplikowane

INŻYNIERIA OPROGRAMOWANIA ZORIENTOWANA NA WIELOKROTNE UŻYCIE



Opiera się na systematycznym podejściu do wykorzystania istniejącego oprogramowania. System integrowany jest z istniejących komponentów lub systemów COTS (Commercial-off-the-shelf)

Fazy procesu:

- analiza komponentów
- modyfikacja wymagań
- projektowanie z wielokrotnym użyciem
- rozwój i integracja

Wielokrotne użycie jest obecnie standardowym podejściem do budowania wielu typów systemów biznesowych.

Typy komponentów oprogramowania

- Usługi internetowe (ang. web services) wytwarzane zgodnie ze standardami usług, wywoływane zdalnie.
- Kolekcje obiektów wytwarzane w postaci pakietów, które integrowane są w ramach implementacji modelu komponentowego (frameworka) np.: .NET, J2EE.
- Autonomiczne systemy oprogramowania (COTS) konfigurowane dla działania w określonym środowisku.

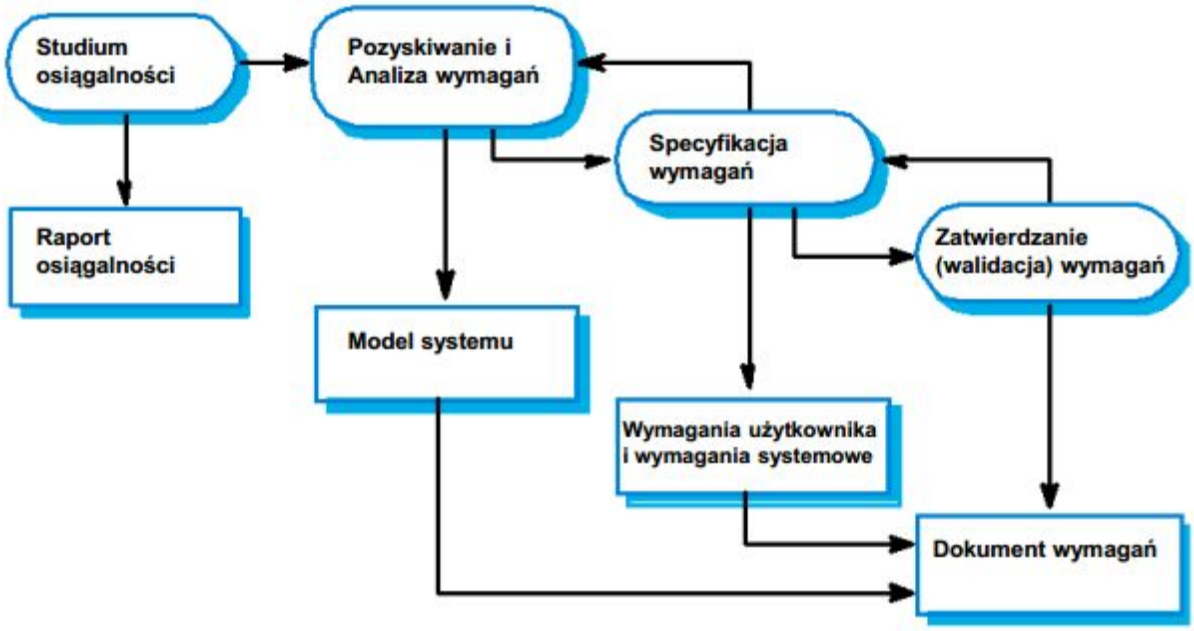
Aktywności procesu

Rzeczywiste procesy IO to sekwencje przeplatających się zespołowych czynności technicznych i zarządczych. Ich ogólnym celem jest wyspecyfikowanie, zaprojektowanie, implementacja oraz przetestowanie systemu oprogramowania.

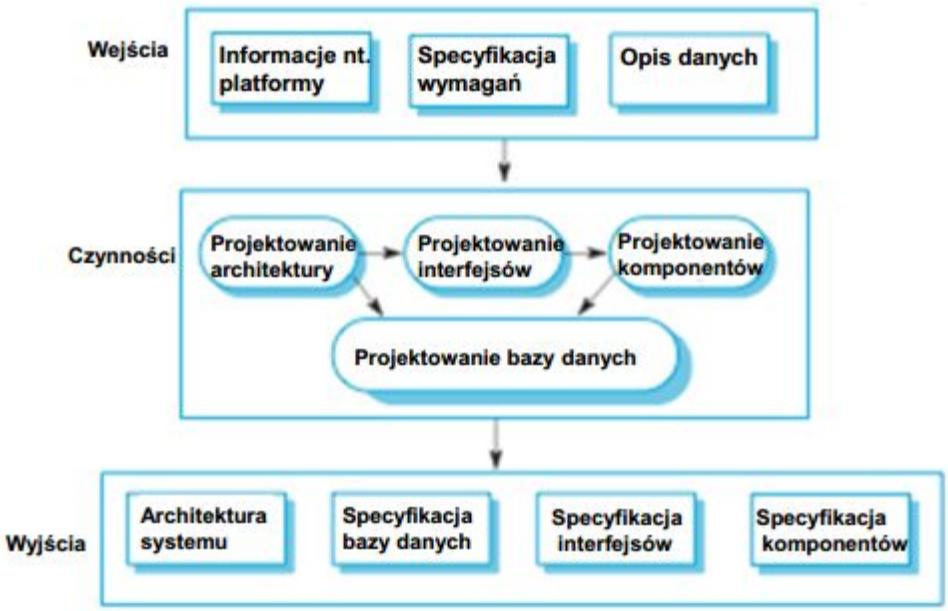
Cztery podstawowe aktywności procesu:

- Specyfikacja, rozwój, zatwierdzanie (walidacja), ewolucja
- Są organizowane różnie, w zależności od procesu (sekwencyjnie, przeplatające się).

Specyfikacja oprogramowania



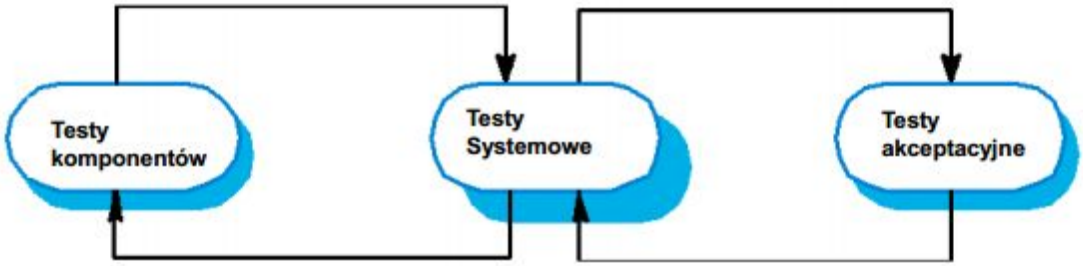
Ogólny model procesu projektowania oprogramowania



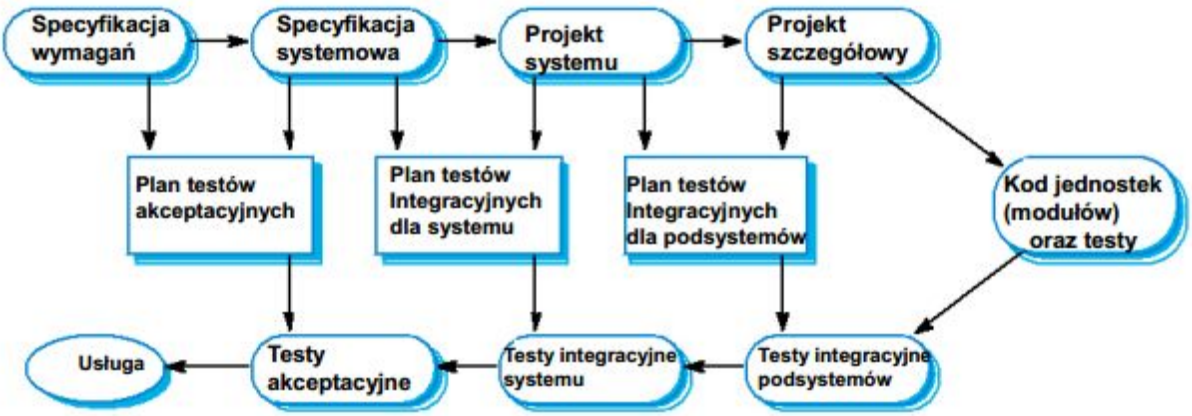
Zatwierdzanie (walidacja) oprogramowania:

- Celem weryfikacji i zatwierdzania jest wykazanie, że system jest zgodny ze specyfikacją i spełnia wymagania klienta systemu.
- Obejmuje procesy kontroli i przeglądów oraz testowania systemu.
- Testowanie systemu to uruchomienie go i wykonanie przypadków testowych, które wywodzą się ze specyfikacji rzeczywistych danych, które będą przetwarzane przez system.
- Testowanie jest najczęściej wykorzystywaną techniką w ramach czynności weryfikacji i zatwierdzania.

Etapy testowania



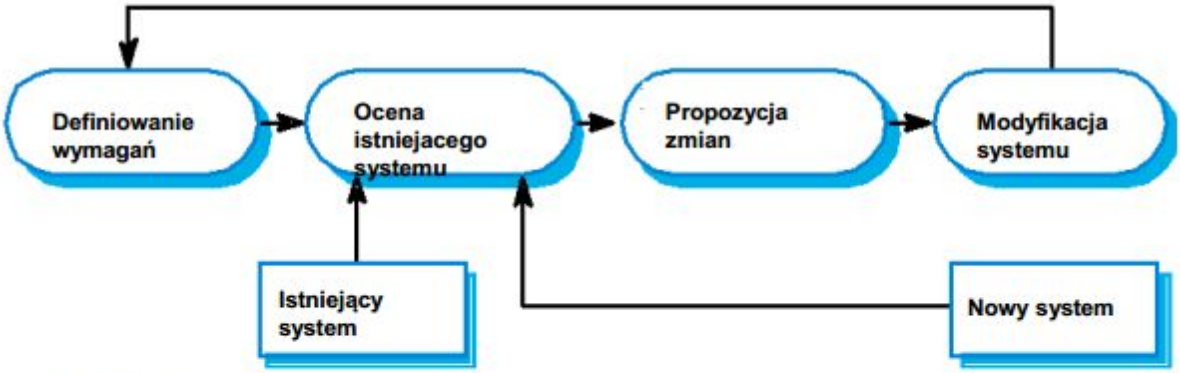
Fazy testowania w procesie IO sterowanym planem



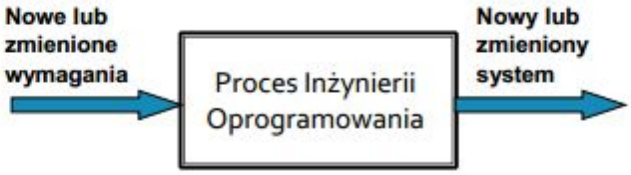
Ewolucja oprogramowania

- Oprogramowanie musi być elastyczne bo zmiany są nieuniknione
- Zmieniają się uwarunkowania biznesowe -> zmieniają się wymagania -> ewoluuje system
- Pomimo pierwotnego wyróżnienia procesu budowy i ewolucji systemu granica ta dzisiaj coraz bardziej się zaciera (coraz mniej systemów jest zupełnie nowych).

Wytwarzanie przez ewolucję



Czyli po prostu…



28

3 AGILE 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.

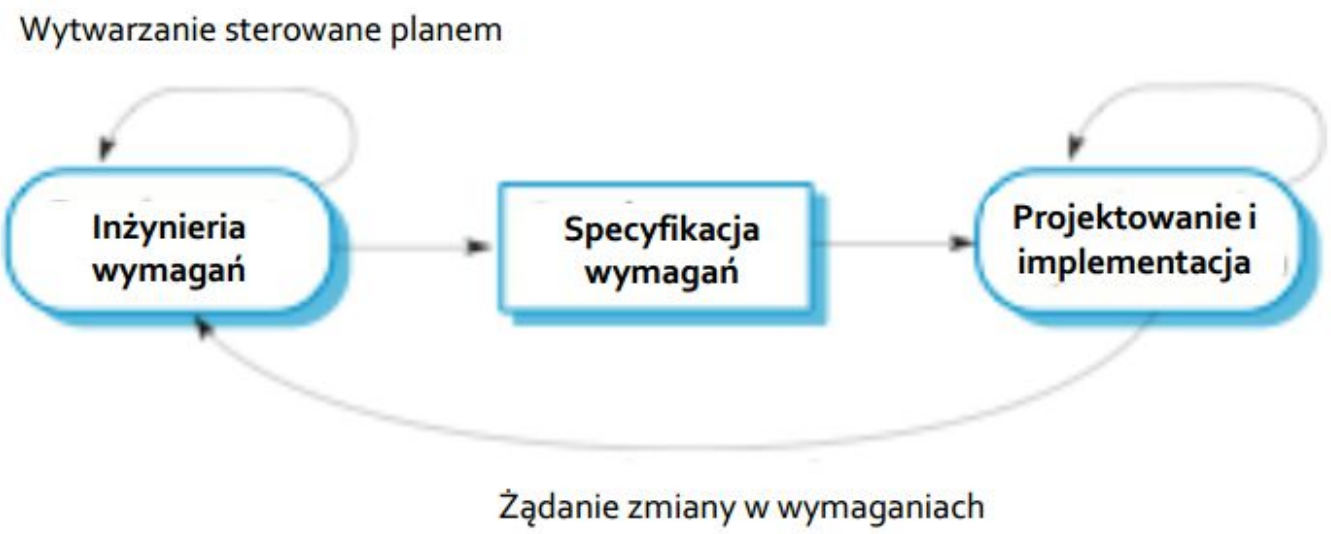
Zasady metod zwinnych

Zaangażowanie klientów	Klienci powinni być zaangażowani w cały proces wytwarzania oprogramowania. Ich rolom jest dostarczanie oraz priorytetyzacja wymagań dla nowego systemu oraz ocena jego iteracji.
Dostarczanie w przyrostach	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.

Problemy metod zwinnych

- Może nie być łatwo utrzymanie odpowiedniego poziomu zainteresowania klienta zaangażowanego w projekt.
- Członkowie zespołu mogą nie być przyzwyczajeni do poziomu intensywności zaangażowania charakterystycznego dla metod zwinnych.
- Jeżeli grup zainteresowania jest więcej mogą wystąpić problemy z priorytetyzacją zmian.
- Utrzymanie prostoty systemy 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.

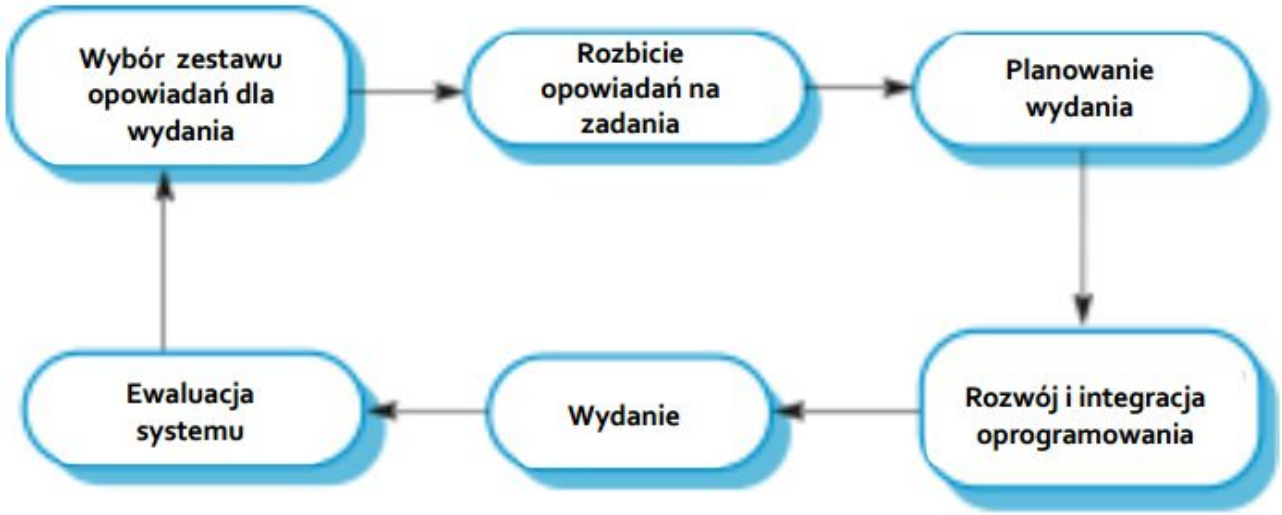
Wytwarzanie sterowane planem a wytwarzanie zwinne



Extreme programming

- Extreme = skrajne, radykalne, ostateczne
- Najbardziej znana i szeroko stosowana 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

Cykl wydania w XP



Praktyki XP

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.
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 testu (które musza 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.

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.

WYTWARZANIE STEROWANE TESTAMI

- 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.

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

- 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 linka kodu jest przejrzana przez więcej niż jedną osobę.
- Takie podejście zachęca do refaktoryzacji – a na tym może skorzystać cały zespół. 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)

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 (sprint 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.

Sprint

1. Sprinty 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 wytworzone w jego ramach
    - Na tej podstawie tworzony jest rejestr zadań sprintu (sprint backlog).
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 przez 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.

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.

Skalowanie wertykalne

- W rozwoju dużych systemów, nie jest możliwe, aby skupić się tylko na kodzie systemu. Wymagane jest projektowanie i dokumentowanie.
- Mechanizmy komunikacji między-zespołowej muszą być zaprojektowane i wykorzystywane. Można wykorzystać rozmowy telefoniczne i wideo konferencje pomiędzy członkami zespołów oraz promować częste, krótkie spotkania elektronicznych, gdzie zespoły aktualizują informacje o swoich postępach.
- Ciągła integracja, gdzie cały system jest budowany za każdym razem oraz programista zatwierdza każdą zmianę, jest praktycznie niemożliwe. Konieczne jest jednak utrzymanie częstych „bulid'ów" i regularnych wydań.

Skalowanie horyzontalne

- Kierownicy projektów nie posiadający doświadczenia w wykorzystaniu metod zwinnych mogą niechętnie akceptować ryzyko wprowadzenia nowego podejścia.
- Duże organizacje często mają wypracowane procedury jakości oraz wewnętrzne standardy, do których muszą stosować się wszystkie projekty. Z powodu ich biurokratycznej natury, są one prawdopodobnie niezgodne z metodami zwinnymi.
- Metody zwinne wydają się być najbardziej efektywne, gdy członkowie zespołu mają stosunkowo wysoki poziom umiejętności. Jednakże w dużych organizacjach, zakres umiejętności i zdolności jest prawdopodobnie szeroki.
- Może istnieć kulturowy opór we wprowadzaniu zwinnych metod, szczególnie w tych organizacjach, które mają długą historię wykorzystywania konwencjonalnych procesów inżynierskich.

CONFIGURATION MANAGEMENT

Oprogramowanie podlega ciągłym zmianom

- System może być postrzegany jako zbiór wersji, z których każda musi być zarządzana i wspierana.
- Wersje reprezentując implementacje zatwierdzonych propozycji zmian, adaptacji dla różnego typu sprzętu czy systemów operacyjnych
- Zarządzanie konfiguracją związane jest ze strategią, procesami oraz narzędziami pozwalającymi na zarządzanie zmieniającym się systemem.
- Bez zarządzania konfiguracją trudno byłoby odpowiedzieć na takie pytania jak:
    - Jakie zmiany zostały wprowadzone w tej wersji systemu
    - Jakie wersje komponentów są elementami tej wersji systemu

Czynności zarządzania konfiguracją

1. Zarządzanie zmianą
    - Śledzenie zgłoszeń dotyczących potrzeby zmiany oprogramowania (od klientów, deweloperów, ...).
    - Określanie kosztów oraz wpływu zmiany.
    - Podejmowanie decyzji o implementacji lub odrzuceniu zmiany
2. Zarządzanie wersjami
    - Śledzenie zmieniających się wersji komponentów systemu
    - Zapewnianie środowiska, w którym praca różnych deweloperów nie interferuje ze sobą
3. Budowanie systemu
    - Proces łączenia komponentów programu, danych, bibliotek do postaci wynikowego systemu
    - Konwersja kodu do postaci wynikowej
4. Zarządzanie wydaniami
    - Przygotowanie oprogramowania do wydania zewnętrznego
    - Śledzenie wersji systemu, które zostały dostarczone klientom.

Terminologia zarządzania konfiguracją

Jednostka konfiguracji lub jednostka konfiguracji oprogramowania (ang. configuration item - **CI** software configuration item - **SCI**)	Każdy element powiązany z wytwarzaniem oprogramowania (projekt, kod źródłowy, dane testowe, dokument, itp.) który został poddany kontroli konfiguracji. Jednostki konfiguracji mogą istnieć w różnych wersjach. Każda jednostka konfiguracji ma unikatową nazwę.
Kontrola konfiguracji (ang. Configuration control)	Proces zapewniania, że wersja systemu oraz komponentów została zapisana i jest zarządzana. Oznacza to, że każda wersja systemu oraz jego komponentów jest identyfikowalna i zapisana w sposób trwały (dostępna przez cały czas życia systemu).
Wersja (ang. Version)	Wersja (ang. Version) Instancja jednostki konfiguracji, różniąca się od innych instancji tej samej jednostki. Wersja zawsze ma unikatowy identyfikator (np. Nazwa jednostki konfiguracji + numer wersji).
Linia bazowa (ang. Baseline)	Zbiór wersji komponentów tworzących system. Linia bazowa jest kontrolowana – wersje komponentów nie mogą zostać zmienione. Oznacza to, że istnieje możliwość ponownego utworzenia linii bazowej z odpowiednich wersji komponentów.
Linia kodu (ang. Codeline)	Zbiór wersji komponentów oprogramowania oraz innych jednostek konfiguracji, od których zależny jest dany komponent.
Linia główna/rozwojowa (ang. Mainline)	Sekwencja linii bazowych reprezentujących różne wersje systemu.
Wydanie (ang. Release)	Wersja systemu, która została udostępniona do użytku klientom (lub innym użytkownikom w ramach organizacji).

Przestrzeń robocza (ang. Workspace)	Prywatna przestrzeń, w obrębie której dokonanie modyfikacji oprogramowania nie ma wpływu na pracę innych deweloperów, który również pracują na oprogramowaniem.
Utworzenie gałęzi (ang. Branching)	Powołanie do życia nowej linii kodu (ang. codeline) na podstawie wybranej wersji w istniejącej linii. Tak utworzona linia może być rozwijana niezależnie.
Łączenie (ang. Merging)	Utworzenie nowej wersji komponentu oprogramowania poprzez połączenie oddzielnych wersji znajdujących się w różnych liniach kodu. Linie te mogły powstać poprzez uprzednie utworzenie gałęzi.
Budowanie systemu (ang. System building)	Utworzenie wykonywalnego systemu poprzez skompilowanie i połączenie odpowiednich wersji komponentów oraz bibliotek wchodzących w skład systemu.

Systemy zarządzania wersjami

- Identyfikacja wersji oraz wydania
  - Wersjom będącym pod kontrolą systemu są przypisywane identyfikatory w momencie wprowadzania ich do systemu.
- Zarządzanie przestrzenią przechowującą dane
  - W celu minimalizacji przestrzeni wymaganej do przechowywania wersji, które częstą różnią się w nieznaczny sposób, system zarządzania wersjami udostępnia dedykowane mechanizmy składowania.
- Zapisywanie historii zmian
  - Każda zmiana w kodzie komponentu będącego pod kontrolą systemu jest zapamiętywana i możliwa do przejrzenia.
- Niezależny rozwój
  - System zarządzania wersjami śledzi komponenty, które zostały pobrane (ang. check out) do edycji w przestrzeni roboczej programisty. System zapewnia, że zmiany w komponencie wykonane przez jednego programistę nie będą miały wpływu na pracę innych programistów.
- Wsparcie dla projektów
  - System zarządzania wersjami może wspomagać rozwijanie kilku projektów, które współdzielą komponenty.

Budowanie systemu

- Proces tworzenia kompletnego, wykonywalnego systemu poprzez kompilację i połączenie komponentów systemu, bibliotek, plików konfiguracyjnych, itp.
- Narzędzia służące do budowania systemu oraz narzędzia służące zarządzaniu wersjami muszą komunikować się ze sobą ponieważ proces budowy wymaga pobrania (checkout) wersji komponentów z repozytorium zarządzanego przez system zarządzania wersjami.
- Opis konfiguracji wykorzystywany do identyfikacji linii bazowej jest również wykorzystywany przez narzędzia budowania systemu.

Funkcjonalności systemu budującego

- Generowanie skryptów budujących
- Integracja z systemem zarządzania wersjami
- Minimalizacja potrzeby rekompilacji komponentów
- Tworzenie wykonywalnego systemu
- Automatyzacja testów
- Raportowanie
- Generacja dokumentacji

Zarządzanie wydaniami

- Wydanie (ang. release) to dystrybucja (publiczna lub prywatna) wersji oprogramowania.
- Wydanie publiczne systemu jest to wersja oprogramowania, która została dostarczona klientom.
- W przypadku oprogramowania generycznego istnieją zazwyczaj dwa typy wydań:
  - Wydania główne (ang. major releases) które dostarczają znaczących zmian w funkcjonalności systemu
  - Wydania podrzędne (ang. minor releases), których celem jest naprawa błędów oraz rozwiązanie problemów zgłaszanych przez klientów.
- W przypadku oprogramowania dedykowanego lub linii produktowych wydania systemu mogą być produkowane niezależnie dla poszczególnych klientów oraz w ramach konkretnego klienta może istnieć (i być w użyciu) kilka wersji jednocześnie.

Czynniki wpływające na planowanie wydań

Jakość techniczna systemu	Jeżeli raportowane są poważne błędy, które mogą mieć wpływ na pracę wielu użytkowników systemu – wydanie poprawiające błędy może być potrzebne. Mniejsze błędy systemu mogą być wydawane w postaci „łatek” (ang. patch), które mogą być zastosowane do bieżącego wydania.
Zmiany platformy	Może istnieć potrzeba utworzenia nowego wydania ponieważ pojawiła się nowa wersja systemu operacyjnego.
Piąte prawo Lehman’a	To „prawo” sugeruje, że jeżeli doda się dużo nowych funkcjonalności do systemu to razem z nimi wprowadzi się również błędy, które ograniczą ilość funkcjonalności w kolejnym wydaniu. Dlatego wydanie systemu, dodające dużą liczbę istotnych nowych funkcjonalności może zazwyczaj wymusić pojawienie się wydania (wydań), związanych z poprawą błędów czy zwiększeniem wydajności.
Konkurencja	Konkurencyjne produkty wprowadzają nowe cechy – jeżeli nie chcemy stracić rynku musimy wprowadzić podobne cechy do systemu.
Wymagania rynku	Dział marketingu w organizacji zobowiązał się do wydania będzie dostępna w danym dniu.
Zmiany proponowane przez klientów	W przypadku systemów dopasowanych klienci mogą zgłosić potrzebę realizacji zestawu zmian i oczekiwać wydania systemu jak tylko zmiany zostaną wprowadzone.

4 INŻYNIERIA OPROGRAMOWANIA

Podstawowe typy wymagań

Wymagania użytkownika

- Zdania języka naturalnego powiązane z diagramami ukazującymi usługi systemu wraz z ograniczeniami. Pisane dla klientów

Wymagania systemowe

- Dokument o określonej strukturze ustalający szczegóły funkcjonalności systemu, jego usług oraz ograniczeń przy których ma działać.
- Definiuje co ma być zaimplementowane – może być podstawą kontraktu pomiędzy zleceniodawcą a wykonawcą.

Charakterystyka wymagań

- **Wymagania funkcjonalne** – zachowanie systemu (jakie akcje ma wykonywać system bez brania pod uwagę ograniczeń)
- **Wymagania niefunkcjonalne** – ograniczenia, które mają wpływ na wykonywane zadania systemu.
- **Ograniczenia projektowe** – ograniczenia dotyczące projektowania systemu, nie mające wpływu na jego zachowanie ale, które muszą być spełnione, aby dotrzymać zobowiązań technicznych, ekonomicznych lub wynikających z umowy

Wymagania funkcjonalne

Określenie wymagania funkcjonalnych obejmuje następujące zadania:

- Określenie wszystkich rodzajów użytkowników, którzy będą korzystać z systemu.
- Określenie wszystkich rodzajów użytkowników, którzy są niezbędni do działania systemu (obsługa, wprowadzanie danych, administracja).
- Dla każdego rodzaju użytkownika określenie funkcji systemu oraz sposobów korzystania z planowanego systemu.
- Określenie systemów zewnętrznych (obcych baz danych, sieci, Internetu), które będą wykorzystywane podczas działania systemu.
- Ustalenie struktur organizacyjnych, przepisów prawnych, statutów, zarządzeń, instrukcji, itd., które pośrednio lub bezpośrednio określają funkcje wykonywane przez planowany system.

Wymagania niefunkcjonalne

Opisują ograniczenia, przy których system ma realizować swoje funkcje.

- Użyteczność (ang. Usability)
  - Wymagany czas szkolenia, czas wykonania poszczególnych zadań, ergonomia interfejsu, pomoc, dokumentacja użytkownika
- Niezawodność (ang. Reliability)
  - Dostępność, średni czas międzyawaryjny (MTBF), średni czas naprawy (MTTR), dokładność, maksymalna liczba błędów.
- Efektywność (ang. Performance)
  - Czas odpowiedzi, przepustowość, czas odpowiedzi, konsumpcja zasobów, pojemność.
- Zarządzalność (ang. Supportability)
  - Łatwość modyfikowania, skalowalność, weryfikowalność, kompatybilność, możliwości konfiguracyjne, serwisowe, przenaszalność.

Zarządzanie wymaganiami

- Zarządzanie wymaganiami dotyczy procesu translacji potrzeb klientów w zbiór kluczowych cech i własności systemu.
- Następnie ten zbiór jest przekształcany w specyfikację funkcjonalnych i niefunkcjonalnych wymagań.
- Specyfikacja jest następnie przekształcana w projekt, procedury testowe i dokumentację użytkownika.

Bariery pozyskiwania wymagań

- Syndrom „tak, ale”
  - „Tak, ale hmmmm, teraz kiedy go już widzę, czy będzie można...? Czy nie lepiej byłoby, gdyby...? Przecież jeżeli..., to trudno będzie...”
- Syndrom „nieodkrytych ruin”
- Syndrom „użytkownik i programista”
  - Użytkownicy, nie wiedzą, czego chcą, lub wiedzą, co chcą, ale nie mogą tego wyrazić.
  - Użytkownicy uważają, że wiedzą, czego chcą, dopóki programiści nie dadzą im tego, o czym mówili, że chcą.
  - Analitycy uważają, że rozumieją problemy użytkownika lepiej niż on sam.
  - Wszyscy uważają, że inni mają określone motywacje polityczne.

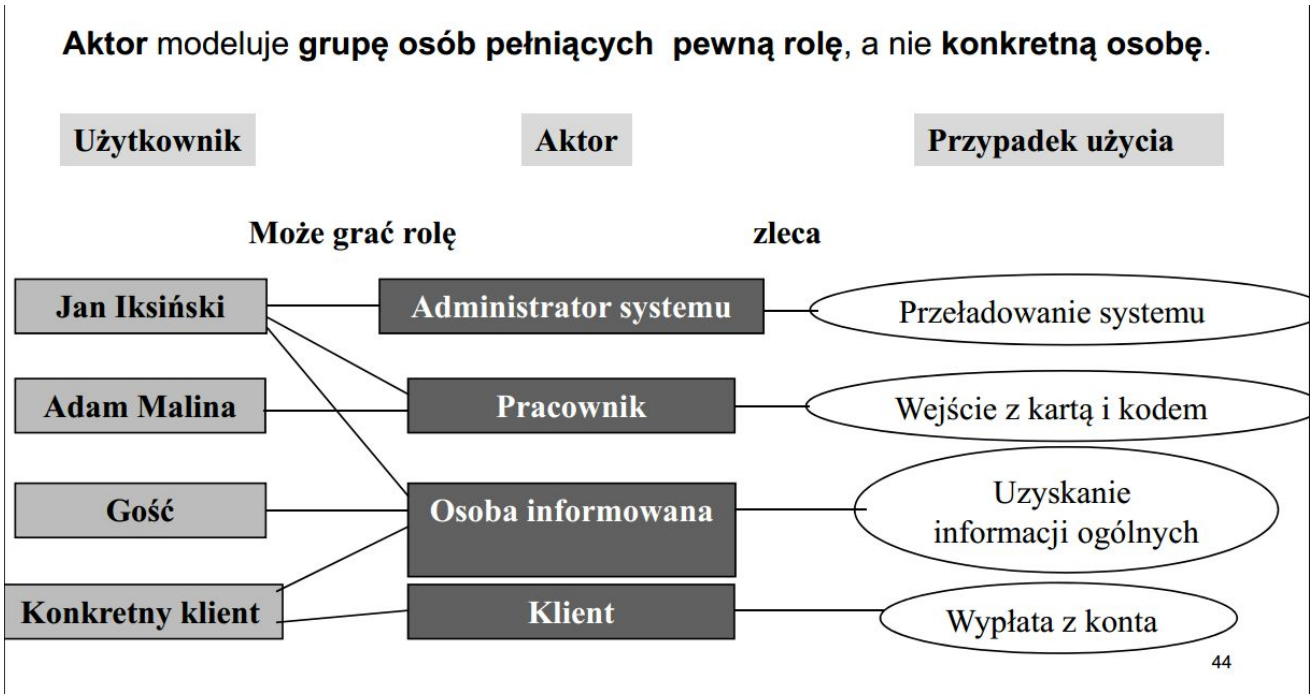
Metody specyfikacji wymagań

- Język naturalny - najczęściej stosowany. Wady: niejednoznaczność powodująca różne rozumienie tego samego tekstu; elastyczność, powodująca wyrazić te same treści na wiele sposobów. Utrudnia to wykrycie powiązanych wymagań i powoduje trudności w wykryciu sprzeczności.
- Formalizm matematyczny. Stosuje się rzadko (dla specyficznych celów).
- Język naturalny strukturalny. Język naturalny z ograniczonym słownictwem i składnią. Tematy i zagadnienia wyspecyfikowane w punktach i podpunktach.
- Tablice, formularze. Wyspecyfikowanie wymagań w postaci (zwykle dwuwymiarowych) tablic, kojarzących różne aspekty (np. tablica ustalająca zależność pomiędzy typem użytkownika i rodzajem usługi).
- Diagramy blokowe: forma graficzna pokazująca cykl przetwarzania.
- Diagramy kontekstowe: ukazują system w postaci jednego bloku oraz jego powiązania z otoczeniem, wejściem i wyjściem.
- Model przypadków użycia: poglądowy sposób przedstawienia aktorów i funkcji systemu. Uważa się go za dobry sposób specyfikacji wymagań funkcjonalnych.

Podstawowe elementy modelu przypadków użycia

Aktor - Reprezentuje rolę, którą może grać w sytemie jakiś jego użytkownik; (np. kierownik, urzędnik, klient)

Przypadek użycia - Reprezentuje sekwencję operacji inicjowaną przez aktora, niezbędnych do wykonania zadania zleconego (zainicjowanego) przez aktora, np. potwierdzenie pisma,złożenie zamówienia, itp.



Notacja

Rejestracja

Przypadek użycia

Student

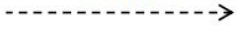
Aktor

Interakcja.

weryfikacja klienta

Blok ponownego użycia

<<include>>



<<extends>>

Zależności (pomiędzy przypadkami użycia)

6 UML

Czym jest UML?

- Notacją graficzną
- Językiem programowania
- Metodyką
- Narzędziem

Na UML można patrzeć przez pryzmat dwóch składowych:

- notacja (elementy graficzne, składnia języka modelowania – ważniejsze przy modelowaniu)
- metamodel (ścisła semantyka poszczególnych elementów – istotne przy programowaniu graficznym i automatycznym generowaniukodu)

Perspektywy modelowania w UML

UML jest określany jako język modelowania z 4+1 perspektywą :

- Perspektywa przypadków użycia – opisuje funkcjonalność systemu widzianą przez użytkowników
- Perspektywa logiczna – sposób realizacji funkcjonalności, struktura systemu widziana przez projektanta
- Perspektywa implementacyjna – zawiera moduły i interfejsy, przeznaczona dla programisty
- Perspektywa procesowa – podział systemu na czynności i jednostki wykonawcze (wątki, procesy, współbieżność) – służy głównie programistom i instalatorom
- Perspektywa wdrożeniowa – fizyczny podział elementów systemu i ich rozmieszczenie w infrastrukturze, ważna dla instalatorów

Model pojęciowy UML

- Podstawowe bloki konstrukcyjne
  - elementy
  - związki
  - diagramy
- Reguły określające sposób łączenia tych bloków
  - bloki konstrukcyjne nie mogą być rozrzucone na chybił trafił. Jak w każdym innym języku tak w UML obowiązują reguły określające jak poprawny model ma wyglądać. Dotyczą one np. nazw, ich zasięgu,kontekstu
- Mechanizmy językowe
  - specyfikacje
  - dodatki
  - rozgraniczenia
  - rozszerzenia

Podstawowe bloki konstrukcyjne UML - elementy

- strukturalne (pełnią rolę rzeczowników modelu UML)
  - klasa
  - interfejs
  - kooperacja
  - przypadek użycia
  - komponent
  - węzeł
- czynnościowe (pełnią rolę czasowników modelu UML)
  - komunikat
  - maszyna stanowa
- grupujące (pełnią rolę organizacyjną, dekompozycja modelu)
  - pakiet
- komentujące (pełnią rolę objaśniającą)
  - notatka

Podstawowe bloki konstrukcyjne UML - diagramy

Diagram – graf, którego wierzchołkami są elementy a krawędziami związki

- Diagram klas
- Diagram obiektów
- Diagram komponentów
- Diagram wdrożenia

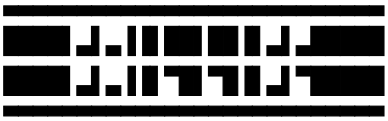
modelowanie strukturalne

- Diagram kooperacji
- Diagram przypadków użycia
- Diagram czynności
- Diagram sekwencji
- Diagram stanów

modelowanie behawioralne

Diagramy klas

- Jest podstawowym diagramem struktury logicznej systemu
- Przedstawia podział odpowiedzialności systemu pomiędzy jego klasy oraz wymienianych pomiędzy nimi komunikatów
- Zawiera największą ilość informacji i stosuje największą liczbę symboli
- Najczęściej używany diagram UML (także do generowania kodu na podstawie modelu)



Modelowanie, Obiektowość, Powtórne użycie

MODELOWANIE

MODELOWANIE POJĘCIOWE:

- dokładne wyobrażenie problemu i metody jego rozwiązania. Procesy tworzenia oprogramowania nie są związane z jakimkolwiek językiem
- pojęcia modelowanie pojęciowe i model pojęciowy odnoszą się do procesów myślowych i wyobrażeń towarzyszących pracy nad oprogramowaniem

ZASADY MODELOWANIA:

- wybór modelu ma wpływ na postrzeganie rzeczywistości
- każdy model może być na dowolnym poziomie szczegółowości
- najlepsze modele odnoszą się do rzeczywistości
- pojedynczy model nie jest wystarczający

MODELOWANIE ANALITYCZNE:

- pomaga w zrozumieniu funkcjonalności systemu i w komunikacji z klientem
- reprezentuje system z różnych punktów widzenia
    - zewnętrznego - kontekst i środowisko
    - zachowania - modeluje zachowania systemu
    - strukturalnego - modeluje architekturę systemu lub strukturę przetwarzania danych

TYPY MODELI:

- przetwarzania danych - jak przetwarzane są dane na różnych etapach pracy systemu
- kompozycji - jak elementy systemu są komponowane z mniejszy fragmentów
- architektoniczny - z jakich podsystemów składa się system
- klasyfikacji - wspólne cechy poszczególnych elementów
- bodziec-reakcja - w jaki sposób system reaguje na zdarzenia poza nim jak i w jego wnętrzu

MODELE KONTEKSTOWE

- ilustrują kontekst system, a otoczenie
- kwestie społeczne i organizacyjne mogą mieć na niego wpływ

MODELE PROCESU

- pokazują czynności wspierane przez system
- pomagają w podjęciu decyzji, które czynności mają być wykonywane automatycznie

MODELE ZACHOWANIA

- przetwarzania danych - pokazują przetwarzanie i przepływ danych w systemie
- stanów - reakcje systemu na zdarzenia

MODELE STANÓW

- opisują zachowanie systemu na wewnętrzne i zewnętrzne zdarzenia
- pokazują odpowiedzi systemu na określone stymulacje
- pokazuje system w postaci zbioru stanów oraz możliwych przejść pomiędzy nimi wraz ze zdarzeniami

DIAGRAMY STANÓW

- w każdym stanie można opisać akcję, która jest wykonywana
- mogą być wspomagane tabelami szczegółowo opisującymi stany i pobudzenia

SKŁADOWE SYSTEMU W PROJEKTOWANIU

- zarządzania pamięcią
- interfejsu użytkownika
- zarządzania zadaniami
- zarządzania danymi
- dziedziny problemu

OBIEKTOWOŚĆ

ZASADA ABSTRAKCJI

- eliminacja lub ukrycie mniej istotnych szczegółów
- wyodrębnienie cech wspólnych i niezmiennych dla pewnego zbioru

ZASADA HERMETYZACJI

- programista ma tyle wiedzieć o obiekcie, ile potrzeba, aby go efektywnie użyć
- wszystko co może być ukryte, powinno być ukryte
- klient jest zależny od interfejsu

ZASADA MODULARYZACJI - DEKOMPOZYCJA

- rozdzielenie czegoś mniejsze, łatwiejsze do zarządzania fragmenty

ZASADA HIERARCHIZACJI

- porządkowanie abstrakcji w strukturę drzewiastą

OBIEKT

- jest to byt obserwowalny w rzeczywistości
- jednostka z dobrze zdefiniowanymi granicami, który hermetyzuje stan oraz zachowanie

WŁASNOŚCI OBIEKTU

- tożsamość - odróżnia go od innych obiektów, jest niezależna od atrybutów obiektu, lokacji
- stan - może zmieniać się w czasie
- zachowanie - zestaw operacji, które wolno stosować do danego obiektu

MODELE OBIEKTOWE

- odzwierciedlają elementy rzeczywistości, którymi manipuluje system
- opisują system w terminach klas obiektów i relacji między nimi
- mogą być rzeczywiste lub abstrakcyjne

OPROGRAMOWANIE ZORIENTOWANE OBIEKTOWO

- OOA - tworzenie modelu obiektowe dla dziedziny problemu
- OOD - tworzenie systemu zorientowanego obiektowo w celu implementacji wymagań
- OOP - realizacji OOD z wykorzystaniem obiektowego języka programowania

CHARAKTERYSTYKA OOD

- obiekty reprezentują elementy rzeczywistości
- obiekty są niezależne, hermetyzują stan, posiadają zachowanie
- nie istnieją współdzielone dane
- obiekty mogą działać sekwencyjnie lub współbieżnie

ZALETY OOD

- łatwiejsze zarządzanie i konserwacja
- obiekty mogą być ponownie użyte
- dla niektórych systemów istnieje zależność pomiędzy światem rzeczywistym a obiektami systemu

KOMUNIKACJA OBIEKTÓW

- obiekty przekazują sobie komunikaty
- komunikat
    - nazwa usługi
    - informacje potrzebne do działania oraz miejsce przechowywania danych
- najczęściej komunikat jest wywołaniem procedury
    - nazwa procedury
    - lista parametrów

SERWERY I AKTYWNE OBIEKTY

- serwer - obiekt reprezentuje równoległy proces z wejściami odpowiadającymi operacjom obiektu
- obiekt aktywny - obiekty są implementowane jako osobne procesy, z możliwością zmiany wewnętrznego stanu obiektu

PROCES OOD

- zrozumienie i zdefiniowanie kontekstu oraz modelu systemu
- zaprojektowanie architektury systemu
- identyfikacja głównych obiektów systemu
- opracowanie modeli projektowych
- wyspecyfikowanie interfejsów obiektów

IDENTYFIKACJA OBIEKTÓW

- wynajdywanie klas obiektów

- elementy z dziedziny zastosowania np. samolot jako rzecz
- podejście czynnościowe
- analiza scenariuszy

MODELE PROJEKTOWE

- pokazują obiekty, klasy i powiązania między elementami
- modele statyczne opisują statyczną strukturę
- modele dynamiczne opisują interakcję między obiektami

GRANULOWOŚĆ PONOWNEGO UŻYCIA

- COTS - cały system można ponownie użyć poprzez włączenie go do innych systemów
- wielokrotne użycie komponentów systemu
- wielokrotne użycie funkcji

KORZYŚCI Z PONOWNEGO UŻYCIA

- zwiększona niezawodność - korzystamy ze sprawdzonych systemów
- redukcja ryzyka w procesie wytwarzanie
- wykorzystani wiedzy specjalistów
- zgodność ze standardami
- przyspieszenie procesu twórczego

PROBLEMY Z PONOWNYM UŻYCIEM

- zwiększone koszty oprogramowania
- brak wspomagania narzędziowego
- syndrom "nie wymyślono tutaj"
- prowadzenie biblioteki komponentów
- adaptowanie komponentów

PODEJŚCIE DO PONOWNEGO UŻYCIA

- wzorce projektowanie - abstrakcje rozwiązujące problemy pojawiające się w wielu aplikacjach
- oprogramowanie komponentowe - tworzenie systemu na zasadzie integracji komponentów zgodnym ze standardem
- zręby aplikacji - kolekcja klas, które mogą być wykorzystane dla utworzenia konkretnej aplikacji
- integracja systemów spadkowych - zdefiniowanie zestawy interfejsów i udostępnienie funkcjonalności już istniejących systemów
- systemy zorientowane usługowo - łączenie współdzielonych usług
- linie produkcyjne programów - adaptacja uogólnionej architektury na różne sposoby
- integracja COTS - tworzenie systemów na zasadzie integracji istniejących plików
- konfigurowalne aplikacje usług pinowych - projektuje się generyczny system, łatwo dopasowalny do wymagań klientów

- biblioteki programowe - funkcje i klasy często używane
- generatory programów - generują całe programy lub ich fragmenty
- programowanie aspektowe - współdzielone komponenty są automatycznie wplatane w różne miejsca aplikacji podczas procesu kompilacji

WZORCE PROJEKTOWE

- ponowne użycie wiedzy na temat problemu i sposobu jego rozwiązania
- abstrakcyjny, aby mógł być wykorzystany w różnych konfiguracjach

WZORZEC OBSERWATOR

- separuje sposób prezentacji stanu obiektu od obiektu
- wykorzystywany, gdy potrzebne są różne sposoby wyświetlania danych

RODZAJE GENERATORÓW

- generatory aplikacji do przetwarzania danych biznesowych
- parsery i analizatory leksykalne dla przetwarzania języków
- generatory kodu w narzędziach CASE

PROGRAMOWANIE ASPEKTOWE

- wszystkie moduły muszą implementować bezpieczeństwo i monitorowanie
- własności przecinają model komponentów funkcjonalnych

ZRĘBY APLIKACJI

- złożone z kolekcji klas i interfejsów
- implementacja systemu na podstawie implementacji abstrakcyjnych klas zrębu

KLASYFIKACJA ZRĘBÓW

- zręby infrastruktury systemowej - wspierają tworzenie infrastruktury
- zręby integracyjne warstwy pośredniej - wspierają komunikację między komponentami JDBC, JAVA BEANS etc.
- zręby aplikacji przemysłowych - wspierają wytwarzanie specyficznych typów aplikacji

COTS

- Computing Off-The-Shelf Systems
- systmey COTS są komplentymi aplikacjami, oferującymi określone API
- tworzenie dużych systemów COTS jest opłacalne dla E-commerce
- szybsze tworzenie i niższe koszty

PROBLEMY COTS

- brak kontroli nad wydajnoścIa i funkcjonalnością
- problemy z integracją i współpracą
- brak kontroli nad ewolucją
- niewielki zakres wsparcia dostawców COTS

CZYSTY KOD

DŁUG TECHNICZNY

- długo techniczny to miara poziomu trudności jaki napotykają programiści podejmując próbę wprowadzenia wymaganych zmian do istniejącego kodu
- wraz ze wzrostem długu rośnie wartość oszacowań, nieprzewidywalność, liczba efektów ubocznych poziom lęku

KIEDY SIĘ ZADŁUŻAMY TECHNICZNIE?

- brak wiedzy
- brak profesjonalizmu
- pośpiech związany z deadlinem'm
- entropia

JAKI POWINIEN BYĆ CZYSTY KOD?

- elegancki i efektywny
- łatwy w ulepszaniu przez innych
- cechuje go dbałość o szczegóły
- nie zawiera powtórzeń
- konsekwentny, prosty, urzekający

NAZEWNICTWO

- nazwy wyjaśniające intencje
- nazwy jednoznaczne, wymowne

KOMENTARZE

- komentujemy gdy chcemy opisać zawiłość kodu
- komentujemy gdy chcemy wyjaśnić intencje
- komentujemy w celu podkreślenia, sprecyzowania

NIE POWTARZAJ SIĘ DRY

- wielokrotnie wykorzystuj kod zamiast go powielać
- wybieraj odpowiednią lokalizację dla kodu
    - pojedyncza zmiana, pojedynczy test
- stosuj zasady poprawnego nazewnictwa

FUNKCJE I POZIOMY ABSTRAKCJI

- SLAP - wszystkie instrukcje w ramach metody powinny działać na tym samym poziomie abstrakcji i być związane z pojedynczym zadaniem

ZASADY SOLID W OBIEKTOWOŚCI

- SRP - Single Responsibility - zasada pojedynczej odpowiedzialności
- OCP - Open/Close
- LSP - Liskov Substitution
- ISP - Interface Segregation
- DIP - Dependency Inversion

SPÓJNOŚĆ I ZALEŻNOŚĆ

- Spójność - zmiana w A dopuszcza zmianę w B, tak że obie zyskują nową wartość
- Zależność - zmiana w B jest wymuszana zmianą w A

OPEN/CLOSE PRINCIPALE

- klasa powinna być otwarta dla rozszerzeń, ale zamknięta dla modyfikacji
- idealnie wprowadzenie nowych cech nie powinno wymagać modyfikacji w istniejącym kodzie

LISKOV SUBSTITUTION PRINCIPLE

- instrukcje, które manipulują obiektem wykorzystując referencje do klasy bazowej muszą być zdolny użyć obiektu klasy potomnej bez wiedzy o jego istnieniu

INTERFACE SEGREGATION PRINCIPLE

- klient nie powinien być zmuszany do zależności od interfejsów, których nie używają

DEPENDENCY INVERSION PRINCIPLE

- moduły wyższego poziomu nie powinny być zależne od modułów niższego poziomu

TESTOWANIE

WERYFIKACJA

- oprogramowanie powinno być zgodne ze specyfikacją

ZATWIERDZANIE

- oprogramowanie powinno być zgodne z potrzebami użytkowników

WZGLĘDNOŚĆ WERYFIKACJI ZARZĄDZANIA

- cel programowania - wymagany poziom zaufania zależy od tego jak programowanie jest krytyczne
- oczekiwania użytkowników
- otoczenie marketingowe - wczesne dostarczenie produktu na rynek może mieć większe znaczenie

INSPEKCJE I TESTOWANIE

- inspekcje oprogramowania - analiza statycznej reprezentacji systemu w celu wykrycia problemów
- testowanie oprogramowanie - dynamiczna analiza poprzez obserwacje zachowania systemu

INSPEKCJE OPROGRAMOWANIA

- analiza źródłowej reprezentacji systemu mająca na celu wykrycie defektów i anomalii
- nie wymagają działającego systemu
- mogą być zastosowane do dowolnej reprezentacji systemu

ZALETY INSPEKCJI

- w czasie pojedynczej inspekcji może zostać wykrytych wiele błędów

INSPEKCJE A TESTOWANIE

- inspekcje nie są w stanie zweryfikować czy system odpowiada użytkownikom
- nie są w stanie sprawdzić realizacji wymagań niefunkcjonalnych

TESTOWANIE

- uruchomienie programu w kontekście sztucznych danych i wnioskowanie na podstawie rezultatów działania
- testowanie wykazuje, że oprogramowanie wykonuje to czego oczekujemy
- testowanie wykrywa defekty zanim oprogramowanie zostanie przekazane do użycia

- testowanie może wykazać istnienie błędów, a nie ich brak

TESTY ZATWIERDZAJĄCE A TESTY USTEREK

- zatwierdzające - oczekujemy poprawnego działania systemu w kontekście zestawu przypadków testowych
- usterek - przypadki testowe są zaprojektowane w celu ujawnienia usterek

ETAPY TESTOWANIA

- testy w trakcie rozwoju - system jest testowany podczas produkcji w celu wykrycia błędów
- testy wydania - odrębny zespół testuje kompletną wersję systemu przed jej udostępnieniem
- testy użytkowników - użytkownicy testują system w swoim środowisku

TESTY W TRAKCIE ROZWOJU

- testy jednostkowe - testowane pojedyncze jednostki oprogramowania, implementacje obiektów, metod
- testy komponentów - testowane są pojedyncze jednostki zintegrowane
- testy systemu - testowane są wybrane lub wszystkie komponenty systemu

TESTY JEDNOSTKOWE

- testowanie pojedynczych jednostek w izolacji
- są to testy usterek
- wykonywane automatycznie

RODZAJE DUBLERÓW DLA TESTÓW JEDNOSTKOWYCH

- dummy objects - wypełniacz
- fake objects - nieprodukcyjna implementacja
- stubs - dostarczają ustalonych odpowiedzi na wybrane wywołania
- mocks - obiekty budowane przez specyfikowanie sposobu w jaki powinny być wywołane

SKUTECZNOŚĆ TESTU JEDNOSTKOWEGO

- przypadki testowe odzwierciedlają normalne operacje i wykazują działanie zgodne z oczekiwaniami
- przypadki testowe wykorzystują nieprawidłowe dane i pokazują jak system na nie reaguje

WSKAZÓWKI DOTYCZĄCE TESTOWANIA

- wybierz takie dane wejściowe, które spowodują wygenerowanie wszystkich komunikatów o błędach
- zaprojektuj wejścia w taki sposób, aby spowodowały przepełnienie bufora
- wielokrotnie powtarzaj te same dane wejściowe lub ich serie

- wymuś wygenerowanie niepoprawnych danych wyjściowych
- wymuś wykonanie obliczeń, których rezultat będzie zbyt mały/duży

TESTY STRUKTURALNE

- są opracowywane na podstawie wiedzy o strukturze i implementacji
- celem jest zapewnienie aby instrukcja programu była wykonana co najmniej raz
- możliwe do pracowania dla niewielkich komponentów

TESTOWANIE ŚCIEŻEK

- odmiana testowania strukturalnego, której celem jest zbadanie każdej niezależnej ścieżki wykonania programu
- punktem startu jest opracowanie grafu przepływu

TESTY KOMPONENTÓW

- testowanie komponentu polega na potwierdzeniu zgodności zachowania interfejsu z jego specyfikacją

TESTOWANIE INTERFEJSÓW

- celem jest wykrycie usterek z powodu
  - niewłaściwego użycia interfejsu
  - niezrozumienia interfejsu
  - błędów synchronizacji

TYPY INTERFEJSÓW

- parametryczne - przekazują dane między komponentami
- w pamięci dzielonej - współdzielą dane w pamięci
- proceduralne - jeden podsystem obudowuje procedury udostępniane innym podsystemom
- przekazywania komunikatów - żąda usługi innego podsystemu przez przesłanie komunikatu

TESTY SYSTEMOWE, A KOMPONENTÓW

- testy systemowe są związane z integracją komponentów w celu utworzenia wersji systemu, która zostaje poddana testowaniu
- testy systemowe sprawdzają kompatybilność komponentów, poprawność interakcji, przesyłanie

WYTWARZANIE STEROWANE TESTAMI - TDD

- metoda przyrostowego rozwoju oprogramowanie, w którym testy stanowią podstawę wytwarzania
- testy pisane są przed kodem

TESTY REGRESYJNE

- kontrola czy zmiany nie spowodowały problemów w istniejącym kodzie
- bardzo kosztowne

TESTY WYDANIA

- test wersji systemu, która ma być używana poza zespołem rozwijającym system

TESTY WYDAJNOŚCIOWE

- testują wydajność i niezawodność
- powinny odzwierciedlać profile wykorzystania systemu

TESTY OBCIĄŻENIA

- testowanie powyżej maksymalnego obciążenia aż do awarii

TESTY UŻYTKOWNIKÓW

- alpha - użytkownicy współpracują z zespołem w celu przetestowania oprogramowania
- beta - użytkownicy samodzielnie eksperymentują i informują producenta o problemach
- testy akceptacyjne - klient testuje system w celu podjęcia decyzji czy jest on gotowy do wdrożenia w środowisku produkcyjnym

TESTOWANIE SYSTEMU W ŚRODOWISKU PRODUKCYJNYM

- A/B testing - obserwacja zachowania użytkowników
- Canary deployment - przekierowanie podzbioru użytkowników do nowej funkcjonalności

SYSTEMY ROZPROSZONE

- zbiór niezależnych komputerów, które z punktu widzenia użytkownika postrzegane są jako pojedynczy system
- przetwarzanie informacji jest rozproszone pomiędzy zespół komputerów

CHARAKTERYSTYKA SYSTEMÓW ROZPROSZONYCH

- współdzielenie zasobów
- wykorzystanie sprzętu i oprogramowania różnych dostawców
- współbieżność w celu zwiększenia wydajności
- zwiększenie przepustowości przez dodanie nowych zasobów

- tolerancja na błędy

PROBLEMY SYSTEMÓW ROZPROSZONYCH

- są skomplikowane
- niezależne zarządzania częściami systemu
- odgórne sterowanie nie jest możliwe

PROBLEMY PROJEKTOWE

- przezroczystość - istnieje warstwa pośrednia mapująca reprezentację logiczną i fizyczną, mimo to pojawiają się widoczne opóźnienia
- otwartość - możliwość niezależnego rozwoju w dowolnym języku programowania
- skalowalność - powinna istnieć możliwość zwiększenia zasobów, rozproszenia geograficznego komponentów i w miarę możliwości zarządzalność
- zabezpieczenie - odrębne organizacje powinny mieć kompatybilne polityki zabezpieczeń, gdyż system taki jest bardziej podatny na atak niż system scentralizowany
- jakość usługi - niezawodność, akceptowalna przepustowość i czas odpowiedzi
- zarządzanie awariami - zaimplementowane mechanizmy wykrywające usterki i automatycznie odtwarzany po awarii

MODELE INTERAKCJI

- komunikacja proceduralna - jeden komputer wywołuje usługę i czeka na odpowiedź
- interakcja oparta na komunikatach - jednostka wysyła zapotrzebowanie o dane, oczekiwanie na odpowiedź nie jest wymagane

WARSTWA POŚREDNIA

- komponenty systemu rozproszonego mogą być realizowane z wykorzystaniem różnych języków programowania

PRZETWARZANIE TYPU KLIENT-SERWER

- użytkownik komunikuje się z aplikacją działającą na lokalnym komputerze, a ona z programem działającym na zdalnym komputerze

WZORCE ARCHITEKTONICZNE

- architektura Master-Slave - systemy czasu rzeczywistego, gdzie odrębne procesory powiązane są ze zbieraniem danych ze środowiska, inne za przetwarzanie, a inne zarządzają sygnalami
- dwuwarstwowa architektura typu klient-serwer - system jest implementowany jako pojedynczy serwer i niezdefiniowana liczba klientów
    - cienki klient to warstwa prezentacji implementowana na kliencie

    - gruby klient to warstwa prezentacji + elementy warstwy aplikacji
- wielowarstwowa architektura klient-serwer - różne warstwy systemu są odrębnymi procesami, wykonywanymi niezależnie
- architektura rozproszonych komponentów - nie istnieje rozróżnienie na klient i serwery, każda jednostka jest obiektem udostępniającym usługi, a komunikacja odbywa się za pośrednictwem warstwy pośredniej
- architektura peer-to-peer - zdecentralizowane, wykorzystuje wiele połączonych komputerów, zdecentralizowane
- oprogramowanie jako usługa - zdalny hosting, udostępnianie za pośrednictwem internetu

SAAS a SOA

- oprogramowanie jako usługa to sposób udostępniania funkcjonalności na zdalnym serwerze, do którego klient uzyskuje dostęp z przeglądarki
- architektura zorientowana usługowo to podejście polegające na budowaniu systemu w postaci zbioru bezstanowych usług

INŻYNIERIA KOMPONENTOWA

- podejście polegające na powtórnym użyciu elementów zwanych komponentami
- pojawiła się jako odpowiedź na porażkę obiektowości w kontekście powtórnego użycia
- komponenty są bardziej abstrakcyjne niż klasy

INŻYNIERIA KOMPONENTOWA WSPIERA

- niezależność - komponenty nie kolidują ze sobą, są niezależne
- hermetyzacja - implementacja ukryta, komunikacja za pomocą interfejsów
- współdzielenie - platformy współdzielą komponenty

KOMPONENTY

- udostępniają usługę niezależnie od miejsca uruchomienia
- komponent to niezależna, wykonywalna jednostka

KOMPONENT JAKO DOSTAWCA USŁUG

- nie musi być kompilowany przed użyciem
- usługa dostępna za pośrednictwem interfejsów
- wewnętrzny stan nie jest eksponowany

WSPARCIE WARSTWY POŚREDNIEJ

- usługi platformy pozwalają na komunikację komponentów zdefiniowanych zgodnie z modelem

USŁUGI INTERNETOWE

- niezależne od aplikacji ją wykorzystującej

ARCHITEKTURA ZORIENTOWANA USŁUGOWO

- komponenty są niezależnymi usługami
- usługo wykonywane na różnych maszynach

ZALETY SOA

- usługi dostarczane lokalnie lub zewnętrznie
- usługi niezależne od języka

STANDARDY USŁUG INTERNETOWYCH

- SOAP - komunikacja z usługami bazującymi na abstrakcji komunikatów
- WSDL - standard pozwalający na opisanie interfejsu usługi oraz sposobu rozwiązania
- WS-BPEL - standard języków pozwalających na definiowane kompozycji usług internetowych

REST

- wzorzec bazujący na przesyłanie przez serwer reprezentacji zasobu
- opiera się na istniejącej infrastrukturze internetowej

INŻYNIERIA USŁUGI

- proces rozwoju usługi w celu jej ponownego wykorzystania w aplikacji zorientowanej usługowo
- usługa zaprojektowana w postaci abstrakcji ponownego użycia

TYPY USŁUG

- usługi ogólne implementujące funkcjonalności wykorzystywane w wielu procesach biznesowych
- usługi biznesowe powiązane ze specyficzną funkcją
- usługi koordynacyjne powiązane z ogólnymi procesami uwzględniające różnych aktorów i wiele aktywności

1. Podstawowym obecnie powodem kryzysu oprogramowania jest:

a) złożoność produktów informatyki i procesów ich wytwarzania

b) zbyt szybko rozwój sprzętu komputerowego

c) użytkownicy systemu i ich nieznajomość informatyki

d) nienaturalność języka maszynowego

2. Hermetyzacja to:

a) Porządkowanie (szregowanie) pojęć w strukturę drzewiastą

b) Rozdzielenie czegoś złożonego na małe łatwiejsze do zarządzania fragmenty

c) Ukrywanie informacji

d) Wyodrębnienie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzenie pojęć lub symboli oznaczających takie cechy

3. Diagramy języka UML dzielą się na dwie grupy:

a) interakcji i klas

b) przypadków użycia i aktywności

c) dynamiczne i statyczne

4. Dwa podstawowe modele architektury repozytorum danych to

a) Modele warstwowy i model obiektowy

b) Model repozytorium i model zdecentralizowany

c) Model zarządcy i model wywołanie-powrót

5. Wymaganie niefunkcjonalne to:

a) akcje jakie system musi wykonywać wraz z ograniczeniami na te akcje

b) funkcje systemu bez specyfikowania implementacji

c) akcje jakie system musi wykonywać bez branie pod uwage ograniczeń

d) ograniczenia, przy których system ma realizować swoje funkcje

6. Modularyzacja to

a) rozdzielenie czegoś złożonego na małe łatwiejsze do zarządzania fragmenty

b) wyodrębnienie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzenie pojęć lub symboli oznaczających takie cechy

c) ukrywanie informacji

d) porządkowanie pojęć w strukturę drzewiastą

7. Skrót CASE oznacza

a) Coumputer Automated of Software Engineering

b) Computer-Aided Software Engineering

c) Computer Augmented Software Evolution

8. Modelowanie pojęciowe oznacza

a) tworzenie modelu projektowego

b) implementację projektu z uwzględnieniem koncepcyjnego języka programowania

c) procesy myślowe towarzyszące pracy nad oprogramowaniem

9. Określenie zakresu przedsięwziecia w fazie strategicznej oznacza

a) określenie systemów, organizacji i użutkowników, z którymi tworzony system ma współpracować

b) określenie fragmentu procesów informacyjnych zachodzących w organizacji, które będą objęte przedsięwzięciem

c) określenie celu biznesowego przedsięwzięcia z punktu widzenia klienta

10. Decyzje, któe powinny być podjętie w fazie strategicznej to

a) wybór inferfejsów oraz podsystemów

b) wybór modelu, narzędzi CASE, stopnia wykorzystania gotowych komponentów

c) wybór obiektów oraz modułów

11. Narzędzia CASE dzielą się na

a) front-case i back-case

b) better-case i worst-case

c) upper-case i lower-case

12. Wymaganie "system musi być dostępny 24h na dobę" należy do grupy

a) wymagań funkcjonalnych

b) wymagań niefunkcjonalnych

c) ograniczeń projektowych

13. Elementami trójkąta kompromisu są

a) funkcje, zakres przedsięwzięcia, cechy systemu

b) harmonogram, zakres, cechy

c) zasoby, harmonogram, cechy systemu

14. Podstawowe zasady obiektowości to

a) Hermetyzacja, abstrakcja, dekompozycja, generalizacja

b) Hermetyzacja, abstrakcja, modularyzacja, hierarchizacja

c) Hermetyzacja, dekompozycja, modularyzacja, hierarchizacja

d) Hermetyzacja, tożsamość, abstrakcja, modularyzacja

15. Dwa podstawowe modele architektury związane z kontrolą przepływu sterowania pomiędzy podsystemami to

a) model warstwowy i model obiektowy

b) model kontroli scentralizowanej oraz model kontroli sterowanej zdarzeniami

c) model zarządcy i modle wywołanie-powrót

16. Wymaganie "użytkownik może wybrać przedział czasowy raportu" zaliczamy do grupy

a) wymagań niefunkcjonalnych

b) wymagań funkcjonalnych

c) ograniczeń projektowych

d)

17. Narzędzia RAD to

a) generatory losowe

b) narzędzia umożliwiające szybką budowę prototypów lub gotowych aplikacji

c) narzędzia umożliwiające testowanie oprogramowanie

18. Model przypadków użycia służy do opisu

a) wymagań funkcjonalnych

b) wymagań niefunkcjonalnych

c) przypadków testowych

19. Jedna z zasad modelowania zwraca uwagę na to, że

a) wybór modelu nie ma wpływu na to jak będziemy postrzegać rzeczywistość

b) pojedynczy model jest niewystarczający

c) należy modelować przy użyciu języka UML

20.Model sterowania, w którym jeden z podystemów steruje rozpoczynaniem, zatrzymaniem i koordynacją pozostałych procesów nazywa się

a) Modelem z przerwaniami

b) Modelem wywołanie-powrót

c) Modelem zarządcy

d) Modelem rozgłaszania

21. Atrybuty dobrego oprogramowania to

a) Funkcjonalność, wydajność, bezpieczeństwo, czytelność

b) Wiarygodnosć, sprawność, użyteczność, zarządzalność

c) Sprawność, wydajność, bezpieczeństwo, jakość

22. Koty, psy, rybki są typami zwierząt domowych. Zdanie to ilustruję zasadę obiektowości nazywaną

a) zasadą abstrakcji

b) zasadą hermetyzacji

c) zasadą modularyzacji

d) zasadą hierarchizacji

23. Wymagania niefunkcjonalne to

a) ograniczenia, przy których system ma realizować swoje funkcje

b) akcje jakie systemu musi wykonywać wraz z ograniczeniami na te akcje

c) funkcje systemu bez specyfikowania implementacji

d) akcje jakie system musi wykonywać bez brania pod uwagę ograniczeń

24. Formalizm matematyczny w specyfikacji wymagań

a) stosuje się rzadko, do specyficznych celów

b) stosuje się często, szczególnie dla aplikacji biznesowych

c) nie jest stosowany

26. Cztery zasadnicze czynności wykonywane w procesie tworzenia oprogramowania to

a) iteracja, walidacja, zatwierdzenie, pielęgnacja

b) specyfikacja, tworzenie, zatwierdzenie, ewolucja

c) planowanie, analiza, projektowanie, programowanie

27. Faza strateficzna jest nazywana również

a) studium osiągalności

b) fazą analizy wymagań

c) fazą projektowania

28. Co to jest proces inżynierii oprogramowania?

a) zestaw działan, których celem jest wytworzenie lub ewolucja oprogramowania

b) proces modelowania systemu

c) zestaw działań, których celem jest oszacowanie nakładów potrzebnych do wytworzenia oprogramowania

29. Wymaganie "system musi być napisany w języku Java" należy do grupy

a) wymagań niefunkcjonalnych

b) ograniczeń projektowych

c) wymagań funkcjonalnych

30. Tworzenie systemu zorientowanego usługowo oznacza

a) integracja istniejących systemów poprzez zdefiniowanie zestawu interfejsów i udostępnienie ich funkcjonalności za pomocą tych interfejsów

b) tworzenie systemu na zasadzie integracji komponentó zgodnym ze standardem modelu komponentowego

c) tworzenie systmeu na zasadzie łączenia współdzielonych usług, które mogą być dostarczane zewnętrznie

31. Wymagania funkcjonalne to

a) funkcje systemu i sposób ich implementacji

b) akcje jakies sytem musi wykonywać wraz z ograniczeniami na te akcje

c) akcje jakies system musi wykonywać bez brania pod uwage ograniczeń

d) atrybuty (cechy) systemu

32. Model sterowania, w którym sterowanie zaczyna się na wierzchołku hierarchi i przez wywoływanie podprogramów przechodzi do najniższych poziomów drzewa wywołań nazywa się

a) modele z przerwaniami

b) modele zarządcy

c) modelem rozgłaszania

d) modelem wywołanie-powrót

33. Wymagania stawiane oprogramowaniu dzielą się na

a) wymagania dotyczące interfejsu, sprzętu i oprogramowania

b) wymagania funkcjonalne, systemowe i niezawodnościowe

c) wymagania funkcjonalne i niefunkcjonalne oraz ograniczenia projektowe

34. Określenie zakresu przedsięwzięcia w fazie strategicznej oznacza

a) określenie fragmentu procesów informacyjnych zachodzących w organizacji, które będą objęte przedsięwzięciem

b) określenie systemów, organizacji i użytkowników, z którymi tworzony system ma współpracować

c) określenie celu biznesowego przedsięwzięcia z punktu widzenia klienta

35. Diagramy języka UML dzielą się na dwie grupy

a) dynamiczne i statyczne

b) interakcji i klas

c) przypadków użycia i aktywności

36. Metoda szacowania kosztów zwana Metoda Punktów Funkcyjnych należy do kategorii

a) szacowania przez osąd ekspertów

b) metod parametrycznych

c) szacowania przez analogię

d) szacowania wstępującego

37. Wzorzec projektowy to

a) projekt podsystemu złożony z kolekcji klas (abstrakcyjnych i konkretnych) oraz interfejsów

b) mechanizm umożliwiający ponowne użycie wiedzy na temat problemu i sposobu jego rozwiązania

c) tworzenie systemó na zasadzie integracji istniejących aplikacji

38. Elementami trójkata kompromisu są

a) funkcje, zakres przedsięwzięcia, cechy systemu

b) zasoby, harmonogram, cechy systemu

c) harmonogram, zakres, cechy

39. Model sterowania, w którym zdarzenie jest przesyłane do wszystkich podsystemów nazywa się

a) modelem z przerwaniami

b) modelem wywołanie-powrtów

c) modelem zarządcy

d) modelem rozgłaszania

40. Dekompozycja obiektowa powoduje, że

a) struktura podsystemu jest zbiorem luźno powiązanych obiektów z dobrze zdefiniowanymi interfejsami

b) podsystem dekomponowany jest do postaci modułów funkcjonalnych transformujących wejscie na wyjście

c) jeden podsystem jest odpowiedzialni za wykonywanie zadań

41. Podstawowym obecnie powodem kryzysu oprogramowania jest

a) złożoność produktów informatyki i procesów ich wytwarzania

b) użytkownicy systemu i ich nieznajomość informatyki

c) zbyt szybko rozwój sprzętu komputerowego

d) nienaturalność języka maszynowego

42. Przukładem dziedinowego modelu odniesienia jest

a) model klient-serwer

b) model warstwowy

c) generyczny model architektury kompilatora

d) model ISO/OSI

43. Generyczny model architektury kompilatora jest przykładem

a) modelu warstwowego

b) dziedzinowego modelu odniesienia

c) dziedicznowego modelu ogólnego

d) modelu architektury klient-serwer

44. Wadą modelu sterowania z rozgłaszaniem jest

a) ograniczona liczba podsystemów obsługi zdarzeń

b) możliwe konflikty w obsłudze zdarzeń

c) skomplikowana ewolucja systemu

d) podsystemu nie wiedzą czy i kiedy zdarzenie zostanie obsłużone

45. Na korki zarządzania zmianą wymagania składają się czynności

a) raport o błędach w wymaganiach

b) analiza zmiany i kosztów

c) analiza problemu i specyfikacja zmiany

d) implementacja zmiany

46. Wydanie oprogramowania

a) strona biznesowa oprogramowania musi być dostępna w momencie wydania

b) może być publiczne i prywatne

c) może być pierwotne i aktualizacyjne

d) jest dostarczane dla klienta tylko na fizycznym nośniku

47. Orkiestra składa się z kilku sekcji, perskusyjnej, smyczkowej, dętej blaszanej i dęte drewnianej. Zadanie to ilustruje zasadę obiektowości zwaną:

a) modularyzacji

b) hermetyzacji

c) abstrakcji

d) hierarchizacji

48. Jednym z najbardziej znanych zrębów do projektowania graficznego interfejsu użytkownika jak i interfejsu webowego nazywa się

a) GUI

b) Singleton

c) MVC

d) ASP.NET

49. Które z poniższych stwierdzeń przedstawia jedną z podstawowych zasad metodyki RUP

a) implementujemy dopiero po zaprojektowaniu całości systemu

b) miarą postępu jest działający kod

c) najważnieje jest posiadanie odpowiedniego personelu projektowego

50. Proces zarządzania ryzykiem składa się z następujących czynności

a) identyfikacja, analiza, planowanie, monitorowanie

b) identyfikacja, planowanie, monitoring, iteracja

c) definiowany macierzy kompromisu, planowanie, szacowanie, zarządzanie prokejtem

d) reorganizacja projektu, aby ryzyko nie miało wpływu, planowanie iteracji, ewaluacja

51. Metodyka to

a) zestaw pojęc, modeli, języków, technik i sposobów psotępowania

b) wymagania dla systemu zapisane w języku formalnym

c) notacja służąca do definiowana modeli

d) realizacja pełnego systemu zgodnie z modelem kaskadowym

52. Testem jednostkowym może być objęta

a) procedura

b) moduł

c) podsystem

d) klasa obiektów

53. Celem testowania defektów/usterek oprogramowania jest

a) wykazanie, że system nie posiada defektów

b) ujawnienie utajnionych defektów w systemie

c) wykazanie, że system spełnia swoją specyfikacje

d) ujawnienie defektów w specyfikacji wymagań dla systemu

54. Dwa podstawowe modele architektury przepływu sterowania między podsystemami to

a) sterowanie współbieżne oraz sterowanie sekwencyjne

b) sterowanie za pomocą rozgłaszania oraz sterowanie scentralizowane

c) sterowanie scentralizowane oraz sterowanie zdarzeniowe

d) sterowanie wywołanie-powrót oraz sterowanie za pomocą przerwań

55. Dwa rodzaje modeli architektonicznych charakterystycznych dla dziedziny nazywane są:

a) wzorcami obiektowymi oraz wzorcami projektowymi

b) wzorcami architektonicznymi oraz wzorcami dziedzinowymi

c) modelami ogólnymi oraz modelami odniesienia

d) modelami środowiskowymi oraz modelami fabryk oprogramowania

56. Test wydania

a) są testami defektów

b) mają za zadanie wykazać, że system spełnia swoją specyfikację

c) testowane jest kompletny system

d) są testami zatwierdzającymi

57. Na proces oparty na elementach wielkrotnego użycia składają się

a) specyfikacja wymagań, analiza komponentów, modyfikacja wymagań, projekt, budowa i integracja, zatwierdzenie

b) nie ma tekigo procesu inżynierii oprogramowania

c) równoległa specyfikacja wymagań, projektowania i budowa systemu

d) specyfikacja wymagań, analiza i projektowanie,e implementacja, testowanie, zatwierdzanie

58. Abstrakcja to

a) wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzenie pojęć lub symboli oznaczających te cechy

b) rodzielenie czegoś złożonego na małe łatwiejsze do zarządzania fragmenty

c) porządkowanie pojęć w strukturę drzewiastą

d) ukrywanie informacji

59. Testowanie metoda białej skrzynki to

a) podejsćie w którym testy wyprowadza się ze specyfikacji programu

b) testowanie integracyjne

c) tstowanie metodą TDD

d) podejście, w którym testy opracowuje się na podstawie znajomości struktury programu

60. Rezultatem procesu analizy wymagań jest opis/model składowej

a) zarządzania zadaniami

b) zarządzania danymi

c) dziedziny problemu

d) zarządzania pamięcią

61. Inspekcje oprogramowani są przykładem

a) statycznej analizy wybranych reprezentacji systemu

b) dynamicznej analizy przez obserwacje zachowania systemu

c) procesu kontroli oprgoramowania w trakcie jego działania w środowisku docelowym

62. Przykładem systemu opartego na kontroli wersji jest

a) oprogramowanie wiki

b) system bankowy

c) oprogramowanie sterowania robotem przemysłowym

d) Eclipse IDE

63. Metodyka to

a) zestaw pojęć, notacji, modeli, języków, technik i sposobów postępowania

b) notacja służąca do definiowania modeli

c) wymagania dla systemu zapisane w języku formalnym

d) realizacja pełnego systemu zgodne z modelem kaskadowym

64. Uczelnia jest instytycją, gdzie wykładowcy prowadzą zajęcia dla studentów na wielu kierunkach. Zdanie to ilustruję zasadę obiektowości nazywaną

a) zasadą abstrakcji

b) zasadą hermetyzacji

c) zasadą hierarchizacji

d) zasadą modularyzacji

65. Wadą modelu kaskadowego procesu IO jest

a) narzuczenie twórcom oprogramowania ścisłej kolejności wykonywania prac

b) brak fazy testowania

c) duże ryzyko utrzymujące się przez cały projekt

d) zbyt duża liczba faz

66. Generyczny model architektury kompilatora jest przykładem

a) dziedzinowego modelu ogólnego

b) dziedzinowego modelu odniesienia

c) modelu warstwowego

d) modelu architekruty klient-serwer

67. Elementami procesu projektowania obiektowego (OOD) są

a) identyfikacja głównych obiektów systemu

b) identyfikacja wymagań systemu

c) wyspecyfikowanie interfejsów obiektów

d) wyspecyfikowanie przypadkow użycia obiektów

68. W rozproszonym modelu kontroli wersji

a) każda stacja robocza przechowuje własną kopię repozytorium

b) jest jedno centrolne repozytorium i rpozproszone stacje robocze

c) wersja jest większana w momencie zapisania zmian na centralnym serwerze

d) nie istnieje potrzeba synchronizacji przy łączeniu niezależnych gałęzi kodu

69. Dwa rodzaje modeli architektonicznych charakterystycznych dla dziedziny nazywane są

a) modelami ogólnymi oraz modelami odniesienia

b) wzorcami architektonicznymi oraz wzorcami dziedzinowymi

c) modelami środowiskowymi oraz modelami fabryk oprogramowania

d) wzorcami obiektowymi oraz projektowymi

70. Testem jednostkowym może być objęta

a) klasa obiektów

b) podsystem

c) moduł

d) procedura

71. Kamień milowy

a) końcowy punkt aktywności

b) podział projektu na zadania

c) rezultat projektu dostarczany użytkownikom

d) reprezentowany zywkle jako aktywność o zerowym czasie trwania i zerowych zasobach

72. Metoda Deplhi to

a) metoda szacowania nakładów pracy

b) metoda określania jakości oprogramowania

c) metoda szacowania czasu trwania projektu

d) nie ma takiej metody

73. Kroki zarządzania zmianą wymagania

a) analiza problemu i specyfikacja zmiany

b) analiza zmiany i ocena kosztów

c) implementacja zmiany

d) raport o błędach w wymaganiach

74. Metoda PDM to

a) metoda pozwalająca na określenie minimalnego czasu trwania projektu

b) graficzna reprezentacja aktywności projektu w postaci sieci aktywności

c) metoda polegająca na konstruowaniu sieci aktywności projektu

d) metoda wykorzystywana w zarządzaniu ryzykiem do identyfikacji zagrożeń

75. Traceability to

a) zarządzanie ramami projektu

b) zdolność śledzenia zależnośći pomiędzy wymaganiami a elementami projektowyi

c) cechy systemu w ujęciu funkcjonalnym

76. W przypadku systemów dopasowanych

a) koszty ewolucji moga znacznie przewyższać koszty budoyw

b) koszty testowania są proporcjonalne do kosztów specyfikowania systemu

c) stosuje się wyłącznie proces ewolucyjny

77. Moduł

a) jest niezależny

b) jest częścią składowa podsystemu

c) jest zawsze wyposażony w interfejs