Oprogramowanie

Oprogramowanie generyczne:

- jak oprogramowanie działa decyduje wykonawca

- decyzje związane ze zmianami podejmuje wykonawca

- systemy tworzone odrębnie i sprzedawane na rynku(szerokiemu zakresowi klientów)

- np. programy graficzne, systemy CAD, systemy dla określonej dziedziny

Oprogramowanie dopasowane:

- jak oprogramowanie działa decyduje klient

- decyzje związane ze zmianami podejmuje klient

- wyprodukowane dla konkretnego klienta na zamówienie(zgodnie z dostarczonymi wytycznymi)

Model dostarczania:

- Instalacja po stronie klienta(np. na komputerze lub serwerze klienta)

Klient –> właściciel

- Udostępnianie zdalne (system na serwerze dostawcy)

Dostawca -> właściciel; Klient -> dzierżawca

Atrybuty dobrego oprogramowania:

-Zarządzalność

- Wiarygodność i Bezpieczeństwo

- Sprawność

- Użyteczność

Inżynieria Oprogramowania:

Inżynieria Oprogramowania to dyscyplina inżynieryjna dotycząca wszystkich aspektów związanych z wytwarzaniem oprogramowania (od koncepcji do wspierania po uruchomieniu)

Dyscyplina inżynieryjna Wykorzystanie teorii i metod do rozwiązywania problemów z ograniczeniami finansowymi oraz organizacyjnymi

Czynności procesu Inżynierii Oprogramowania:

Specyfikacja oprogramowania – klient i inżynier definiują oprogramowania oraz ograniczenia przy których ma dalej działać

Rozwój oprogramowania – oprogramowanie jest projektowane i implementowane

Zatwierdzanie (walidacja) oprogramowania – sprawdzanie programu pod kątem wymagań klienta

Ewolucja Oprogramowania – modyfikacja programu w celu uwzględnienia zmian w wymaganiach klienta

Problemy mające wpływ na oprogramowanie:

-Heterogeniczność – systemy muszą działać na różnych środowiskach (mobile, PC)

- **Zmiany środowiska biznesowego i społecznego** – technologia idzie do przodu, więc trzeba szybko produkować oprogramowanie lub wprowadzać zmiany w już istniejących

- **Bezpieczeństwo i zaufanie** – trzeba stworzyć oprogramowanie które będzie budzić zaufanie

Inżynieria systemów – jest związana z wszelkimi aspektami budowy i ewolucji złożonych systemów

Niejednolitość Inżynierii Oprogramowania:

- Istnieje wiele różnych typów oprogramowania – nie ma uniwersalnych narzędzi i technik

- Metody i narzędzia wykorzystywane do produkcji oprogramowania różnią się w zależności:

***Typ rozwijanej aplikacji**

***Wymagania klienta**

***Umiejętności, wiedza i doświadczenie zespołu projektowego**

***Dostępność narzędzi**

Typy aplikacji:

- **Aplikacje samodzielne(lokalne)** – działają na lokalnym komputerze bez podłączania do sieci

- **Interaktywne aplikacje transakcyjne** – aplikacje wykonywane na komputerze zdalnym za pośrednictwem własnego komputera

- **Wbudowane systemy kontrolne** – Oprogramowanie kontrolujące i zarządzające działaniem urządzeń. (Dominuje rynek)

- **Systemy przetwarzania wsadowego** – Przeznaczone do przetwarzania dużej ilości danych

- **Systemy rozrywkowe** – do użytku osobistego, np. gry

- **Systemy do modelowania i symulacji** – opracowywane przez naukowców do modelowania fizycznych zjawisk (np. femm)

- **Systemy gromadzenia danych** – zbierają dane za pomocą czujników i przesyłają dalej w celu przeanalizowania

- **Systemy systemów** – składają się z wielu innych systemów

Niezależnie od typu aplikacji, 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 oprogramowania

- Recykling oprogramowania – jeśli mamy jakiś fragment już stworzony należy go wykorzystać

Inżynieria Oprogramowania a Internet

- Internet jest platformą wykonawczą coraz większej ilości aplikacji

- Usługi Internetowe pozwalają na dostęp do funkcjonalności za pomocą Internetu

- Chmury obliczeniowe to podejście gdzie użytkownik dzierżawni oprogramowanie korzystając z chmury

Internet:

- Internet doprowadził do znaczącej zmiany w sposobie organizacji oprogramowania biznesowego.

- Ważnym aspektem w aplikacjach webowych jest wielokrotne użycie

- Systemy webowe powinny być rozwijane i dostarczane przyrostowo.

- Interfejsy użytkownika są ograniczane możliwościami przeglądarek Internetowych.

Proces Inżynierii 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;

- **Zatwierdzanie** – sprawdzanie czy system wykonuje to czego oczekuje klient;

- **Ewolucja** – modyfikowanie systemu w odpowiedzi na zmiany potrzeb klienta.

Procesy:

Procesy sterowane planem to procesy, gdzie wszystkie czynności procesu są zaplanowane

W **Procesach zwinnych** planuje się na bieżąco co pozwala szybciej zaadaptować zmiany w wymaganiach

Modele procesu Inżynierii Oprogramowania:

- Model Kaskadowy

- Realizacja przyrostowa

- Inżynieria zorientowana na wielokrotne użycie

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

Wady tego modelu to:

- trudność adaptacji zmian pojawiających się w czasie realizacji procesu
- Brak elastyczności poprzez podział projektu na oddzielne etapy

Model kaskadowy jest wykorzystywany dla realizacji dużych systemów, gdzie prace odbywają się w kilku miejscach.

Realizacja przyrostowa:

Zalety:

- Redukcja kosztów adaptacji zmian w wymaganiach
- Łatwiej jest uzyskać informację zwrotną na temat dotychczas wykonanych prac.
- Możliwe jest szybsze dostarczenie i wdrożenie użytecznego oprogramowania.

Wady:

- Niejawność procesu(program jest wytwarzany szybko więc nie tworzymy dokumentacji)
- Struktura systemu ma tendencję do degradacji wraz z dodawaniem nowych zmian

Inżynieria zorientowana na wielokrotne użycie:

Fazy procesu:

- Analiza komponentów/usług;
- Modyfikacja wymagań;
- Projektowanie z wielokrotnym użyciem;
- Rozwój i integracja.

Opiera się na systematycznym podejściu do wykorzystania istniejącego oprogramowania

Redukcja kosztów modyfikacji:

- Unikanie zmian – przewidywanie ewentualnych zmian na poziomie pisania programu
- Tolerowanie zmian – proces tworzenia skonstruowany tak aby obniżyć koszty zmian

Prototypowanie:

Prototyp – to początkowa wersja systemu, gdzie demonstrujemy i próbujemy koncepcje projektowe

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.

Prototyp może wykorzystywać dedykowane języki i narzędzia

Prototyp 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.

Prototypy powinny być porzucane po opracowaniu ponieważ nie stanowią dobrej bazy do produkcji systemu

Dostarczanie przyrostowe:

- Podzielić dostarczanie systemu na przyrosty, w którym każdy z nich będzie zawierał część 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.

Realizacja 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

Agile Software Development

Metody zwinne:

- Niskie koszty

Zasady metod zwinnych:

- Zaangażowanie klienta – dostarczenie oraz priorytetyzacja wymagań oraz ocena iteracji
- Dostarczanie w przyrostach
- Ludzie a nie proces
- Akceptowanie zmian na bieżąco
- Utrzymanie prostoty

Kiedy stosujemy metody zwinne:

- produkty niewielkie i średnie
- Niestandardowy system w ramach organizacji gdzie łatwo zaangażować klienta
- przy małych i silnie zintegrowanych zespołach

Problemy metod zwinnych:

- Klient może stracić zainteresowanie
- zespół nie przyzwyczajony do poziomu intensywności
- problemy z priorytetyzacją zadań
- utrzymanie prostoty systemu
- Kontrakt z klientem może stanowić problem

Extreme Programming(XP):

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

XP i zasady metod zwinnych:

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

- Utrzymywanie prostoty przez ciągłą refaktoryzację.

Praktyki w XP:

- Planowanie przyrostowe
- Małe wydania
- Prosty projekt
- Rozwój sterowany testami
- Refaktoryzacja
- Programowanie w parach (programiści kontrolują i wspierają się wzajemnie)
- Wspólna własność kodu
- Ciągła integracja
- zrównoważone tępo
- klient „w miejscu" – klient jest częścią grupy i jest dostępny cały czas

Scenariusze wymagań:

- Klient będący członkiem podejmuje decyzje dotyczące wymagań
- wymagania użytkowników są wyrażane w scenariuszach „user stories"(historyjki)
- scenariusze są zapisywane na kartkach a zespój je rozbija na zadania do zaimplementowania
- klient wybiera scenariusze do realizacji w ramach kolejnego zadania

W XP zakłada się że nie warto poświęcać czas na przewidywanie zmian, w zamian za to proponowane jest podejście oparte o stałe ulepszanie kodu(refaktoryzację).

Refaktoryzacja

- jeśli zespół znajdzie możliwość udoskonalenia programu to to robi niezależnie czy jest to potrzebne
- zaletą jest zwiększenie zrozumiałości kodu co niweluje potrzebę tworzenia dokumentacji
- Wprowadzanie zmian jest prostsze bo kod jest lepszy
- wadą jest to że niektóre problemy wymagają zmian na poziomie architektury co generuje koszty
-przykłady: stosowanie „Inline", Reorganizacja hierarchii klas

Testowanie w XP:

Zakłada się że cały kod jest testowany po każdej zmianie

Przystosowanie do testów:

- Rozwój sterowany testami
- przyrostowe testy
- Zaangażowanie użytkowników w opracowywanie testów i ich walidacje(np. BETA)
- Używanie narzędzi automatycznie uruchamiających testy w trakcie budowania

Rolą klienta w testowaniu jest pomoc w opracowaniu testów, pisanie testów

Programowanie w parach:

- programiści siedzą w parze przy jednym komputerze i pracują nad kodem
- pary są tworzone dynamicznie więc pomaga to w poczuciu wspólnej własności kodu
- Każda linijka jest przejrzana przez więcej niż jedną osobę
- takie podejście zachęca do refaktoryzacji
- współdzielenie wiedzy
- Produktywność jest podobna lub większa do dwóch programistów pracujących niezależnie

Zwinne zarządzanie projektem:

- kierownik projektu zarządza projektem, (zarządza: budżetem, czasem)
- kierowniczy przygotowują plan dla projektu określając: Co?, Kiedy?, Kto?, Gdzie?, I jak?.

Metodyka Scrum:

Scrum składa się z trzech faz:

- Faza inicjacyjna – ustalanie ogólnych cech projektu
- Sprinty – seria iteracji, w której każda realizuje przyrost systemu
- Faza końcowa projektu – zamykanie projektu i uzupełnianie dokumentacji

Sprint

Sprinty mają określoną długość(2-4 tygodnie) – są odpowiednikiem wydania systemu w metodzie XP

Punktem wyjścia jest tzw. „**product backlog”**, czyli priorytetyzowany rejestr wymagań do wykonania w trakcie trwania projektu. Ustalany z klientem który ma role„właściciel produktu”(„**product owner**”)

Dla pojedynczego sprintu zespół wraz z klientem wybiera elementy z **rejestru wymagań**, które mają zostać utworzone w jego ramach (na tej podstawie tworzymy rejestr zadań sprintu „**sprint backlog**”)

Po ustaleniu wszystkiego zespół zbiera się do tworzenia oprogramowania. Teraz zespół zostaje odizolowany od klienta i komunikacja między nim za zespołem realizowana jest przez członka zespołu o roli **Scrum master**.

Scrum master ma za zadanie chronić zespół przed zakłóceniami z zewnątrz, mentorować i monitorować postępy

Po zakończeniu prezentowane są rezultaty i rozpoczyna się kolejny sprint

Scrum master:

- organizuje spotkania
- dokonuje pomiarów
- zapisuje podjęte decyzje
- komunikuje się z klientem

Cały zespół:

- bierze udział w spotkaniach
- informują się o postępach
- informują się o problemach
- informują się o planach na najbliższy okres pracy

Zalety metody Scrum:

- podzielony produkt jest łatwiejszy do zarządzania
- lepsze zrozumienie wymagań nie powoduje przerw w pracy
- lepsza komunikacja, cały zespół wie wszystko
- klienci otrzymują przyrosty na czas i wysyłają informacje zwrotne
- budowane jest zaufanie pomiędzy klientem a zespołem

Skalowane metod zwinnych:

- Mogą być realizowane przy małych i średnich projektach, przez małe i zintegrowane zespoły
- ze względu na mały zespół komunikacja jest usprawniona, a zatem większy sukces
- Skalowanie w większych projektach wymaga zmiany ich, aby można było wykorzystać w większych, dłużej trwających projektach oraz przy dużej ilości osób

Wytwarzanie dużych systemów:

- jest to kolekcja niezależnych współpracujących systemów realizowanych osobno
- są mocno powiązane z istniejącymi systemami
- przy integrowaniu systemów więcej pracy jest z konfiguracją niż pisaniem nowego kodu
-są często ograniczane przez zewnętrzne regulacje
- Długi czas realizacji zamówienia i rozwoju
- składa się z wielu zespołów które ciężko ze sobą zintegrować

Scaling out i up:

Scaling up – skalowanie wertykalne(metody zwinne które nie mogą być realizowane przez małe zespoły)

Scaling out – skalowanie horyzontalne(wprowadzanie metod zwinnych w dużych organizacjach)

Skalowanie musi zachować fundamenty zwinności

Inżynieria Oprogramowania

Model procesu inżynierii:

- abstrakcyjna reprezentacja procesu

- Opis procesu z określonej perspektywy

Projektowanie i implementacja oprogramowania:

-Proces którego celem jest przekształcenie specyfikacji w działający system

- Projektowanie – określenie struktury oprogramowania, która realizuje specyfikacje

- implementacja – translacja struktury na wykonalny program

Projektowanie i implementacja są ze sobą mocno powiązane i często się przeplatają

Zatwierdzanie (walidacja) oprogramowania:

- Celem weryfikacji i zatwierdzania jest wykazanie, że system jest zgodny ze specyfikacją i spełnia wymagania

- 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.

Ewolucja oprogramowania:

- Oprogramowanie musi być elastyczne bo zmiany są nieuniknione
Zmieniają się uwarunkowania biznesowe -> zmieniają się wymagania -> ewoluuje system

Granica pomiędzy ewolucją a budową systemu się zaciera, więc jest coraz mniej nowych systemów

/*Podsumowanie*/

Procesy IO to czynności obejmujące produkcję oprogramowania. Modele procesów IO to abstrakcyjne ich reprezentacje.

Ogólne modele procesu opisuję podstawową organizację procesu IO. Przykładami są: **model kaskadowy, realizacja przyrostowa** oraz **wytwarzanie zorientowane na powtórne użycie**.

Inżynieria wymagań to proces rozwoju specyfikacji oprogramowania.

Projektowanie i implementacja związane są z transformacją specyfikacji wymagań w wykonywalne oprogramowanie.

Zatwierdzanie oprogramowania to proces kontroli, której celem jest sprawdzenie czy system jest zgodny ze specyfikacją i czy spełnia rzeczywiste potrzeby użytkowników systemu.

Ewolucja oprogramowania to dostosowywanie istniejącego oprogramowania do nowych wymogów. Aby pozostać użyteczne – oprogramowanie musi ewoluować.

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. – nie wiemy co będą potrzebować konkretne systemy tworzone w kolejnych przyrostach

- Istotą procesu iteracyjnego jest to, że specyfikacja jest opracowywana w połączeniu z oprogramowaniem.

RUP – The Rational Unified Process

UML

Łączy w sobie elementy 3 generycznych procesów (kaskadowy, realizacja przyrostowa i zorientowanie na wielokrotne użycie).

Opisywany z trzech perspektyw:

- Dynamicznej – fazy funkcji czasu

- statycznej – czynności procesu (workflow)

- Pragmatycznej – sugestie wskazówki, dobre praktyki, narzędzia

Dobre praktyki w **RUP**:

- Buduj przyrostowo

- Zarządzaj wymaganiami

- wykorzystuj architekturę komponentową

- Modeluj wizualnie

- Stale weryfikuj jakość

- Zarządzaj zmianami

Perspektywa statyczna w RUP – **Proces pracy**

- **Modelowanie biznesowe** - Procesy biznesowe modelowane są za pomocą biznesowych przypadków użycia

- **wymagania** - Aktorzy będący w interakcji z systemem oraz przypadki użycia modelujące wymagania dla systemu.

- **analiza i projektowanie** - Model projektowy budowany jest z wykorzystaniem modeli architektonicznych, modeli komponentowych, modeli obiektowych oraz modeli sekwencji.

- **Implementacja** - Komponenty systemu są implementowane i reprezentowane w postaci podsystemów. Wykorzystuje się możliwości automatycznej generacji kodu w celu przyspieszenia procesu.

- **Testowanie** - Testowanie jest procesem iteracyjnym realizowanym r powiązaniu z implementacją. Po zakończeniu implementacji wykonywane są testy systemowe

- **Wdrożenie** - Tworzone jest wydanie systemu, które jest dystrybuowane do użytkowników.

- **Zarządzanie konfiguracją i zarządzanie** zmianą – proces wspierający – zarządzanie zmianami w systemie

- **Zarządzanie projektem** – proces wspierający – zarządzanie rozwojem systemu

- Środowisko - Proces pracy związany z zapewnianiem dostępności odpowiedniego zestawu narzędzi CASE.

/*Podsumowanie*/

Proces musi obejmować czynności umożliwiające radzenie sobie ze zmianami. Przykładem takiej czynności jest etap prototypowania pozwalający na zmniejszenie ryzyka błędnych decyzji dotyczących wymagań i projektu systemu.

Proces może być tak skonstruowany aby umożliwiać rozwój oraz dostarczenie przyrostowe. Dzięki iteracyjności zmiany mogą być wprowadzane przy minimalizacji ryzyka naruszenia całości systemu.

Przykładem nowoczesnego generycznego modelu procesu IO jest **Rational Unified Process**. Proces ten zorganizowany jest w fazy (rozpoczęcie, opracowanie, konstrukcja, przekazanie) odseparowane od konkretnych czynności (wymagania, analiza i projektowanie,...).

Inżynieria wymagań

Wymaganie:

Może być definiowane:

- od abstrakcyjnego opisu usług lub ograniczeń systemu

- do szczegółowej matematycznej specyfikacji funkcjonalności

Z punktu widzenia celu wymaganie:

- Może być podstawą oferty kontraktowej – czyli musi być otwarte na interpretacje

- Może być podstawą samego kontraktu – czyli musi być jednoznaczne i szczegółowe

Wymagania użytkownika:

- dokument napisany tak by klient mógł zrozumieć o co chodzi, znajdują się tam także diagramy, ukazuje usługi systemu wraz z ograniczeniami

Wymagania systemowe:

- Dokument o określonej strukturze, ma w sobie szczegóły funkcjonalności systemu, jego usług i ograniczeń

- Definiuje co ma być zaimplementowane(może być podstawą kontraktu)

Rodzaje 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 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, 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** – Wymagany czas szkolenia, czas wykonania poszczególnych zadań, ergonomia interfejsu, pomoc, dokumentacja użytkownika

- **Niezawodność** – Dostępność, średni czas międzyawaryjny (MTBF), średni czas naprawy (MTTR), dokładność, maksymalna liczba błędów.

- **Efektywność** – Czas odpowiedzi, przepustowość, czas odpowiedzi, konsumpcja zasobów, pojemność

- **Zarządzalność** – Łatwość modyfikowania, skalowalność, weryfikowalność, kompatybilność, możliwości konfiguracyjne, serwisowe, przenaszalność

Wymagania niefunkcjonalne powinny być **weryfikowalne**, tj. powinna istnieć możliwość sprawdzenia czy system je rzeczywiście spełnia.

Cechy:

- **wydajność**

- Liczba transakcji obsłużonych w ciągu sekundy
- Czas odpowiedzi
- Szybkość odświeżania ekranu

- **Zasoby**
  - Wymagana pamięć RAM
  - Wymagana pamięć dyskowa

- **Łatwość użytkowania**
  - Czas niezbędny dla przeszkolenia użytkowników
  - Liczba stron dokumentacji

- **niezawodność**
  - Prawdopodobieństwo błędu podczas realizacji transakcji
  - Średni czas pomiędzy błędnymi wykonaniami
  - Dostępność (procent czasu w którym system jest dostępny)
  - Czas restartu po awarii systemu
  - Prawdopodobieństwo zniszczenia danych w przypadku awarii

- **Przenaszalność**
  - Procent kodu zależnego od platformy docelowej
  - Liczba platform docelowych
  - Koszt przeniesienia na nową platformę

Ograniczenia projektowe

Ten rodzaj wymagań nakłada ograniczenia na projekt systemu lub proces, którego używamy do budowy.

- Produkt musi spełniać normę ISO 601
- Proces wytwarzania musi być zgodny ze standardem DOD 1200-34

Mają często negatywny wpływ na elastyczność projektantów

Użyj systemu Oracle, programuj w Visual Basic, użyj biblioteki klas XYZ.

Wymagania a inżynieria wymagań

Wymagania – opis usług i ograniczeń systemu generowany w procesie inżynierii wymagań

Inżynieria wymagań - proces pozyskiwania, analizowania, dokumentowania oraz weryfikacji wymagań (czyli zarządzanie wymaganiami)

Dlaczego inżynieria wymagań?

Niezbędna umiejętność – pozyskiwanie wymagań od użytkowników i klientów

- Zamiana celów klienta na konkretne wymagania zapewniające osiągnięcie tych celów.
- Klient rzadko wie, jakie wymagania zapewnią osiągniecie jego celów.
- Jest to tak naprawdę proces, konstrukcji zbioru wymagań zgodnie z postawionymi celami.

Organizacja i dokumentowanie wymagań

- Setki, jeżeli nie tysiąca wymagań jest prawdopodobnie związanych z systemem
- Według klienta prawdopodobnie wszystkie z nich są najważniejsze w 100%.
- Większość z nas nie może pamiętać więcej niż kilkadziesiąt informacji jednocześnie.

Śledzenie, kontrola dostępu i weryfikacja wymagań

- sprawdzanie kto? Kiedy? Gdzie? Jak? Czy? Napisał dane wymaganie

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

Przykładowe techniki pozyskiwania wymagań:

Śledzenie – polega na obserwowaniu użytkownika podczas wykonywania przez niego codziennych zadań w rzeczywistym środowisku

Wywiady – Spotkanie członka zespołu projektowego z użytkownikiem lub klientem

Focus groups – Sesja w której wymagania ustala się w większej grupie(np. burza mózgów)

Surveys, ankiety - Zbiór pytań utworzony w celu zebrania określonych informacji

Instruowanie przez użytkowników - W technice tej użytkownicy instruują przeprowadzającego badanie w sposobie w jaki wykonują określone zadania.

Prototypowanie

Specyfikacja wymagań

Metody specyfikacji wymagań:

- Język naturalny
- Formalizm matematyczny.
- Język naturalny strukturalny
- Tablice, formularze.
- Diagramy blokowe

- Diagramy kontekstowe

- Model przypadków użycia

Dokument Specyfikacji Wymagań Oprogramowania (SWO)

- Wymagania powinny być zebrane w dokumencie – specyfikacji wymagań oprogramowania.

- Dokument ten powinien być podstawą do szczegółowego kontraktu między klientem a producentem oprogramowania.

- Powinien także pozwalać na weryfikację stwierdzającą, czy wykonany system rzeczywiście spełnia postawione wymagania.

- Powinien to być dokument zrozumiały dla obydwu stron.

- Tekstowy dokument SWO jest najczęściej powiązany z innymi formami specyfikacji wymagań.

Model przypadków użycia opisuje:

- zachowanie systemu

- System

- jego środowisko

- Związki pomiędzy systemem a jego środowiskiem

Sporządzanie słownika pojęć w modelu przypadków użycia

- Słownik dotyczy dziedziny problemowej

- Tworzenie go polega na wyłowieniu wszystkich terminów z wymagań użytkownika.

- Terminy mogą odnosić się do aktorów, przypadków użycia, obiektów, operacji, zdarzeń, itp.

- Terminy w słowniku powinny być zdefiniowane w sposób precyzyjny i jednoznaczny.

- Posługiwanie się terminami ze słownika powinny być regułą przy opisie każdego kolejnego problemu, sytuacji czy modelu.

/*Podsumowanie*/

Zarządzanie wymaganiami - pozyskiwanie, analizowanie, dokumentowanie oraz weryfikacja wymagań

Wymagania: **funkcjonalne i niefunkcjonalne**

Pozyskiwanie wymagań jest procesem trudnym, wymagającym odpowiedniego przygotowania

UML – Unified Modeling Language

Konstrukcja UML:

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 generowaniu kodu)

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. 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 – elementy:

Strukturalne:

- klasa

- interfejs

- kooperacja

- przypadek użycia

- komponent

- węzeł

Czynnościowe:

- komunikat

- maszyna stanowa

Grupujące:

- pakiet

Komentujące:

- notatka

Podstawowe bloki konstrukcyjne – diagramy:

Diagram – graf, którego wierzchołkami są elementy a krawędziami związki

Modelowanie strukturalne:

- Diagram klas

- Diagram obiektów

- Diagram komponentów

- Diagram wdrożenia

Modelowanie behawioralne:

- Diagram kooperacji

- Diagram przypadków użycia

- Diagram czynności

- Diagram sekwencji

- Diagram stanów

Zależność – najprostszy i najsłabszy rodzaj relacji

Asocjacja – Silniejsza niż zależność, reprezentuje czasowe powiązanie między obiektami dwóch klas, ale czas życia obu nie jest od siebie zależny (oznaczamy ją czarnym trójkącikiem)

Agregacja – silniejsza forma asocjacji (biały romb)

Kompozycja – najsilniejszy związek łączący klasy (czarny romb)

Kompozycja > Agregacja > Asocjacja > Zależność

Klasa abstrakcyjna – deklaruje wspólną funkcjonalność grupy klas. Nie może posiadać obiektów i musi definiować podklasy

Interfejs – deklaruje grupę operacji bez podawania ich implementacji

Cztery zasady modelowania:

- Wybór modelu ma wpływ na to jak będziemy postrzegać rzeczywistość.
- Każdy model może być wyrażony na dowolnym poziomie szczegółowości
- Najlepsze model są bezpośrednio odnoszą się do rzeczywistości
- Pojedynczy model jest niewystarczający.

Modelowanie analityczne systemu

Modelowanie systemu pomaga analitykowi w zrozumieniu funkcjonalności systemu oraz w komunikacji z klientem.

Modele mogą prezentować system z różnych punktów widzenia :

- **Zewnętrznego** – pokazuje kontekst lub środowisko systemu;
- **Zachowania** – modeluje zachowanie systemu;
- **Strukturalnego** – modeluje architekturę systemu lub strukturę przetwarzania danych

Typy modeli:

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

Modele kontekstowe:

- Ilustrują operacyjny kontekst systemu-otoczenie.
- Kwestie społeczne i organizacyjne mogą mieć wpływ na ustalenia co do systemu należy a co jest poza jego granicami.
- Modele architektoniczne pokazują system i jego powiązania z innymi systemami.

Modele procesu:

- Pokazują czynności wspierane przez system.
- Uzupełniają model kontekstowy i pomagają w podjęciu decyzji, które czynności będą wykonywane automatycznie.

Modele zachowania:

- Opisują ogólne zachowanie systemu.
- Typy:

- Modele przetwarzania (przepływu) danych – pokazują sposób w jaki dane są przetwarzane i jak przepływają przez system;

- Modele stanów (maszyna stanów) pokazujące reakcje systemu na zdarzenia.

- Pozwalają spojrzeć na zachowanie systemu z różnych punktów widzenia.

Maszyny stanów

- Opisują zachowanie systemu w reakcji na wewnętrzne lub zewnętrzne zdarzenia.

- Pokazują odpowiedzi systemu na określone stymulacje, dlatego często wykorzystywane są do modelowania systemów czasu rzeczywistego.

- Maszyna stanu pokazuje system w postaci zbioru stanów oraz możliwych przejść pomiędzy nimi wraz ze zdarzeniami, które to przejście powodują.

- Do graficznego przedstawienia maszyny stanów wykorzystuje się diagramy stanów.

Diagramy stanów

- Umożliwiają poziomowanie modelu (dekompozycję na pod-modele).

- W każdym stanie można opisać akcję która jest wykonywana.

- Mogą być wspomagane tabelami szczegółowo opisującymi stany i pobudzenia.

Modele danych

- Opisują logiczną strukturę danych, które przetwarza system (np. model Encja-Związek i jego odmiany).

- Encja – koncepcyjnie niezależna jednostka (obiekt)

- Związek/relacja – koncepcyjny związek pomiędzy encjami.

- Atrybut – własność encji

- Szeroko stosowane w projektowaniu baz danych (mogą być łatwo przekształcone w model relacyjny).

Słownik Danych

- Alfabetyczna lista nazw używanych w różnych modelach systemu, również elementów modelu danych (encji, związków, atrybutów)

- Zalety:

- Wspiera proces zarządzania nazwami i umożliwia unikanie duplikatów;

- Przechowuje wiedzę nabywaną podczas projektu; wiąże ze sobą różne poziomy modelowania oraz implementację;

- Proces tworzenia słownika jest często wspierany przez narzędzia CASE.

Obiektowość zmniejsza lukę pomiędzy myśleniem o rzeczywistości (dziedzinie problemowej) a myśleniem o danych i procesach, które zachodzą na danych.

Zasada abstrakcji

- Eliminacja lub ukrycie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji.

- Wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzanie pojęć lub symboli oznaczających takie cechy.

- Abstrakcja definiuje granice zależne od perspektywy obserwatora