Model wodospadowy (kaskadowy, liniowy):

1. specyfikacja wymagań,
2. projektowanie
3. implementacja
4. testowanie
5. użytkowanie i pielęgnowanie

Zalety: łatwość zarządzania, narzucenie kolejności wykonywania prac
Wady: narzucenie kolejnosci wyk. prac, wysoki koszt błędów popełnianych we wczesnych fazach

Model ewolucyjny (odkrywczy):

1. budowa systemu
    2. specyfikacja
    3. użytkowanie systemu
    4. (system własciwy ? koniec : goto 2)

Zalety: możliwość stosowania nawet w przypadku kłopotów z określeniem wymagań klienta
Wady: zagmatwana struktura systemu, konieczność szybkiej produkcji, brak weryfikacji wymogów, brak możliwości pielęgnowania

Prototypowanie:

1. określenie wymagań,
2. opracowanie szybko prototypu,
3. weryfikacja prototypu przez klienta,
4. określenie szczegółowych wymagań,
5. opracowanie pełnego systemu

Cel prototypu: wykrycie braków w specyfikacji, nieporozumienia między klientem a developerem
Dev: model ewolucyjny, istniejące komponenty, niepełna realizacja, język hi-level, generatory int.

Formalne transformacje:
Wymagania wobec sys. są zapisywane w języku formalnym. Podlegają one automatycznym przekształceniom do programu.

Zalety: wysoka niezawodność,
Wady: mała efektywność kodu, trudności formalnego specyfikowania

Realizacja przyrostowa:
Realizacja sys. w skończonej liczbie kroków.

1. określenie wymagań
2. projekt ogólny
    3. wybór funkcji
    4. projekt, implementacja, testy
    5. dostarczenie częsci sys. goto 3

Zalety: częsty kontakt z klientem, wczesne wykorzystanie częsci sys.
Wady: dodatkowy koszt związany z realizacją fragmentów sys. Frameworki, biblioteki.

Obiekty:
Mają: stan, zbiór operacji do tego stanu i jego modyfikacji, obiekty się komunikują.

Obiekt: aktor (steruje), serwer (jest sterowany), agent(obie te role)

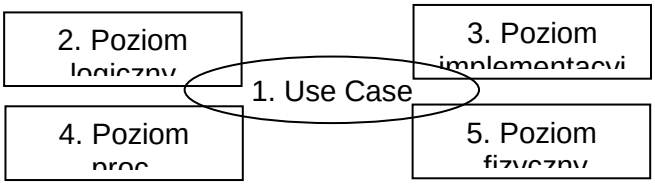
Analiza zorientowana obiektowo:

1. identyfikacja obiektów
2. organizowanie obiektów
3. opis interakcji
4. definicja operacji obiektu
5. definicja wnętrza obiektu

Wymagania:

1. funkcjonalne: funkcjonalnośc
2. niefunkcjonalne: czas odpowiedzi, zasoby

Projektowanie z wykorzystaniem notacji UML:



Model use case (CO sys. robi, a nie JAK):
Sys. z punktu widzenia użytkownika. Modeluje zachowanie sys. w odpowiedzi na polecenia użytkownika.

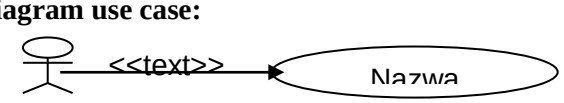
Model logiczny:
Przedstawia sys. w postaci klas, powiązań i interakcji między nimi,
Diagramy: klas, sekwencji, współpracy, przejść stanów

Model implementacyjny:
Przedstawia sys. jako moduły, podsys., zadania.
Diagramy: komponentów

Model procesów:
Sys. modelowany z punktu widzenia sys. operacyjnego, w jakim będzie działał. Zestaw procesów, wątków, zadań.
Diagramy: procesów

Model deployment model:
Modeluje fizyczne rozmieszczenie modułów sys. na komputerach, wymagania sprzętowe, obszary krytyczne.
Diagramy: deployment (montażowy)

Diagram use case:

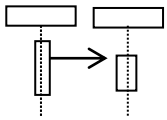


Relacje: <<include>>, <<uses>>, <<interacts>>, <<extends>>

Diagram klas:
atrybuty: private(-), public(+), protected(#), implementacyjny(>), wprowadzony(/), kluczowy(*)

Agregacja (* do N): ——————◇
(coś zawiera coś innego, jest zbudowane)
Kompozycja (0 lub 1): ——————◆
(zbudowana jest z – fizycznie)
Generalizacja – dziedziczenie klas: ——————▷
może być: {exclusive}, {disjoint}, {overlapping}, {complete}, {incomplete}
Asocjacja: zwykła linia, relacja między klasami, może być: {ordered}, {subset}, {exclusive}
Asocjacje trenarne: połączone diamentem pustym
Asocjacje kierunkowe (kanały nawigacyjne): zwykłe strzałki
Atrybuty: linia przerywana, klasa wiążąca

Diagram sekwencji (interakcji):
Modeluje dynamiczne cechy systemu, pomoc w tworzeniu diagramu stanów. Każdy dot. jednej ścieżki wywołania systemu. Przedstawia sekwencję odwołań obiektów w czasie.

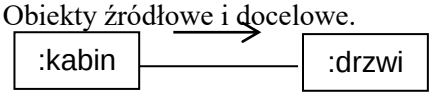


Komunikat A do B: zwykła strzałka do B
Komunikat asynch.: strzałka bez jednej kreski
Komunikat wywołania procedutry: pełna czarna strz.

Diagram stanów:
Opisuje zachowanie obiektów jednej klasy. entry/operacja, do/akcja, exit/operacja, zdarzenie/operacja
Istnieją: agregacje (and, przedzielone kreską przerywaną) i generalizacje (or, oddzielnie pogrupowane)

Diagram aktywności (czynności):
Odmiana diagramów stanu z uproszczeniem, reprezentują zachowanie metody lub przypadek użycia. Mogą być warunkowe, pokazywać równoległe czynności. Stan obiektu w []. Przepływ informacji, przerywana strzałka.

Diagram współpracy:
Przedstawia komunikację między obiektami. Nie w czasie. Obiekty źródłowe i docelowe.



{local}, {new}, {deleted}, {transient}
Obiekty aktywne (te co mogą czymś sterować) grubą obwódką. Synchronizacja, sekwencja, równoległość komunikatów.

Model implementacyjny:
Prostokąty (obiekty) w pakietach, strzałki co w use-case. <<thread>> , <<podsystem>>, <<implementation>>, eg: main.c -> (inc) program.h

Diagram montażowy:
Sprzęt wchodzący w skład sys. i rozmieszczenie oprog. na sprzęcie. Prostokąty eg: modem <<device>> - PC <<processor>>

Adapter: gdy konieczność dopasowania interfejsu klasy do potrzeb innej klasy.
Ambasador: gdy nie jest konieczne stałe utrzymanie zainicjowanego obiektu w sys. z powodu np. mem.
Obserwator: jeden-do-wielu w bazach danych.

Diagram przepływu danych (DFD):
Procesy – elipsy, przepływy – strzałki, magazyny – prostokąty, terminatory/interakcje – kółka.
Odmiany: diagram SADT, ERD (związków encji), diag. Jacksona, STD (sieci przejść)

COCOMO (constructive cost model):
Atrybuty: produktu, komputera, personelu, projektu

Testowania:
TD (top-down): od komponentu najbardziej abstrakcyjnego w głąb,
BU (bottom-up): od komponentów fundamentalnych w górę,
Wątków: – w sys. czasu rzeczywistego, sterowanych zdarzeniami,
Stresowe: obciążenie systemu,
Porównawcze (back-to-back): gdy dostępna więcej niż jedna wersja systemu, sprawdzanie po modyfikacji,
Defect test: test na specyfikację
Funkcjonalne: testy na podstawie specyfikacji, sys traktowany jak czarna skrzynka.
Strukturalne: analiza kodu,
Ścieżek: jedna z metod strukturalnego,
Interfejsu: typy interfejsu: parametryczne, SHM, proceduralne, komunikaty
Statystyczne: służy do mierzenia miar niezawodności.

Inżynieria oprogramowania to dziedzina inżynierii systemów zajmująca się wszelkimi aspektami produkcji oprogramowania: od analizy i określenia wymagań, przez projektowanie i wdrożenie, aż do ewolucji gotowego oprogramowania. Podczas gdy informatyka zajmuje się teoretycznymi aspektami produkcji oprogramowania, inżynieria

oprogramowania koncentruje się na stronie praktycznej.

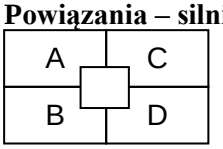
UML to język:

1. wizualizacji
2. specyfikacji
3. konstrukcji
4. dokumentacji

Poziomy spójności:

1. przypadkowa,
2. logiczna
3. czasowa
4. proceduralna
5. komunikacyjna
6. sekwencyjna
7. funkcjonalna

Powiązania – silnie powiązane:



Pielęgnowanie (łatwe):

- sys spójny
- sys ma mało powiązań
- sys jest łatwo adaptowalny