Model wodospadowy - został zaczerpnięty z innych źródeł inżynierskich, polega na produkcji oprogramowania w następujących po sobie fazach:

1) specyfikacja wymagań - co oprogramowanie ma robić i w jakich ograniczeniach ma pracować
2) wykonanie projektu, który miałby zastosować te funkcje
3) implementacja - zastosowanie projektu szczegółowe wypełnienie diagramu klas
4) faza testowania
5) faza wdrożenia
6) faza pielęgnowania

Model wodospadowy = Model kaskadowy

Faza strategiczna - nakłada się z fazą określenia wymagań, zaczyna się trochę wcześniej, jest to określenie zgrubnych wymagań i zastanowienie się co projekt ma realizować ale również decyzji, czy warto się zabierać za ten projekt i czy uda nam się na tym zarobić

Faza analizy - nakłada się na początkowe projektowanie i określenie uszczegółowionych wymagań, bardzo szczegółowe określenie, co system ma robić. W fazie analizy robiona jest architektura systemu, określenie z czego projekt będzie się składał, jakie będą zależności między podsystemami, projekt interfejsu, zrobienie diagramu klas takiego ogólnego (są klasy i powiązania między nimi, ale wnętrze tych klas nie jest szczegółowo wypełniane

Dokumentacja - zaczyna się w fazie określenia wymagań i jest tworzona przez cały proces produkcji oprogramowania. W fazie określania wymagań tworzona jest zgrubna dokumentacja systemu, określająca co system ma robić. Szkic dokumentacji projektowej

Określenie wymagań | Projektowanie | Implementacja | Testowanie | Praca Pielęgnowania

Faza strategiczna | Analiza | Instalacja

Dokumentacja

Estymacja kosztów

Systemy	Wymagania/ projekt	Implementacja	Testowanie
sterowania	46 %	20 %	34 %
operacyjne	33 %	17 %	50 %
bankowe	44 %	26 %	30 %
biznesowe	44 %	28 %	28 %

Wiele czasu zajmuje określenie wymagań i przygotowanie całego projektu. Dla systemów operacyjnych udział dwóch pierwszych faz jest najmniejszy, ponieważ dokładnie znane co system powinien robić. Budowa systemów operacyjnych jest dobrze znana dlatego zrobienie projektu systemu operacyjnego przychodzi bardzo szybko. Widać wysoki udział w pierwszej fazie układu sterowania, ponieważ bardzo trudno określić wymagania takiego systemu, często nie ma wzorców, które możemy powielać. Implementacja zajmuje mało procent ponieważ jeżeli mamy dobrze zrobiony projekt jego zaimplementowanie jest bardzo łatwe. Faza testowania w przypadku systemów operacyjnych zajmuje dużo procent, ponieważ testowanie takich systemów jest bardzo kosztowne i czasochłonne.

Tendencja - rosną koszty testowania systemów, sięgają nawet 60%, spadają koszty implementacji ponieważ są gotowe narzędzia, spadają koszty wymagań.

Zaletą modelu wodospadowego jest to, że łatwo jest takim projektem kierować, ponieważ łatwo można monitorować co się dzieje w danej fazie, łatwo sprawdzić czy faza jest zakończona, łatwo zrobić harmonogram. Wady modelu wodospadowego to wysoki koszt błędu, który jest wykryty w pierwszych fazach oraz mało kontaktu z klientem.

MODEL WODOSPADOWY

Faza modelu wodospadowego	Rezultat
Określenie wymagań: Analiza wymagań – określenie co system powinien robić, ogólne funkcje, ograniczenia na poziomie użytkownika	Studium wykonalności – sprawdzenie czy jesteśmy w stanie zrobić to oprogramowanie i czy nam się to opłaca
Określenie wymagań: Definicja wymagań – uszczegółowienie wymagań, tak żeby były zrozumiałe dla informatyka, który będzie tworzył projekt	~~Dokumentacja opisująca wymagania~~
Określenie wymagań: Specyfikacja systemu + określenie architektury systemu	Funkcjonalna specyfikacja, plan testów akceptowalnych, szkic podręcznika użytkownika
Projektowanie: Projektowanie architektury – zbiór funkcji, które system ma realizować jest przydzielany poszczególnym funkcjom	Specyfikacja architektury, testy systemowe – testują cały system
Projektowanie: Projektowanie ~~jednostek~~ interfejsów – określany jak system będzie się komunikował	Specyfikacja interfejsów, testy integracyjne – sprawdzają czy podsystemy współpracują

Projektowanie	Projektowanie jednostek - projektowanie podsystemu, równolegle projektowane jest kilka jednostek	Projekt szczegółowy, testy jednostkowe: - testy funkcjonalne, sprawdzające czy jednostka robi to co powinna (black box) - testy strukturalne, wyprowadzone na podstawie tego jak jednostka została zaprojektowana (white box)
Implementacja	Kodowanie - przekształcenie diagramów klas, diagramów sekwencji... w kod programu	Kod programu
Testowanie	Testowanie jednostek - testowanie pojedynczych klas	Raport testowania jednostkowego
	Testowanie modułu - testowanie zbioru klas - komponentów	Raport testowania modułowego
	Testowanie integracyjne - testowanie modułów potocznych komponentów	Raport testowania integracyjnego, podręcznik użytkownika
	Testowanie systemowe - test całego systemu	Raport testowania systemowego
	Testowanie akceptacyjne - sprawdzające czy oprogramowanie robi to co było uzgodnione z klientem	Dokumentacja systemu

W każdej fazie modelu wodospadowego otrzymujemy jakiś rezultat, jednak model ten ze względu na dużą pracę w kontaktach z klientem jest stosowany do małych systemów

Weryfikacja systemu jest przeprowadzana wtedy, gdy uzyskujemy odpowiedź na pytanie czy system spełnia wymagania, po przeprowadzeniu testów, czy produkt jest taki jak opisano w dokumencie, który opisuje wymagania. Weryfikacje przeprowadza firma produkująca oprogramowanie.

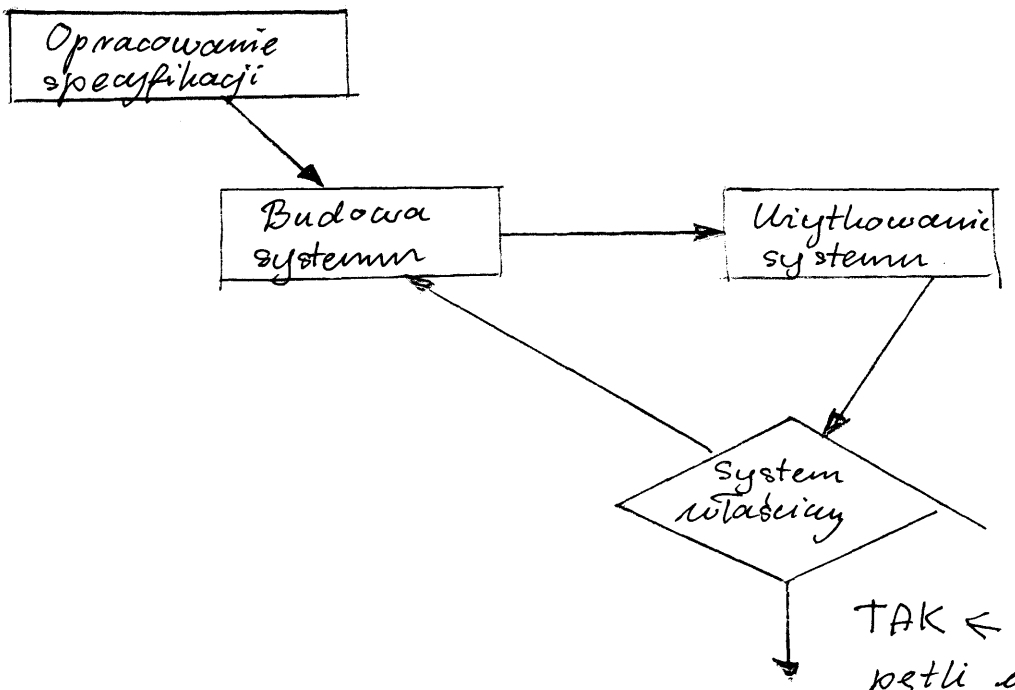
Walidacja systemu określa czy dane oprogramowanie jest dobre dla klienta, czy funkcje są przydatne klientowi, przeprowadza ją klient
Specyfikacja powinna zawierać dokładne wymagania klienta.

Model wodospadowy może być użyty, gdy jesteśmy w stanie dokładnie określić specyfikację.

Model ewolucyjny (odkrywczy) - jest stosowany wtedy, kiedy mamy, że określenie dokładnej specyfikacji wymagań nie jest możliwe, bo jest to np. nowa dziedzina zastosowań, bo klient jeszcze sam dobrze nie wie co by chciał z tym systemem zrobić (systemy sztucznej inteligencji).

W modelu ewolucyjnym robimy zgrubną specyfikację i bardzo szybko budujemy prototypowy system, który przekazujemy do wypróbowania klientowi. Klient w trakcie pracy z systemem odkrywa jakie możliwości powinien mieć ten system. Szybko dokonujemy nowe wymagania klienta, znów dajemy mu do wypróbowania system. Stosuje się środowiska, które wspomagają szybką budowę aplikacji, stosuje się języki takie jak Lisp i Prolog, czasem korzysta się z gotowych elementów

Model ewolucyjny



Czasami może się zdarzyć, że klient będzie chciał żebyśmy coś jeszcze dobudowali do systemu, a my nie będziemy w stanie tego zrobić, ponieważ jeśli budowa systemu następuje w sposób bardzo szybki to nie robi się dokumentacji, a struktura takiego systemu robi się zagmatwana (kod spaghetti - bundel w kodzie)
Przy modelu ewolucyjnym nie można mówić o weryfikacji, bo nie ma dokumentacji.
Model ewolucyjny jest stosowany do produkcji oprogramowania prototypowego, które działa krótko, dopóki klient nie odkryje co system powinien robić. Jak już będzie wiedział to robimy oprogramowanie od początku, korzystając np. z metody wodospadowej, tworząc dokumentację.

Prototypowanie - określamy zgrubne funkcje tego systemu, szybko przygotowujemy prototyp, korzystając z szybkich języków programowania, używając gotowych komponentów. Na prototypie pracuje klient, który przeprowadza walidację oprogramowania. Na tej podstawie przygotowywane są szczegółowe wymagania i często system jest pisany od początku. Prototypowanie stosuje się po to, aby klient pracując na prototypie odkrył pełne funkcje, które mu są potrzebne, a o których wcześniej nie wiedział. Prototypy często produkowane są przez firmy produkujące oprogramowanie na swoje własne potrzeby, po to aby pracując na prototypie uczynić wkład w specyfikacji, pamiętając wkład w specyfikacji są bardzo kosztowne. Prototypy budowane są po to aby pokazać klientowi funkcje systemu, które trudne są do opisania a łatwe do pokazania (interfejs użytkownika). Prototypy stosuje się również po to aby na nim szkolić użytkowników podczas produkcji właściwego oprogramowania.

Prototypy budowane są w modelu ewolucyjnym, często korzystając z gotowych komponentów, korzystając z generatorów interfejsów, stosuje się nie pełną realizację (mówi się że będzie jakaś funkcja, której jeszcze nie ma!

Formalne transformacje - określamy dokładne określenia wymagań, ale nie w języku naturalnym, lecz w języku matematycznym, formalnym. Niektóre specyfikacje formalne można automatycznie przekształcić do programu źródłowego np w języku C++.
Specyfikacje formalne możemy podzielić na
- specyfikacje algebraiczne i bazujące na modelu.

Przy metodzie formalnych transformacji mamy bardzo wysoką niezawodność systemu, ponieważ przy transformacjach nie są popełniane błędy. Jeżeli wymagania w stosunku do systemu mamy zapisane w pewnym aparacie formalnym to są tam narzędzia, które udowadniają pewne twierdzenia, które stawiamy i wykazują nam te błędy. Poprawione specyfikacje są wprowadzane do narzędzi, które generują kod i nie ma w nim błędów.
Metoda formalnych specyfikacji wymaga bardzo dużych umiejętności zespołu, jest bardzo trudne nauczenie się metod formalnych specyfikacji.

Formalne transformacje stosuje się w przypadku systemów, które muszą mieć bardzo duże parametry niezawodności, czyli np. w systemach sterowania elektrowni jądrowej, gdzie błąd systemu może powodować zagrożenie życia ludzkiego.

Nie wszystkie rzeczy dają się formalnie wyspecyfikować (np. interfejsy użytkownika)

Wadą formalnych specyfikacji jest to że kod automatycznie wygenerowany (chociaż bez błędów) jest wolniejszy od napisanego ręcznie. Porównuje się te kody i wtedy programiści coś poprawiają.

Z - schematy - fragmenty specyfikacji formalnej, wprowadzają specyfikację obiektu, określają jego zmienne i ustanawiają relacje między nimi

pojemnik	nazwa
zawartość : N pojemność : N	sygnatura - określamy zmienne opisujące stan pojemnika
zawartość <= pojemność	predykat schematu - określa coś co zawsze jest prawdziwe dla tego schematu.

wskaźnik
---

lampa {on, off}
odczyt : N
niebezpieczny poziom : N
---
lampa = on <=> odczyt <= niebezpieczny poziom

Teraz te dwa Z - schematy składamy w Z - schemat zbiornik

zbiornik
---

pojemnik
wskaźnik
---
odczyt = zawartość
pojemność = 4000
niebezpieczny poziom = 40

napełnianie - OK (Δ schemat - delta schemat)

Δ zbiornik
ilość? : N
---
zawartość + ilość? <= pojemność
zawartość' = zawartość + ilość

zmienna stanu tego schematu napełnia ulegną zmianie

< pokazujemy jak zawartość się zmienia.

W Z-schematach można korzystać z operacji wejściowych i wyjściowych. Znak zapytania ? jest to operacja wejściowa, która mówi o tym, że ilość będzie wprowadzona z wejścia.

Zawartość' - zawartość prim - oznacza nową wartość tej zawartości (do starej zawartości dodana będzie ilość).

przepełnienie

Ξ zbiornik
ilość ? : N
t ! : seq char

Ξ - (Xi schemat) - wartości zmiennych stanu nie zostaną zmienione
przez operacje

pojemność < zawartość + ilość ?
t ! = „niemieszczająca pojemność zbiornika - napełnianie zlasowane"

! operacja wyjściowa - generuje komunikat wyjściowy

napełnianie

napełnianie _ OK ∪ przepełnienie