



ZARZĄDZANIE CYKLEM ŻYCIA APLIKACJI
INŻYNIERIA OPROGRAMOWANIA I SDLC

PLAN ZAJĘĆ

- ▶ Wprowadzenie do inżynierii oprogramowania
- ▶ Plan kursu - Zagadnienia z zakresu IO i SDLC
- ▶ Wprowadzenie do Cyklu Produkcji Oprogramowania (SDLC)
- ▶ Modele SDLC
- ▶ Omówienie planu ćwiczenia na zajęciach
- ▶ Podział na grupy



WPROWADZENIE

INŻYNIERIA
OPROGRAMOWANIA



INŻYNIERIA OPROGRAMOWANIA

ZASTOSOWANIE SYSTEMATYCZNEGO,
ZDYSYPLINOWANEGO, ILOŚCIOWEGO
PODEJŚCIA DO ROZWOJU, EKSPOLOATACJI I
UTRZYMANIA OPROGRAMOWANIA.

IEEE Standard Glossary of Software Engineering Terminology

Zgodnie ze słownikiem IO opracowanym przez Institute of Electrical and Electronics Engineers, IO to zastosowanie:

- 1) systematycznego
- 2) zdyscyplinowanego
- 3) ilościowego podejścia do
 - 1) rozwoju
 - 2) eksploatacji
 - 3) utrzymania oprogramowania.

Oznacza to, że do wytwarzania oprogramowania używamy jasno zdefiniowanych i reguł wynikających z praktyk naukowych, procedur i metod, w wyniku czego otrzymujemy wydajne i niezawodne oprogramowanie, produkt (działający na realnych maszynach).

IEEE (Institute of Electrical and Electronics Engineers) Computer Society wraz z **ACM (Association for Computing Machinery)** to 2 kluczowe organizacje powstałe po zakończeniu 2 Wojny Światowej, które zajmują się określaniem standardów nauczania IO zwanym Computing Curricula. IO jest 1 z 14 obszarów informatyki wyodrębnionych w tym dokumencie.

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ Ewolucja
- ▶ Walidacja
- ▶ Narzędzia
- ▶ API

Inżynieria Oprogramowania definiuje 8 głównych obszarów wiedzy, z którymi musi zapoznać się każdy programista.

INŻYNIERIA OPROGRAMOWANIA

▶ Wymagania

▶ Projektowanie

▶ Procesy

▶ Zarządzanie

▶ Ewolucja

▶ Walidacja

▶ Narzędzia

▶ API

Dwa pierwsze dotyczą czynności poprzedzających samo pisanie kodu. Jest to:

- specyfikacja wymagań, czyli ustalenie co budowany system ma robić, jakie spełniać wymogi bezpieczeństwa, wydajności, wsparcia dla systemów i urządzeń
- projektowanie oprogramowania, czyli – w dużym uproszczeniu – zaproponowanie jego struktury.

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ Ewolucja
- ▶ Walidacja
- ▶ Narzędzia
- ▶ API

2 następne obszary to

- procesy wytwarzania oprogramowania - rozpatruje się tutaj różne modele cyklu życia oprogramowania, co wpływa bezpośrednio na planowanie przedsięwzięć programistycznych - i
- zarządzanie przedsięwzięciami programistycznymi.

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ **Ewolucja**
- ▶ **Walidacja**
- ▶ Narzędzia
- ▶ API

Kolejne 2 obszary to

- ewolucja, czyli rozwój i utrzymania użyteczności oprogramowania i umiejętności wprowadzania do niego koniecznych zmian oraz
- walidacja i weryfikacja oprogramowania czyli, inaczej mówiąc kontrola jakości) i jego ewolucji, czyli utrzymania użyteczności programu i umiejętności wprowadzania do niego koniecznych zmian.

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ Ewolucja
- ▶ Walidacja
- ▶ **Narzędzia**
- ▶ **API**

Ostatnie 2 obszary wiedzy dotyczą narzędzi i środowisk programistycznych oraz interfejsów programistycznych – w skrócie API (Application Programming Interface).

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ Ewolucja
- ▶ Walidacja
- ▶ Narzędzia
- ▶ API

W ramach tego kursu zajmiemy się omówieniem 6 spośród tych obszarów. Skupimy się przy tym na wybranych zagadnieniach, kluczowych dla cyklu wytwarzania oprogramowania a jednocześnie dająccych mam nadzieję wystarczający pogląd na wielość możliwych podejść do tego zagadnienia.

INŻYNIERIA OPROGRAMOWANIA

- ▶ Wymagania
- ▶ Projektowanie
- ▶ Procesy
- ▶ Zarządzanie
- ▶ Ewolucja
- ▶ Walidacja
- ▶ Inżynieria wymagań
- ▶ Projektowanie oprogramowania
- ▶ Zarządzanie procesem produkcyjnym
- ▶ Klasyczne Metodyki wytwarzania oprogramowania
- ▶ Zwinne metodyki wytwarzania oprogramowania
- ▶ Testowanie i debuggowanie

I tak, te 6 obszarów inżynierii oprogramowania będzie bezpośrednio powiązane z tematyką naszych kolejnych zajęć.

INŻYNIERIA WYMAGAŃ

- ▶ Czym jest Inżynieria Wymagań?
- ▶ Wymagania funkcjonalne, pozafunkcjonalne
- ▶ Model FURPS
- ▶ 4 podejścia do opisu wymagań
- ▶ Przypadki użycia, User Stories
- ▶ Omówimy szablon dokumentu wymagań produktu (PRD)

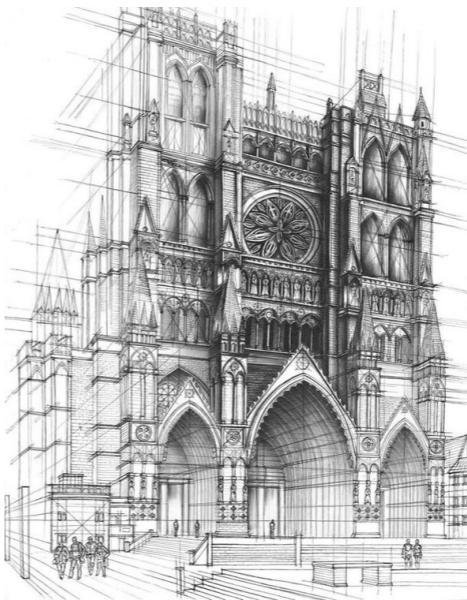


Na najbliższych zajęciach zajmiemy się tematem IW, i

- postaramy się odpowieć na pytanie „Czym jest inżynieria wymagań?”,
- zdefiniujemy czym są wymagania funkcjonalne i pozafunkcjonalne oprogramowania,
- omówimy model FURPS, który opisuje 5 powiązanych z nimi obszarów.
- Następnie opowiemy sobie o możliwych podejściach do sposobu opisywania wymagań oraz
- szczegółowo przyjrzymy się dwóm z nich: Przypadkom Użycia oraz User Stories (zwanych też historyjkami).
- W ramach ćwiczeń podejmujemy próbę napisania w grupach i omówienia kilku przykładów UC i US.
- Na koniec chciałbym omówić przykładowy dokument wymagań produktu (PRD) - jakie sekcje i jakie informacje powinien zawierać.

PROJEKTOWANIE OPROGRAMOWANIA

- ▶ Funkcja i znaczenie UML
- ▶ Podstawy notacji języka UML
- ▶ Modelowanie procesów biznesowych
- ▶ Diagramu klas
- ▶ Wzorce projektowe



Kolejnym obszarem, któremu chciałbym się przyjrzeć jest projektowanie oprogramowania - czy mieliście już państwo?

W ramach tego spotkania chciałbym w kilku slajdach przedstawić państwu:

- historię powstania języka modelowania UML, jego funkcję i znaczenia w procesie wytwarzania oprogramowania
- wprowadzić państwa w podstawy notacji języka UML
- zaprezentować na czym polega modelowanie procesów biznesowych
- omówić diagramy klas
- Następnie omówić wybrane wzorce projektowe oraz wyjaśnić okoliczności w jakich winny być stosowane
- W ramach ćwiczeń spróbujemy stworzyć diagram przypadków użycia lub diagram klas do projektów grupowych lub Państwa projektów indywidualnych, które będziecie pisać w sprincie 4.
- Na koniec chciałbym spróbować omówić wybrane diagramy, które uda się państwu stworzyć podczas zajęć

ZARZĄDZANIE PROCESEM PRODUKCYJNYM

- ▶ PMBOK i kluczowe obszary zarządzania projektem
- ▶ Procesy i produkty zarządcze
- ▶ Macierz RACI
- ▶ Rejestr Ryzyk



Ostatnie w tym tygodniu zajęcia, dotyczące Zarządzania Procesem Produkcyjnym, chciałbym

- rozpocząć od wprowadzenia pojęcia projektu
- a następnie omówić 10 kluczowych z perspektywy project managementu obszarów oraz powiązanych z nimi procesów i produktów zarządczych.
- chciałbym następnie abyśmy przyjrzaeli się 2 szczególnie ważnym z perspektywy każdego projektu procesom i powstałym w ich wyniku artefaktom: Macierzy odpowiedzialności RACI oraz Rejestrowi Ryzyk
- i w ramach ćwiczeń spróbowali przygotować oba z nich i w miarę możliwości omówić

KLASYCZNE METODYKI

- ▶ Wprowadzenie do metodyki Prince2
- ▶ Role i odpowiedzialności w Prince 2
- ▶ Fazy projektu i produkty zarządcze w Prince 2
- ▶ Metoda Ścieżki Krytycznej
- ▶ PERT - szacowanie pracochłonności
- ▶ Harmonogramowanie i kontrola przebiegu prac przy użyciu wykresu Gantta



Opowiem o klasycznych metodykach wytwarzania oprogramowania na przykładzie Prince2. W ramach zajęć chciałbym zapoznać Państwa

- z regułami i kluczowymi procesami tej metodyki
- opowiedzieć o strukturze zespołu projektowego oraz podziale ról i odpowiedzialnościach w jego ramach
- przybliżyć standardowe w Prince2 fazy projektu oraz charakterystyczne dla nich produkty zarządcze
- następnie chciałbym przybliżyć państwu metodę ścieżki krytycznej oraz metody szacowania pracochłonności przy użyciu metody PERT
- a na koniec przeprowadzić ćwiczenie w ramach którego postaramy się wycenić pracochłonność kilku zadań i w oparciu o wycenę przygotować harmonogram prac w MS Project

ZWINNE METODYKI

- ▶ SCRUM - definicja, filary wartości
- ▶ Zespół SCRUMowy, wydarzenia i artefakty
- ▶ Kanban, XP
- ▶ Agile PM
- ▶ Metodyki Hybrydowe: Prince2 Agile



Omawiając zwinne metodyki zarządzania procesem produkcyjnym chciałbym przede wszystkim, skupić się na SCRUMie,

- omówić jego filary i wartości, którym hołduje
- przedstawić charakterystykę zespołu scrumowego wraz z jego podziałem na role i przynależne odpowiedzialności
- omówić nieodzowne w SCRUMie wydarzenia i artefakty
- Następnie przybliżyć inne metodyki zwinne, jak: Kanban, Extreme Programming, Agile PM
- Chciałbym podjąć również tematykę metodyk hybrydowych na przykładzie Prince2 Agile
- A w ramach ćwiczenia chciałbym abyśmy na chwilę przeistoczyli się w zespół scrumowy i opierając się na przygotowanym wcześniej założeniach spróbowali przeprowadzić jedno z wydarzeń Scrumowych (planning lub retrospektywę).

TESTOWANIE I DEBUGGOWANIE

- ▶ Testowanie jako weryfikacja i walidacja
- ▶ Statyczna i dynamiczna weryfikacja
- ▶ Testy czarno- i biało skrzynkowe
- ▶ Podział testów ze względu na przedmiot lub cel testowania
- ▶ Scenariusze i przypadki testowe
- ▶ TDD - Test Driven Development



Na przedostatnich zajęciach chciałbym abyśmy podjęli tematykę testowania i debuggowania i różnic pomiędzy tymi procesami.

- omówimy czym różni się walidacja od weryfikacji oprogramowania
- wprowadzimy pojęcie statycznej i dynamicznej weryfikacji, testów czarnoskrzynkowych i biało skrzynkowych.
- oraz porozmawiamy o podziale testów ze względu na przedmiot testowania (jednostkowe, integracyjne, systemowe, akceptacyjne) lub cel (regresywne, sanity/smoke).
- chciałbym abyśmy również mieli okazję spojrzeć na dokumentację testową, na przykładzie scenariuszy testowych i przypadków testowych - są to dokumenty, których niejednokrotnie wymagają klienci, szczególnie ci korporacyjni, dla których istotne są standardy i dokumentacja działań związanych z gwarancją jakości
- na koniec chciałbym przybliżyć Państwu jeszcze jedną zwinną metodykę produkcji oprogramowania a mianowicie Test Driven Development (TDD), gdzie sam proces wytwarzania oprogramowania rozpoczyna się od pisania testów.
- Na ostatnich zajęciach podsumujemy sobie również zakres niniejszego kursu i porozmawiamy o egzaminie.



WPROWADZENIE

**SOFTWARE DEVELOPMENT
LIFE CYCLE (SDLC)**



SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

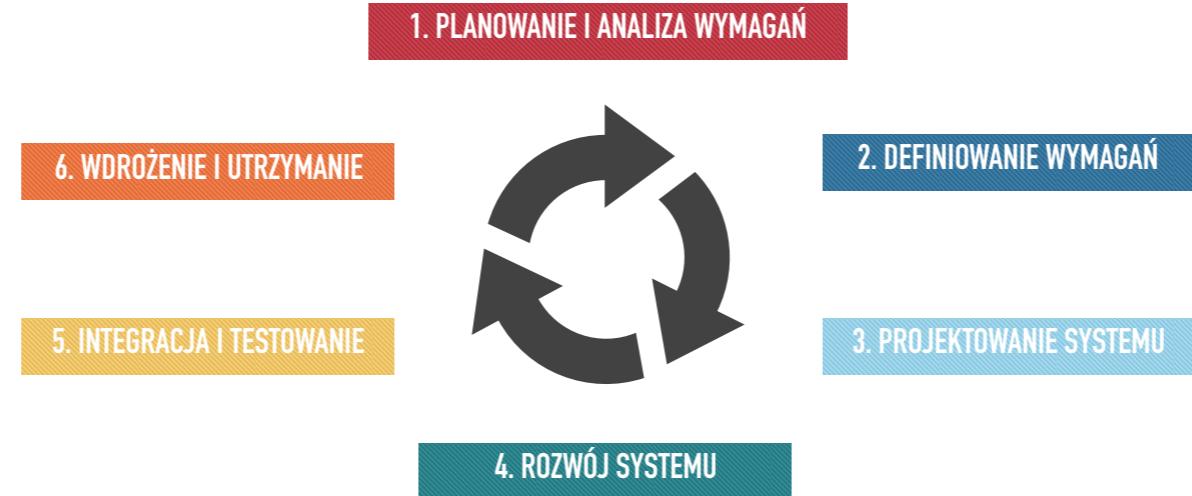
FRAMEWORK DEFINUJĄCY ZADANIA
WYKONYWANE NA KAŻDYM ETAPIE
PROCESU TWORZENIA OPROGRAMOWANIA.

ISO/IEC/IEEE 12207:2017 (Listopad 2017)

SDLC zwany także Software Development Process - framework definiujący zadania wykonywane na każdym etapie procesu tworzenia oprogramowania.

Obowiązującym międzynarodowym standardem, który definiuje wszystkie zadania wymagane do rozwoju i utrzymania oprogramowania jest ISO/IEC/IEEE 12207:2017 opublikowany listopadzie ubiegłego roku.

SOFTWARE DEVELOPMENT LIFE CYCLE



SDLC określa kilka kluczowych etapów procesu wytwarzania oprogramowania. Różne źródła operują różnymi nazwami, natomiast zasadniczo sprowadzają się one do następujących czynności:

1. PLANOWANIE I ANALIZA WYMAGAŃ

- ▶ Zbieranie i analiza wymagań
- ▶ Źródła wymagań:
 - ▶ wywiady z interesariuszami, użytkownikami
 - ▶ analiza istniejących rozwiązań
- ▶ Studium wykonalności
- ▶ Plan testów
- ▶ Identyfikacja ryzyk i planowanie reakcji

Zbieranie i analiza wymagań jest najważniejszym i podstawowym etapem w SDLC.

zespół odpowiedzialny za wytworzenie produktu (lub usługi), na różne sposoby stara się pogłębić wiedzę na temat wymagań klienta względem tego co dany produkt ma robić, jak ma je realizować, kto jest odbiorcą danej usługi lub użytkownikiem produktu.

Źródłem wymagań mogą być:

- wywiady przeprowadzone z:
 - interesariuszami po stronie klienta o celach, które mają być realizowane przez produktu
 - z potencjalnymi użytkownikami o oczekiwaniach względem produktu
 - ze specjalistami z danej dziedziny problemowej
- odwoływanie się do istniejących konkurencyjnych rozwiązań
- analizowanie istniejącego systemu i oprogramowania

Studiom wykonalności

Po zebraniu wymagań zespół opracowuje ogólny plan procesu tworzenia oprogramowania. Na tym etapie zespół analizuje czy w ramach organizacji można wykonać oprogramowanie w celu spełnienia wszystkich wymagań użytkownika. Produktem tej fazy cyklu powinno być potwierdzenie (lub nie), że produkt jest wykonalny pod względem finansowym, czasowym, technicznym.

Częścią tego etapu jest również planowanie zakresu testów (gwarancji jakości) oraz identyfikacja ryzyk związanych z projektem.

2. DEFINIOWANIE WYMAGAŃ

- ▶ Jasne zdefiniowanie wymagań
- ▶ Udokumentowanie wymagań
 - ▶ PRD
 - ▶ Product Backlog
- ▶ Akceptacja wymagań przez klienta, interesariuszy

Po przeprowadzeniu analizy wymagań kolejnym krokiem jest

- jasne zdefiniowanie wymagań produktu - tak, aby były zrozumiałe dla wszystkich stron -
- udokumentowanie wymagań produktu w
 - Product Requirements Document - który obejmuje wszystkie wymagania dotyczące produktu, które należy zaprojektować i rozwinąć w trakcie cyklu produkcyjnego.
 - Backlogu produktu - spriorytetyzowana lista wymagań (opisanych w formie user stories)
- oraz uzyskanie zatwierdzenia od klienta

Klient - nie koniecznie zewnętrzna postać, może być klient wewnętrzny, np. dyrektor działu marketingu w naszej firmie.

3. PROJEKTOWANIE

- ▶ Projekt architektury aplikacji
- ▶ Projekt infrastruktury serwerowej
- ▶ Projekt bazy danych
- ▶ Diagram przepływu danych, komunikacji z wewnętrznymi i zewnętrznymi modułami
- ▶ Analiza i zatwierdzenie projektu przez interesariuszy

Dokumentacja projektu jest punktem odniesienia dla architektów rozwiązania (produkту), którzy przygotowuje w oparciu o nią projekt architektury aplikacji, architektury serwerowej, baz danych.

Podejście projektowe w jasny sposób definiuje wszystkie moduły architektoniczne produktu wraz z jego komunikacją i reprezentacją przepływu danych z modułami wewnętrznymi i zewnętrznymi (jeśli występują).

Projekt taki następnie podlega zatwierdzeniu (lub zostaje odesłana do rewizji) przez kluczowych interesariuszy (udziałowców, osoby które będą czerpać korzyści i najczęściej sponsorują wytwarzanie produktu).

4. ROZWÓJ PRODUKTU

- ▶ Rozwój i budowa modułów produktu
 - ▶ Wybór języka programowania, bibliotek
 - ▶ Pisanie kodu
 - ▶ Kompilowanie
 - ▶ Unit testy
 - ▶ Kontrola wersji

Na tym etapie rozpoczyna się właściwy rozwój i budowa produktu w oparciu o przygotowany w poprzedniej fazie projekt.

O ile nie zostało to określone wcześniej w oparciu o wymagania klienta lub wnioski z analizy, dokonuje Język programowania wybierany jest w zależności od rodzaju tworzonego oprogramowania.

5. INTEGRACJA I TESTY

- ▶ Integracja modułów w system
- ▶ Testy integracyjne
- ▶ Testy funkcjonalne
- ▶ Testy akceptacyjne
- ▶ Testy regresyjne

Ten etap jest zwykle podzbiorem wszystkich etapów, jako że w nowoczesnych modelach SDLC, testowanie jest częścią wszystkich etapów wytwarzania oprogramowania.

W ramach etapu Integracja i testy realizowane są czynności mające na celu zagwarantowanie jakości produktu - podczas testów zgłoszane są wady produktu a także ma miejsce debuggowanie, czyli usuwanie błędów oraz re-testy (ponowne testy), dopóki produkt nie osiągnie standardów jakości określonych w dokumentacji wymagań.

6. WDROŻENIE I UTRZYMANIE

- ▶ Plan deployment
- ▶ Wdrożenie (deployment) na środowiska produkcyjne
- ▶ Publikacja produktu
- ▶ Wdrożenie etapami:
 - ▶ Publikacja produktu o ograniczonym zakresie
 - ▶ Testy użytkownika UAT
 - ▶ Decyzja o wdrożeniu lub dalszym rozwoju w oparciu o wyniki UAT
- ▶ Utrzymanie

Po przetestowaniu produktu i przygotowaniu do wdrożenia, zostaje on wdrożony / zdeployowany (zainstalowany) na środowisku produkcyjnym, a następnie udostępniony publicznie.

Wdrożenie może być szczególnym wyzwaniem wymagającym planu, kiedy klient dysponuje jednym środowiskiem, na którym aktualnie działa poprzednia wersja oprogramowania / systemu / produktu.

Czasami wdrożenie produktu odbywa się etapami - Produkt może być najpierw wydany w ograniczonym segmencie i przetestowany w rzeczywistym środowisku biznesowym (testy akceptacyjne UAT-u).

Następnie, w oparciu o opinie, produkt może zostać wydany bez zmian lub z sugerowanymi ulepszeniami w segmencie rynku kierowania.

Po wprowadzeniu produktu na rynek jego konserwacja odbywa się dla istniejącej bazy klientów.

MODELE WYTWARZANIA OPROGRAMOWANIA

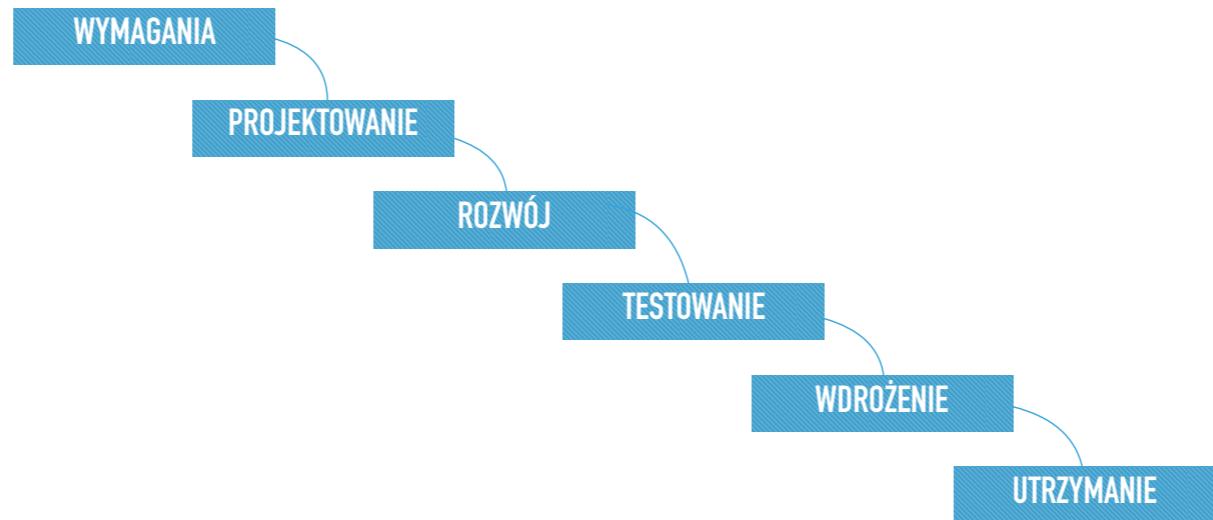
- ▶ Model Waterfall (Kaskadowy)
- ▶ Model Iteracyjny
- ▶ Model Spiralny
- ▶ Model V (Weryfikacja i Walidacji)

Opracowano różne modele cyklu życia oprogramowania, za którymi można podążać w procesie tworzenia oprogramowania. Każdy z nich charakteryzuje się serią unikalnych kroków, mających zagwarantować sukces projektu i skuteczne dostarczenie produktu.

Najpopularniejsze z nich to:

Każdy produkt digitalowy i projekt w ramach którego jest realizowany posiada specyficzne charakterystyki, dzieje się w specyficznej organizacji i jest realizowany dla specyficznego klienta, dlatego w każdej takiej sytuacji powinno się podjąć decyzji o tym jakie podejście do produkcji oprogramowania przyjmiemy. Nie ma dobrych i złych podejść, są tylko dobrze i źle dobrane do sytuacji i okoliczności.

MODEL WATERFALL (KASKADOWY)



Model Waterfall (Kaskadowy), zwany też modelem linearno-sekwencyjnym, był pierwszym modelem stosowanym do wytwarzania oprogramowania. Zakłada on, że każda z faz procesu musi zostać zakończona, zanim rozpoczęta zostanie kolejna. Nie dopuszczalne jest nakładanie się faz projektu.

- Wymagania - fazie zbierania wymagań, zostają zebrane wszystkie możliwe wymagania systemowe i spisane w formie dokumentu specyfikacji wymagań
- Projektowanie - wymagania zebrane w dokumentacji są analizowane, a następnie projektowany jest system. W oparciu o projekt systemu określana jest jego architektura oraz definiowane są dodatkowe wymagania, np dla hardware'u.
- Rozwój - w oparciu o architekturę systemu rozpoczyna się dewelopment małych części programu, tzw. unit'ów, które są integrowane w etapie następnym. Każdy z unitów jest wytwarzany a następnie testowany pod kątem jego funkcji (stąd pojęcie Unit Test).
- Integracja i testowanie - kiedy unit testy są poprawnie ukończone, unity są integrowane w system a następnie przeprowadzane są testy integracyjne, funkcjonalne, niefunkcjonalne, akceptacyjne dla całości systemu na środowiskach (serwerach) testowych.
- Wdrożenie - po zamknięciu testów, produkt zostaje wdrożony (zdeployowany) na środowiska produkcyjne klienta i opublikowane / wypuszczone na rynek.
- Utrzymanie - w sytuacji gdy na środowisku klienta ujawniają się błędy lub klient oczekuje usprawnień/ulepszeń systemu, przygotowywane są fixy / patch'e które są następnie wdrażane na środowisku klienta.

TEKST

KIEDY STOSOWAĆ WATERFALL?

- ▶ Wymagania są bardzo dobrze udokumentowane, jasne i niezmienne.
- ▶ Definicja produktu jest stabilna
- ▶ Technologia jest znana i nie ewoluuje
- ▶ Nie ma niejednoznacznych wymagań
- ▶ Projekt jest krótki

Waterfall można zastosować przede wszystkim w sytuacji realizacji produktu, który jest znanych, prostych i powtarzalnych - np. zbudowanie strony produktowej na istniejącym szablonie.

TEKST

ZALETY WATERFALL

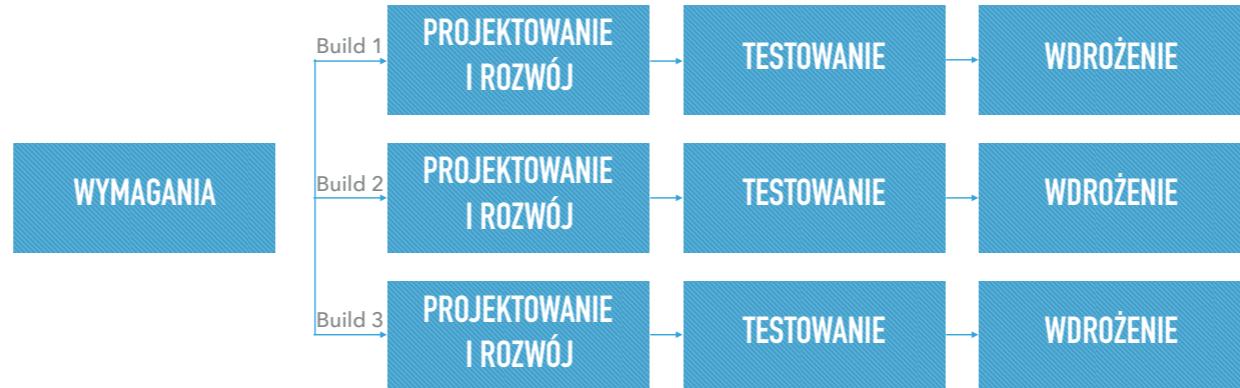
- ▶ Łatwy zrozumienia i używania
- ▶ Każda z faz ma jasno określone produkty, które ma dostarczyć oraz proces ich weryfikacji
- ▶ Tylko jedna faza na raz
- ▶ Jasno zdefiniowane etapy oraz „kamienie milowe”
- ▶ Łatwość w organizacji zadań
- ▶ Procesy i rezultaty są dobrze udokumentowane
- ▶ Sprawdza się na małych projektach z dobrze zdefiniowanymi wymaganiami

TEKST

WADY WATERFALL

- ▶ Działające oprogramowanie nie jest udostępnione, aż do późnych faz procesu
- ▶ Jest to powodem niepewności i zwiększonego ryzyko w projekcie
- ▶ Nierrekomendowany do złożonych i długoterminowych projektów
- ▶ Niewskazany w projektach o wysokim prawdopodobieństwie zmian w wymaganiach produktowych
- ▶ Nie uwzględnia zarządzanie zmianą
- ▶ Trudność w mierzeniu postępu prac w ramach danej fazy

MODEL ITERACYJNY



Podstawowym założeniem modelu iteracyjnego jest wytwarzania oprogramowania w toku powtarzalnych cykli i w małych porcjach (przyrostowo). Na koniec każdego cyklu (iteracji) powstaje nowa wersja oprogramowania

Model iteracyjny rozpoczyna się zaprojektowaniem i rozwinięciem małego zestawu wymagań i iteracyjnie rozszerza kolejne wersje, aż do momentu kiedy system jest kompletny, gotowy do wgrania i opublikowania na serwerach produkcyjnych.

W modelu iteracyjnym, podczas produkcji oprogramowania, możliwe jest występowanie wielu jednocześnie iteracji cyklu produkcyjnego w tym samym czasie.

Kluczem do sukcesu w tym modelu jest rygorystyczna walidacja wymagań i testowanie każdej kolejnej wersji oprogramowania w oparciu o te wymagania w każdym kolejnym cyklu.

KIEDY STOSOWAĆ MODEL ITERACYJNY?

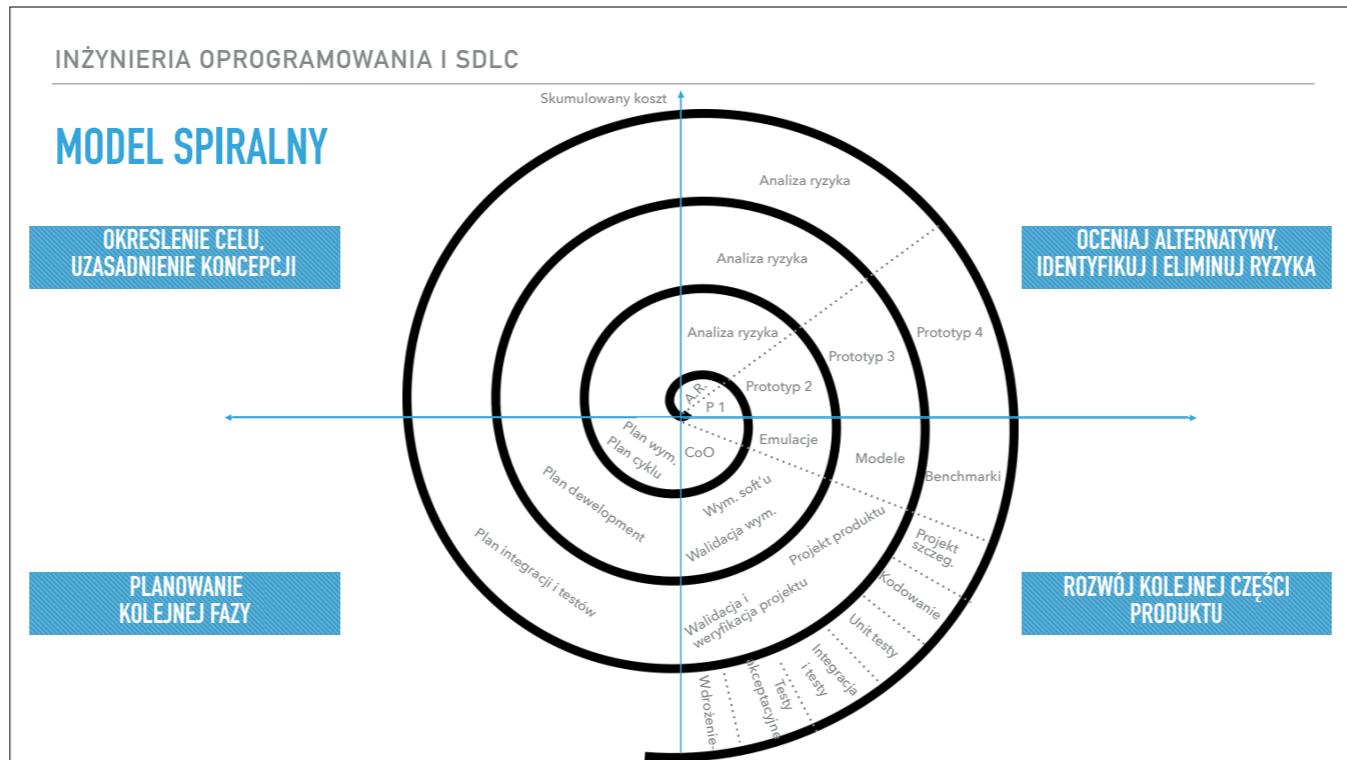
- ▶ Oczekiwania względem kompletnego systemu są jasne i zrozumiałe
- ▶ Główne wymagania są zdefiniowane, natomiast niektóre funkcje lub rozszerzenia mogą ewoluować z czasem
- ▶ Do budowy systemu zostanie użyta nowa technologia, której zespół projektowy musi się nauczyć w trakcie trwania projektu
- ▶ Członkowie zespołu o wymaganych kompetencjach nie są dostępni od reki, lecz ich udział jest zaplanowany na zasadach kontraktu w konkretnych iteracjach
- ▶ W zakresie projektu występują funkcje lub cele wysokiego ryzyka, które z czasem mogą ulec zmianie

ZALETY MODELU ITERACYJNEGO?

- ▶ Rezultaty są uzyskiwane wcześnie i cyklicznie
- ▶ Z każdym przyrostem dostarczany jest działający produkt
- ▶ Testowanie i debugowanie niewielkiego zakresu danej iteracji jest łatwiejsze
- ▶ Dostępność działającego modelu systemu na wczesnym etapie cyklu produkcji oprogramowania, co wspiera ewaluację przez klienta na wczesnym etapie i zbieranie feedbacku
- ▶ Możliwość zrównoleglenia prac deweloperskich
- ▶ Łatwiejszy pomiar postępu
- ▶ Zmiany w zakresie są mniej kosztowne łatwiejsze do zarządzenia
- ▶ Ryzyka są identyfikowane i rozwiązywane w ramach iteracji, elementy wysokiego ryzyka są rozwiązywane w pierwszej kolejności
- ▶ Rekomendowany do dużych projektów o wysokim priorytecie dla organizacji

WADY MODELU ITERACYJNEGO?

- ▶ Często wymaga większej liczby zasobów
- ▶ Wymaga większej uwagi kadry zarządzającej, zarządzanie jest bardziej złożone
- ▶ Problemy związane z architekturą lub projektem systemu mogą wystąpić się na późniejszym etapie projektu
- ▶ Nie odpowiedni do małych projektów
- ▶ Ocena ryzyka wymaga zaangażowania osób o wysokich kompetencjach
- ▶ Postęp projektu jest uzależniony od analizy ryzyka



Proces tworzenia ma postać spirali, której każda pętla reprezentuje jedną fazę procesu. Najbardziej wewnętrzna pętla przedstawia początkowe etapy projektowania, np. studium wykonalności, kolejna definicji wymagań systemowych, itd.

Każda pętla spirali podzielona jest na cztery sektory:

- Ustalanie celów – definiowanie konkretnych celów wymaganych w tej fazie przedsięwzięcia. Identyfikacja ograniczeń i zagrożeń. Ustalanie planów realizacji.
- Rozpoznanie i redukcja zagrożeń – przeprowadzenie szczegółowej analizy rozpoznanych zagrożeń, ich źródeł i sposobów zapobiegania. Podejmuje się odpowiednie kroki zapobiegawcze.
- Tworzenie i zatwierdzanie – tworzenie oprogramowania w oparciu o najbardziej odpowiedni model, wybrany na podstawie oceny zagrożeń.
- Ocena i planowanie – recenzja postępu prac i planowanie kolejnej fazy przedsięwzięcia bądź zakończenie procesu produkcyjnego.

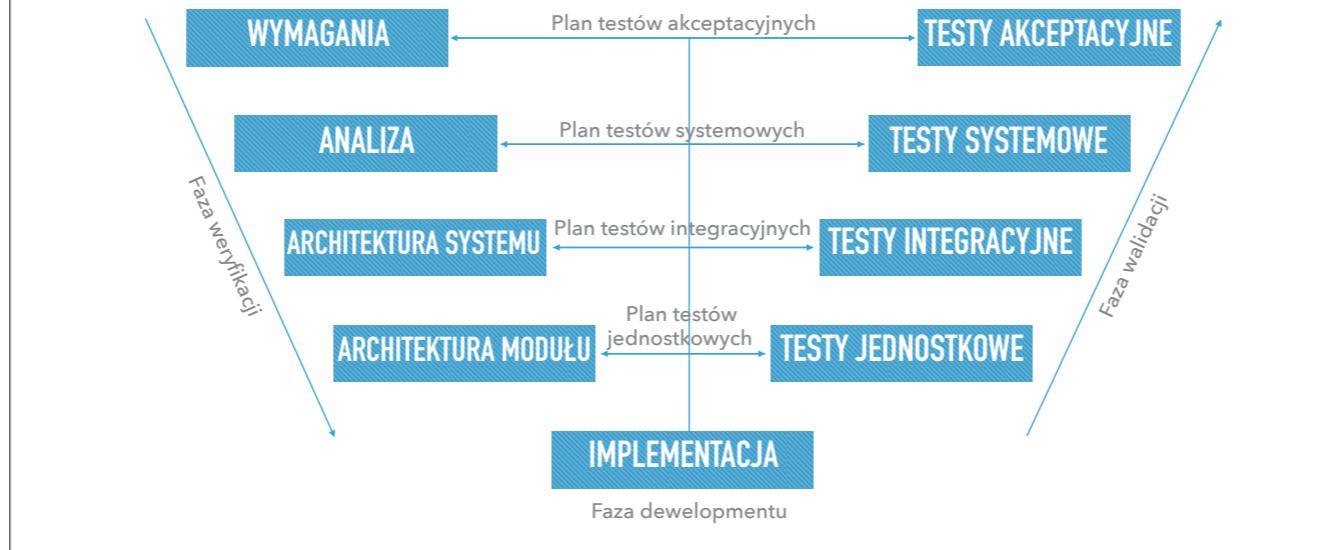
ZALETY MODELU SPIRALNEGO?

- ▶ Nastawienie na wykrywanie błędów, zapobieganie
- ▶ Zamiany wymagań są łatwe do zarządzania
- ▶ Wspiera szerokie użycie prototypów
- ▶ Wymagania mogą być trafniej uchwycone
- ▶ Użytkownik ma wczesny dostęp do działającego systemu
- ▶ Dewelopment może zostać podzielony na mniejsze części
- ▶ Elementy wysokiego ryzyka mogą być dostarczane wcześniej

WADY MODELU SPIRALNEGO?

- ▶ Proces i zarządzanie są bardzo złożone
- ▶ Koniec projektu, może nie być znany na wczesnym etapie
- ▶ Niestosowny do małych projektów lub projektów niskiego ryzyka, może być dla nich bardziej kosztowny
- ▶ Duża liczba pośrednich faz wymaga obszernej dokumentacji

MODEL V (WALIDACJA I WERYFIKACJA)

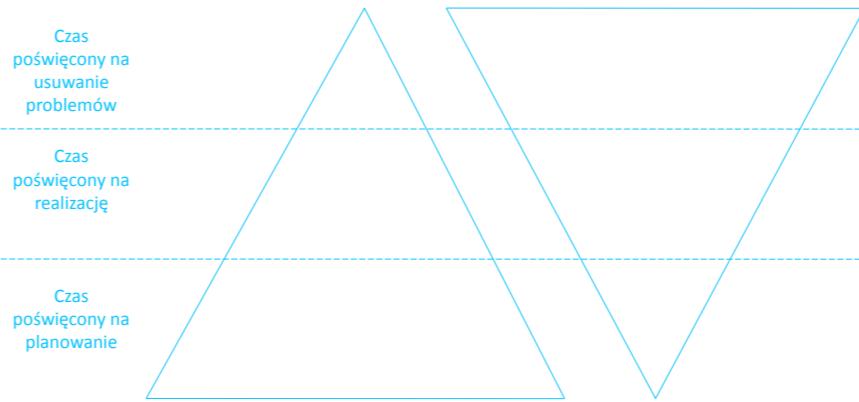


Model V jest modelem sekwencyjnym, w którym proces przebiega po ramionach litery V. Jest ulepszoną wersją modelu kaskadowego. Został stworzony w latach 80, po dziś dzień jest standardem instytucji rządowych Republiki Federalnej Niemiec.

Założenia:

- w kolejnych fazach **zespoł produkcyjny** opracowuje produkty poszczególnych faz, podczas gdy równolegle opracowywane są plan testów i test case'y aby zweryfikować i zwalidować produkt zgodnie z wymaganiami dla danej fazy

MODEL V – ZASADA 2 TRÓJKĄTÓW

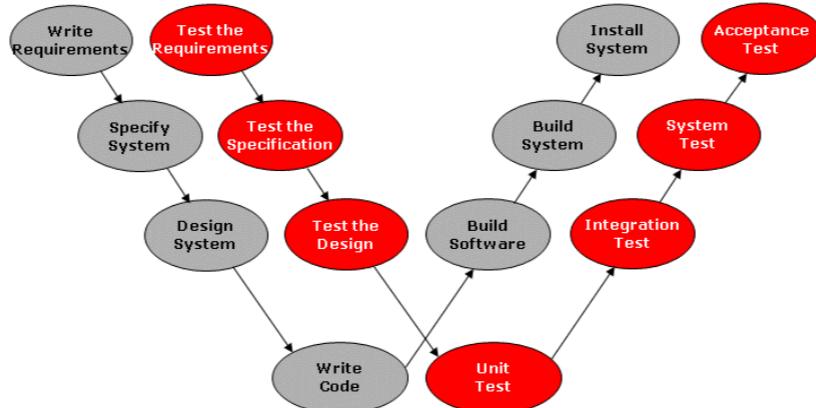


Model V jest modelem sekwencyjnym, w którym proces przebiega po ramionach litery V. Jest ulepszoną wersją modelu kaskadowego. Został stworzony w latach 80, po dziś dzień jego model V jest wykorzystywany przez struktury Państwowe Republiki Federalnej Niemiec

ZAłożenia:

- w kolejnych fazach **zespoł produkcyjny** opracowuje produkty poszczególnych faz, podczas gdy
- w tych samych fazach zespół QA opracowuje plan testów dla poszczególnych faz projektu

MODEL V - DUAL VEE



Dual VEE to modyfikacja modelu VEE, w którym na bieżąco weryfikacji podlegają produkty kolejnych faz projektu.

ZALETY MODELU V

- ▶ Wysoko zdyscyplinowany model - tylko jedna faza na raz może być realizowana
- ▶ Łatwość planowania, harmonogramowania, monitorowania
- ▶ Eliminacja zagrożeń na wczesnych etapach projektu
- ▶ Obniżenie kosztów usuwania usterek
- ▶ Wyczerpująca, szczegółowa specyfikacja
- ▶ Zaangażowanie zespołu w cały proces - wysoka jakość produktu
- ▶ Zarządzanie zmianą

WADY MODELU V

- ▶ Czasochłonność
- ▶ Wysoki koszt wytwarzania
- ▶ Wymaga zaangażowania dużej liczby zasobów
- ▶ Dokumentowanie na każdym z etapów

(anitycy, architekci, testerzy, deweloperzy, pm)



ĆWICZENIA

PODZIAŁ NA GRUPY PROJEKTOWE,
PRZEGŁĄD NARZĘDZIA

TEKST

PLAN ĆWICZEŃ

- ▶ 2-osobowe zespoły „projektowe”
- ▶ Pomysł na produkt stanowiący wyzwanie technologiczne, organizacyjne
- ▶ Dokumentowanie wymagań: Przypadki Użycia, User stories
- ▶ Przygotowanie diagramu przypadków użycia, klas lub procesu biznesowego
- ▶ Przygotowanie RACI i Rejestru Ryzyka
- ▶ Wycena przy użyciu PERT i harmonogramowanie (Gantt)
- ▶ Planowanie Sprintu / Retrospektyna

TEKST

NARZĘDZIA

- ▶ <https://github.com/join>
- ▶ <https://www.draw.io/>
- ▶ Excel
- ▶ MS Project
- ▶ Szablony dokumentów