BYT

1. Model wytwarzania oprogramowania według modelu UP. Opis z diagramem. W jakiej sytuacji warto decydować się na wytwarzanie oprogramowania w podanym modelu?. Wady i zalety.

Model UP jest zunifikowanym procesem wytwarzania oprogramowania. W tym modelu oprogramowanie wytwarzane jest w kolejnych iteracjach, po których otrzymujemy pewien funkcjonalny prototyp. Kolejne iteracje składają się z rozpoczęcia, analizy wymagań, budowy prototypu wraz z weryfikacją z klientem. **Zaletami** tego podejścia jest wprowadzanie poprawek na bieżąco, co ostatecznie daje produkt zgodny z oczekiwaniami klienta. **Wadą** tego rozwiązania może być rozszerzenie wymagań pod pretekstem wprowadzania poprawek.
Wytworzone prototypy powinny być archiwizowane.

Iteracja	Rozpoczęcie	Analiza wymagań	Budowa prototypu	Weryfikacja
iteracja IV	n R	n A	n B	n W
iteracja III	III R	III A	III B	III W
iteracja II	II R	II A	II B	II W
iteracja I	I R	I A	I B	I W

Model UP warto stosować w dużych projektach rozciągniętych w czasie.

2. Wyjaśnić czym są wymagania STRQ, FEAT i TERM, podać przykłady.

STRQ - wymagania od pracowników firmy (na podstawie wywiadów)
np. "Chcemy mieć możliwość załączania plików do wiadomości"
FEAT - określają co będzie robiła tworzona aplikacja/system. Realizują wymagania STRQ.
np. "Przechowywanie załączonych plików wiadomości na serwerze"
TERM - Słownik pojęć definiujący terminologię używaną w dokumentach projektowych
np. *pracownik* - osoba zatrudniona w firmie

***3.* Pozycja konfiguracji** - dokumentuje status aplikacji np. użyte biblioteki i ich wersje, historię zmian (kto i kiedy ich dokonał), opis systemu co robi każdy moduł, ekrany interfejsu użytkownika, pliki z danymi (teksty, bazy, słowniki). Opis środowiska użytego w projekcie (kompilatory, sprzęt itd.). Pozycja konfiguracji jest wyróżnialnym elementem w projekcie lub produkcie.

Przykład tablicy statusu konfiguracji

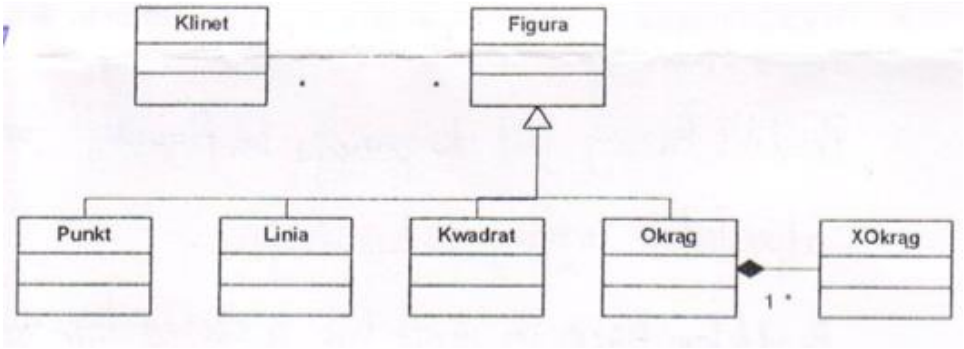
Produkty bazowe ->	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Kamienie milowe ->	*Zatwierdzenie DWU*	*Zatwierdzenie DWO*	*Zatwierdzenie DAP*	*Pośredni produkt*	*Zatwierdzenie DDP*	*Akceptacja wstępna*	*Akceptacjakońcowa*
DWU	*1.0*	*1.1*	*1.2*	*1.3*	*1.4*	*1.5*	*1.6*
DWO		*1.0*	*1.1*	*1.2*	*1.3*	*1.4*	*1.5*
DAP			*1.0*	*1.1*	*1.2*	*1.3*	*1.4*
Podr. uż.				*1.0*	*1.1*	*1.2*	*1.3*

Kompil.				*6.4*	*6.4*	*6.5*	*6.5*
Program				*1.0*	*1.1*	*1.2*	*1.3*

***DWU**-Dokument Wymagań Użytkownika, **DWO**-Dokument Wymagań na Oprogramowanie, **DAP**-Dokument Analityczno-Projektowy*

4. Zasada "miej koniec na względzie" - ma zastosowanie zarówno w życiu osobistym jak i w zawodowym. Na początku definiujemy cel długoterminowy np. "za rok zostanę magistrem, dyrektorem itp.". W drugim kroku definiujemy co musimy zrobić (jakie kroki podjąć aby osiągnąć wyznaczony cel). Np. aby zostać magistrem za rok teraz musze zdać egzamin.

5.Jaki to wzorzec projektowy

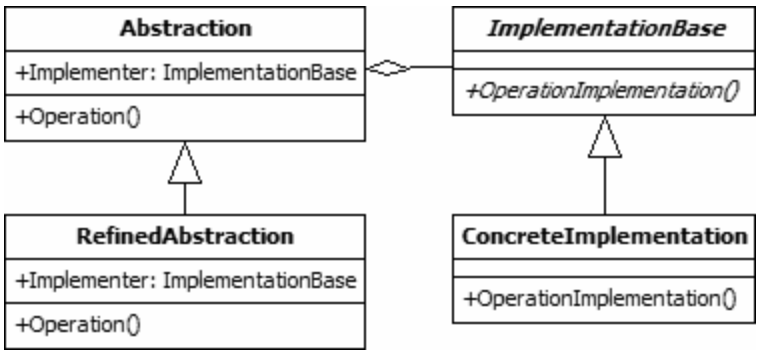


Schemat przedstawia dwa połączone wzorce *ADAPTER i FASADA.*

Adapter - rozwiązuje problem współpracy niekompatybilnych klas. Adapter niejako opakowuje interfejs klasy niekompatybilnej i udostępnia kompatybilny z drugą klasą interfejs.

Fasada - umożliwia ukrycie złożoności systemu. Tworzy dodatkowy obiekt który udostępnia najbardziej potrzebne funkcje ukrywając złożoność tego co jest za fasadą.

6. Most



Oddziela abstrakcję od jej implementacji. Przykładowo abstrakcyjny element może być logiką biznesową aplikacji, która została stworzona bez wiedzy o szczegółach implementacji np. dostępu do danych.
Wzorzec umożliwia zmianę implementacji bez zmiany abstrakcji. Np. zmiana systemu operacyjnego czy bazy danych nie wpływa na logikę biznesową.

INNE

Kryzys oprogramowania

Podstawowym powodem kryzysu oprogramowania jest złożoność produktów informatyki i procesów ich wytwarzania.

- Sprzeczność pomiędzy odpowiedzialnością, jaka spoczywa na współczesnych SI, a ich zawodnością wynikającą ze złożoności i ciągle niedojrzałych metod tworzenia i weryfikacji oprogramowania.
- Ogromne koszty utrzymania oprogramowania.
- Niska kultura ponownego użycia wytworzonych komponentów projektów i oprogramowania; niski stopień powtarzalności poszczególnych przedsięwzięć.
- Długi i kosztowny cykl tworzenia oprogramowania, wysokie prawdopodobieństwo niepowodzenia projektu programistycznego.
- Długi i kosztowny cykl życia SI, wymagający stałych (często globalnych) zmian.
- Eklektyczne, niesystematyczne narzędzia i języki programowania.
- Frustracje projektantów oprogramowania i programistów wynikające ze zbyt szybkiego postępu w zakresie języków, narzędzi i metod oraz uciążliwości i długotrwałości procesów produkcji, utrzymania i pielęgnacji oprogramowania.
- Uzależnienie organizacji od systemów komputerowych i przyjętych technologii przetwarzania informacji, które nie są stabilne w długim horyzoncie czasowym.
- Problemy współdziałania niezależnie zbudowanego oprogramowania, szczególnie istotne przy dzisiejszych tendencjach integracyjnych.
- Problemy przystosowania istniejących i działających systemów do nowych wymagań, tendencji i platform sprzętowo-programowych.

Walka z kryzysem oprogramowania

- Stosowanie technik i narzędzi ułatwiających pracę nad złożonymi systemami;
- Korzystanie z metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń;
- Usystematyzowanie procesu wytwarzania oprogramowania, tak aby ułatwić jego planowanie i monitorowanie;
- Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia.

Jak walczyć ze złożonością:

- Zasada dekompozycji:
  rozdzielenie złożonego problemu na podproblemy, które można rozpatrywać i rozwiązywać niezależnie od siebie i niezależnie od całości.

- Zasada abstrakcji:
  eliminacja, ukrycie lub pominięcie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji; wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzaniu pojęć lub symboli oznaczających takie cechy.
- Zasada ponownego użycia:
  wykorzystanie wcześniej wytworzonych schematów, metod, wzorców, komponentów projektu, komponentów oprogramowania, itd.

- Zasada sprzyjania naturalnym ludzkim własnościom:
  dopasowanie modeli pojęciowych i modeli realizacyjnych systemów do wrodzonych ludzkich własności psychologicznych, instynktów oraz mentalnych mechanizmów percepcji i rozumienia świata.

SCRUM

„Scrum nie jest procesem, ani techniką tworzenia produktów, lecz stanowi ramę metodyczną, w obrębie, której można stosować inne procesy i techniki"

Dlaczego Scrum działa:

- Zorientowanie na klienta i jego zmieniające się potrzeby;
- Skrócenie czasu pomiędzy „listą życzeń" a zaimplementowaną funkcjonalnością;
- Skupienie się na dostarczeniu funkcjonalności o najwyższym priorytecie jak najszybciej się da;
- Wzrost rangi zespołu. Jeden problem, wiele umysłów.

Role w Scrum:

- **Mistrz Młyna (ScrumMaster)** - jest osobą odpowiedzialną za prawidłowe przeprowadzenie Scrum-a, dopilnowanie, aby jego zasady były przestrzegane przez wszystkich biorących udział w procesie wytwórczym
- **Właściciel Produktu** - reprezentuje osoby zainteresowane projektem i jego rezultatami. Tworzy listę wymagań, zwaną Zaległościami Produktowymi (Product Backlog)
- **Zespół** - składa się ze specjalistów mających na celu stworzenie funkcjonalności opartej na Zaległościach Produktowych

Etapy SCRUM:

1. Spotkanie planujące projekt (opcjonalne)
    - cel projektu i plan jak go zrealizować
    - jak będzie wyglądał produkt końcowy (cechy, funkcjonalność)
    - czas ukończenia projektu i ryzyko z nim związane
    - wstępny kosztorys projektu.
2. Sprint
    - dowolna ich ilość w procesie wytwórczym
    - każdy sprint jest iteracją i trwa 30 dni
    - koniec sprintu = przyrost funkcjonalności o najwyższym priorytecie
    - składa się z iteracji, czyli codziennego Scrum.

Podział Sprintu:

- Spotkanie planujące sprint, cz. 1:
    - Czas trwania: 4 h
    - Cel: utworzenie Zaległości Sprintu, czyli wybranych elementów z listy Zaległości Produktowych
    - Uczestnicy czynni: ScrumMaster, Właściciel Produktu, Zespół

- Spotkanie planujące sprint, cz. 2 (początek Sprintu):
    - Czas trwania: 4 h (bezpośrednio po części 1)
    - Cel: strategia realizacji wybranych Zaległości w przyrost funkcjonalności, przydział zadań
    - Uczestnicy czynni: Zespół

Codzienny Scrum:

- Czas trwania: 15 minut
- Cele: Analiza postępów każdego członka zespołu od ostatniego spotkania
    - Co się udało zrobić.
    - Co się planuje zrobić teraz.
    - Trudności w realizacji zadań.
- Uczestnicy czynni: Zespół, ScrumMaster

Spotkanie przeglądu sprintu:

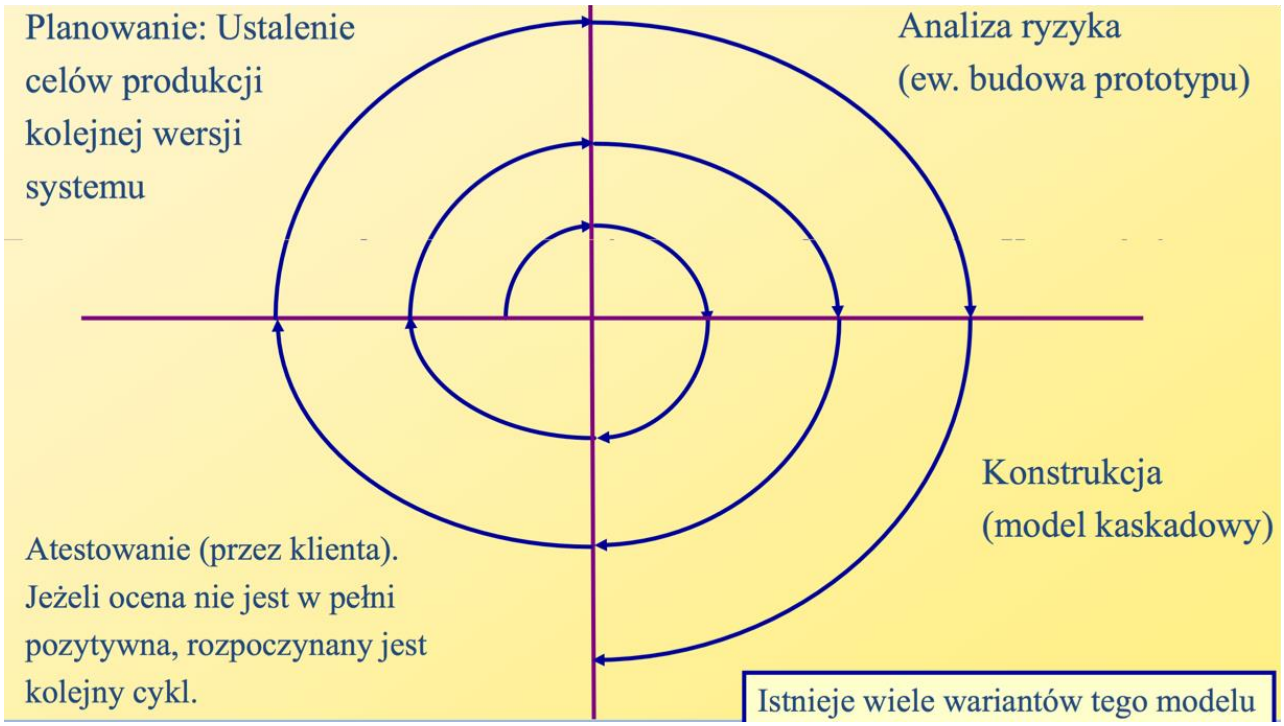
- Czas trwania: 4 h
- Cel: zaprezentowanie wykonanej funkcjonalności
- Uczestnicy:
    - Zespół - prezentuje funkcjonalność
    - Osoby zainteresowane projektem – spostrzeżenia, obserwacje, wymagane zmiany do wprowadzenia, wprowadzenie nowych funkcjonalności
    - ScrumMaster – określa miejsce i datę kolejnego przeglądu sprintu

Retrospektywne spotkanie Sprintu:

- Czas trwania: 3 h
- Cel: analiza przebiegu sprintu, co poszło dobrze a co mogłoby zostać ulepszone w następnym sprincie
- Uczestnicy:
    - Zespół - każdy członek zespołu odpowiada na powyższe pytania
    - ScrumMaster – zapisuje odpowiedzi na formularzu podsumowywującym, pomaga w poszukiwaniu lepszych sposobów wykorzystania Scrum

Model spiralny

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.



Model (opis)

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** - Tworzenia 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 projektu.

Cechy modelu spiralnego

Widoczną cechą modelu spiralnego jest szczegółowe potraktowanie zagrożeń realizacji projektu. Dobrze rozpoznane zagrożenia i przedsięwzięte kroki im zapobiegania lub redukcji skutkują m.in. wysoką niezawodnością (dependability) powstającego oprogramowania, bądź pewnością, że projekt ma szanse dalszej realizacji.

W modelu spiralnym nie ma takich faz jak specyfikowanie albo projektowanie. Jeden cykl spirali może przebiegać w oparciu o model kaskadowy procesu tworzenia oprogramowania, w innym można użyć prototypowania lub przekształceń formalnych, w zależności od aktualnego etapu przedsięwzięcia / realizowanej części systemu (np. inny dla tworzenia interfejsu użytkownika, inny dla krytycznych funkcji bezpieczeństwa)

Każdy cykl wymaga formalnej decyzji o kontynuacji projektu.

Zalety modelu spiralnego

- Można wykorzystać gotowe projekty
- Faza oceny w każdym cyklu pozwala uniknąć błędów lub wcześniej je wykryć
- Cały czas istnieje możliwość rozwijania projektu
- Częste kontrole jakości w kolejnych cyklach spirali
- Nastawienie na wykrywanie błędów i działania kontrolne, a nie na zapobieganie
- Orientacja na zarządzanie, czas i budżet.

Wady modelu spiralnego

- Model nie do końca dopracowany. Każdy projekt jest inny i powstaje w innych warunkach. Ciężko określić jakie warunki brać pod uwagę.
- Tworzenia w oparciu o model spiralny wymaga doświadczenia w prowadzeniu tego typu projektów oraz często wiedzy ekonomicznej w zarządzaniu
- Wysoki koszt usuwania błędów wykrytych w finalnych etapach projektu