INŻYNIERIA OPROGRAMOWANIA

wykład 5:
ANALIZA WYMAGAŃ

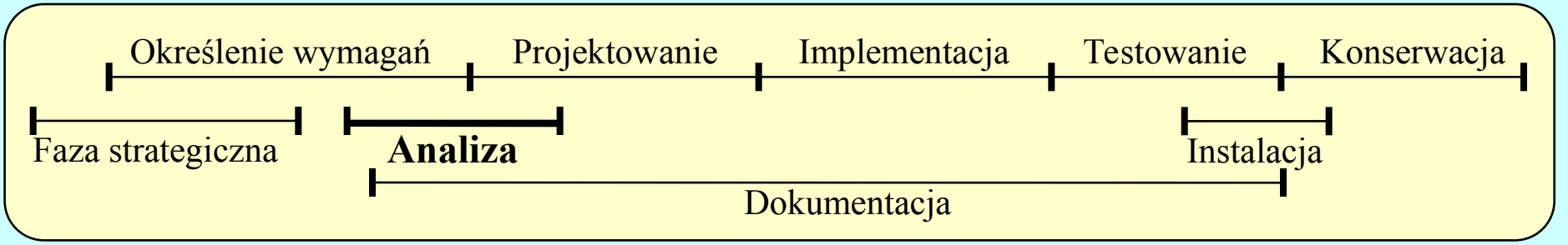
dr inż. Leszek Grocholski

Zakład Inżynierii Oprogramowania
Instytut Informatyki
Uniwersytet Wrocławski

Plan wykładu

1. Model analityczny
2. Czynności w fazie analizy
3. Wymagania na oprogramowanie
4. Notacje w fazie analizie
5. Metodyki strukturalne i obiektowe
6. Metodyki obiektowe → UML
7. Proces tworzenia modelu obiektowego
8. Dokument wymagań na oprogramowanie
9. Kluczowe czynniki sukcesu i rezultaty fazy analizy

Analiza



Analiza jest fazą niezbędną do **przejścia** pomiędzy wymaganiami użytkownika (wynikającymi z jego procesów biznesowych) do wymagań na system informatyczny (wspomaganiem tych procesów).

Celem **początkowych** czynności analizy jest rozpoznaniem wszystkich tych czynników lub warunków w dziedzinie przedmiotowej, w otoczeniu projektu, w istniejących lub planowanych systemach, które mogą wpłynąć na **decyzje projektowe** i na **przebieg procesu projektowego**.

Na początku analizy chodzi nam wyłącznie o **rozpoznanie** wszelkich krytycznych aspektów w dziedzinie biznesowej, które mogą zaważyć na rezultacie lub przebiegu projektu. Nie interesujemy się zmianą procesów biznesowych wynikającą z wprowadzenia do nich systemu informatycznego.

Wynik analizy

Celem **końcowych** czynności analizy jest zamiana wymagań użytkownika oraz wszelkich krytycznych cech procesów biznesowych i otoczenia tych procesów na:

- wymagania wobec projektowanego systemu,
- wymagania wobec procesu jego wytwarzania.

Końcowe czynności analizy są więc początkowymi decyzjami projektowymi. Moment, w którym analiza zamienia się w projektowanie, jest umowny i często nieokreślony.

Istotne w analizie jest ustalenie **logicznego modelu systemu**, opisującego sposób realizacji przez system postawionych wymagań, bez szczegółów implementacyjnych.

Celem końcowego dokumentu analizy jest uzyskanie odpowiedzi na pytania: "jakie cechy ma posiadać przyszły system?"
oraz „jak ma być on tworzony?”.

Model analityczny

Każda dyscyplina techniczna korzysta z modeli.

Proszę podać przykłady.

Dlaczego stosujemy modele ?

Film Altkom Akademia
User Story – zwinne definiowanie wymagań:
https://www.youtube.com/watch?v=NRiaTM9t5oc&t=2175s

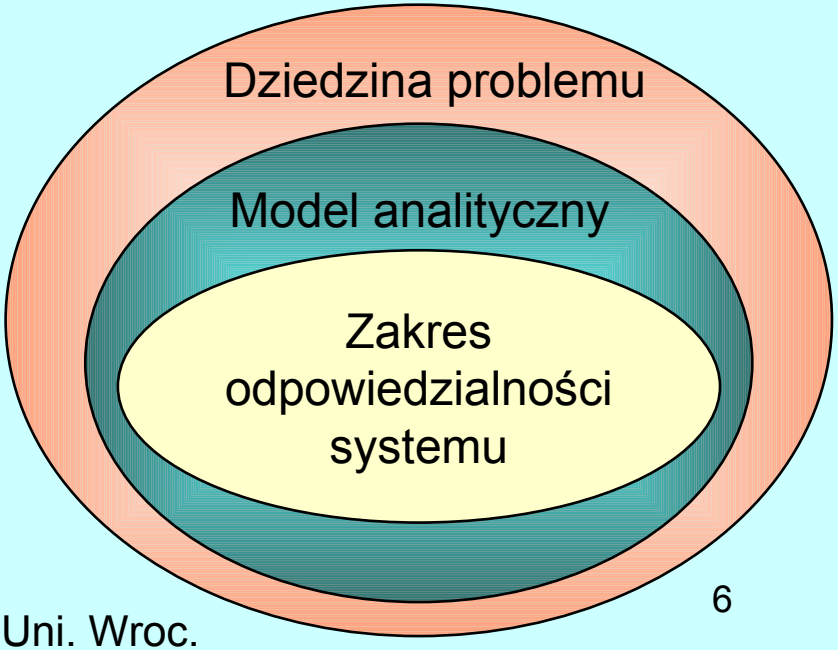
Model analityczny

Z reguły wykracza poza zakres odpowiedzialności systemu. Przyczyny:

Ujęcie w modelu pewnych elementów dziedziny problemu nie będących częścią systemu czyni model bardziej zrozumiałym. Przykładem jest ujęcie w modelu systemów zewnętrznych, z którymi system ma współpracować.

Na etapie modelowania może nie być jasne, które elementy modelu będą realizowane przez oprogramowanie, a które w sposób sprzętowy lub ręcznie.

Dostępne środki mogą nie pozwolić na realizację systemu w całości.
Celem analizy może być wykrycie tych fragmentów dziedziny problemu, których wspomaganie za pomocą oprogramowania będzie szczególnie przydatne.



Cechy modelu analitycznego (logicznego)

- Uproszczony opis systemu
- Hierarchiczna dekompozycja funkcji systemu
- Model logiczny jest opisany przy pomocy notacji zgodnej z pewną konwencją
- Jest on zbudowany przy użyciu dobrze rozpoznanych metod i narzędzi
- Jest on używany do wnioskowania o przyszłym oprogramowaniu

Model oprogramowania powinien być jego uproszonym opisem, opisującym wszystkie istotne cechy oprogramowania na wysokim poziomie abstrakcji.

Model ten jednakże nie zastępuje doświadczenia i wnikliwości projektantów, lecz pomaga projektantom w zastosowaniu tych walorów.

Logiczny model oprogramowania:

- pokazuje co system musi robić,
- jest zorganizowany hierarchicznie, wg poziomów abstrakcji,
- unika terminologii implementacyjnej,
- pozwala na wnioskowanie „od przyczyny do skutku” i odwrotnie.

Czynności w fazie analizy

Rozpoznanie, wyjaśnianie, modelowanie, specyfikowanie i dokumentowanie rzeczywistości lub problemu będącego przedmiotem projektu

Ustalenie kontekstu projektu

Ustalenie wymagań użytkowników

Ustalenie wymagań organizacyjnych

Inne ustalenia, np. dotyczące preferencji sprzętowych, preferencji w zakresie oprogramowania, ograniczeń finansowych, ograniczeń czasowych, itd.

Uwaga:
W zasadzie analiza nie powinna stawiać nacisku na zmianę rzeczywistości poprzez wprowadzenie tam nowych elementów np. w postaci systemu komputerowego. Jej **celem jest rozpoznanie** wszystkich tych aspektów rzeczywistości, które mogłyby mieć wpływ na postać, organizację lub wynik projektu.

Tematy i techniki analizy

- Budowa fizyczna systemu (składniki systemu)
- Analiza funkcji (historyjek użytkownika)
- Analiza przypadków użycia
- Identyfikacja i weryfikacja składowych logicznych systemu
- Identyfikacja i definiowanie algorytmów
- Modelowanie stanów i przejść między stanami
- Modelowanie procesów i przepływów danych
- Modelowanie przepływu sterowania
- Inne

Wiele tych technik jest omówione podczas wykładu nt. obiektowego projektowania oprogramowania.

Wymagania na oprogramowanie

W trakcie analizy **wymagania użytkownika** są przekształcane w **wymagania na oprogramowanie.**

Mogą one dotyczyć:

- Funkcji systemu
- Wydajności systemu
- Zewnętrznych interfejsów
- Wykonywanych operacji
- Wymaganych zasobów
- Sposobów weryfikacji
- Sposobów testowania
- Dokumentacji
- Ochrony (losowe zniszczenie)
- Przenośności
- Jakości
- Niezawodności
- Pielęgnacyjności
- Bezpieczeństwa (włamania)

Uwaga:
Wymagania powinny być zorganizowane hierarchicznie.
Wymagania niefunkcjonalne powinny być skojarzone z wymaganiami funkcjonalnymi (np. poprzez wzajemne odsyłacze).

Reguły modelu logicznego opartego na funkcjach

- Funkcje musza wynikać z historyjek użytkownika

- Funkcje muszą mieć pojedyncze, zdefiniowane cele.

- Funkcje powinny być zdefiniowane na tym samym poziomie (np. funkcja *Oblicz Sumę Kontrolną* jest niższego poziomu niż funkcja *Obsługa Protokołu Sieciowego*).

- Interfejsy do funkcji (wejście i wyjście) powinny być minimalne. Pozwala to na łatwiejsze oddzielenie poszczególnych funkcji.

- Przy dekompozycji funkcji należy trzymać się zasady „nie więcej niż 7 funkcji podrzędnych”.

- Opis funkcji nie powinien odwoływać się do pojęć i terminów implementacyjnych (takich jak plik, zapis, zadanie, moduł, stacja robocza).

- Określenie wskaźników wydajnościowe funkcji (szybkość, częstość, itd).

- Powinny być zidentyfikowane funkcje krytyczne (od których zależy istota systemu).

- Nazwy funkcji powinny odzwierciedlać ich cel i mówić ***co*** ma być zrobione, a nie ***jak*** ma być zrobione.

- Nazwy funkcji powinny mieć charakter deklaracyjny (np. *Walidacja Zlecenia Zewnętrznego),* a nie proceduralny (np. *Czynności Systemu po Otrzymaniu Zlecenia*).

Notacje w fazie analizy

Rodzaje notacji

- Język naturalny
- Notacje graficzne
- Specyfikacje - ustrukturalizowany zapis tekstowy i numeryczny

Szczególne znaczenie maja notacje graficzne. Inżynieria oprogramowania wzoruje się na innych dziedzinach techniki, takich jak elektronika i mechanika. **Zalety notacji graficznych potwierdzają badania psychologiczne.**

Funkcje notacji

- Narzędzie pracy analityka i projektanta, zapis i analiza pomysłów
- Współpraca z użytkownikiem
- Komunikacja z innymi członkami zespołu
- Podstawa implementacji oprogramowania
- Zapis dokumentacji technicznej

Notacje powinny być przejrzyste, proste, precyzyjne, łatwo zrozumiałe, umożliwiające modelowanie złożonych zależności.

Metodyki obiektowe w analizie

Metodyka wykorzystująca **pojęcia obiektowości** dla celów **modelowania pojęciowego** oraz analizy i projektowania systemów informatycznych.

Podstawowym składnikiem jest **diagram klas**, będący zwykle wariantem notacyjnym i pewnym rozszerzeniem diagramów encja-związek.

Diagram klas zawiera: **klasy**, w ramach klas specyfikacje **atrybutów** i **metod**, związki **generalizacji**, związki **asocjacji** i **agregacji**, **liczności** tych związków, różnorodne **ograniczenia** oraz inne oznaczenia.

Uzupełnieniem tego diagramu są inne: **diagramy dynamiczne** uwzględniające **stany** i przejścia pomiędzy tymi stanami, **diagramy interakcji** ustalające zależności pomiędzy wywołaniami metod, **diagramy funkcjonalne** (będące zwykle pewną mutacją diagramów przepływu danych), itd.

Koncepcja **przypadków użycia** (*use cases*) zakłada odwzorowanie struktury systemu z punktu widzenia jego użytkownika.

Różnice pomiędzy metodykami

Podejścia proponowane przez różnych autorów różnią się częściowo, nie muszą być jednak ze sobą sprzeczne. **Nie ma metodyk uniwersalnych.** Analitycy i projektanci wybierają kombinację technik i notacji, która jest w danym momencie najbardziej przydatna.

Poszczególne metodyki zawierają elementy rzadko wykorzystywane w praktyce.

Notacje proponowane przez różnych autorów nie są koniecznie nierozerwalne z samą metodyką.

Narzędzia CASE nie narzucają metodyki; raczej, określają one tylko notację. Twierdzenia, że jakieś narzędzie CASE “jest oparte” na konkretnej metodyce jest często wyłącznie hasłem reklamowym. Nawet najlepsze metodyki i narzędzia CASE nie są w stanie zapewnić jakości projektów.

Uwaga:
Kluczem do dobrego projektu jest zespół doświadczonych, zaangażowanych i kompetentnych osób, dla których metodyki, notacje i narzędzia CASE służą jako istotne wspomaganie.

Notacja a metodyka

Dowolny język, w tym notacje stosowane w metodykach, oprócz *składni* posiada dwa znacznie od niej ważniejsze aspekty: *semantykę* i *pragmatykę*.

Składnia określa, jak wolno łączyć (składać) ze sobą przyjęte oznaczenia.

Semantyka określa, co należy rozumieć pod przyjętymi oznaczeniami.

Pragmatyka określa, w jaki sposób i do czego należy używać przyjętych oznaczeń.

Pragmatyka określa, jak do konkretnej sytuacji dopasować pewien wzorzec notacyjny. Pragmatyka wyznacza więc *procesy* prowadzące do wytworzenia zapisów wyników analizy i projektowania, które są zgodne z intencją autorów tej notacji. Jakakolwiek notacja nie ma większego sensu bez wiedzy o tym, w jaki sposób może być ona użyta w ramach pewnego procesu analizy i projektowania.

W metodykach pragmatyka stosowanej notacji (czyli jak i do czego jej użyć) jest sprawą podstawową. Jest ona zazwyczaj trudna do objaśnienia: nie ma innego sposobu oprócz pokazania sposobów użycia na przykładach przypominających realne sytuacje. **Realne sytuacje są zazwyczaj skomplikowane, co powoduje wrażenie, że przykłady zamieszczane w podręcznikach wydają się banalne.**

UML - przykład notacji

UML (Unified Modeling Language) powstał jako synteza trzech metodyk/notacji:

- OMT (Rumbaugh): metodyka ta była dobra do modelowania dziedziny przedmiotowej. Nie przykrywała jednak dostatecznie dokładnie zarówno aspektu użytkowników systemu jak i aspektu implementacji/konstrukcji.

- OOSE (Jacobson): metodyka ta dobrze podchodziła do kwestii modelowania użytkowników i cyklu życiowego systemu. Nie przykrywała jednak dokładnie modelowania dziedziny przedmiotowej jak i aspektu implementacji/konstrukcji.

- OOAD (Booch): metodyka dobrze podchodziła do kwestii projektowania, konstrukcji i związków ze środowiskiem implementacji. Nie przykrywała jednak dostatecznie dobrze fazy analizy i rozpoznania wymagań użytkowników.

Istniało wiele aspektów systemów, które nie były właściwie przykryte przez żadne z wyżej wymienionych podejść, np. włączenie prototypowania w cykl życiowy, rozproszenie i komponenty, przystosowanie notacji do preferencji projektantów, i inne. Celem UML jest przykrycie również tych aspektów.

Diagramy definiowane w UML

Autorzy UML wychodzą z założenia, że żadna pojedyncza perspektywa spojrzenia na projektowany system nie jest wystarczająca. Stąd wiele środków:

- **Diagramy przypadków użycia (***use case***)**
- **Diagramy klas,** w tym diagramy pakietów
- **Diagramy zachowania się (***behavior***)**
  - Diagramy stanów
  - Diagramy aktywności
  - Diagramy sekwencji
  - Diagramy współpracy (*collaboration*)

- Diagramy implementacyjne

  - Diagramy komponentów
  - Diagramy wdrożeniowe (*deployment*)

Diagramy te zapewniają mnogość perspektyw systemu podczas analizy i rozwoju.

Diagramy definiowane w UML

Film :
Modelowanie oprogramowania z użyciem UML
Altkom Akademia:

https://www.youtube.com/watch?v=G9zTK90ZNXQ

Film:
Diagramy aktywności czyli co łączy procesy i scenariusze
Altkom Akademia:

https://www.youtube.com/watch?v=zpTQN__UPUk

Proces tworzenia modelu obiektowego

Zadania:

- Identyfikacja klas i obiektów
- Identyfikacja związków pomiędzy klasami
- Identyfikacja i definiowanie pól (atrybutów)
- Identyfikacja i definiowanie metod i komunikatów

Czynności te są wykonywane iteracyjnie. Kolejność ich wykonywania nie jest ustalona i zależy zarówno od stylu pracy, jak i od konkretnego problemu.

Inny schemat realizacji procesu budowy modelu obiektowego polega na rozpoznaniu funkcji, które system ma wykonywać. Dopiero w późniejszej fazie następuje identyfikacja klas, związków, atrybutów i metod. Rozpoznaniu funkcji systemu służą modele funkcjonalne (diagramy przepływu danych) oraz model przypadków użycia.

Identyfikacja klas i obiektów

Typowe klasy:

- przedmioty namacalne (samochód, czujnik)
- role pełnione przez osoby (pracownik, wykładowca, student)
- zdarzenia, o których system przechowuje informacje (lądowanie samolotu, wysłanie zamówienia, dostawa),
- interakcje pomiędzy osobami i/lub systemami o których system przechowuje informacje (pożyczka, spotkanie, sesja),
- lokalizacje - miejsce przeznaczone dla ludzi lub przedmiotów
- grupy przedmiotów namacalnych (kartoteka, samochód jako zestaw części)
- organizacje (firma, wydział, związek)
- wydarzenia (posiedzenie sejmu, demonstracja uliczna)
- koncepcje i pojęcia (zadanie, miara jakości)
- dokumenty (faktura, prawo jazdy)
- klasy będące interfejsami dla systemów zewnętrznych
- klasy będące interfejsami dla urządzeń sprzętowych

Obiekty, zbiory obiektów i metadane

W wielu przypadkach przy definicji klasy należy dokładnie ustalić, z jakiego rodzaju abstrakcją obiektu mamy do czynienia.

Należy zwrócić uwagę na następujące aspekty:

- czy mamy do czynienia z konkretnym obiektem w danej chwili czasowej?
- czy mamy do czynienia z konkretnym obiektem w pewnym odcinku czasu?
- czy mamy do czynienia z opisem tego obiektu (dokument, metadane)?
- czy mamy do czynienia z pewnym zbiorem konkretnych obiektów?

Np. klasa „samochód”. Może chodzić o:

- egzemplarz samochodu wyprodukowany przez pewną fabrykę
- model samochodu produkowany przez fabrykę
- pozycję w katalogu samochodów opisujący własności modelu
- historię stanów pewnego konkretnego samochodu

Podobnie z klasą „gazeta”. Może chodzić o:

- konkretny egzemplarz gazety kupiony przez czytelnika
- konkretne wydanie gazety (niezależne od ilości egzemplarzy)
- tytuł i wydawnictwo, niezależne od egzemplarzy i wydań
- partię egzemplarzy danej gazety dostarczaną codziennie do kiosku

Wymagania na oprogramowanie

Informacje organizacyjne

Streszczenie (maksymalnie 200 słów)
Spis treści
Status dokumentu (autorzy, firmy, daty, podpisy, itd.)
Zmiany w stosunku do wersji poprzedniej

Zasadnicza zawartość dokumentu

1. Wstęp
- 1.1. Cel
- 1.2. Zakres
- 1.3. Definicje, akronimy i skróty
- 1.4. Referencje, odsyłacze do innych dokumentów
- 1.5. Krótki przegląd

2. Ogólny opis
- 2.1. Relacje do bieżących projektów
- 2.2. Relacje do wcześniejszych i następnych projektów
- 2.3. Funkcje i cele
- 2.4. Ustalenia dotyczące środowiska
- 2.5. Relacje do innych systemów
- 2.6. Ogólne ograniczenia
- 2.7. Opis modelu

..... cd. na następnym slajdzie

Norma
ANSI/IEEE Std 830-1993 „Recommended Practice for Software Requirements Specifications”

Wymagania na oprogramowanie (2)

..... (*poprzedni slajd*)

3. Specyficzne wymagania (ten rozdział może być podzielony na wiele rozdziałów zgodnie z podziałem funkcji)

3.1. Wymagania dotyczące funkcji systemu
3.2. Wymagania dotyczące wydajności systemu
3.3. Wymagania dotyczące zewnętrznych interfejsów
3.4. Wymagania dotyczące wykonywanych operacji
3.5. Wymagania dotyczące wymaganych zasobów
3.6. Wymagania dotyczące sposobów weryfikacji
3.7. Wymagania dotyczące sposobów testowania
3.8. Wymagania dotyczące dokumentacji
3.9. Wymagania dotyczące ochrony dokumentacji i nośników
3.10. Wymagania dotyczące przenośności
3.11. **Wymagania dotyczące jakości**
3.12. Wymagania dotyczące niezawodności
3.13. Wymagania dotyczące pielęgnacyjności
3.14. Wymagania dotyczące bezpieczeństwa

Dodatki (to, co nie zmieściło się w powyższych punktach)

Analiza – zapewnienie jakości

PLAN ZAPEWNIENIA JAKOŚCI dotyczy WYMAGAŃ odnośnie:

- Sposobów weryfikacji
- Sposobów testowania
- Norm, standardów
- Niezawodności
- Pielęgnacyjności
- Bezpieczeństwa
- Standardów wewnętrznych

PLAN ZAPEWNIENIA JAKOŚCI powinien zakładać MONITOROWANIE następujących aktywności:

- Zarządzanie
- Dokumentowanie (jakość dokumentacji)
- Standardy, praktyki, konwencje i metryki
- Przeglądy i audyty
- Testowanie
- Raporty problemów i akcje korekcyjne
- Narzędzia, techniki i metody
- Kontrolowanie wytwarzanego kodu i mediów
- Kontrolowanie dostaw
- Pielęgnowanie i utrzymywanie kolekcji zapisów
- Szkolenie
- Zarządzanie ryzykiem

Analiza - kluczowe czynniki sukcesu

- Zaangażowanie właściwych osób ze strony klienta

- Kompleksowe i całościowe podejście do problemu, nie koncentrowanie się na partykularnych jego aspektach

- Nie unikanie trudnych problemów (bezpieczeństwo, skalowalność, szacunki objętości i przyrostu danych, itd.)

- Zachowanie przyjętych standardów, np. w zakresie notacji

- Sprawdzenie poprawności i wzajemnej spójności modelu

- Przejrzystość, logiczny układ i spójność dokumentacji

Analiza - podstawowe rezultaty

- Poprawiony dokument opisujący wymagania

- Słownik danych zawierający specyfikację modelu

- Dokument opisujący stworzony model, zawierający:

- diagram klas
- diagram przypadków użycia
- diagramy sekwencji komunikatów (dla wybranych sytuacji)
- diagramy stanów (dla wybranych sytuacji)
- raport zawierający definicje i opisy klas, atrybutów, związków, metod, itd.

- Harmonogram fazy projektowania

- Wstępne przypisanie ludzi i zespołów do zadań

INŻYNIERIA OPROGRAMOWANIA

Dziękuję za uwagę