Wprowadzenie do modelowania systemów informacyjnych

Kryzys oprogramowania

- długi i kosztowny cykl tworzenia oprogramowania
- wysokie prawdopodobieństwo niepowodzenia projektu (USA, 2003: 33% niepowodzeń, 33% sukcesów, 33% zakończeń problematycznych)
- sprzeczność: odpowiedzialność oprogramowania a jego zawodność (skutek złożoności i źle stosowanych metod tworzenia i weryfikacji oprogramowania)
- wysokie koszty utrzymywania oprogramowania
- frustracje projektantów i programistów, powód: zbyt szybki postęp języków, narzędzi i metod tworzenia oprogramowania, uciążliwość i długi czas trwania procesu
- problemy ze współdziałaniem niezależnie budowanego oprogramowania
- problemy z dostosowaniem istniejącego oprogramowania do nowych wymagań

Próby walki z kryzysem

- stosowanie technik i narzędzi ułatwiających pracę nad złożonym oprogramowaniem,
- wykorzystanie metod wspierających analizę nieznanych problemów
- usystematyzowanie procesu budowy oprogramowania

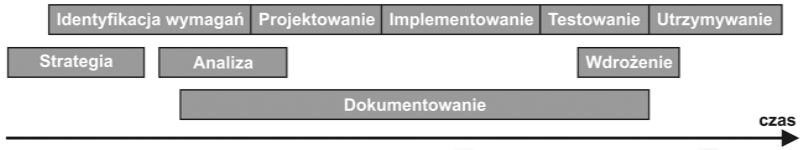
zastosowanie odpowiedniej
metodyki projektowania

Metodyka projektowania

- zestaw pojęć, notacji, modeli, języków, technik i sposobów postępowania używanych do:
  - analizy dziedziny będącej przedmiotem projektu,
  - projektowania pojęciowego,
  - projektowania logicznego,
  - projektowania fizycznego.
- ustala:
  - fazy projektu, role uczestników w poszczególnych fazach,
  - modele tworzone w każdej z faz,
  - reguły przechodzenia między fazami,
  - notacje, używanie w poszczególnych fazach,
  - dokumentację poszczególnych faz.

Fazy życia systemu informacyjnego

1. Strategia
2. Identyfikacja wymagań
3. Analiza
4. Projektowanie
5. Implementowanie, testowanie i dokumentowanie
6. Wdrożenie
7. Utrzymywanie



Strategia

- nazwy alternatywne: strategiczny plan rozwoju informatyzacji (SPRI), studium osiągalności
- realizowana przed decyzją o wykonaniu przedsięwzięcia
- cele:
  - określenie głównych celów projektu z punktu widzenia klienta (np. "*wzrost wydajności obsługi klientów*", "*skrócenie czasu realizacji zamówienia*", "*rozszerzenie zakresu oferowanych usług*", itd.)
  - przygotowanie odpowiednich przesłanek do podjęcia najważniejszych decyzji związanych z projektem (np. czy projekt zostanie uruchomiony)

Strategia – czynności

- określenie zakresu i kontekstu przedsięwzięcia (np. "*Zakresem przedsięwzięcia jest działalność wypożyczalni filmów. Systemami zewnętrznymi są: system dystrybucji filmów, pracownicy wypożyczalni, klienci wypożyczalni.*")
- oszacowanie rozmiaru przedsięwzięcia (np. metodą punktów funkcyjnych),
- identyfikacja ograniczeń (czasowych, zasobowych, technologicznych) i warunków wstępnych,
- wybór modelu realizacji przedsięwzięcia,
- opracowanie harmonogramu,
- wybór technik stosowanych w fazach: analizy i projektowania,
- wybór narzędzia CASE i środowiska implementacji.

Strategia – wyniki

- raport dla klienta:
  - definiujący cele przedsięwzięcia,
  - opisujący zakres przedsięwzięcia,
  - opisujący systemy zewnętrzne dla projektowanego systemu,
  - z ogólnym opisem wymagań,
  - z ogólnym modelem systemu,
  - ze wstępnym oszacowaniem kosztów,
  - ze wstępnym harmonogramem prac.
- raport o rozważanych rozwiązaniach, ich oceny i powód wyboru jednego z nich,
- opis wymaganych zasobów.

Identyfikacja wymagań

- cel: określenie wymagań klienta wobec projektowanego systemu i ich precyzyjna dokumentacja w postaci zrozumiałego dla obu stron dokumentu
- przeprowadzana przy dużym zaangażowaniu przedstawicieli klienta
- cele klienta z fazy strategii są zamieniane w konkretne wymagania, zapewniające osiągnięcie tych celów.
- problemy:
  - brak wiedzy klienta jak osiągnąć założone cele strategiczne,
  - brak precyzji klienta przy określaniu oczekiwań wobec systemu,
  - brak zrozumienia między klientem a wykonawcą,
  - sprzeczność celów różnych użytkowników przyszłego systemu.

Identyfikacja wymagań – techniki

- wywiady z pracownikami organizacji,
- analiza ankiet wypełnianych przez pracowników organizacji,
- analiza dokumentów, produkowanych przez organizację,
- obserwacja pracy organizacji,
- analiza dotychczas używanego oprogramowania,
- analiza wymagań systemowych (gdy nowy system ma być częścią większego systemu lub ściśle współpracować z innymi systemami)

Wymagania funkcjonalne (1)

- funkcje, które ma wykonywać projektowany system
- przykład: *"użytkownik ma mieć możliwość wyświetlenia informacji o filmach wypożyczonych przez klienta"*, *"każdy klient będzie identyfikowany unikalnym, 10-znakowym symbolem"*
- określane na różnych poziomach:
  - na poziomie użytkownika końcowego – mają postać ogólnego opisu
  - na poziomie systemu – szczegółowo definiują funkcje systemu wraz z ich wejściami i wyjściami

Wymagania funkcjonalne (2)

- konieczne jest określenie:
  - rodzajów użytkowników:
    - korzystających z systemu,
    - niezbędnych do działania systemu,
  - funkcji systemu dla każdego rodzaju użytkownika wraz ze sposobem ich użycia,
  - systemów zewnętrznych, wykorzystywanych przez system (zewnętrzne bazy danych, internet, itd.),
  - struktur organizacyjnych, przepisów prawnych, itd. mających wpływ na projektowany system.

Wymagania funkcjonalne (3)

- metody zapisu:
  - język naturalny,
  - tablice, formularze,
  - diagramy blokowe,
  - diagramy przypadków użycia,
  - formularz wymagań funkcjonalnych
- często uporządkowane w postaci hierarchii:
  - łączenie funkcji w grupy – podejście bottom-up,
  - podział funkcji na podfunkcje – podejście top-bottom
  - na każdym poziomie ten sam poziom szczegółowości,
  - brak znaczenia kolejności w hierarchii

Przykład specyfikacji wymagań

Numer i nazwa funkcji	2.1. Rejestracja nowego klienta
Opis	Funkcja zapisuje dane nowego klienta w rejestrze klientów wypożyczalni, generując dla klienta unikalny identyfikator. Dodatkowo drukuje kartę klienta.
Priorytet	Ważna
Użytkownicy	Pracownik recepcji
Warunki wstępne	Brak
Dane wejściowe	Dane osobowe klienta
Źródło danych wejściowych	Klient
Wynik	Pozycja opisująca klienta w rejestrze klientów. Wygenerowany unikalny identyfikatora klienta. Wydrukowana karty klienta.
Warunki końcowe	Brak

Wymaganie niefunkcjonalne (1)

- definiują ograniczenia, w których system ma działać
- mogą wynikać z:
  - potrzeb użytkownika,
  - ograniczeń finansowych,
  - potrzeby współpracy systemu z systemami zewnętrznymi,
  - potrzeby współpracy systemu ze specyficznym sprzętem,
  - ...

Wymaganie niefunkcjonalne (2)

- typowe grupy wymagań:
  - dotyczące produktu – narzucane na oprogramowanie, np.:
    - sposób obsługi ("wszystkie operacje muszą być realizowalne za pomocą klawiatury"),
    - wydajność działania ("czas oczekiwania użytkownika na odpowiedź nie może przekroczyć 10s przy 20 pracujących jednocześnie użytkownikach"),
    - używane protokoły ("system nie może korzystać z protokołu http")
  - organizacyjne – wynikają ze strategii firmy, jej procedur wewnętrznych, np. "realizacja funkcji X musi być zgodna ze standardem XYZ.102"
  - zewnętrzne – związane z czynnikami zewnętrznymi (systemami zewnętrznymi, środowiskiem działania, itd.), np.: "dane klienta mogą być dostępne tylko dla pracowników recepcji"

Dokument wymagań

- SRS – Software Requirements Specifications
- zbiera wszystkie wymagania dot. budowanego systemu
- nie jest projektem – mówi, co system ma robić, nie mówiąc, jak to ma robić
- jest podstawą zawarcia kontraktu między klientem a producentem oprogramowania
- definiowany normą IEEE Std 830-1998 *Recommended Practice for Software Requirements Specifications*

Analiza

- cel: ustalenie wszystkich czynników, które mogą wpłynąć na decyzje projektowe, przebieg procesu projektowego i realizację wymagań
- wynik: analityczny (logiczny) model systemu:
  - uproszczone ale rzeczywiste odzwierciedlenie rzeczywistości,
  - opisuje sposób realizacji wymagań,
  - nie wskazuje sposobu implementacji wymagań.
- system powinien być opisany przez zbiór spójnych, wzajemnie uzupełniających się modeli

Analityczny model systemu

- przedstawia sposób realizacji wymagań przez system bez wskazania szczegółów implementacyjnych
- posługuje się terminem "funkcja" jako realizowanym wymaganiem
- ujmuje dodatkowo elementy środowiska przyszłego systemu (nie będące częścią systemu):
  - w celu lepszego zrozumienia funkcjonowania systemu (np. systemy, z którymi modelowany system będzie współpracował),
  - z powodu braku jasności, które wymagania będą realizowane przez system, a które ręcznie

Cechy modelu analitycznego

- jest uproszczonym modelem systemu
- opisuje system na wysokim poziomie abstrakcji
- ma postać hierarchiczną – funkcje ogólne podlegają dekompozycji na funkcje niższego poziomu
- jest zbudowany w oparciu o wybraną notację

Przebieg fazy analizy (1)

- zamodelowanie rzeczywistości, będącej przedmiotem analizy
- uszczegółowienie wymagań użytkowników
- uszczegółowienie wymagań organizacyjnych
- inne ustalenia:
  - preferencje sprzętowe i programowe,
  - ograniczenia finansowe,
  - ograniczenia czasowe,
  - ...

Przebieg fazy analizy (2)

- diagramy:
  - diagram procesów,
  - diagram związków encji,
  - diagram przepływów danych,
  - diagram hierarchii funkcji

Metodyki

- metodyki strukturalne (klasyczne):
  - Yourdona, SSADM (Structured System Analysis and Design Methodology), SADT (Structured Analysis and Design Techniques)
  - diagramy procesów, diagramy przepływów danych, diagramy związków encji, schemat danych, ...
- metodyki obiektowe:
  - OMT (Object Modeling Technique), OOSE (Object-Oriented Software Engineering), OOAD (Object-Oriented Analysis and Design),
  - UML

Projektowanie

- cel:
  - opracowanie architektury systemu,
  - projekt logicznej struktury bazy danych systemu,
  - projekty aplikacji – formularzy ekranowych, raportów, bibliotek, inych,
- stosowane techniki:
  - transformacje encji i związków między nimi do schematu bazy danych,
  - transformacje funkcji do projektów aplikacji.

Implementowanie i dokumentowanie

- cel:
  - uzyskanie fizycznej struktury bazy danych,
  - uzyskanie działających aplikacji,
  - przetestowanie aplikacji pod kątem poprawnego działania,
  - uzyskanie założonej wydajności działania systemu,
  - uzyskanie dokumentacji technicznej i użytkowej systemu,
- stosowane techniki:
  - generatory baz danych,
  - generatory aplikacji,
  - generatory dokumentacji.

Wdrożenie i utrzymywanie

- wdrożenie – przekazanie działającego systemu końcowym użytkownikom,
- utrzymywanie:
  - usuwanie z systemu przeoczonych w fazie testowania błędów,
  - poprawianie efektywności działania systemu,
  - dostosowywanie funkcjonalności systemu do nowych potrzeb, zmieniających się wymagań, itd. (np. na skutek zmian przepisów, profilu działania organizacji)

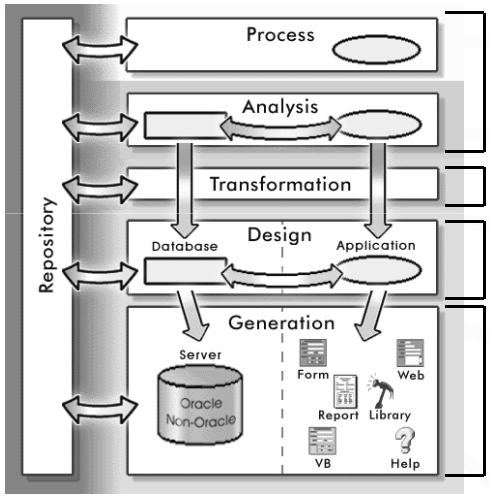
Narzędzia CASE (1)

- CASE – ang. *Computer Aided Software Engineering*,
- użycie narzędzi programistycznych w procesie rozwoju i utrzymywania oprogramowania,
- typowe narzędzia CASE:
  - edytory,
  - generatory kodu,
  - narzędzia refaktoryzujące,
  - narzędzia transformujące,
  - ...
- wyniki działania narzędzi CASE:
  - kod programu,
  - diagramy,
  - schemat bazy danych,
  - dokumentacja techniczna i użytkowa,
  - ...

Narzędzia CASE (2)

- rodzaje:
  - narzędzia CASE wysokiego poziomu – ang. upper CASE:
    - zastosowanie: etapy strategii, analizy i projektowania systemu
    - narzędzia: do tworzenia diagramów, sprawdzania poprawności informacji na diagramach
  - narzędzia CASE niskiego poziomu – ang. lower CASE:
    - zastosowanie: etapy implementowania, testowania i wdrażania,
    - narzędzia: generatory bazy danych, formularzy i raportów, testujące, zarządzające oprogramowaniem

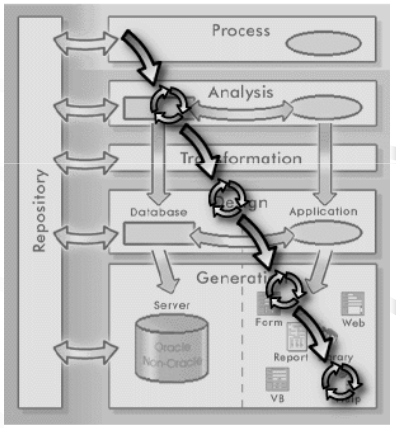
Projektowanie SI z wykorzystaniem Oracle Designer



- narzędzia do modelowania: PD, ERD, FHD, DFD,
- transformacje,
- struktura logiczna bazy danych (model relacyjny), definicje aplikacji,
- generowanie obiektów bazy danych i kodu po stronie serwera,
- generowanie aplikacji.

Model kaskadowy (1)

- synonimy: model wodospadowy, model liniowy
- kolejne fazy następują sekwencyjnie, przejście do następnej fazy dopiero po zakończeniu fazy poprzedniej,

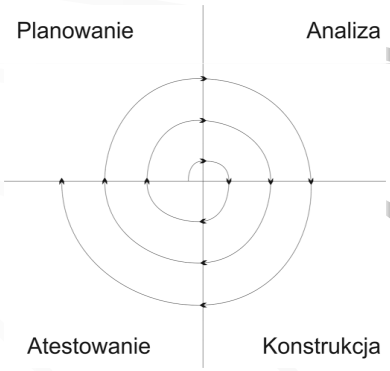


Model kaskadowy (2)

- zalety:
  - inwestycja w początkowe etapy tworzenia systemu skutkuje lepszą jakością wyników etapów następnych,
  - łatwość zarządzania projektem – podział na jasno oddzielone fazy
- wady:
  - narzucenie ścisłej kolejności wykonywania prac,
  - trudno dokładnie określić momenty przejścia do następnego etapu,
  - wysoki koszt błędów popełnionych na początkowych etapach,
  - brak sprzężenia zwrotnego pomiędzy kolejnymi etapami,
  - długie przerwy w kontaktach z klientem – efekt końcowy może nie odpowiadać potrzebom użytkownika,
- model dobry dla stabilnych organizacji, o dokładnie zdefiniowanych celach.

Model spiralny (1)

- cztery fazy:
  - planowanie – ustalenie celów produkcji kolejnej wersji systemu,
  - analiza – ocena celów i ryzyka ich realizacji,
  - konstrukcja – jej struktura jest zgodna z modelem kaskadowym
  - atestowanie – ocena przez klienta zbudowanej wersji systemu

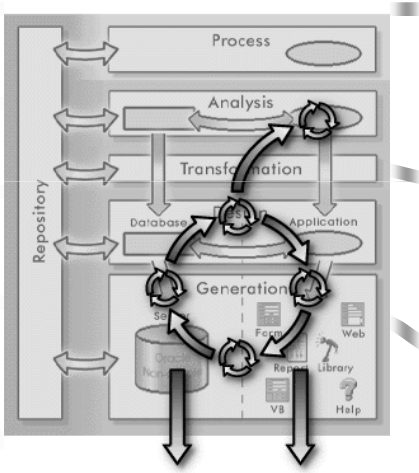


Model spiralny (2)

- zalety:
  - minimalizacja ryzyka – częste kontakty z klientem
  - łatwość wprowadzania poprawek w kolejnych wersjach
- wady:
  - wymaga większych nakładów organizacyjnych
  - może powodować przyłożenie mniejszej uwagi do identyfikacji wymagań i budowy modeli przez łatwość wprowadzanie poprawek w kolejnych wersjach
- model dobry dla projektów o dużej zmienności wymagań, gdzie konieczne są częste kontakty z klientem

Prototypowanie (1)

- RAD – ang. *Rapid Application Development*
- założenia:
  - szybkie tworzenie szkieletów aplikacji (prototypów) celem prezentacji ich użytkownikowi,
  - użytkownik: „nie wiem dokładnie, czego chcę, ale powiem, czy to jest to, gdy to zobaczę”



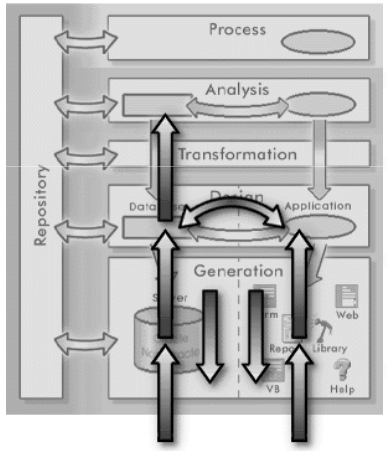
Prototypowanie (2)

- przebieg:
  - ogólne określenie wymagań
  - budowa prototypu
  - weryfikacja prototypu przez klienta
  - pełne określenie wymagań
  - realizacja pełnego modelu zgodnie z modelem kaskadowym
- prototyp nie jest elementem gotowego systemu!

Prototypowanie (3)

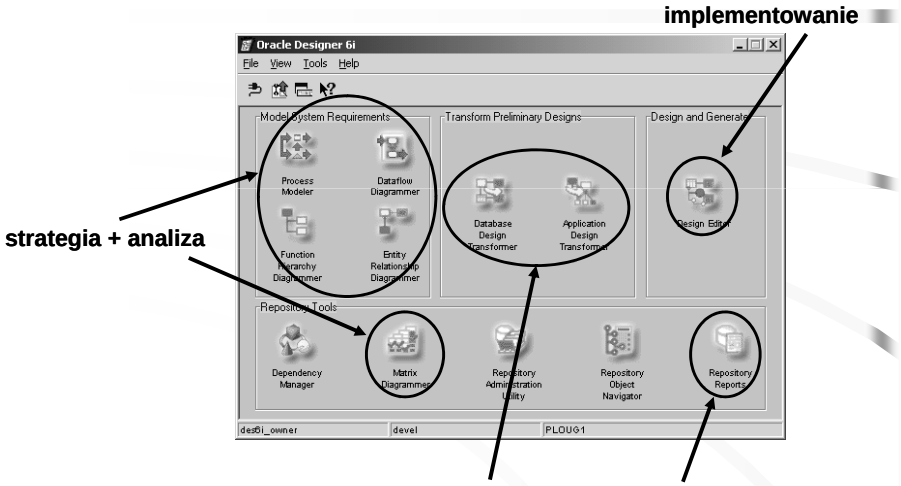
- zalety:
  - szybka demonstracja pracującej wersji systemu
  - dokładniejsze zaspokojenie potrzeb użytkowników,
  - możliwość przeprowadzenia szkoleń użytkowników przed zbudowaniem pełnego systemu
- wady:
  - dodatkowy koszt budowy prototypu,
  - zmniejszona stabilność działania systemu,
  - brak dokładnej analizy funkcjonalnej,
  - zdziwienie klienta: "Czemu muszę tak długo czekać? Przecież widziałem już działający program!"

Model DCD (Design Capture Driven)



- stosowany w przypadku istnienia systemów w organizacji,
- wykorzystuje mechanizmy reverse-engineering,
- pozwala na tworzenie nowych systemów korzystając ze starych definicji,
- pozwala realizować nowe potrzeby przedsiębiorstwa przy minimalnych nakładach czasowych i finansowych.

Narzędzia Oracle Designer



Referencje

- Richard Barker: CASE*Method: Function and Process Modelling. Addison-Wesley, 1992
- Włodzimierz Dąbrowski, Kazimierz Subieta: Podstawy inżynierii oprogramowania. Wydawnictwo PJWSTK 2005.
- Andrzej Jaszkiewicz: Inżynieria oprogramowania. Helion, 1997
- Edward Yourdon: Modern Strutured Analysis. Prentice Hall, 1989
- What is a CASE Environment? – Carnegie Melon Software Engineering Institute. http://www.sei.cmu.edu/legacy/case/case_whatis.html