Wydział Informatyki PB

Inżynieria oprogramowania
i analiza biznesowa

Wykład 6:

Projektowanie i implemetacja

Wdrażanie systemów informatycznych

Marek Krętowski
pokój 13a
e-mail: m.kretowski@pb.edu.pl
http://aragorn.pb.bialystok.pl/~mkret

Wersja 1.3 st. podyplomowe

Modelowanie i projektowanie

Modelowanie a projektowanie

Modelowanie (analiza)

- Celem analizy jest ustalenie wszystkich tych czynników lub warunków w dziedzinie przedmiotowej, w otoczeniu realizatorów projektu, w istniejących lub planowanych systemach komputerowych, które mogą wpłynąć na decyzje projektowe, na przebieg procesu projektowego i na realizację wymagań.
- Wynik: Logiczny model systemu, opisujący sposób realizacji przez system postawionych wymagań, lecz abstrahujący od szczegółów implementacyjnych; model staje się podstawą tworzenia projektu

Jak system ma działać?

Projektowanie

- W odróżnieniu od analizy dużą rolę odgrywa środowisko implementacji; projektanci muszą więc posiadać dobrą znajomość języków, bibliotek, i narzędzi stosowanych w trakcie implementacji
- Dążenie do tego, aby struktura projektu zachowała ogólną strukturę modelu stworzonego podczas analizy; niewielkie zmiany w dziedzinie problemu nie powinny powodować dużych zmian w projekcie
- Wynik: szczegółowy opis implementacji systemu

Jak system ma być zaimplementowany?

Cechy modelu analitycznego

- uproszczony opis systemu, zawierający wszystkie istotne cechy oprogramowania na wyższym poziomie abstrakcji
- hierarchicznie dekomponuje funkcje systemu
- opisany przy pomocy notacji zgodnej z pewną konwencją
- zbudowany przy użyciu dobrze rozpoznanych metod i narzędzi
- może być używany do wnioskowania o przyszłym oprogramowaniu

Nawet b. dobry model nie jest warunkiem wystarczającym do stworzenia dobrego projektu systemu (np. bez udziału doświadczonych i wnikliwych projektantów)

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

Karty CRC

- **C**lass-**R**esponsibility-**C**ollaborators = Klasa-Zobowiązanie-Współpraca
- Klasy przedstawiane są na kartach katalogowych (papierowe kartoniki) o określonych wymiarach (4x6 cali)
  - możliwość łatwego uzupełniania, układania, niszczenia, ...
- Karty wypełniane są podczas nieformalnego spotkania (burza mózgów) i następnie mogą być układane w celu modelowania interakcji
  - każdy uczesnik może odpowiadać za jedną lub więcej kart
- Na karcie poza nazwą klasy notowane są jej zobowiązania i współdziałania a nie atrybuty i metody
  - ew. atrybuty i metody na odwrotnej stronie karty
- **Zobowiązanie** - wysokopoziomowy opis zadań klasy (kilka punktów)
  - unikanie opisywania fragmentów danych i procesów
  - ograniczony rozmiar karty ma wymusić unikanie zbyt rozbudowanych klas i ewentualnie ich podział na mniejsze
- **Współdziałania** - odnoszą się do klas z którymi dana klasa ma współpracować
  - wysokopoziomowe pojęcie o związkach pomiędzy klasami

Nazwa klasy:	
Zobowiązania	Współdziałania

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
- konkretnym obiektem w pewnym odcinku czasu
- opisem tego obiektu (dokument, metadane)
- pewnym zbiorem konkretnych obiektów

W przypadku klasy „samochód” obiekt może reprezentować:

- konkretny egzemplarz samochodu (unikalny numer nadwozia i silnika) wyprodukowany przez określoną fabrykę
- model (rodzaj) samochodu oferowany klientom przez znany koncern
- pozycję w katalogu samochodów opisujący własności modelu
- historię stanów pewnego konkretnego samochodu

Dlaczego potrzebujemy architektury?

- **Zrozumienie systemu** - systemy oprogramowania są duże, skomplikowane i muszą spełniać sprzeczne wymagania; architektura zapewnia szkielet rozwiązania, który abstrahuje od szczegółów implementacyjnych, ale ustawia wzajemnie podstawowe elementy tak aby możliwe byłoby spełnienie wymagań
- **Organizowanie rozwoju oprogramowania** - pozwala odseparować różne części, tak aby ich tworzenie było możliwie niezależne; zależności pomiędzy częściami są jasno określone i punkty styku dokładnie wyspecyfikowane
- **Promowanie ponownego wykorzystania** - architektura pomaga w zidentyfikowaniu na możliwie wysokim poziomie analogicznych systemów krytycznych i podsystemów, dzięki czemu ponowne wykorzystanie nie ogranicza się do poziomu klas; wspólne podsystemy mogą być przystosowane na etapie rozwoju do późniejszego wyk.
- **Promowanie ciągłego rozwoju** - większość systemów musi reagować na nowe potrzeby i wymagania przed nimi stawiane; dobra architektura pozwala zapanować nad ewolucją systemu w czasie i kontrolować jej przebieg; zapoznanie się z architekturą pozwala na zrozumienie działania systemu przez przyszłych twórców i eliminuje potencjalne zagrożenia

Tradycyjne spojrzenie na architekturę

Proces projektowania architektonicznego zależy od wiedzy o zastosowaniu oraz umiejętności i intuicji architekta; wspólne czynności dla wszystkich procesów (czynności te zwykle nakładają się na siebie):

- **strukturalizacja systemu** - system jest dzielony na kilka podstawowych podsystemów (podsystem - niezależna jednostka oprogramowania, jej usługi nie zależą od usług oferowanych przez inne podsystemy); identyfikuje się komunikację między podsystemami
  - podział podsystemów na moduły (moduł jest zwykle komponentem systemu, który oferuje co najmniej jedną usługę innym modułom; zwykle nie jest niezależny)
- **modelowanie sterowania** - określa się ogólny model związków sterowania między częściami systemu; uzupełnia użyty model struktury

Standardowe modele struktury

- **Repozytorium** - wszystkie współdzielone dane są umieszczone w centralnej bazie danych, z której mogą korzystać wszystkie podsystemy; zwykle generowane przez jeden podsystem a wykorzystywane przez pozostałe
  - systemy sterowania i kontroli, systemy zarządzania informacjami, systemy CAD, zestawy narzędzi CASE
- **Klient-serwer** - model rozproszonego systemu, w którym dane i przetwarzanie są rozdzielone między zbiór procesorów (samodzielne serwery oferujące usługi + klienci korzystający z usług oferowanych przez serwery + sieć, dająca dostęp klientom do usług)
  - **cienki klient** - całość przetwarzania i zarządzania danymi na serwerze, zadaniem klienta jest uruchomienie oprogramowania prezentacyjnego
  - **gruby klient** - serwer odpowiada za zarządzanie danymi; oprogramowanie klienta implementuje logikę programu użytkowego i interfejs użytkownika
- maszyna abstrakcyjna (**warstwowy**) - opisuje sprzęganie podsystemów; układa system w ciąg warstw, z których każda oferuje pewne usługi

Modele sterowania

Sterowanie **scentralizowane** - jeden z podsystemów jest wybrany do roli sterownika i odpowiada za zarządzanie działaniem innych

- **model wywołanie-powrót** - sterowanie zaczyna się na wierzchołku hierarchii podprogramów i przez wywołania podprogramów przechodzi do niższych poziomów; jedynie systemy sekwencyjne
- **model menedżera** - stosuje się do systemów współbieżnych; jeden z komponentów systemu jest menedżerem i steruje rozpoczynaniem, zatrzymywaniem i koordynacją innych procesów systemu

Sterowanie **zdarzeniowe** - każdy podsystem musi reagować na zdarzenia zachodzące na zewnątrz

- **model rozgłaszania** - zdarzenie jest w zasadzie ogłoszeniem dla wszystkich podsystemów, każdy podsystem, który może obsłużyć to zdarzenie, reaguje na nie
- **modele z przerwaniami** - używane wyłącznie w systemach czasu rzeczywistego, gdzie zewnętrzne przerwania są wykrywane przez obsługę przerwań; następnie są one przekazywane do innego komponentu, który je przetworzy

Zadania wykonywane podczas projektowania

- Uszczegółowienie opracowanego modelu logicznego
  - projekt musi być wystarczająco szczegółowy, aby mógł być podstawą implementacji, przy czym stopień szczegółowości zależy od poziomu zaawansowania programistów (ale np. szczegółowy projekt jest dobrą dokumentacja techniczną)
- Projektowanie składowych systemów, które nie są związane z dziedziną problemu
- Optymalizacja systemu
- Dostosowanie do ograniczeń i możliwości środowiska implementacji
- Określenie fizycznej struktury systemu
  - określenie struktury kodu źródłowego, tj. wyróżnienie plików źródłowych, zależności pomiędzy nimi oraz rozmieszczenie składowych projektu w plikach źródłowych
  - podział systemu na poszczególne aplikacje
  - fizyczne rozmieszczenie danych i aplikacji na stacjach roboczych i serwerach

Uszczegółowienie modelu (1)

- Wybór sposobu implementacji konstrukcji, które mogą być zrealizowane na wiele sposobów (np. zgodnie ze standardami przyjętymi w firmie)
- Określenie poziomów widoczności atrybutów i dostępu do nich
- Podanie reguł odwzorowanie notacji w struktury konkretnego języka programowania (np. polegające na dokładnym określeniu typów danych)
- Jeżeli język implementacji narzuca ograniczenia, wówczas możliwe jest tworzenie np. nowych klas, które będą jedynie częścią implementacji (nie należą do modelu)
- Podanie nagłówków metod oraz ich parametrów
- Określenie, które z metod będą realizowane jako funkcje wirtualne a które jako zwyczajne funkcje
- Zastąpienie niektórych prostych metod bezpośrednim dostępem do atrybutów
- Zastąpienie niektórych atrybutów redundantnych przez odpowiednie metody, np. Wiek = BieżącaData - DataUrodzenia;

Uszczegółowienie modelu (2)

Określenie sposobów implementacji związków (asocjacji), gdyż zwykle można je zaimplementować na wiele sposobów, z reguły poprzez wprowadzenie dodatkowych atrybutów (pól):

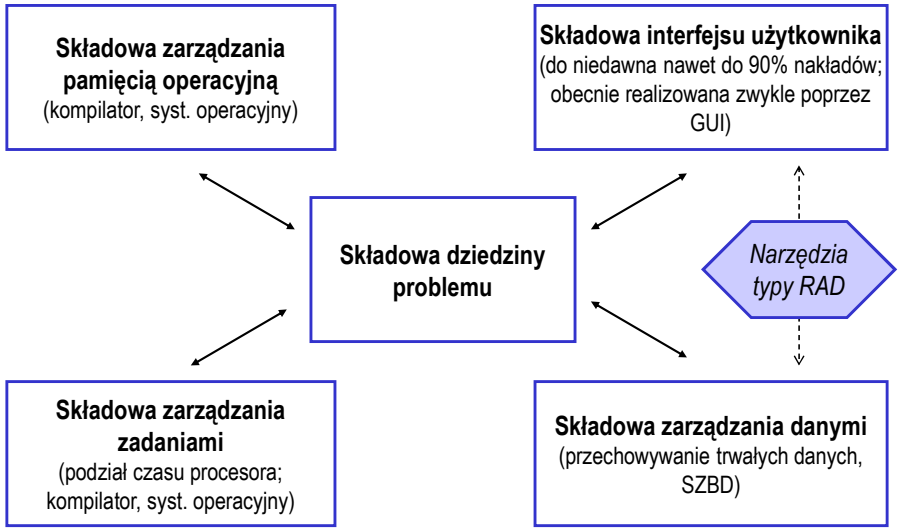
- obiekty powiązanej klasy
- wskaźniki (referencje) do obiektów powiązanej klasy
- identyfikatory obiektów powiązanej klasy
- klucze kandydujące obiektów powiązanej klasy

W zależności od przyjętego sposobu oraz od liczności związków (1:1, 1:n, n:1, m:n) możliwe są bardzo różne deklaracje w przyjętym języku programowania:

- tablica obiektów
- lista wskaźników
- tablica wskaźników

Określenie dodatkowych reguł przy transformacji schematów obiektowych na relacyjne

Dodatkowe składowe systemu
(niezwiązane z dziedziną problemu)



Interakcja z użytkownikiem

Za pomocą linii komend (w niektórych sytuacjach szybsza od interfejsu graficznego):

- nieduże systemy lub prototypy
- programy typowo obliczeniowe
- dla zaawansowanych użytkowników

Za pomocą plików skryptowych:

- celem automatyzacji powtarzanych wielokrotnie tych samych działań

Przez graficzny interfejs okienkowy:

- większe systemy
- doceniane zwłaszcza przez początkują-cych i średnio zaawansowanych użytkowników

Typowe sposoby wydawania przez użytkownika poleceń:

- wpisywanie poleceń w postaci tekstowej (np. w linii komend)
- wybór opcji z menu lub przycisku w dialogu
- korzystanie z ikon w paskach narzędziowych lub ze skrótów (kombinacje klawiszy)
- nawigacja przy użyciu urządzenia wskazującego np. myszy (kursor, przyciski, ...)
- dotykanie aktywnego ekranu
- głosem
- ...

Złote zasady projektowania interfejsu użytkownika (1)

- Spójność - wygląd oraz obsługa interfejsu powinna być podobna podczas korzystania z różnych funkcji; poszczególne programy tworzące system powinny mieć zbliżony interfejs, podobnie powinna wyglądać praca z rozmaitymi dialogami, podobnie powinny być interpretowane operacje wykonywane przy pomocy myszy; przykładowe reguły:
  - umieszczanie etykiet zawsze nad lub obok pól edycyjnych,
  - umieszczanie przycisków zawsze od dołu lub od prawej,
  - spójne tłumaczenie nazw angielskich, spójne oznaczenia pól
  - ...
- Skróty dla doświadczonych użytkowników - możliwość zastąpienia komend dostępnych przez menu przez kombinację klawiszy
- Potwierdzenie przyjęcia polecenia użytkownika - realizacja niektórych operacji może trwać stosunkowo długo; w takich sytuacjach, aby użytkownik nie był zdezorientowany, należy potwierdzić przyjęcie polecenie, oraz informować o przebiegu operacji (np. jakiś wskaźnik lub najlepiej czas do zakończenia)

Złote zasady projektowania interfejsu użytkownika (2)

- Zrozumiała terminologia komunikacji - nawet (zwłaszcza) w sytuacjach krytycznych oprogramowanie powinno posługiwać się terminologią zrozumiałą dla użytkownika
- Eliminacja prostych błędów edycji - oprogramowanie nie powinno dopuszczać do wprowadzenia niepoprawnych danych; jeżeli użytkownik wprowadzi dane, które dopiero po zweryfikowaniu można zakwalifikować jako błędne, wówczas należy zasygnalizować rodzaj błędu oraz umożliwić poprawienie wprowadzonych wartości lub umożliwić powrót do wartości startowych (poprzednich)
- Odwoływanie poprzednio wykonanych operacji - możliwość cofnięcia ostatnio wykonanej operacji lub cofnięcie się „dowolnie" daleko (w granicach rozsądku)
- Wrażenie kontroli nad systemem - nikt nie lubi, kiedy system sam zaczyna robić coś, czego użytkownik nie zainicjował, lub kiedy akcja systemu nie daje się przerwać; system nie powinien inicjować długich akcji (np. składowania) nie informując użytkownika co w tej chwili robi oraz powinien możliwie szybko reagować na sygnały przerwania akcji (Esc, Ctrl+C, Break,...)

Złote zasady projektowania interfejsu użytkownika (3)

- Dostępność powiązanych informacji - w momencie wypełniania konkretnej informacji, użytkownik powinien zawsze mieć możliwość podejrzenia (lub wyboru z listy)
- Nieobciążanie pamięci krótkotrwałej użytkownika - użytkownik może zapomnieć o tym po co i z jakimi danymi uruchomił dialog; system powinien w miarę możliwości wyświetlać stale te informacje, które są niezbędne do tego, aby użytkownik wiedział, co aktualnie się dzieje i w którym miejscu interfejsu się znajduje
- Grupowanie powiązanych operacji -jeżeli zadanie nie da się zamknąć w prostym dialogu lub oknie, wówczas trzeba je rozbić na szereg powiązanych dialogów; użytkownik powinien być prowadzony przez ten szereg, z możliwością łatwego powrotu do wcześnie wprowadzanych informacji

Reguła Millera 7(±2) - reguła psychologiczna określająca liczba obiektów nad którymi człowiek może jednocześnie pracować w sposób efektywny

- Ograniczenie powyższe można przełamywać poprzez grupowanie powiązanych ze sobą elementów w wyraźnie wydzielone zespoły

Optymalizacja projektu

Optymalizacja (rozumiana jako poprawa efektywności) może być dokonana na poziomie:

• projektu • implementacji

Kluczowe zagadnienia na etapie projekowania:

- Zmiana algorytmu przetwarzania - np. zmiana algorytmu sortującego poprzez wprowadzenie pośredniego pliku zawierającego tylko klucze i wskaźniki do sortowanych obiektów może przynieść nawet 100-krotny zysk.
- Wykrycie potencjalnych "wąskich gardeł" w przetwarzaniu i staranne ich zaprojektowanie; stosuje się tutaj twierdzenie, że 20% kodu jest wykonywane przez 80% czasu; możliwe jest również rozwiązanie polegające na zleceniu implementacji kluczowego przetwarzania na niższym poziomie
- Świadoma denormalizacja relacyjnej bazy danych - wprowadzenie nadmiarowości danych w celu zwiększenia efektywności (np. łączenie dwóch lub więcej tablic w jedną)
- Stosowanie struktur pomocniczych - indeksów, tablic wskaźników, ...
- Analiza mechanizmów buforowania danych w pamięci operacyjnej i ewentualna zmiana tego mechanizmu (np. zmniejszenie liczby poziomów)

Implementacja

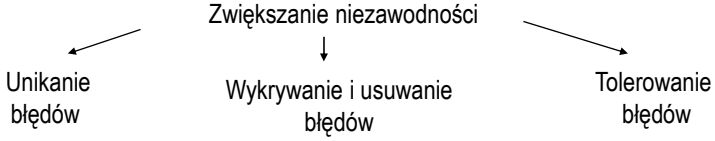
Ułatwienie implementacji

- Implementacja ulega znacznej automatyzacji wynikającej ze stosowania:
  - Języków wysokiego poziomu
  - Gotowych elementów
  - Narzędzi szybkiego wytwarzania aplikacji - RAD (ang. Rapid Application Development)
  - Generatorów kodu - są to najczęściej składowe narzędzi CASE, które na podstawie opisu projektu automatycznie tworzą kod programu; zwykle jest to szkielet programu, który następnie jest uzupełniany przez programistów; typowe elementy kodu:
    - skrypty tworzące relacje w bazie danych
    - definicje struktur danych
    - nagłówki procedur i funkcji
    - definicje klas
    - nagłówki metod

Niezawodność oprogramowania

Znaczenie niezawodności:

- Rosnące oczekiwania klientów wynikające m.in. z coraz większej powszechności wykorzystania oprogramowania oraz z wysokiej niezawodności sprzętu; w kwestiach niezawodności oprogramowanie znacznie ustępuje sprzętowi (być może wynika to z mniejszej powtarzalności oraz dużej złożoności oprogramowania)
- Potencjalnie duże koszty błędnych wykonań (wysokie straty finansowe wynikające z błędnego działania funkcji oprogramowania, nawet zagrożenie dla życia)
- Nieprzewidywalność efektów oraz trudność usunięcia błędów w oprogramowaniu
- Często pojawia się konieczność znalezienia kompromisu pomiędzy efektywnością i niezawodnością; teoretycznie łatwiej jest jednak pokonać problemy zbyt małej efektywności niż zbyt małej niezawodności.



Unikanie błędów

Można przyjąć, że stworzenie bardziej złożonego systemu, który nie zawiera błędów, jest praktycznie niemożliwe; trzeba natomiast starać się zmniejszać prawdopodobieństwo wystąpienia błędu dzięki:

- unikaniu niebezpiecznych technik (np. programowanie oparte na wskaźnikach)
- stosowaniu zasady ograniczonego dostępu (reguły zakresu, hermetyzacja, ...)
- językom z mocną kontrolą typów i kompilatorom sprawdzających zgodność typów
- wykorzystaniu języków o wyższym poziomie abstrakcji
- dokładnemu i konsekwentnemu specyfikowaniu interfejsów pomiędzy modułami oprogramowania
- zwróceniu szczególnej uwagi na sytuacje skrajne (puste zbiory, pętle z graniczną ilością obiegów, wartości zerowe, niezainicjowane zmienne, ...)
- wykorzystaniu gotowych komponentów (np. gotowych bibliotek procedur lub klas) z zastosowaniem zasady ograniczonego zaufania
- minimalizowaniu różnic pomiędzy modelem pojęciowym i modelem implementacyjnym
- ...

Właściwy styl programowania
("złote myśli")

- Używaj znaczących oraz unikaj podobnych nazw zmiennych
- Nie używaj tych samych zmiennych do różnych celów
- Dla jednoznaczności używaj nawiasów
- Pisz tylko jedną instrukcję kodu w wierszu i używaj wcięć
- Ucz się i używaj prostych cech języka
- Ucz się i wykorzystuj dostępne funkcje biblioteczne i standardowe
- Dopóki program nie jest poprawny nie myśl o jego efektywności
- Nie poświęcaj czytelności kodu dla (często pozornych) zysków w efektywności
- Unikaj trików językowych
- Nigdy nie ulepszaj programu jeśli nie musisz
- Komentuj to co istotne, ale unikaj zbędnych komentarzy
- Komentując odpowiadaj na pytania czytelnika
- Komentuj wszystkie niebanalne zmienne (pola)
- ...

Techniki z natury niebezpieczne (1)

- Instrukcje typu *goto* (skocz do) oraz wykorzystanie etykiet prowadzą zwykle do programów, których działanie jest trudne do zrozumienia i prześledzenia
- Stosowanie liczb ze zmiennym przecinkiem, których dokładność jest ograniczona i może być przyczyną nieoczekiwanych błędów, najczęstsze błędy pojawiają się podczas porównań
- Wskaźniki i arytmetyka wskaźników: dają możliwość dowolnej penetracji całej pamięci operacyjnej i w przypadku popełnienia błędu prowadzić mogą do zupełnie nieoczekiwanych (losowych) zachowań, które ciężko jest diagnozować
- Obliczenia równoległe: mogą prowadzić do złożonych zależności czasowych i tzw. *pogoni* (wynik zależy od tego, który z procesów szybciej dojdzie do pewnego punktu w obliczeniach); z uwagi m.in. na niedeterminizm zachowania bardzo trudne do testowania; np. *wątki* są określane jako *zatrute jabłko*
- Przerwania i wyjątki - wprowadzają również pewien rodzaj równoległości; dodatkowo, ryzyko przerwania operacji krytycznych czasowo

Techniki z natury niebezpieczne (2)

- Rekurencja - może prowadzić do krytycznego zapętlenia albo przepełnienia stosu wywołań; utrudnia śledzenie programu,
- Procedury i funkcje, które realizują wyraźnie odmienne zadania w zależności od parametrów lub stanu zewnętrznych zmiennych,
- Dynamiczna alokacja pamięci - stosowana bez zapewnienia mechanizmu zwalniana pamięciu już niewykorzystywanej,
- Zmienne globalne,
- Niewyspecyfikowane, nieoczekiwane efekty uboczne funkcji i procedur,
- Przeciążanie operatorów - problemy z kolejnością wywołań operatorów

Stosowanie powyższych technik wymaga zachowania dużej ostrożności, ale w niektórych sytuacjach może być przydatne lub wręcz niezbędne

Dokumentowanie

Dokumentacja

Producenta
(opisuje szczegółowo proces wytwórczy i rozwiązania wykorzystane do stworzenia systemu)

- Kierownictwo
- Zespół projektowy (projektanci, programiści)

Użytkowa
(opisuje system jako produkt, ukierunkowana na ułatwienie korzystania z oprogramowania)

- Użytkownicy systemu
- Administratorzy

- Dokumentacja jest integralną częścią systemu i powinna powstawać równolegle z procesem wytwórczym oprogramowania
- Często jakość dokumentacji jest dobrym miernikiem stopnia dojrzałości procesu tworzenia oprogramowania w organizacji
- Opracowanie dobrej dokumentacji pozwala uniknąć wielu problemów na etapie wdrożenia i następnie rzeczywistego wykorzystania systemu

Dokumentacja producenta

W trakcie trwania przedsięwzięcia powstają następujące dokumenty:

- dokumentacja procesu produkcji oprogramowania
- dokumentacja techniczna opisująca wytworzony produkt

Dokumentacja procesu obejmuje:

- **Plany, szacunki, harmonogramy** - dokumenty tworzone przez kierownictwo przedsięwzięcia jako propozycje; odbiorcami ich są przełożeni wyższego szczebla; po zaakceptowaniu dokumenty tego typu pełnia rolę poleceń dla wykonawców
- **Raporty** - dokumenty (przygotowywane przez kierowników dla przełożonych) opisujące przebieg i rezultaty prac
- **Standardy** - dokumenty opisujące pożądany sposób realizacji
- **Dokumenty robocze** - rozmaite dokumenty zawierające propozycje rozwiązań tworzone przez członków zespołu; zaakceptowane mogą stać się standardami, raportami, planami, ...
- **Komunikaty** - rozmaite, z reguły krótkie dokumenty służące do wymiany informacji pomiędzy członkami zespołu

Dokumentacja techniczna

- Zawiera dokładny opis systemu (projekt + kod + testy + oszacowanie niezawodności...); przeznaczona jest zwykle wyłącznie dla producenta
- Dokumentacja techniczna przed oddaniem oprogramowania do eksploatacji powinna być poddana weryfikacji celem wyeliminowania błędów i nieścisłości

Istotne jest wypracowanie w firmie standardów dokumentacji technicznej:

- procesów wytwarzania dokumentacji: tworzenia wstępnej wersji dokumentów, wygładzania, drukowania, powielania, oprawiania, wprowadzania zmian w istniejących dokumentach, zatwierdzania; konieczne jest ścisłe określenie odpowiedzialnych za to osób
- treści i formy dokumentów: strona tytułowa, spis treści, budowa rozdziałów, podrozdziałów i sekcji, indeks, słownik
- sposobu dostępu do dokumentacji: niezbędne jest stworzenie rodzaju biblioteki dokumentów technicznych, z zapewnieniem sprawnego dostępu do dowolnego dokumentu

Dokumentacja użytkownika

- **Opis funkcjonalny** - wstępna część dokumentacji opisująca w sposób zwarty przeznaczenie i główne możliwości systemu; może być wykorzystywany do pierwszego zapoznania się z oprogramowaniem
- **Podręcznik użytkownika** - przeznaczony głównie dla początkujących użytkowników; powinien zawierać podstawowe informacje niezbędne do korzystania z systemu (najlepiej na podstwie przykładów) oraz odnośniki do bardziej szczegółowych informacji
- **Kompletny opis systemu** - przeznaczony głównie dla doświadczonych użytkowników; opisujący całą funkcjonalność oraz ograniczenia systemu, zawierający opisy formatów danych, możliwych błędów, ...
- **Instrukcja instalacji** - przeznaczona głównie dla administratora, zawiera procedurę instalacyjną
- **Podręcznik administratora** - opisuje możliwości zmian konfiguracji dostępne bez ingerencji twórców systemu oraz sposoby udostępniania systemu użytkownikom końcowym; tworzenie kopii zapasowych, ...

Jakość dokumentacji

Czynniki wpływające na postrzeganie i akceptację dokumentacji przez użytkowników:

- czytelna struktura dokumentu
- zachowywanie standardów
- sposób przekazu informacji:
  - stosowanie formy aktywnej i zwracanie się bezpośrednio do użytkownika (dyskusyjne w odniesieniu do dokumentacji w j. polskim)
  - bezwzględna poprawność gramatyczna i ortograficzna
  - zwięzła forma (krótkie zdania zawierające pojedyncze fakty; rozsądnej długości akapity)
  - jasność wypowiedzi, eliminacja dwuznaczności (w tekstach technicznych można mniej rygorystycznie przestrzegać np. unikania powtórzeń)
  - precyzyjne definicje używanych terminów (najlepiej zebrane dodatkowo w słowniczku)
  - powtarzanie trudnych opisów (w przypadku kluczowych kwestii dopuszczalne jest powtórzenie opisu w kilku miejscach)
  - stosowanie tytułów (podtytułów) sekcji, wyliczeń i wyróżnień
  - zrozumiałe odwołania do innych części dokumentacji

Instalacja i wdrożenia

Instalacja systemu

Szczególnie istotna w przypadku oprogramowania wykonywaniego na konkretne zamówienie; w jej trakcie następuje przekazanie systemu klientowi, który w momencie zakończenia instalacji staje się właścicielem systemu

Podstawowe czynności podczas instalacji systemu:

- przygotowanie środowiska pracy: dostarczenie nowego lub wykorzystanie istniejącego sprzętu (serwery, stacje, sieć, drukarki, ...); odpowiednie skonfigurowanie systemów operacyjnych; instalacja i/lub dostrojenie systemów bazodanowych
- przeniesienie stworzonych aplikacji (instalacje na serwerach i stacjach)
- dostosowanie oprogramowania do wymogów konkretnego stanowiska
- inicjalizacja baz danych (zdefiniowanie reguł postępowania, wypełnianie słowników i kartotek, przeniesienie - import danych z funkcjonujących systemów,...)
- zdefiniowanie użytkowników (ew. grup lub ról), przydzielenie im uprawnień i weryfikacja dostępu do odpowiednich części systemu
- sprawdzenie możliwości wykorzystania przynajmniej podstawowej funkcjonalności systemu

Problemy podczas instalacji

- **Wypełnienie startowe bazy danych** jest często bardzo żmudnym procesem, wymagającym wprowadzenia danych z różnych nośników oraz uzgodnienia ich poprawności i spójności. Niekiedy część danych może być dostępna w formie elektronicznej (np. w postaci zapisów poprzedniego systemu), ale wymaga to przygotowania specjalnych programów konwersji (możliwe to jest w zasadzie tylko pod warunkiem znajomości struktury bazy)
- Ważne jest **planowanie i harmonogramowanie prac** - pojawia się szereg problemów, np. konieczność usunięcia błędów i wprowadzenia modyfikacji; należy dożyć do zarezerwowania odpowiedniego czasu na prace związane z instalacją pamiętając przy tym, że użytkownicy nie mogą zaniechać wykonywania przez nich bieżących prac
- **Opór użytkowników przed zmianą sposobu pracy** - często użytkownicy końcowi systemu są to osoby mniej zaawansowane i na starcie nieufne do nowego systemu; zwykle nie uczesniczyli bowiem w poprzednich pracach prowadzących do powstania systemu; bardzo ważne jest uzyskanie ich akceptacji

Szkolenia i wdrożenie

- Część wstępna szkoleń (najczęściej w postaci seminarium, bez bezpośredniej pracy użytkowników z systemem) może być przeprowadzona jeszcze przed ukończeniem systemu
- Szkolenia muszą być dostosowane do poziomu użytkowników (np. odzielnie dla użytkowników końcowych i administratorów systemu) i powinny być organizowane w taki sposób, aby osoby uczestniczące były zainteresowane całością zagadnienia
- Zaleca się, aby szkolenia przeprowadzały (lub przynajmniej je rozpoczynały) osoby, które były zaangażowane w prowadzenie przedsięwzięcia; jest im łatwiej nawiązać kontakt z przyszłymi użytkownikami
- Zwykle stosowane jest tzw. nadzorowane korzystanie z systemu (użytkownicy pracują w nowym systemie pod kontrolą wdrożeniowca), często równoległe z tradycyjnym sposobem pracy, po którym następuje weryfikacja rezultatów
- Usuwanie błędów w oprogramowaniu i dokumentacji użytkowej