PROJEKTOWANIE

określenie wymagań | **specyfiko-wanie** | **projektowanie** | **kodowanie implementacja** | **testowanie** | **produkt konserwacja**

Faza strategiczna

Analiza

Dokumentacja

Instalacja

PROJEKT – most pomiędzy specyfikowaniem a kodowaniem

- Cele projektowania (jak program będzie działał)

  - Algorytm podstawowy („jak” można to zrobić – nie ma reguł, łatwy do przekształcenia w efektywny, możliwie bezawaryjny, modyfikowalny program – iteracja wokół „istoty” problemu)

  - Dekompozycja („jak” rozwiązania problemu dzieli się na podproblemy..... – proces twórczy, przekształcanie zalążka koncepcji rozwiązania problemu – „stopniowe doskonalenie” (projektowanie od góry do dołu))

OGÓLNE ZASADY PROJEKTOWANIA

PROJEKT – most pomiędzy specyfikowaniem a kodowaniem

TREŚĆ PROJEKTU – algorytmy i struktury rozwiązujące problem

„Algorytmy + Struktury danych = Programy” Nicklaus Wirth (projekt nie widoczny)

- Podejście iteracyjne (nie próbujemy aż do skutku, lecz „wzmacniamy” działanie systemu, każdy kolejny projekt (wzmocnienie) powinien móc być zrealizowany i testowany tak jakby był końcowym systemem)
- Stopniowe doskonalenie (zapis dekompozycji na komponenty z opisem interfejsów i komunikacyjnych struktur danych, gdzie dla każdego komponentu określa się funkcjonalność razem z elementami testów) – parafraza strategii dziel i zwyciężaj.„
- **Ukrywanie informacji** (Lokalizowanie informacji wykorzystywanych przez wiele modułów w jednym miejscu (np. obsługa urządzeń zewnętrznych poza odwołaniami do drukowania)
- **Uzupełnianie planu testów** (każde rozwiązanie sugeruje test!!)

SZTUKA PROJEKTANTA

Sztuka projektowania (w praktyce) **– sztuka przekształcania wymagań lub specyfikacji w kod programu lub coś zbliżonego do kodu.**

- Przejście od wymagań do projektowania
  - forma wynika z funkcji
  - należy podejmować jedynie decyzje konieczne
- **Od wymagań do projektu:**
  - rozpoznanie każdego wymagania
  - wyszukanie w wymaganiach, zbioru wspólnych, koniecznych „funkcji” niezbędnych dla działania systemu (coś musi być zrobione – jeśli warunek to odpowiedź (dobry model na funkcję) – potrzebna jednak gdy częste odwołania do warunku)
  - skatalogowanie głównych, potrzebnych struktur danych (trwał i tymczasowe)
  - określenie unikatowych lub ważnych wymaganych algorytmów (często na początku proste przetwarzanie wyrasta ze struktur danych i funkcji)

Projektowanie podsystemów

Projektowanie wysokiego poziomu - uzupełnienie dekompozycji- wynalezienie głównych podsystemów aplikacji, opisanie interfejsów między nimi oraz do interfejsu pomiędzy systemem a otoczeniem.

- **Projektowanie głównych podsystemów**

  - z których każdy może być miniaturowym systemem następnie udoskonalanym podczas szczegółowego projektowania

  - jedyna wskazówka dla tworzenia podsystemów – doświadczenie w danej dziedzinie oraz z konstrukcji poprzednich systemów

- **Zastosowanie poprzednio zaprojektowanych systemów:**

  - oszczędza ogrom pracy
  - zabezpiecza przed błędami (były testowane)
  - jest ich ogromna ilość:
    - podsystemy we-wy i sterowniki urządzeń
    - biblioteki matematyczne, graficzne
    - systemy zarządzania baza danych

Projektowanie podsystemów (2)

Macierze śladu - pozwalają na sprawdzenie realizacji wszystkich wymagań poprzez określone podsystemy

Wymagania	Podsystemy
.....	.....
[1.5.1]	We/wy, Obsługa sesji, Matem
[1.5.2]	We/wy, Obsługa sesji, Matem, Obsługa błędów
[1.5.3]	We/wy, Obsługa sesji, Nadzór reaktora, Obsługa błędów
.....	.....
[2.3.1]	File, Obsługa błędów
[2.3.2]	Matem, Nadzór reaktora

..... (macierz odwrócona – definicja podsystemów)

Podsystem nadzoru reaktora	Podsystem We/wy
[1.5.4]	[1.5.1], [1.5.2]
[2.3.2]	[1.5.3], [1.5.4]

Projektowanie podsystemów (3)

Interfejsy podsystemów - komunikacja podsystemów (podsystemy izolowane)

Zasady:

Jedna wymagana funkcja, jeden interfejs (minimalizacja liczby sposobów wywoływania podsystemu, jasność interfejsu *(equivalence))*

Ustalenie kto będzie wywoływał podsystem i dlaczego (dane dla podsystemu i zwracane rezultaty

Zapisz ograniczenia użycia interfejsu (np. kolejność i zależność danych podsystemu – jeżeli to możliwe należy natychmiast uniemożliwić naruszenie tych ograniczeń

Utrzymuj złożone, trudne do zmiany struktury danych, poza interfejsami

Interfejsy ze środowiskiem - z systemami operacyjnymi i bibliotekami mogą nie być traktowane jako część projektu. Co innego interfejs systemu z użytkownikiem – ma swoje wymagania.

Projektowanie szczegółowe

Projektowanie szczegółowe - dostarcza informacji dla implementacji, kodowania

Projekt szczegółowy identyfikuje i definiuje „moduły” (jednostki) implementacji programowania na tyle szczegółowe aby mogły być kodowane i testowane.

Komponent (moduł) może być definiowany jako porcja kodu

- realizowana w postaci jednego podprogramu
- spełniająca wymagania logicznej lub funkcjonalnej części podsystemu
- niezależna od reszty systemu, z wyjątkiem zdefiniowanych interfejsów

Projektowanie szczegółowe(2)

Dekompozycja podsystemów na moduły - podobna do dekompozycji na etapie projektowania wysokiego poziomu.

1. Określenie głównych modułów podsystemu
2. Definicje interfejsów pomiędzy modułami
3. Określenie funkcji do wykonania przez każdy moduł

Projektowanie głównych modułów - wykorzystanie odwróconej macierzy śladu.

Zasady: **niezależność modułów,**

minimalna liczba modułów,
połącz każdy moduł z podsystemem który z kolei jest połączony z wymaganiami

[1.8.1] System będzie wyznaczał poziom promieniowania co 200 milisekund. Jeśli poziom promieniowania przekroczy wartość krytyczną, to system uruchomi alarm

-Wyznacza poziom promieniowania

-Przekracza wartość krytyczną

-Uruchamia alarm

(kombinacje czasownik-rzeczownik mogą wyznaczać moduły)

Projektowanie szczegółowe(3)

Definiowanie interfejsów modułów - określenie danych przekazywanych do i z modułu

Określenie funkcji modułu - krótki opis uzupełniony formalna notacja projektową.

Obsługa błędów – aspekt funkcjonalności każdego modułu nie do zaniedbania:

Błędy zakresu danych

Błędy systemowe (nie powiodło się wywołanie np. biblioteki)

Błędy wewnętrzne – uszkodzenie struktury danych

Zasada: jeżeli jest cień wątpliwości, czy dla danego modułu jest potrzebna obsługa błędów to jest ona NA PEWNO potrzebna

Projektowanie szczegółowe(4)

Dokumentacja modułu - ostatni krok projektowania szczegółowego niezbędna dla realizacji podsystemu przez programistów

Dokumentacja modułu powinna zawierać:

- **Nazwę modułu i opis jego parametrów (nagłówek podprogramu)**
- **Nazwę pliku zawierającego dany moduł**
- **Opis globalnych lub zewnętrznych wejść i wyjść dla modułu, poza jego parametrami**
- **Opis funkcjonalny (specyfikacja) modułu, łącznie z obsługą błędów**
- **Dodatkowe założenia do sprawdzenia przy wywoływaniu modułu**

Zasada: Każda firma produkująca oprogramowanie posiada standardy dokumentacji modułów.

TESTOWANIE

(optymistycznie) Testowanie oprogramowania ma na celu sprawdzenie, czy oprogramowanie spełnia wymagania.

(pesymistycznie) Celem testowania oprogramowania jest znalezienie błędów.

Testowanie oprogramowania powinno być realizowane zgodnie z przyjętym planem testów.

Dwa aspekty błędów:

awaria- oprogramowanie źle realizuje wymagania

usterka – błąd w kodzie programu powodujący awarię

Zasada IEEE: Celem testowania oprogramowania jest takie wyszukiwanie awarii, że gdy oprogramowanie ulegnie awarii podczas testów, usterki odpowiedzialne za te awarie będą mogły być znalezione i poprawione.

TESTOWANIE(2)

Testowanie jednostek a testowanie systemu

Gdy dany moduł jest zakończony może być testowany na poziomie jednostki przez programistę (cykl test-poprawianie).

Moduły są łączone w podsystemy i następnie w cały system który jest testowany na poziomie systemu

Testowanie jednostek jest efektem dekompozycji problemu i nie wykrywa błędów w interfejsach pomiędzy nimi.

Samodokumentowanie się kodu procedur bez ich funkcjonalnej specyfikacji często prowadzi do błędów (kod jest swoją własną definicją – to nie może być oceniany)

TESTOWANIE(3)

Testowanie jednostek a testowanie systemu

Testowanie jednostek systemu wymaga wprowadzenia tzw. *namiastek* będącymi pozornymi procedurami uruchamiającymi testowaną jednostkę. Są to oczywiście warunki odmienne od występujących w rzeczywistej pracy systemu. Najczęściej namiastki są interaktywnymi interfejsami pomiędzy testowaną jednostką a testującym ją testerem (człowiekiem).

Często stanowią również jednostki tzw. puste nie robiące nic, a jedynie zwracające sterowanie do jednostki testowanej

Uprzęże testowe, służą automatyzacji testowania jednostek poprzez automatyczne programy testujące np. typy zmiennych

TESTOWANIE(4)

Testowanie jednostek a testowanie systemu

Testowanie podsystemów - ukryte informacje w danych wewnętrznych prowadzi do losowego zachowania systemu – wprowadzenie dodatkowych funkcji drukowania czytelnej dla testera wersji struktur wewnętrznych (polepszanie testowalności kodu)

Testowanie kodu obiektowego – metody klasy obiektowej są podprogramami dostępu i również tutaj powstaje problem badania danych ukrytych. Pomaga tutaj dziedziczenie – jeżeli metoda nadklasy dziedziczona przez podklasy była testowana jednostkowo to nie ma potrzeby testowania jej w każdej podklasie.

Testowanie ciągów wejść do podsystemów – generowanie stanów które powinny być obsłużone przez system – eliminacja ślepych zaułków.

TESTOWANIE(5)

Testowanie systemu

Testy systemowe - ich efekt będzie widoczny dla użytkowników. Inne działania testowe nie będą widoczne dla użytkowników.

Testowanie integracyjne – przyrostowa metoda integracji systemu, dokładanie kolejnych podsystemów. Testowanie z „góry” i z „dołu”.

Tak naprawdę dopiero w testowaniu systemowym widać spełnianie wymagań i realizację opracowanych specyfikacji.

Inspekcja kodu a testowanie jednostki lub podsystemu mają doprowadzić do znalezienia usterki kodzie jednostki.