

TEORETYCZNE PODSTAWY INFORMATYKI

22/01/2018

WFAiS UJ, Informatyka Stosowana
I rok studiów, I stopień

Wykład 14d

2

Metody programistyczne

- Wyidealizowany scenariusz tworzenia oprogramowania
- Metody programistyczne:
 - ▣ Projektowanie zstępujące
 - ▣ Projektowanie wstępujące
- Weryfikacja poprawności
 - ▣ Specyfikacja
 - ▣ Niezmienniki pętli
- Problem „STOPu”

Tworzenie oprogramowania: wyidealizowany scenariusz

3

1. **Definicja problemu i specyfikacja**
Analiza wymagań użytkownika (często nieprecyzyjne i trudne do zapisania),
budowa prostego prototypu lub modelu systemu
2. **Projekt**
Wyróżniamy najważniejsze komponenty, specyfikujemy wymagania
związane z wydajnością systemu, szczegółowe specyfikacje niektórych
komponentów
3. **Implementacja**
Każdy implementowany komponent poddajemy serii testów
4. **Integracja i testowanie systemu**
5. **Instalacja i testowanie przez użytkowników**
6. **Konserwacja**
Niezwykle istotna jakość stylu programowania (w wielu wypadkach to
ponad 50% nakładów poniesionych na napisanie systemu)

Metody programistyczne

4

- Przy rozwiązywaniu prostych zadań **podejście „ad hoc”** może dawać szybsze rezultaty. Jednak, gdy mamy do czynienia z bardziej złożonymi problemami efektywniejsze jest **podejście systematyczne**
- **Projektowanie zstępujące (top-down design)**
 - Rozpoczynamy od zdefiniowania problemu który chcemy rozwiązać
 - Problem dzielimy na główne kroki – pod-problemy
 - Pod-problemy są dzielone na drobniejsze kroki tak długo, aż rozwiązania drobnych pod-problemów stają się łatwe nazywamy to stopniowym uszczegółowianiem
 - W idealnej sytuacji pod-problemy mogą być rozwiązywane niezależnie, a ich rozwiązania łączone w celu otrzymania rozwiązania całego problemu. W odniesieniu do tworzenia kodu oznacza to, że poszczególne kroki powinny być kodowane niezależnie, przy czym dane wyjściowe jednego kroku są używane jako dane wejściowe do innego
- **Realnie**
 - celem projektowania zstępującego jest **zminimalizowanie tych współzależności**

Metody programistyczne

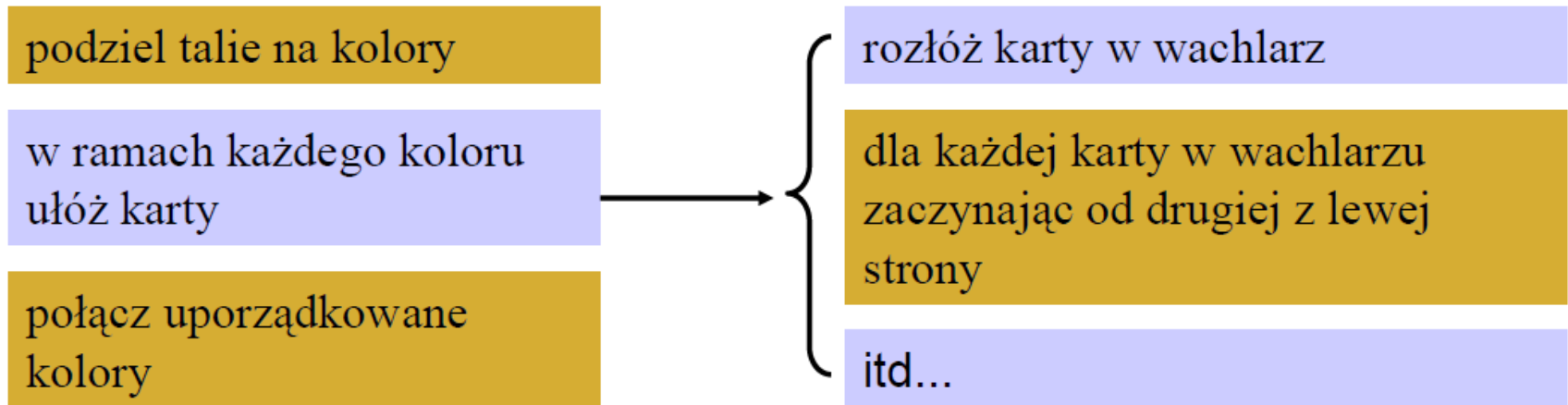
5

- Zalety projektowania zstępującego
 - Jest to systematyczna metoda rozwiązywania problemów
 - rozwiązanie jest **modularne**, poszczególne kroki mogą być uruchamiane, modyfikowane i ulepszone niezależnie od pozostałych, rozwiązanie składa się z klarownych fragmentów które można niezależnie zrozumieć
 - dobrze zaprojektowane fragmenty mogą być **ponownie zastosowane** w innych zadaniach
 - można rozpoznać wspólne problemy na samym początku i **uniknąć wielokrotnego ich rozwiązywania**

Metody programistyczne

6

Przykład: Uporządkuj talie kart:



Metody programistyczne

7

□ Projektowanie wstępujące (bottom-up design)

- Polega na wyjściu od samego języka i wzbogacaniu go nowymi operacjami, dopóki nie będzie można wyrazić rozwiązania problemu w rozszerzonym języku
- Każde oprogramowanie działające na maszynie cyfrowej rozszerza jej możliwości o nowe funkcje. W efekcie, uruchomienie programu na komputerze tworzy nową maszynę, która nie wykonuje swoich operacji bezpośrednio, ale zmienia je na prostsze wykonywalne przez sprzęt
- Nową maszynę nazywamy **maszyną wirtualną**, ponieważ istnieje w świecie abstrakcyjnym a nie fizycznym
- To samo dotyczy sytuacji gdy dodajemy nową operację do języka programowania, pisząc funkcje. Zwiększa ona jego funkcjonalność tworząc nowy, silniejszy język. O zbiorze nowych funkcji można myśleć jako o wirtualnej maszynie zbudowanej na bazie starego języka.

Metody programistyczne

8

- Przykład: Policz wyrażenie $(3/28 + 2/7) * 4/3 - 1$
 - Aby to wykonać należałoby wzbogacić istniejący język przez dodanie
 - funkcji implementujących działania $+$, $-$, $*$, $/$ na ułamkach,
 - funkcji wczytywania i wypisywania ułamków,
 - ten pakiet funkcji mógłby też zawierać funkcję skracającą ułamki, chociaż nasz program nie musiałby z niej korzystać.
 - Zazwyczaj programy projektuje się **łącząc metodę wstępującą i zstępującą**
 - zaczynamy od podzielenia problemu na pod-zadania,
 - przekonujemy się, że byłby przydatny określony pakiet funkcji,
 - rozstrzygamy, jakie funkcje wejdą w skład pakietu i rozszerzą język,
 - powtarzamy to iteracyjnie, dzieląc problemy na prostsze i wzbogacając tym samym język.... dopóki rozwiązania wszystkich problemów składowych nie dadzą się zapisać bezpośrednio w rozszerzonym języku.

Metody programistyczne

9

- Abstrakcyjne typy danych (ATD)
 - Jest to istotne ulepszenie metody projektowania programów.
 - Podstawą metody jest **oddzielenie operacji wykonywanych** na danych i sposobów ich przechowywania **od konkretnego typu danych**. Pozwala ona na podzielenie zadania programistycznego na dwie części:
 - wybór struktury danych reprezentującej ATD i napisanie funkcji implementujących operacje
 - napisanie „programu głównego” który będzie wywoływał funkcje ATD
- Program ma dostęp do danych ATD wyłącznie przez wywołanie funkcji; nie może bezpośrednio odczytywać ani modyfikować wartości przechowywanych w wewnętrznych strukturach ATD.

Veryfikacja poprawności programu

10

- Sprawdzenie częściowej poprawności.
 - Weryfikacja opiera się o sprawdzenie specyfikacji, która jest czymś niezależnym od kodu programu.
 - Specyfikacja wyraża, co program ma robić, i określa związek między jego danymi wejściowymi i wyjściowymi.
 - W specyfikacji definiujemy warunek dotyczący danych wejściowych, który musi być spełniony na początku programu, tzw. **warunek wstępny**. Jeżeli dane nie spełniają tego warunku, to nie ma gwarancji że program będzie działał poprawnie ani nawet że się zatrzyma.
 - W specyfikacji definiujemy również **warunek końcowy**, czyli co oblicza program przy założeniu że się zatrzyma. Warunek końcowy jest zawsze prawdziwy jeżeli zachodzi warunek wstępny.

Veryfikacja poprawności programu

11

- Aby wykazać, że program jest zgodny ze specyfikacją, trzeba podzielić ją na wiele kroków, podobnie jak dzieli się na kroki sam program. Każdy krok programu ma swój warunek wstępny i warunek końcowy.
- Przykład: prosta instrukcja przypisania
 - $x=v$, x – zmienna, v – wyrażenie
 - warunek wstępny: $x \geq 0$
 - wyrażenie: $x = x+1$
 - warunek końcowy: $x \geq 1$
- Dowodzimy poprawności specyfikacji przez podstawienie wyrażenia $x+1$ pod każde wystąpienie x w warunku końcowym.
- Otrzymujemy formułę $x+1 \geq 1$, która wynika z warunku wstępnego $x \geq 0$.
- Zatem specyfikacja tej instrukcji podstawienia jest poprawna.

Weryfikacja poprawności programu

12

□ Istnieją **cztery sposoby** łączenia prostych kroków w celu otrzymania kroków bardziej złożonych.

1. stosowanie instrukcji złożonych (bloków instrukcji wykonywanych sekwencyjnie)
2. stosowanie instrukcji wyboru
3. stosowanie instrukcji powtarzania (pętli)
4. wywołania funkcji

Weryfikacja poprawności programu

13

- Każdej z tych metod budowania kodu odpowiada metoda przekształcania warunku wstępnego i końcowego w celu wykazania poprawności specyfikacji kroku złożonego:
 - Aby wykazać prawdziwość specyfikacji instrukcji wyboru, należy dowieść, że warunek końcowy wynika z warunku wstępnego i warunku testu w każdym z przypadków instrukcji wyboru.
 - Funkcje też mają swoje warunki wstępne i warunki końcowe; musimy sprawdzić że warunek wstępny funkcji wynika z warunku wstępnego kroku wywołania, a warunek końcowy pociąga za sobą warunek końcowy kroku wywołania.
 - Wykazanie poprawności programowania dla pętli wymaga użycia **niezmiennika pętli**.

Niezmienniki pętli

14

- Niezmiennik pętli określa warunki jakie muszą być zawsze spełnione przez zmienne w pętli, a także przez wartości wczytane lub wypisane (jeżeli takie operacje zawiera).
- Warunki te muszą być prawdziwe przed pierwszym wykonaniem pętli oraz po każdym jej obrocie.
(mogą stać się chwilowo fałszywe w trakcie wykonywania wnętrza pętli, ale muszą ponownie stać się prawdziwe przy końcu każdej iteracji)
- Dowodzimy że pewne zdanie jest niezmiennikiem pętli wykorzystując **zasadę indukcji matematycznej**.
- Mówi ona że:
 - jeśli (1) pewne zdanie jest prawdziwe dla $n = 0$
 - oraz (2) z tego, że jest prawdziwe dla pewnej liczby $n \geq 0$ wynika że musi być prawdziwe dla liczby $n+1$,
 - to (3) zdanie to jest prawdziwe dla wszystkich liczb nieujemnych.

Dowodzenie niezmienników pętli

15

- Stwierdzenie jest prawdziwe przed pierwszym wykonaniem pętli, ale po dokonaniu całej inicjacji; wynika ono z warunku wstępnego pętli.
- Jeśli założymy, że stwierdzenie jest prawdziwe przed jakimś przebiegiem pętli i że pętla zostanie wykonana ponownie, a więc warunek pętli jest spełniony, to stwierdzenie będzie nadal prawdziwe po kolejnym wykonaniu wnętrza pętli.

- Proste pętle mają zazwyczaj proste niezmienniki.
- Pętle dzielimy na trzy kategorie
 - pętla z wartownikiem: czyta i przetwarza dane aż do momentu napotkania niedozwolonego elementu,
 - pętla z licznikiem: zawczasu wiadomo ile razy pętla będzie wykonana,
 - pętle ogólne: wszystkie inne.

Problem „STOPU”

16

- Aby udowodnić że program się zatrzyma, musimy wykazać, że zakończą działanie wszystkie pętle programu i wszystkie wywołania funkcji rekurencyjnych.
 - Dla pętli z wartownikiem wymaga to umieszczenia w warunku wstępnym informacji, że dane wejściowe zawierają wartownika.
 - Dla pętli z licznikiem wymaga to dodefiniowania granicy dolnej (górnjej) tego licznika.

Uwagi końcowe

17

- Istnieje wiele innych elementów, które muszą być brane pod uwagę przy dowodzeniu poprawności programu:
 - możliwość przepełnienia wartości całkowitoliczbowych,
 - możliwość przepełnienia lub niedomiar dla liczb zmiennopozycyjnych,
 - możliwość przekroczenia zakresów tablic,
 - prawidłowość otwierania i zamykania plików,
 - itd.

Uwagi końcowe

18

Na wybór najlepszego algorytmu dla tworzonego programu wpływa wiele czynników, najważniejsze to:

- prostota,
- łatwość implementacji
- efektywność