The Pedagogy of Program Design

Professor Viera Krňanová Proulx
College of Computer and Information Science
Northeastern University
Boston, Massachusetts, USA

DIDINFO 2009, Brusno, Slovakia
March 25, 2009
Outline:

Curriculum Foundation
- Design Recipes
- Supporting Software

Curriculum Components
- BOOTSTRAP - young children
- TeachScheme - secondary schools, universities
- ReachJava - secondary schools, universities

Pedagogy of Program Design
- Design Recipes: Details

Our Experiences; Resources
Curriculum Foundation

Function design based on algebra

• every function consumes data, produces new data
• no change of state; no input/output difficulties

Design Recipe for understanding the design process

• several steps, clearly defined
• each step has goals and measure of success

Understanding information versus data

• represent information as data
• interpret the information that data represents

Supporting software: to focus on the essential concepts

• language levels and test support
• libraries for graphics, interaction, client/server
Understanding the design process

DESIGN RECIPE for functions

• 1: Problem analysis and data definition
• 2: Purpose statement and the header
• 3: Examples with expected outcomes
• 4: Inventory/Template of available data fields and methods
• 5: Function body
• 6: Tests

Pedagogical advantages:

Each step is well defined
-- with a tangible result
-- with a guidance on what questions to ask

DESIGN RECIPEs for data definitions, abstractions...
DESIGN RECIPE for data definitions

- can it be represented by a primitive type? - select the type
- are there several parts that represent one entity? - a class
- are there several related variants? - a union of classes
- add arrows to connect data definitions

Make examples of data

- convert information to data
- interpret data as information
Supporting software: to focus on the concepts

Software appropriate for students current knowledge

Language levels

• add complex features gradually

Test support

• it should be easy to test every function

Supporting libraries: students focus on the essentials

• graphics: define color, size, shape/image, position
• interactions: define scene after a time tick, key or mouse event
• client/server programming: design the communication protocol
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program
10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program
10 lessons, 90 minutes each
Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each

Goals: to understand mathematical principles of computation
Curriculum Components: BOOTSTRAP

6 - 8 grade afterschool program

10 lessons, 90 minutes each
Curriculum Components: BOOTSTRAP

Student code:

```
; Updating Code
; update-object : Number -> Number
; move the object up the screen
(define (update-object y)
  (- y 20))

(define-expect "update-object: 24 -> 14")

; update-target-x : Number -> Number
; return the new x-coordinate of the target
(define (update-target-x x)
  (+ x 7))

(define-expect "update-target-x: 31 -> 38")

; update-target-y : Number -> Number
; return the new y-coordinate of the target
(define (update-target-y y)
  (+ y 50))

; update-player : Number String -> Number
; make a new player that is shifted to the left or right
(define (update-player x dir)
  (cond
   ([string? dir "left"] [(- x 60)])
   ([string? dir "right"] [(+ x 60)])
   [else x]))

; Geometry Functions
; distance : Number Number Number Number -> Number
; determines the distance between the target's tx,ty and the object's tx,ty
(define (distance x1 y1 x2 y2)
  (sqrt (+ (* (- x1 x2) 2) (* (- y1 y2) 2))))

; collide? : Number Number Number Number -> Boolean
; return true if the coordinates are within 15 of each other
(define (collide? tx ty ox oy)
  (<= (distance tx ty ox oy) 60))

; offscreen? : Number Number -> Boolean
; return true if the target's coordinates put it offscreen
(define (offscreen? x y)
  (or
   (<= x 640)
   (+ x 0)
   (< y 0)
   (< y 460)))

; PIC18F2020: update-object collide? target-player object-offscreen
```

```
```
Curriculum Components: BOOTSTRAP

Pedagogy

simple but exact language

• expressions: circles of evaluation
• functions: substitution
• conditionals

library with required functions:

(update-object x)
(update-target-x x)
(update-target-y y)
(update-player dir)
(collide tx ty ox oy)

• images are data
Curriculum Components: TeachScheme!

How to Design Programs

Interactive functional environment

Simple Scheme-like language(s)

Focus on understanding the design of data

The shape of a function follows the shape of the data

Authors:

Matthias Felleisen, Robert Bruce Findler,
Matthew Flatt, Shriram Krishnamurthi
Curriculum Components: TeachScheme!

How to Design Programs
Textbook free online:  htdp.org
DrScheme programming environment:  DrScheme.org

Tools for the programmer:
  • Stepper:  shows expression evaluation
  • Test support:  evaluates tests, reports results
  • Test coverage:  show program parts not tested
  • Dependency arrows:  definition to use, or use to definition

http://www.teach-scheme.org/
Curriculum Components: TeachScheme!

DrScheme

;; convert the longitude to x coordinate
;; to-x: flon -> num
(define to-x (lambda (longitude)
  (- 100 (* 100 (abs (sin (angle longitude)) 180) 180))))

;; Examples
(check-expect (to-x 89.649538) 51)
(check-expect (to-x 86.049538) 57)
(check-expect (to-x 83.639538) 44)
(check-expect (to-x 85.749538) 40)
(check-expect (to-x 85.564379) 37)

;; convert the latitude to y coordinate
;; to-y: flon -> num
(define to-y (lambda (latitude)
  (- 100 (* 100 (abs (sin (angle latitude)) 180) 180))))

;; Examples
(check-expect (to-y 39.640845) 54)
(check-expect (to-y 39.720852) 54)
(check-expect (to-y 41.600853) 38)
(check-expect (to-y 40.942103) 37)
(check-expect (to-y 38.545116) 35)

;; convert the location to a pcom
(define (to-pcom p
  (make-pcom to-x pto-long pto-lat pto-alt))

;; Examples
(check-expect (to-pcom op-11-loc) [make-pcom 51 54])
(check-expect (to-pcom sp-11-loc) [make-pcom 57 54])
(check-expect (to-pcom dr-11-loc) [make-pcom 44 38])
(check-expect (to-pcom to-ks-10) [make-pcom 40 37])
(check-expect (to-pcom se-by-10) [make-pcom 53 39])
Curriculum Components: TeachScheme!

DrScheme

```scheme
(define city (make-city "Springfield" "IL"))
(define mp-ll-loc (make-location 89649531/1000000 955001/250000))
(define ar-ll-loc (make-location 8504533/1000000 950523/750000))
(define de-ll-loc (make-location 914325953/1000000 4103003/1000000))
(define in-ll-loc (make-location 91064979/1000000 9506279/250000))
```

[Diagram showing DrScheme interface and code execution]

20
Curriculum Components: ReachJava

How to Design Classes

Textbook draft: 670 pages used in classrooms for five years

DrScheme programming environment: ProfessorJ languages

• language levels
• testing support
• libraries for interactive graphics based games

Curriculum materials:

• Laboratory tutorials/projects
• Assignments

Standard Java support
The Pedagogy of Program Design

The main themes:

• Data vs. Information
• Program Design
• The Role of Testing
• Designing Reusable Programs: Abstractions

The team:

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt
Kathryn E. Gray, Shriram Krishnamurthi, Viera K. Proulx
The Pedagogy of Program Design

Let's 'play' with the design of a simple game:

Fish swim across the screen, each is replaced by a new one when it escapes or is eaten.

Shark waits, swimming up and down in response to the keys, gets hungrier as the time goes on.

When the shark eats a fish it grows.

The game ends when the shark dies of starvation.
The Pedagogy of Program Design

• How do we build the game?
  ◦ We need a frame, a panel with graphics to draw
  ◦ We need to learn how to use the **Timer**
  ◦ We need to learn how to use the **onKeyEventListener**

Then we can start thinking about the game actions

*Technical details hide the program design*
The Pedagogy of Program Design

• How do **we** build the game? - what are the parts we need?
  ◦ How do we draw? - make a scene from rectangles, circles with size, color
  ◦ How do we animate? - create a scene for each tick
  ◦ How do we respond to the keys? - define **onKeyEvent** method

No complicated system interaction - focus on the game actions
The Pedagogy of Program Design

Data vs. Information

Think about the problem, what information is available?

• How do we build the game? - what are the parts we need?
  ◦ There is a shark - that moves up and down
  ◦ There is a fish - or more than one - that swims
  ◦ All should stay within the game area
The Pedagogy of Program Design

Data vs. Information

Think about the problem, what information is available?

• How do we build the game? - what are the parts we need?
  ◦ There is a shark - that moves up and down
  ◦ There is a fish - or more than one - that swims
  ◦ All should stay within the game area
The Pedagogy of Program Design

Data vs. Information

Think about the problem, what information is available?

- **Shark:** what do we know about him?
  - where is the shark
  - how hungry is the shark

- **Fish:** where is the fish?
  - How fast is it swimming?
  - Did it swim out of the game area?

- **Game area:** how wide, how tall?
  - Background color?
The Pedagogy of Program Design

Data vs. Information

- **World** consists of the area, the fish and the shark
- **Shark**
  - Position - consists of the x and y coordinate
  - Life time remaining
- **Fish**
  - Position - consists of the x and y coordinate
  - ... maybe the speed
- **Game area**
  - width and height
  - we also have to draw the shapes
Data vs. Information

Data definition for the world with CartPt: a class diagram
The Pedagogy of Program Design

Data vs. Information

Sample data

Fish fish = new Fish(new CartPt(200, 100), 5);

Shark shark = new Shark(new CartPt(20, 100), 30);

Box box = new Box(200, 200);

World w = new World(fish, shark, box);
The Pedagogy of Program Design

Data vs. Information

Sample data

*Fish* fish = new Fish(new CartPt(200, 100), 5);  
a fish that swims at speed 5 starting from the mid-right of the box

*Shark* shark = new Shark(new CartPt(20, 100), 30);  
a shark with 30 lives starting 20 pixels in from the mid-left of the box

*Box* box = new Box(200, 200);  
the box of width and height 200

*World* w = new World(fish, shark, box);  
the scene 200 by 200 with one fish on the right, one shark on the left
The Pedagogy of Program Design

Data vs. Information

• This is complicated enough to warrant separate attention
• We must make sure students understand what data the program works with

• Design Recipe for Data Definition:
  ◦ can it be represented by a primitive type? - select the type
  ◦ are there several parts that represent one entity? - a class
  ◦ are there several related variants? - a union of classes
  ◦ add arrows to connect data definitions

• Convert information to data
• Interpret data as information
The Pedagogy of Program Design

Data vs. Information

Sample data

Fish fish = new Fish(new CartPt(200, 100), 5);
a fish that swims at speed 5 starting from the mid-right of the box

Shark shark = new Shark(new CartPt(20, 100), 30);
a shark with 30 lives starting 20 pixels in from the mid-left of the box

Box box = new Box(200, 200);
the box of width and height 200

World w = new World(fish, shark, box);
the scene 200 by 200 with one fish on the right, one shark on the left
Designing the functionality

• Move the shark up and down in response to the arrow keys
• Move the fish left as the time goes on
• Replace the fish with a new one if it gets out of bounds
• Check if the shark ate the fish - if yes, replace the fish with a new one
• Starve the shark as the time goes on, check if he is dead
The Pedagogy of Program Design

Designing the functionality

• Move the shark up and down in response to the arrow keys
• Move the fish left as the time goes on
• Replace the fish with a new one if it gets out of bounds
• Check if the shark ate the fish - if yes, replace the fish with a new one
• Starve the shark as the time goes on, check if he is dead
The Pedagogy of Program Design

Designing the program

How do you eat an elephant?  one bite at a time

- One task  __  one function/method
- Make a wish list (a list of things to do later)  
  ... when the task is too complex
- Think systematically about each small task
Designing the program: Select a sub-problem

- Move the shark up and down in response to the arrow keys
- Move the fish left as the time goes on
- Replace the fish with a new one if it gets out of bounds
- Check if the shark ate the fish - if yes, replace the fish with a new one
- Starve the shark as the time goes on, check if he is dead
The Pedagogy of Program Design

Designing the program: One Task --- One Function/Method

• Check if the shark ate the fish

• Replace the fish with a new one
The Pedagogy of Program Design

Designing the program: One Task --- One Function/Method

Check if the shark ate the fish

Replace the fish with a new one

put the second task on a wish list
The Pedagogy of Program Design

Method Design - Step 1: Problem Analysis, Data Definition

Check if the shark ate the fish

What data do we need?
-- one Shark and one Fish

What class is responsible for this task?
-- could be either - choose Shark
-- the Fish becomes the method argument

What type of result do we produce?
-- a boolean value
Method Design - Step 2: Purpose Statement and the Header

In the class Fish:

```java
// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}
```

What should we do next?

... well, when can the shark eat the fish?

... -- when they are close enough to each other

We can now reason about the method behavior
The Pedagogy of Program Design

Method Design - Step 3: Examples with Expected Outcomes

// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}

The method produces a boolean result

... we need at least two examples

The shark and the fish far away from each other
The shark and the fish are close to each other
Method Design - Step 3: Examples with Expected Outcomes

// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}

Fish fish1 = new Fish(new CartPt(200, 100), 5);
Fish fish2 = new Fish(new CartPt(25, 100), 5);
Shark shark = new Shark(new CartPt(20, 100), 30);

shark.ateFist(fish1) ... expect false
shark.ateFist(fish2) ... expect true
The Pedagogy of Program Design

Method Design - Step 4: Inventory/Template

// check if this shark ate the given fish
boolean ateFish(Fish fishy) {...}

Make an inventory of what we know about the shark and the fish

this.loc -- CartPt
this.life -- int
fishy.loc -- CartPt
fishy.speed -- int

it depends on how close are the this.loc and fishy.loc
Method Design - Step 4: Inventory/Template

// check if this shark ate the given fish
boolean ateFish(Fish fishy) {...}

this.loc       -- CartPt
this.life      -- int
fishy.loc      -- CartPt
fishy.speed    -- int

it depends on how close are the this.loc and fishy.loc

Remember: one task  __  one function/method

Design a method boolean distTo(CartPt that) in the class CartPt
The Pedagogy of Program Design

Method Design - Step 4: Inventory/Template

// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}

this.loc         -- CartPt
this.life        -- int
fishy.loc        -- CartPt
fishy.speed      -- int

Design a method in the class CartPt
// compute the distance of this point to that
boolean distTo(CartPt that)
The Pedagogy of Program Design

Method Design - Step 4: Inventory/Template

// check if this shark ate the given fish
boolean ateFish(Fish fishy){...}

design a method in the class CartPt
// compute the distance of this point to that
boolean distTo(CartPt that)
Method Design - Step 5: Method Body

Designing method body:
... one question remains:
-- how close does the fish have to be for the shark to eat it?

-- we decide it must be within 20
-- of whatever unit we use to measure the distance

Here is the complete method - we hope:

// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

Are we done? ... NO
The Pedagogy of Program Design

Method Design - Step 6: Tests

What else needs to be done?

... how do we know we are correct?
... does the method work as we expected it to?

We already have examples with the expected outcomes!

Convert the examples into tests and test the method

```java
// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;
}
```
Method Design - Step 6: Tests

// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

Fish fish1 = new Fish(new CartPt(200, 100), 5);
Fish fish2 = new Fish(new CartPt(25, 100), 5);
Shark shark = new Shark(new CartPt(20, 100), 30);

checkExpect(shark.ateFist(fish1), false);
checkExpect(shark.ateFist(fish2), true);
The Pedagogy of Program Design

Method Design - Step 6: Tests

// check if this shark ate the given fish
boolean ateFish(Fish fishy){
    return this.loc.distTo(fishy.loc) < 20;}

Fish fish1 = new Fish(new CartPt(200, 100), 5);
Fish fish2 = new Fish(new CartPt(25, 100), 5);
Shark shark = new Shark(new CartPt(20, 100), 30);

checkExpect(shark.ateFist(fish1), false);
checkExpect(shark.ateFist(fish2), true);
... add more tests if needed
The Pedagogy of Program Design

Designing a Method: The DESIGN RECIPE

1: Problem analysis and data definition
2: Purpose statement and the header
3: Examples with expected outcomes
4: Inventory/Template of available data fields and methods
5: Method body
6: Tests

Each step is well defined
-- with a tangible result
-- with a guidance on what questions to ask
The Pedagogy of Program Design

Other sub-problems --- use the same design process

• Move the shark up and down in response to the arrow keys
• Move the fish left as the time goes on
• Replace the fish with a new one if it gets out of bounds
• Check if the shark ate the fish - if yes, replace the fish with a new one
• Starve the shark as the time goes on, check if he is dead
The Pedagogy of Program Design

Designing Abstractions

A skill on its own: transcends programming

- motivated by observing repeated code patterns
- students are taught to design abstractions
- each abstraction motivates a new language construct or style
The Pedagogy of Program Design

Designing Abstractions - Why Teach Abstractions?

Eliminate code duplication - reduce maintenance costs
Design reusable code
Build libraries
Learn to use libraries
Software Support

Essential for students to understand the concepts

• Language levels
• Exploration of the program behavior and evaluation
• Test design, evaluation, reporting
• Graphics and interactions
Software Support

Graphics and interactions

• Drawing with minimal work needed
• In Scheme: images are data
• Games: interface with methods to program the behavior:
  ◦ `onTick()` produce the scene after one clock tick
  ◦ `onKeyEvent(String ke)` produce the scene in response to the key press `ke`
  ◦ `bigBang(width, height, speed)` run the animation of the given size at the given speed
• Support for client/server programming with messages
Experiences

**BOOTSTRAP**

- schools in five states, summer camps: over the past four years

**TeachScheme!**

- 700 secondary schools, many universities
  - excellent results over the past 15 years	extbook translated into Polish, Spanish, German, ...

**ReachJava -- How to Design Classes**

- a number of universities - very good results
- workshops for teachers/instructors over the past five years

**Testing Library**

- a number of schools and universities - great response
THANK YOU

Resources

Web sites:

Main site for the TeachScheme/ReachJava! project: http://www.teach-scheme.org

Lab materials, lecture notes, assignments: http://www.ccs.neu.edu/home/vkp/HtDC.html

Tester library, World libraries: http://www.ccs.neu.edu/javalib

My home page: http://www.ccs.neu.edu/home/vkp