On Teaching *How to Design Programs*

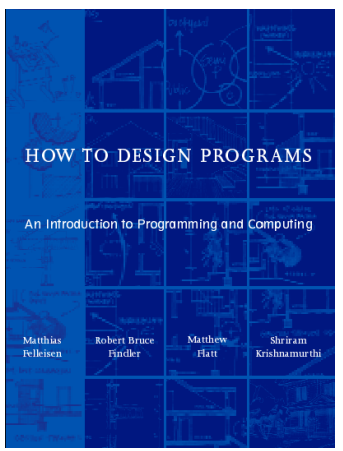
Norman Ramsey
Department of Computer Science
Tufts University

I want to help teachers

How to teach

How to Design Programs

(Felleisen, Findler, Flatt, Krishnamurthi)



(Even if you have never touched Racket)

Paper has hints, observations, problems

Paper has hints, observations, problems

This talk:

- **The design method**
- **What's hard for students**
- **An outsider's view of the technology**
- **One open problem**

Paper has hints, observations, problems

This talk:

- The design method
- What's hard for students
- An outsider's view of the technology
- One open problem

In the paper:

- Traps and pitfalls
- Much more of all the above

“Systematic problem-solving,” not “functional programming”

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data **Make data examples**

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data **Make data examples**

Describe function

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data	**Make data examples**
Describe function	**Make functional examples**

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data	**Make data examples**
Describe function	**Make functional examples**
Create code template from types	

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data	**Make data examples**
Describe function	**Make functional examples**
Create code template from types	**Fill in template**

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data	**Make data examples**
Describe function	**Make functional examples**
Create code template from types	**Fill in template**

Test

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data	**Make data examples**
Describe function	**Make functional examples**
Create code template from types	**Fill in template**

Test

Review and refactor

“Systematic problem-solving,” not “functional programming”

This is (my revised) design process:

Describe data

Make data examples

Describe function

Make functional examples

Create code template from types

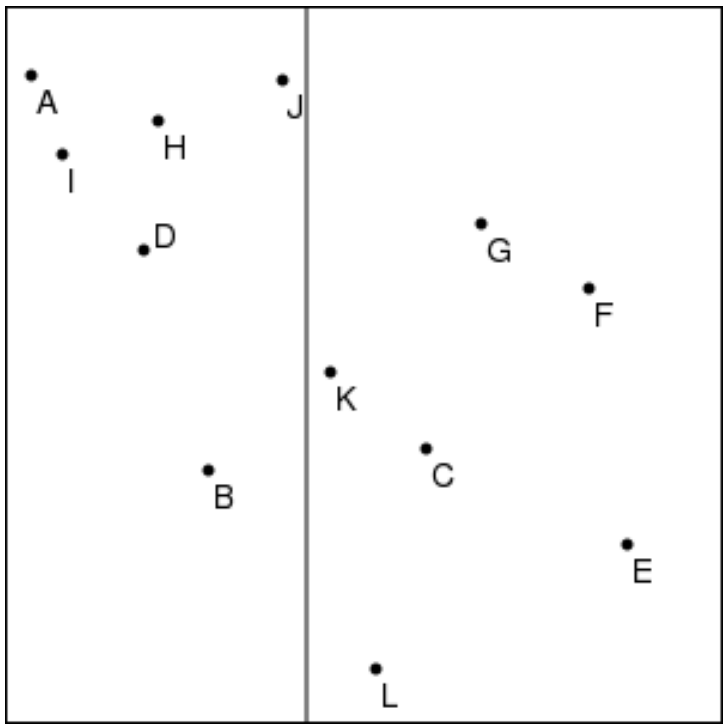
Fill in template

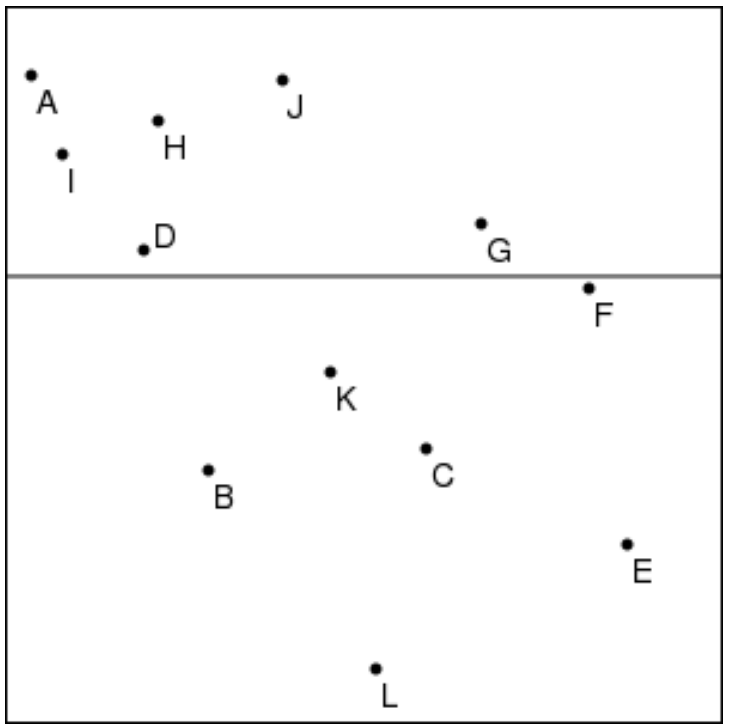
Test

Review and refactor

Example: Beginners can work with 2D-trees

Data examples:





Example: 2D-tree data definition

A `(2Dpoint A)` is a structure `(make-point x y value)`
where `x` and `y` are numbers and `value` is an `A`.

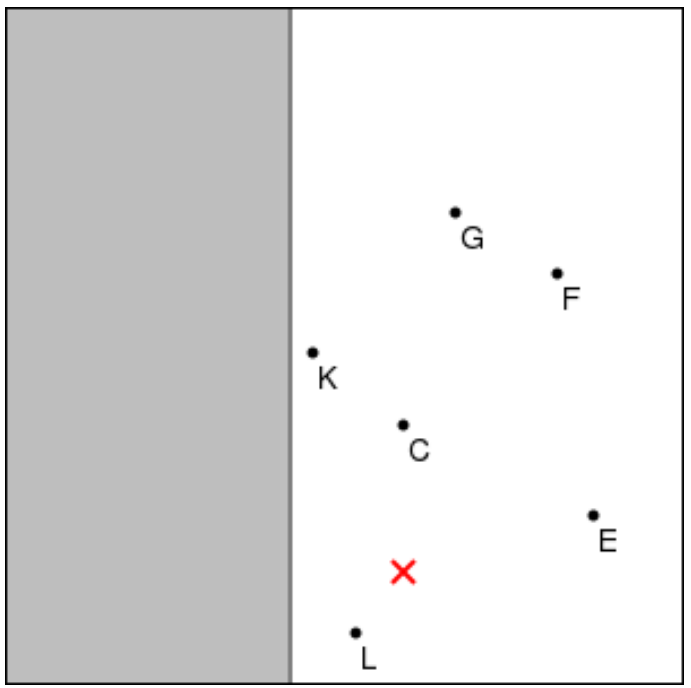
A `(2Dtree A)` is one of the following:

- A `(2Dpoint A)`
- A structure `(make-v-boundary left x right)`, where
  - `x` is a number,
  - `left` is a `(2Dtree A)` in which every point has an x coordinate at most `x`, and
  - `right` is a `(2Dtree A)` in which every point has an x coordinate at least `x`
- A structure `(make-h-boundary above y below)` ...

Nearest point: functional example

Problem 5, homework 9 (of 11)

Find the point nearest ×:

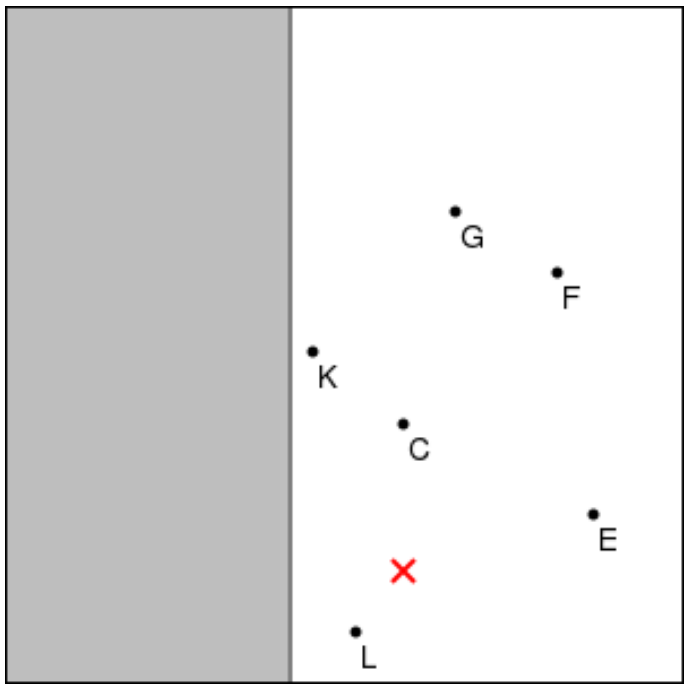


Answer: L

Nearest point: functional example

Problem 5, homework 9 (of 11)

Find the point nearest ×:



Answer: L

(Later: nearest hospital)

Nearest point: description

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                  (... x y
                       (h-boundary-y tree)
                       (h-boundary-above tree)
                       (h-boundary-below tree))]
        [(v-boundary? tree)
                  (... x y
                       (v-boundary-x tree)
                       (v-boundary-left tree)
                       (v-boundary-right tree))]))
```

Nearest point: description

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                   (... x y
                        (h-boundary-y tree)
                        (h-boundary-above tree)
                        (h-boundary-below tree))]
        [(v-boundary? tree)
                   (... x y
                        (v-boundary-x tree)
                        (v-boundary-left tree)
                        (v-boundary-right tree))]))
```

Nearest point: description

```
; nearest-point :
;      number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                    (... x y
                         (h-boundary-y tree)
                         (h-boundary-above tree)
                         (h-boundary-below tree))]
        [(v-boundary? tree)
                    (... x y
                         (v-boundary-x tree)
                         (v-boundary-left tree)
                         (v-boundary-right tree))]))
```

Nearest point: template

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                   (... x y
                        (h-boundary-y tree)
                        (h-boundary-above tree)
                        (h-boundary-below tree))]
        [(v-boundary? tree)
                   (... x y
                        (v-boundary-x tree)
                        (v-boundary-left tree)
                        (v-boundary-right tree))]))
```

Nearest point: template

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                  (... x y
                       (h-boundary-y tree)
                       (h-boundary-above tree)
                       (h-boundary-below tree))]
        [(v-boundary? tree)
                  (... x y
                       (v-boundary-x tree)
                       (v-boundary-left tree)
                       (v-boundary-right tree))]))
```

Nearest point: template

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                  (... x y
                       (h-boundary-y tree)
                       (h-boundary-above tree)
                       (h-boundary-below tree))]
        [(v-boundary? tree)
                  (... x y
                       (v-boundary-x tree)
                       (v-boundary-left tree)
                       (v-boundary-right tree))]))
```

Nearest point: template

```
; nearest-point :
;     number number (2Dtree A) -> (2Dpoint A)
; *efficiently* returns the point in the given tree
;   that is closest to the given (x, y) coordinates
(define (nearest-point x y tree)
  (cond [(point? tree) tree]
        [(h-boundary? tree)
                    (... x y
                         (h-boundary-y tree)
                         (h-boundary-above tree)
                         (h-boundary-below tree))]
        [(v-boundary? tree)
                    (... x y
                         (v-boundary-x tree)
                         (v-boundary-left tree)
                         (v-boundary-right tree))]))
```

Part II

What's Hard?

Type-directed programming is hard

Templates are constructed based on type of input.

If input is		
	Atomic	use library functions
	Sum	use `cond`
	Product	use selector functions
	Arrow	Apply it

Type-directed programming is hard

Templates are constructed based on type of input.

If input is	**Atomic**	**use library functions**
	Sum	**use `cond`**
	Product	**use selector functions**
	Arrow	**Apply it**

OMG the flaws!

Most common: repeated elimination of sums, products

(Paper, Section 3)

“Purpose statements” are hard

Hard in semesters 1, 2, 3, and 4

Reasonably good early examples:

```
;;meters->english; number -> string
;;input the distance in meters then
;;    converts them to english
```

```
;; move-big-hand : time -> time
;; adds one minute to time structure
```

After 5 weeks:

```
;; stations-on : railway -> list-of-stations
;; returns an ordered list of all stations
;;   on the railway, southernmost first
```

Recommendation: “Review and refactor”

Purpose statement's acid test: recursion

Vague purpose statement? Mentally inline the code.

- **"Works" until functions become recursive**

Students are very aggressive inliners

Purpose statement's acid test: recursion

Vague purpose statement? Mentally inline the code.

- **"Works" until functions become recursive**

Students are very aggressive inliners

Diagnosis: difficulty with procedural abstraction

Part III
The technology

I didn’t need five “language levels”

I didn't need five "language levels"

. . . Scheme . . .	Full Racket
Simple	Advanced Student Lang
Pure	Intermediate Student Lang+
⋮	
No local/let vars (first-order)	Beginning Student Lang

I didn't need five "language levels"

. . . Scheme . . .	Full Racket
Simple	Advanced Student Lang
Pure	Intermediate Student Lang+
⋮	
No local/let vars (first-order)	Beginning Student Lang

I didn't need five "language levels"

. . . Scheme . . .	Full Racket
Simple	Advanced Student Lang
Pure	Intermediate Student Lang+
⋮	
No local/let vars (first-order)	Beginning Student Lang

Don’t be fooled by the DrRacket IDE

GUI often distracted or frustrated students

- Designed for full Racket, including Help
- Mysterious program analyses (colored arrows)

Don’t be fooled by the DrRacket IDE

GUI often distracted or frustrated students

- Designed for full Racket, including Help
- Mysterious program analyses (colored arrows)

The major win:

Don’t be fooled by the DrRacket IDE

GUI often distracted or frustrated students

- **Designed for full Racket, including Help**
- **Mysterious program analyses (colored arrows)**

The major win:

Every time you compile,
untested code is thrown in your face

Result: students think testing is essential

Go deep into “world programs”

ICFP’09:

- **Interactive apps by composing pure functions**
- **“Build a program like applications students use”**

Go deep into “world programs”

ICFP’09:

- **Interactive apps by composing pure functions**
- **“Build a program like applications students use”**

Easily overlooked opportunities:

Go deep into “world programs”

ICFP’09:

- **Interactive apps by composing pure functions**
- **“Build a program like applications students use”**

Easily overlooked opportunities:

- **Look at world; see data; define representation**

Go deep into “world programs”

ICFP’09:

- **Interactive apps by composing pure functions**
- **“Build a program like applications students use”**

Easily overlooked opportunities:

- **Look at world; see data; define representation**
- **Design *programs*, not just functions**

Go deep into “world programs”

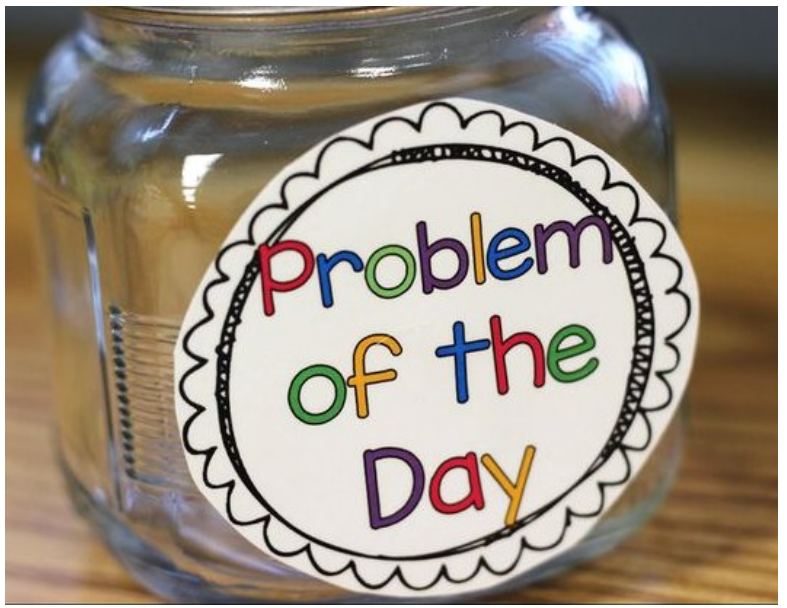
ICFP’09:

- **Interactive apps by composing pure functions**
- **“Build a program like applications students use”**

Easily overlooked opportunities:

- **Look at world; see data; define representation**
- **Design *programs*, not just functions**
- **Make choices that matter**

Part IV: An open problem



Assessment is too binary

What we really care about:

- **Was the code developed by systematic design?**

Assessment is too binary

What we really care about:

- **Was the code developed by systematic design?**

What experienced instructors do:

Assessment is too binary

What we really care about:

- **Was the code developed by systematic design?**

What experienced instructors do:

- **Deduct “points”**

Assessment is too binary

What we really care about:

- **Was the code developed by systematic design?**

What experienced instructors do:

- **Deduct “points”**

Open problems: Find a middle ground

Assessment is too binary

What we really care about:

- **Was the code developed by systematic design?**

What experienced instructors do:

- **Deduct "points"**

Open problems: Find a middle ground

Identify "primary traits?"

Conclusion:

Try it yourself

HtDP: low cost, high reward

- Delivers effective problem-solving
  - Good for students
  - Easy to sell “systematic software development”
- The technology really helps
- Plenty of functional-programming goodness
- I had lots of fun

HtDP: low cost, high reward

- Delivers effective problem-solving
  - Good for students
  - Easy to sell “systematic software development”
- The technology really helps
- Plenty of functional-programming goodness
- I had lots of fun

Please contribute!

- Dare to make changes
- Help solve some open problems
  (Paper, Section 6)

From end-of-term self-assessment

I didn't grasp the importance of laying out a template. I got into the habit of coding without templates. . . I didn't understand that by outlining the function based on input data, the design of the function was in many ways simplified and structured, making it harder to veer off course. Unfortunately, when data began to grow more complex, I failed to transition to using the template approach, and this resulted in a few uncomfortable weeks of coding. Without knowing how to establish a template based on the data of a function, I often felt very lost.

From end-of-term self-assessment

. . . As the problems became more complicated with more conditions, I realized that I couldn't do everything in my head. Writing [functional] examples concretizes each condition, its input and expected output, so that I can focus on solving one case at a time, which is significantly less daunting. Functional examples are also test cases which help me debug my code part by part.

From end-of-term self-assessment

While purpose statements are only written into the code for each individual function, they are useful for thinking about a problem as whole. [When] the entire problem is executed through one function which uses multiple helper functions, the signature and purpose statement make it easy to break down a problem into the individual parts addressed in each helper function. Often the change from one type of data with a certain meaning into another cannot be done in one step. Here it is the purpose statements and signatures that show each smaller step of the problem.