THE FIRST YEAR

Matthias Felleisen
PLT
Northeastern University, Boston
WHERE IT ALL BEGAN

- Starting in 1995 (January 26) @ Rice University
- The "How to Design" Project
  - TeachScheme!
  - DrScheme and ProfessorJ
- ... but in reality, it all began in 1978.
The Problem
in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods
in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods

in 2007 @ Anywhere (College, University): variables, assignments, printing, arrays, loops, procedures, classes and methods, ...
in 1978 @ Karlsruhe (Technische Universität): variables, assignments, printing, arrays, loops, procedures, classes and methods

in 2007 @ Anywhere (College, University): variables, assignments, printing, arrays, loops, procedures, classes and methods, ...

... and perhaps interfaces and inheritance.
Programming 101

- Algol 60/Simula 67
- Pascal
- C
- Scheme
- C++
- Eiffel
- Haskell
- Java
- Alice
1978 - 2007:
9 languages, 29 years:

- Algol 60/Simula 67
- Pascal
- C
- Scheme
- C++
- Eiffel
- Haskell
- Java
- Alice
1978 - 2007:
9 languages, 29 years:

Are we really just a fashion industry?
Is your story any better?
Why are we still thrashing around when it comes to the first year?

Do we really lack a “product” to sell?

Would a Physicist ask “how do I make the introductory course more Glamour-ous?”
The Constraints
Programming 1 & 2, You, and “Best Intentions”
Programming 1 & 2, You, and “Best Intentions”
CONSTRAINTS

Programming 1 & 2, You, and "Best Intentions"

Student Preparation

Student Motivation
CONTRAINTS

Programming 1 & 2, You, and "Best Intentions"

Student Preparation
Student Expectation
Student Motivation

Software Eng
CONSTITUENTS

Student Preparation

Student Expectation

Student Motivation

Programming 1 & 2, You, and “Best Intentions”

Software Eng

Graphics
CONSTRAINTS

Programming 1 & 2,
You, and "Best Intentions"

Parents

HCI
Software Eng
Graphics

Student Preparation
Student Expectation
Student Motivation
CONSTRAINTS

Programming 1 & 2, You, and “Best Intentions”

Parents

HCI
Softw Eng
Graphics

Student Preparation
Student Expectation
Student Motivation

Intern.
CONSTRAINTS

- HCI
- Softw Eng
- Graphics

Parents ➔ Student Preparation ➔ Student Expectation ➔ Intern.

Student Motivation ➔ ? ➔ Student Expectation ➔ Student Preparation ➔ Parents
THE COLLEGE TIMELINE

College

Year 1  Year 2  Year 3  Year 4  Year 5
THE COLLEGE TIMELINE
THE COLLEGE TIMELINE
THE COLLEGE TIMELINE

College
Year 1
Year 2
Year 3
Year 4
Year 5
Semester 2
Co-op
My First Job
THE COLLEGE TIMELINE

College

Year 1

Year 2

Year 3

Year 4

Year 5

Co-op

Principles

Preparation for Industry

My First Job
The Principles
COLLEGE
College empowers *life-long learning*. 
College empowers *life-long learning*.

Students need *early exposure to best-practices*.
College empowers *life-long learning*.

Students need *early exposure to best-practices*.

Faculty *aims high and has spine* to stick to principles.
Best practices means “get it right.”
Best practices means “get it right.”

Best practices means “think it through.”
Best practices means “get it right.”
Best practices means “think it through.”
Best practices means “to program is to design.”
PROGRAM. LANGUAGE
Object-oriented programming has won.
Object-oriented programming has won.
So has scripting (light-weight programming).
Object-oriented programming has won.

So has scripting (light-weight programming).

The quality of programs depends on how often the programmer has switched languages and programming paradigms. It does \textit{not} depend on the numbers of years spent with \textit{one} language [PPig].
THE FIRST LANGUAGE

Current:

variables, assignments, printing, arrays, loops, procedures, ...

10th grade math
Wanted: ???
Wanted: An tenth grader should naturally understand the computational model behind TFL.
THE FIRST LANGUAGE

Wanted: An tenth grader should naturally understand the computational model behind TFL.

TFL must accommodate (and display ideals of) principled program design.
The second language must prepare students for co-op and should be fashionable.

Use the second language to demonstrate that principles work everywhere.
PRINCIPLES & CONSTRAINTS

College

Year 1

- programming
- computation
- easy!

principles:

Year 2

- programming
- computation
- relevant

principles for OO
Linguistics I
THE FIRST LANGUAGE
THE FIRST LANGUAGE

Sather
Eiffel
C#
C++
VB
C
Basic

imperative or OOP
The First Language

- Basic
- C
- C#
- Java
- Eiffel
- Sather
- C++
- Java
- C
- VB
- Basic

Scripting Languages:
- JavaScript
- Ruby
- Python
- Tcl/Tk
- Perl

Imperative or OOP Languages:
- Sather
- Eiffel
- C#
- C++
- Java
- C
- VB
- Basic
It's Scheme!

- JavaScript
- Ruby
- Tcl/Tk
- Python
- Perl

scripting

- Sather
- Eiffel
- C#
- C++
- Java
- VB
- C
- Basic

imperative or OOP
The First Language

No, they are all wrong. It's Scheme!

Scripting

Imperative or OOP
// a first C++ program
main() {
    double price;
    int quantity;
    double total;
    ...
    price * quantity = total;
    ...
}
// a first C++ program
main() {
  double price;
  int quantity;
  double total;

  ...
  price * quantity = total;
  ...
}

LHS value expected here!
// a first C++ program
main() {
    double price;
    int quantity;
    double total;

    ... price * quantity = total; ...
}

LHS value expected here!

- 100s of hours of observations
- inner city, urban, suburban, public, private high schools, college freshmen
- Not a C++ problem.
- Every language suffers.
// my first Java program with extends

```java
class UP {
    UP(int x) {}  
}

class DOWN extends UP {  
    DOWN(int y) {}  
}
```
// my first Java program with extends

class UP {
    UP(int x) {}  
}

class DOWN extends UP {
    DOWN(int y) {}  
}

constructor-error-message.java:6: cannot find symbol: constructor UP()
Quick: What does this mean?

```java
// my first Java program with extends

class UP {
    UP(int x) {}
}

class DOWN extends UP {
    DOWN(int y) {}
}
```

constructor-error-message.java:6: cannot find symbol: constructor UP()
“You have to know everything all at once.”
“You have to know everything all at once.”

- We can’t change the “all at once” part.
The First Language

“You have to know everything all at once.”

- We can’t change the “all at once” part.
- We can change the “everything” part.
THE FIRST LANGUAGE

Beginner L

Full L
The First Language

- Full L
- Advanced L
- Intermediate L
- Beginner L
... and all of them are enforced and checked.
The First Language

Our choice: Scheme, a broken language


Beginning Scheme is 8th grade algebra, plus structures

The others chosen based on design principles.
Design (HtDP)
Design is what happens before, while, and after you program.

Design means to create programs in a systematic manner.

Design is the opposite of “tinker with examples until it works.”
Problem Statement
Problem Statement
Problem Statement

How do you “unstick”
A design method for beginners must unstick students at almost any point.

A design method must therefore guide students step by step from problem statement to program.

It is only step-by-step if it is continuous (a small change in the problem statement leads to a systematic change to the function).
We are designing functions: ... 
... things that consume data ...
... and produce data.
A design method must start from and must exploit the nature of input/output data. A design method must do this systematically.
The Design Recipes:

1. a series of steps from problem statement to solution expressed as questions;

2. the steps are fine-tuned to the nature of data that the function consumes.
HTDP

- read the problem, extract a data description, illustrate the data description with examples
- state a purpose and type signature
- illustrate with behavioral examples
- translate the data descr. into a program organization (aka “take inventory”)

Systematic Process
Data comes in sets (misnamed “classes”).

Naive set theory guides development:

- atomic (numbers, characters, ...)
- unions (intervals, classification)
- products (structures, records)
- “inductive” (arbitrary size, ...)
Problem:

Control a UFO via Keystrokes on Screen

Move the anchor point of a geometric shape along the horizontal or vertical according to key presses by the user.
Data Descriptions

A **Key** is one of:
- char (#\a)
- ‘left
- ‘right
- ‘up
- ‘down
- other symbol

A **Point** is a structure:

```
(define-struct point (x y))
```

**Examples:**

-- ‘left
-- (make-posn 20 30)
Contract:

;; Point Key -> Point

Purpose Statement:

;; +/- Xdelta to p if key is ‘right or ‘left
;; +/- Ydelta to p if key is ‘up or ‘down

Function Signature:

(define (move p key) ... )
**Functional Examples:**

```
move((make-point 10 20), 'left)
```

should be
```
(make-point (+ ... 10) 20)
```

```
move((make-point 10 20), 'up)
```

should be
```
(make-point 10 (+ ... 20))
```
Inventory:

(define (move p key)
  (cond
    ["key is a char" ... p ...]
    ["key is 'left"   ... (point-x p) ... (point-y p) ...]
    ["key is 'right"   ... (point-x p) ... (point-y p) ...]
    ["key is 'up"   ... (point-x p) ... (point-y p) ...]
    ["key is 'down"   ... (point-x p) ... (point-y p) ...]
    ["otherwise"   ... p ...]
  )
The Program:

from here, the program is self-evident
Tests:

(check-expect (move (make-posn 10 20) 'left) (make-posn 5 20))
HTDP: FUN

- Run: the function is used via a callback hook: (on-key-event move)
- Students write GUI-controlled programs from week 2.
- And they get to know the distinction between running and testing a program.
Problem:

Process the Files in A Directory ...

Design a function that finds out whether a file with some given name exists in a folder structure.
Data Description:

A **folder** is a structure with two fields:
- name, list of files and folders.

A **list of files and folders** is one of:
- the empty list
- a list of files and folders extended with a file
- a list of files and folders extended with a folder

A **file** is a name.
Data Description:

A **folder** is a structure with two fields:
- name, list of files and folders.

A **list of files and folders** is one of:
- the empty list
- a list of files and folders extended with a file
- a list of files and folders extended with a folder.

A **file** is a name.
Socratic Scripts:

- Does the data description consist of several disjoint subsets? How many?
- For each subset (subclass): Is it a structure? What are its fields?
- Does the data description refer to itself? Make the function recursive in analogous places!
Socratic Scripts:

- Does the data description consist of several disjoint subsets? How many?
- For each subset (subclass): Is it a structure? What are its fields?
- Does the data description refer to itself? Make the function recursive in analogous places!

This is material for the 7th week for total novices at NU.
HTDP: MORE DESIGN

- iterative refinement (data modeling)
- abstraction over similar data (polym.)
- abstraction over similar functions (h.o.)
- generative recursion (graph traversal)
- functions with accumulators
- memory (state and assignment)
Details too much for a talk.

Warning: Commercial Plug
Dewarning: http://www.htdp.org/
Linguistics 2
... and all of them are enforced and checked.
How do you enforce and support them?
Each language comes with a compiler and its own error reporting.

Each language comes with an algebraic stepper.

Each language comes with interactive REPL.
The First Languages

- Each language comes with a compiler and its own error reporting.
- Each language comes with an algebraic stepper.
- Each language comes with interactive REPL.

So we did it for all five of them, in one programming environment.
THE FIRST ENVIRONMENT
THE FIRST ENVIRONMENT

Thanks Viera Proulx
A GOOD FIRST IDE

Our Response to the 747-IDE
;;; draw : Position -> Image
;;; create a drawing of the rocket at the proper position
(define (draw p)
  (place-image XX p (empty-scene WIDTH HEIGHT)))

;;; move : Position -> Position
;;; move a rocket straight up
(define (move p)
  (+ 3 p))

;;; tests:
(check-expect (move 10) 13)
(check-expect (draw 10) (place-image XX 10 (empty-scene WIDTH HEIGHT)))
(generate-report)
;; draw : Position -> Image
;; create a drawing of the rocket at the proper position
(define (draw p)
  (place-image XX p (empty-scene WIDTH HEIGHT)))

;; move : Position -> Position
;; move a rocket straight up
(define (move p)
  (+ 3 p))

;; tests:
(check-expect (move 10) 13)

(check-expect (draw 10) (place-image XX 10 (empty-scene WIDTH HEIGHT)))
(generate-report)

;; run program run

Welcome to DrScheme, version 369.8-svn26feb2007 [3m].
Language: Beginning Student.
Teachpack: /Users/matthias/plt/collections/teachpack/htdp/testing.ss.
true
>(draw 20)
(define WIDTH 300)
(define HEIGHT 300)
(define XX 100)

;; draw : Position -> Image
;; create a drawing of the rocket at the proper position
(define (draw p)
  ;; code for drawing the rocket
)

(check-expect (move 10) 10)
(check-expect (draw p) (draw-rocket))

Welcome to DrScheme, version 369.8-svn26feb2007 [3m].
Language: Beginning Student.
Teachpack: /Users/matthias/plt/collects/teachpack/htdp/testing.ss.
true
true
true
true

Programing language: Beginning Student
not running
(define (world-move ke x)
  (cond
   ((char? ke) x)
   ((symbol=? ke 'left) (- x 2))
   ((symbol=? ke 'right) (+ x 2))
   (else x)))

(world-move 'left 20)

(cond
  ((char? 'left) 20)
  ((symbol=? 'left 'left) (- 20 2))
  ((symbol=? 'left 'right) (+ 20 2))
  (else 20))
(define (world-move ke x)
  (cond
    ((char? ke) x)
    ((symbol=? ke 'left) (- x 2))
    ((symbol=? ke 'right) (+ x 2))
    (else x)))

(cond
  ((char? 'left) 20)
  ((symbol=? 'left 'left) (- 20 2))
  ((symbol=? 'left 'right) (+ 20 2))
  (else 20))

(cond
  (false 20)
  ((symbol=? 'left 'left) (- 20 2))
  ((symbol=? 'left 'right) (+ 20 2))
  (else 20))
(define (world-move ke x)
  (cond
    ((char? ke) x)
    ((symbol=? ke 'left) (- x 2))
    ((symbol=? ke 'right) (+ x 2))
    (else x)))

(cond
  (false 20)
  ((symbol=? 'left 'left) (- 20 2))
  ((symbol=? 'left 'right) (+ 20 2))
  (else 20))
(define XX 100)
(define (draw p)
  (place-image
    XX
    p
    (empty-scene WIDTH HEIGHT)))
(place-image
  XX
  p
  (empty-scene 300 300))
(define XX 100)
(define (draw p)
  (place-image
   XX
   P
   (empty-scene WIDTH HEIGHT)))
(place-image
  100
  42
  (empty-scene WIDTH HEIGHT)))
(define (world-move ke)
  (cond
   [(char? ke) 20]
   [(symbol=? ke 'left) (-> 2)]
   [(symbol=? ke 'right) (-> 2)]
   [else 0]]))

;; Tests:
(= (world-move 'left 20) 18)
(= (world-move 'right 20) 22)
(= (world-move 'top 20) 20)
class State {
    int x;
    State(int x) { this.x = x; }

    // move this left or right by 2, depending on keystroke
    State move(String keystroke) {
        if (keystroke.equals("left"))
            return new State(this.x - 2);
        else if (keystroke.equals("right"))
            return new State(this.x + 2);
        else return this;
    }
}

Welcome to DrScheme, version 299.100.
Language: ProfessorJ: Beginner.
> new State(20).move("left")
State(x = 18)
Design (HtDC)

Semester 2
- Classes for Objects, Types to Check
- Does the Design Process Apply?
- Does Java Support it All?
Split the Design Recipe
Split the Design Recipe

Design Data:
- Diagrams & Classes
- Sample Instances
- Representation & Interpretation
HTDC: DESIGN RECIPE (1)

Split the Design Recipe

Design Data:
- Diagrams & Classes
- Sample Instances
- Representation & Interpretation

Design Methods:
- Signature & Purpose
- Functional Examples
- Template = Chase the Diagram
- Program!
- Tests
HTDC: CLASS DESIGN
“atomic” types
HTDC: CLASS DESIGN

“atomic” types

PlainClass
HTDC: CLASS DESIGN

“atomic” types

PlainClass

RefClass

ClassSec
"atomic" types

**HTDC: CLASS DESIGN**

PlainClass

RefClass

ClassSec

IUnion

Variant1

Variant2

Variant3
HTDC: CLASS DESIGN

“atomic” types

PlainClass

RefClass

ClassSec

ISelf

IUnion

Variant1

Variant2

Variant3

Variant1

Variant2

Variant3
WarOfWorlds extends World

UFO myUFO

UFO

Vector location
Design Example

WarOfWorlds extends World

UFO myUFO

Examples
Vector uLoc = ...
UFO u = ...
World w = ...

UFO
Vector location
Design Example

WarOfWorlds

UFO myUFO

World onKeyEvent(String ke)

Examples

Vector uLoc = ...  
UFO u = ...  
World w = ...

UFO

Vector location

UFO move(Vector delta)
Design Example

WarOfWorlds

UFO myUFO

World onKeyEvent(String ke)

Examples

Vector uLoc = ...
UFO u = ...
World w = ...

UFO

Vector location

UFO move(Vector delta)
HTDC: ABSTRACTION

- IUnion
  - Variant1
  - Variant2
  - Variant3
HTDC: ABSTRACTION

IUnion

Variant1
Variant2
Variant3

AUnion

Variant1
Variant2
Variant3

Inheritance
HTDC: ABSTRACTION
HTDC: ABSTRACTION

Inheritance and Overriding
HTDC: ABSTRACTION

Diagram:

- **IListInt**
  - **MtInt**
  - **ConsInt**

- **IListPict**
  - **MtPict**
  - **ConsPict**
HTDC: ABSTRACTION

Generics
detect similar but distinct traversals

abstract over distinct action with “first-class” methods ...

... which don’t exist: represent as instances of command class
detect similar but distinct traversals

abstract over distinct action with “first-class” methods ...

... which don’t exist: represent as instances of command class

Note: systematics “rediscovery” of patterns
HTDC: ENCAPSULATION

- idea: program for others, protect invariants
- for state, protection is encapsulation
HTDC: JAVA

- Java Teaching Languages: Beginning Student, Intermediate Student, Advanced Student
- switch to Eclipse and Java 1.5, with JUnit
- introduce Java run-time library
Results
RESULTS: FIRST SEMESTER

- The first semester: Ten Years
- Evaluation with respect to C++ course(s)
- Hand-over test at Rice: five instructors
- Field tests at 20 local high-schools
- Now used at 12+ colleges and 200+ high schools
RESULTS: RICE

Rice Engineering Freshmen
RESULTS: RICE

Rice Engineering Freshmen

HtDP

CS/CE/Eng

C/C++ Intro

CE/EE/Eng

Rice Engineering Freshmen
RESULTS: RICE

CS 212: OO Data Structures in C++

HtDP

C/C++ Intro

CS/CE/Eng

CE/EE/Eng

Rice Engineering Freshmen
RESULTS: RICE

Rice Engineering Freshmen

CS 212: OO Data Structures in C++

HtDP

C/C++ Intro

CS/CE/Eng

CE/EE/Eng

Rice Engineering Freshmen

HtDP Students routinely outperform C/C++ students on C++
RESULTS: HIGH SCHOOL

Same Teacher
RESULTS: HIGH SCHOOL

Class 1

Class 2

Class 3

Same Teacher
RESULTS: HIGH SCHOOL

Same 2 Curricula

Class 1  Class 2  Class 3

Same Teacher
RESULTS: HIGH SCHOOL

Same 2 Curricula

Class 1

Class 2

Class 3

Same Teacher
RESULTS: HIGH SCHOOL

- All students: HtDP preferred by ~70%

Same 2 Curricula

Class 1  Class 2  Class 3

Same Teacher
RESULTS: HIGH SCHOOL

- All students: HtDP preferred by ~70%
- The more C++, the more they prefer HtDP

<table>
<thead>
<tr>
<th>Class 1</th>
<th>Class 2</th>
<th>Class 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same 2 Curricula</td>
<td>Same Teacher</td>
<td></td>
</tr>
</tbody>
</table>
RESULTS: HIGH SCHOOL

- All students: HtDP preferred by ~70%
- The more C++, the more they prefer HtDP
- Female students: prefer HtDP by a ratio over 4:1
The second semester: four full years at NU

Three follow-up courses, 3 different instructors: all confirm that the students have made a quantum leap in programming performance.

Field tests: at a few high schools and colleges
RESULTS

Fall 2008: Bootcamp for MS
Conclusions, Future
<table>
<thead>
<tr>
<th>Semester 1</th>
<th>Programming</th>
<th>Mathematics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>How to Design Programs</em></td>
<td><em>Discrete</em></td>
</tr>
<tr>
<td>Semester 2</td>
<td><em>How to Design Classes</em></td>
<td><em>How to Prove Programs (ACL2)</em></td>
</tr>
</tbody>
</table>
Combining principles with pragmatics in the first year is
Combining principles with pragmatics in the first year is feasible.
Combining principles with pragmatics in the first year is

- ... feasible
- ... effective
Combining principles with pragmatics in the first year is

- ... feasible
- ... effective
- ... productive
Combining principles with pragmatics in the first year is

- ... feasible
- ... effective
- ... productive
- It is the *right* thing!
What it **Really** Takes

Design Principles
WHAT IT *Really* TAKES

Design Principles

Series of PLs
WHAT IT Really TAKES

Design Principles

Series of PLs

Peda. IDE
WHAT IT *Really* TAKES

- Design Principles
- Series of PLs
- Peda. IDE
WHAT IT *Really* TAKES

Design Principles

5 - 10 Years of Development Work with between 3 and 20 people

Series of PLs

Peda. IDE
WHAT IT *Really* TAKES

Design Principles

5 - 10 Years of Development Work
with between 3 and 20 people

Series of PLs

Peda. IDE

Good Luck!
Thank You!

Matthew Flatt
Robert Findler
Shriram Krishnamurthi
Kathi Fisler
Kathy Gray
Viera Proulx
John Clements
and many, many more