THE BEGINNING (1992/95)

AP, high schools
the "better math"
"computational" physics
economics "come alive"

C++, C, Pascal, Ratfor, Fortran

CS I
THE BEGINNING (1992/95)

- Robby Findler
- Kathi Fisler
- Matthew Flatt
- Shriram Krishnamurthi
- Emmanuel Schanzer

CS I

C, Pascal, Ratfor, Fortran

TeachScheme!

Program By Design

Bootstrap

CS II: if CS I is about “Scheme”, what roles does CS it serve?

- Robert Cartwright (Rice)
- Robby Findler
- Peter Druschel (MPI-SWS)
- Mike Ernst (UW)

Sw Dev ~ just before students study Sw Eng

Dist Sys Dev
THE BEGINNING (1992/95)

CS I

C, Pascal, Ratfor, Fortran

CS II: if CS I is about " Scheme", what roles does CS it serve?

Sw Dev ~ just before students study Sw Eng

Dist Sys Dev
WHERE I AM TODAY
TODAY’S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS

- **Sw Dev**: Technical skills: systematic creation of code
- **CO OP**: Communication skills: conversing about code
- **OOD**: Technical skills: systematic creation of code
- **CS I**: Systematic creation of code
- **CS II**: Systematic creation of code
- **LiCS**: Systematic creation of code
systematic design, typed & OOPL (Java)
pair programming, code review

pair programming, panel/peer review, memos on code

scale problem complexity and size; consolidate

6-month job-like setting, code in “the real world”

pair programming, code review

scale it up in Java, logic in interface

proving theorems about (functional) code, dual to systematic design
pair programming

functional programming for systematic design
Why should we care about software development?

What are doing wrong and what can we do better?

How can we change our introductory software development curriculum?
WHY CARE ABOUT SOFTWARE DEVELOPMENT?
SOFTWARE DEVELOPMENT IS A CHALLENGE & AN OPPORTUNITY

Do our colleagues really not care?

- research problems for the lone ranger
- software as prototypes, at most
- few maintain software over years

- there is no research here, just teaching
- coding is easy anyways
- kids get jobs if they can spell “C”
THE MORAL IMPERATIVE

Thesis

Our graduates will find jobs as long as they can spell the name of the C programming language. Every minute we spend on them, we won’t spend on research and papers and grants.
AntiThesis

Our graduates will find jobs as long as they can spell the name of the C programming language. Every minute we spend on them, we won’t spend on research and papers and grants.

99%

SynThesis

Colleges promise – in our name – that we add value to our undergraduates for the rest of their lives. We have a moral obligation to live up to our premise and a commercial one, too.
Programming is easy, we can teach it one or two courses. The software architects design, and programmers just code. But architecture is software engineering, not software development.
DEVELOPING SOFTWARE IS HARD.

**AntiThesis**

Programming is easy, we can teach it one or two courses. The software architects design, and programmers implement. But architecture is software engineering, not software development.

*David Pye, The Nature and Art of Workmanship, Cambium 2002*

**workmanship of certainty vs workmanship of risk**

**SynThesis**

*Software development* is “workmanship of risk” because (most of) it is a *thinking* activity and articulating thoughts. And that is hard.
DEVELOPING SOFTWARE IS HARD.

Programming is easy, we can teach it one or two courses. But architecture is software engineering, not software development.

David Pye, The Nature and Art of Workmanship, Cambium 2002

Programs must be written for people to read, and only incidentally for machines to execute.

Abelson and Sussman, Structure and Interpretation of Computer Programs, MIT Press, 1984
WE MUST LEARN TO APPRECIATE DEVELOPMENT TIME & QUALITY.

What is the cost of turning thoughts into code?

Thesis
The total cost of development consists of all the time developers touch the code.
We must learn to appreciate development time & quality.

- Developers are scarce.
- Ergo, developer time is scarce (expensive).
- Companies should worry about how they use their developers' time.
- Developers should care where they spend their (collective) time.

Your developers hate vacations.

Do they all have relationship trouble all the time?

All developers have teenagers at home. Been there, done that.
IN SUMMARY: WHY DO WE CARE

We have a *moral* and *commercial* obligation.

We actually don’t know how to teach software development properly.

There is a *research* and a *teaching* opportunity.
WHAT ARE WE DOING WRONG, WHAT CAN WE DO DIFFERENTLY
WHAT WE TEACH WHEN WE TEACH ‘CODING’

‣ Algol 60/Simula 67
‣ Pascal
‣ C
‣ Scheme
‣ C++
‣ Eiffel
‣ Haskell
‣ Java
‣ Alice/Scratch
‣ Python

10 cool languages in 30 years

Can we do better?
WHAT WE TEACH WHEN WE TEACH ‘CODING’

‣ “hello world”
‣ puzzles
‣ graphics
‣ GUIs
‣ web connections
‣ apps for your phone
‣ parallel processing tricks
‣ hack fests
‣ 3D printing

10 sexy tricks in 30 years

Is this all we offer?
WHAT WE TEACH WHEN WE TEACH ‘CODING’

\[
\begin{align*}
\text{def} \quad \text{main}() \quad \{
\quad \text{print} \quad \text{"hello world"} \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{int} \quad \text{main}() \quad \{
\quad \text{printf} \quad \text{("hello world")} \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{public} \quad \text{static} \quad \text{void} \quad \text{main}($\text{String} \quad \text{argv[]}$) \quad \{
\quad \text{System} \quad \text{out} \quad \text{.println} \quad \text{("hello world")} \\
\}
\end{align*}
\]

PROGRESS IN COMPUTER SCIENCE

SYNTAX

MORE SYNTAX

MIMIC MY EXAMPLES

LET’S ADD A PRINT STATEMENT

OKAY, TIME FOR THE DEBUGGER
Design all the way down.

Empower students to help themselves.

Inspect and review code.

**BUT WHAT IS DESIGN?**
WHAT COULD WE TEACH? DEVELOPMENT ~ SYSTEMATIC DESIGN

multiple stages

multiple representations

multiple viewpoints

(This slide stolen from Shriram Krishnamurthi)
WHAT COULD WE TEACH? DEVELOPMENT ~ SYSTEMATIC DESIGN

multiple stages
multiple viewpoints
multiple representations

EXPPLICATE SEPARATE PIECES

BREAK DOWN THE CODING PROCESS

HAVE STUDENTS STUDY THE VARIOUS PIECES
At every scale of software development, students must learn to

- stage the development process.
- understand software via multiple representations
- view code from at least two perspectives: producer and consumer.
HOW CAN WE CHANGE OUR SOFTWARE DEVELOPMENT CURRICULUM?
HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE

- We need *several courses* that inspect students’ code for its communicative qualities.

- Every course must enhance both
  - design skills
  - communication skills

- The courses must be coordinated.
HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE

Sw Dev
- Scale problem complexity and size; consolidate
- 6-month job-like setting, code in “the real world”
- Scale it up in Java, logic in interface
- Pair programming, code review

CO OP
- Thinking about code, dual to systematic design
- Pair programming

OOD
- Systematic design, typed & OOPL (Java)
- Pair programming, code review

CS II
- Pair programming, code review

CS I
- Pair programming

LiCS
- Systematic design, “student languages”
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE PROGRAM LEVEL

- data analysis, data definition, data examples
- signature and purpose statement
- functional examples
- function template
- function definition
- tests and testing

multiple stages

systematic design, “student languages”
HOW CAN WE TEACH SYSTEMATIC DESIGN ACROSS THE SCALE

- data analysis, data definition, data examples
- signature and purpose statement
- functional examples
- function template
- function definition
- tests and testing

### EXAMPLES

<table>
<thead>
<tr>
<th>given</th>
<th>wanted</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>37</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
</tr>
</tbody>
</table>

```
(define (f x) (+ (sqr x) 1))
```

```
(define (f x) (.. x ..))
```

```
;; Number -> Number
```

系统化设计，"学生语言"
How can we teach systematic design at the component level

What are all these ()s and ;s doing here?

CS I

CS II

LiCS

pair programming

pair programming

pair programming

pair programming

multiple viewpoints

“Pilot”

“Co-pilot”

reader

writer

What are all these ()s and ;s doing here?
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

class Mathy {
    int f(int x) {
        return x*x+1;
    }
}

(defun f (x) (+ (sqr x) 1))
(defthm F (implies (natp x) (> (f x) x)))

systematic design, typed & OOPL (Java)

thinking about code, dual to systematic design

multiple representations across courses
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

(defthm F (implies (listp l) (natp (f l)))

IH(l) <=>
(implies (listp l) (natp (f l))
by cases on the structure of l:
— l is ':(): 0
— l is (cons A k) .. IH(k) ..

(defun f (l)
  (cond
    ((endp l) 0)
    (t (+ (f (cdr l)) l))))
Type checking enforces signatures before damage is done.

Object-oriented design turns functional design on its side (but that’s it).

First test, then formulate theorems.

Induction is the dual of structural recursion.

systematic design, typed & OOPL (Java)

thinking about code, dual to systematic design
HOW CAN WE TEACH SYSTEMATIC DESIGN AT THE COMPONENT LEVEL

First test, then formulate theorems.

Induction is the dual of structural recursion.

thinking about code, dual to systematic design

(defthm F (implies (listp l) (natp (f l)))

IH(l) <=>
(implies (listp l) (natp (f l))
by cases on the structure of l:
- l is '() : 0
- l is (cons A k) .. IH(k) ..

(defun f (l)
  (cond
    ((endp l) 0)
    (t (+ (f (cdr l)) l))))
HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

Bring distinct representations together in one unit of code

scale it up in Java, logic in interface

join multiple representations
interface ISpecies {
    @pre this.oneIsHungry()
    
    @post !@result.isPresent() || @result.get() = s +1
    
    Optional<Integer> feed1(int s)
}

HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

join multiple representations

scale it up in Java, logic in interface
HOW CAN WE TEACH SYSTEMATIC DESIGN FOR SMALL SYSTEMS

My first co-op: “Day 4 and I am already demoing code. I LOVE MY LIFE.”

A co-op employer often expects students to pick up yet another language.

systematic design, typed & OOPL (Java)
pair programming, code review

pair programming, code review

thinking about code, dual to systematic design
pair programming

pair programming

systematic design, “student languages”
pair programming, panel/peer review, memos on code

scale problem complexity and size; consolidate

SO WHAT'S THIS ALL ABOUT?
A FINAL COURSE ON SOFTWARE DEVELOPMENT

(NOT SOFTWARE ENGINEERING)
The Situation

SPECIALIZATIONS & CAP STONES

COOP 3

SPECIALIZATIONS: AI, BIG DATA, SYSTEMS, PL, .....

COOP 2

pair programming, panel/peer review, memos on code

scale problem complexity and size; consolidate

TODAY'S WORLD @ NU CCIS: TECHNICAL SKILLS & COMMUNICATION SKILLS

junior & senior years

``middler“ year

sophomore year

freshman year
The Goal

Learn to produce software for, judge it by,

- its design organization,
- its clarity in ideas, and
- its testability.

Do not judge it by its functionality.
The Outline

13 weekly assignments on sw dev ideas

10 weeks dedicated to public code walks

- Your favorite programming language
- Living up to interfaces
- Development includes maintenance
- From interfaces to protocols
- Incremental refinement, step 2
- Incremental refinement, step 3
- Changing an API

- GUIs
- Refactoring
- Designing your own protocol
- Integration time
- Remote proxying
- Strategy [optional]

USE BOARD GAME BUT MAKE SURE TO DISCOUNT THE RESULTS OF ANY COMPETITION. IT'S ABOUT SW DEV NOT AI DEV.
SOFTWARE DEVELOPMENT, THE COURSE

Sw Dev

scale problem complexity and size; consolidate

pair programming, panel/peer review, memos on code

the students choose their favorite teenage-heartbreak language

change pairs
switch pairs to a different code base

describe problems in formal memos

conduct formal code walks to find design flaws, bugs

present code to panel and class

testing harnesses & “test fests” across languages

libraries & JSON parsing
echo servers on STDIN & STDOUT

streaming JSON parsers
deal with TCP sockets

planning @ scale and across time

a taste of distributed systems
How do you test in world of many different programming languages?

- Functional units of code.
- Design test languages in a data exchange language.

The students choose their favorite teenage-heartbreak language.

- Testing harnesses & “test fests” across languages
- Libraries & JSON parsing
- Echo servers on STDIN & STDOUT
- Streaming JSON parsers
- Deal with TCP sockets
- A taste of distributed systems
- Planning @ scale and across time
SOFTWARE DEVELOPMENT, THE COURSE

test fests, running everyone’s tests against everyone’s code.

---

**Testfest for homework 12**

Test cases are in **Name**. Programs submitted by pairs are in **Team**.

In each cell, a check mark indicates that the program passed the test. A cross, that it failed the test. A squiggle, that it passed for some combinations of pretty-printed or one-line output, and rapidly-transmitted vs trickle-fed output, but failed for other combinations.

Pairs are identified by the last four digits of their NUIDs (in the same order as their CCS ids appear in their class repo name). Tests are identified similarly, but with the addition of the test number after the two NUID segments.

Each cell in each row in the matrix is a hyperlink to the test case inputs and outputs.
SOFTWARE DEVELOPMENT, THE COURSE

- How can we increase complexity and size in a staged manner?
  - Functional protocols.
  - Design distributed programs from sequential ones.

- scale problem complexity and size; consolidate
- testing harnesses & “test fests” across languages
- libraries & JSON parsing
- echo servers on STDIN & STDOUT
- streaming JSON parsers
- a taste of distributed systems
- deal with TCP sockets
- planning @ scale and across time

the students choose their favorite teenage-heartbreak language
SOFTWARE DEVELOPMENT, THE COURSE

NO, NOT WITH SUFFICIENT DETAIL.

Can students figure out the architecture of such systems?

- Interfaces for Foobarmistan.
- Week-by-week training:
  - they design an interface.
  - then we use mine.

- scale problem, complexity and size; consolidate
- testing harnesses & "test fests" across languages
- libraries & JSON parsing
- echo servers on STDIN & STDOUT
- streaming JSON parsers
- deal with TCP sockets
- a taste of distributed systems
- planning @ scale and across time
- Can students figure out the architecture of such systems?
SOFTWARE DEVELOPMENT, THE COURSE

pair programming, panel/peer review, memos on code

the students choose their favorite teenage-heartbreak language

change pairs

switch pairs to a different code base

describe problems in formal memos

conduct formal code walks to find design flaws, bugs

present code to panel and class

How can students practice “software dev as articulation of thoughts”

▸ Students must practice continuously.

▸ Students must present to a panel. The goal is to help the panel discover errors in the devs’ thinking.
SOFTWARE DEVELOPMENT, THE COURSE

- Pair programming, panel/peer review, memos on code
- The students choose their favorite teenage-heartbreak language
- Change pairs
- Switch pairs to a different code base
- Present code to panel and class
- Conduct formal code walks to find design flaws, bugs
- Describe problems in formal memos

Does focused error discovery affect the students’ psyche?

- Practice with melt-downs.
SOFTWARE DEVELOPMENT, THE COURSE

Sw Dev

pair programming, panel/peer review, memos on code

the students choose their favorite teenage-heartbreak language

change pairs

switch pairs to a different code base

present code to panel and class

describe problems in formal memos

conduct formal code walks to find design flaws, bugs

How do we check whether and what panelists learn?

▸ Secretaries write memos.

▸ Grades come in four flavors: ok+, ok, ok-, zero.

▸ Students vote what the grades mean.
SOFTWARE DEVELOPMENT, THE COURSE

pair programming, panel/peer review, memos on code

the students choose their favorite teenage-heartbreak language

Does working with partners always work well?

- Allow divorces.
- Force changes.
- Let students vote on “choice of partner” or “choice of code base.”
TAKE AWAY
DEVELOPMENT COST IS HIGH FOR DEVELOPERS AND EMPLOYERS

... AND EVENTUALLY THIS WILL POSE A PROBLEM FOR THEM AND FOR US.
TAKE AWAY

STUDENTS NEED **TECHNICAL DESIGN SKILLS**.

- Teach systematic design explicitly.
- Teach it in several courses.
- Teach it at increasingly large scales.
- Teach it in different languages & contexts.
- Teach it until it becomes second nature.
STUDENTS NEED **TECHNICAL COMMUNICATION SKILLS**.

- Teach programming as communication of thoughts.
- Teach it in several courses.
- Teach it in different contexts.
- Teach it in for pairs and in class.
- Teach it until it becomes second nature.
Your students and their employers will appreciate these skills in time.
systematic design, typed & OOPL (Java)

pair programming, code review

CS I

pair programming, code review

Sw Dev

scale problem complexity and size; consolidate

type code, dual to systematic design

pair programming

thinking about code, dual to systematic design

pair programming

systematic design, “student languages”

pair programming

HOW?
THE END

- Robby Findler, for co-creating “Hell” and pointing me in the right direction
- Matthew Flatt, for teaching me the value of rapid feedback in design
- Shriram Krishnamurthi and Kathi Fisler, for many exchanges on design and planning
- .. and many others for discussions and push-back and telling me how wrong I was and often am
QUESTIONS?