How do people naturally think about computation?

Cyrus Omar
Computer Science Department
Carnegie Mellon University

05-899D: Human Aspects of Software Development (HASD)
Spring 2011 – Lecture 11
Programming is difficult
Programming is difficult

- Difficult to learn
  - 30% of students fail or withdraw from CS1

[Bennedsen and Caspersen 2007]
Programming is difficult

- Difficult to learn
  - 30% of students fail or withdraw from CS1
    
    [Bennedsen and Caspersen 2007]

- Difficult to do well
Programming is difficult

- **Difficult to learn**
  - 30% of students fail or withdraw from CS1
    
    [Bennedsen and Caspersen 2007]

- **Difficult to do well**

  Write a [Pascal] program that repeatedly reads in positive integers, until it reads the integer 99999. After seeing 99999, it should print out the average.

  *Rainfall Problem* [Soloway et al, 1983]
Programming is difficult

- **Difficult to learn**
  - 30% of students **fail or withdraw from CS1**
    - [Bennedsen and Caspersen 2007]

- **Difficult to do well**

Write a [Pascal] program that repeatedly reads in positive integers, until it reads the integer 99999. After seeing 99999, it should print out the average.

*Rainfall Problem* [Soloway et al, 1983]

- 14% of CS1 students (3/4 through course)
- 36% of CS2 students (3/4 through course)
- 69% of students in Jr./Sr. Systems course
Why?
What do people have trouble with?
What do people have trouble with?

- Conceiving of a solution?
What do people have trouble with?

- Conceiving of a solution?
- Formalizing the solution?
What do people have trouble with?

- **Conceiving of a solution?**
  - Q: Can people develop natural language solutions to programming problems?

- **Formalizing the solution?**
What do people have trouble with?

- **Conceiving of a solution?**
  - Q: Can people develop natural language solutions to programming problems?

- **Formalizing the solution?**
  - Languages and APIs are user interfaces
  - Q: Are they intuitive / natural?
  - Q: If not, how could we do better?
Q: Can people develop **natural language solutions** to programming problems?
Q: Can people develop natural language solutions to programming problems?

Write a [Pascal] program that repeatedly reads in positive integers, until it reads the integer 99999. After seeing 99999, it should print out the average.

Rainfall Problem [Soloway et al, 1983]

repeat
    Sum := 0 + I
    N := 1
    Sum := I + I
    N := 2
until I = 99999

[Bonar & Soloway, 1983]
Q: Can people develop natural language solutions to programming problems?

Write a [Pascal] program that repeatedly reads in positive integers, until it reads the integer 99999. After seeing 99999, it should print out the average.

Rainfall Problem [Soloway et al, 1983]

```pascal
repeat
    Sum := 0 + I
    N := 1
    Sum := I + I
    N := 2
until I = 99999
```

Subject: Input to [pause] so that the computer will know that, for each [pause] for each integer entered, you add 1, you add the integer to the sum [points to “Sum := 0 + I”], and that this is the first format of that, zero plus integer, N equals 1, sum equals integer plus integer, number = 2, until ...

[Bonar & Soloway, 1983]
Q: Can people develop **natural language solutions** to programming problems?

Write a [Pascal] program that repeatedly reads in positive integers, until it reads the integer 99999. After seeing 99999, it should print out the average.

*Rainfall Problem* [Soloway et al, 1983]

```
repeat
  Sum := 0 + I
  N := 1
  Sum := I + I
  N := 2
until I = 99999
```

Even though the subject seems fairly confused about how to express the program in Pascal, he has a very clear idea about the actions needed for a correct solution. We have found that this is typical -- novice programmers are not totally confused about what needs to be done, just about how to express that need.

[Bonar & Soloway, 1983]
Q: Can people develop **natural language solutions** to programming problems?

**Goal:** Create directions for *somebody else.*

Make one list of employees who meet either of the following criteria:

1. They have a job title of technician and they make 6 dollars/hr. or more.
2. They are unmarried and make less than 6 dollars/hr.

List should be organized by employee name.

[Miller, 1981]
**Q:** Can people develop **natural language solutions** to programming problems?

**Goal:** Create directions for somebody else.  

- **Successful:** other humans could accomplish tasks with their instructions  
- **Set operations**, not loops: “For all the last names starting with G…”  
- **If operations**, but no **else**.

Make one list of employees who meet either of the following criteria:

1. They have a job title of technician and they make 6 dollars/hr. or more.  
2. They are unmarried and make less than 6 dollars/hr.

List should be organized by employee name.  

[Miller, 1981]
Q: Can people develop natural language solutions to programming problems?

Suppose we sell concert tickets over the telephone in the following way – when a customer calls in and asks for a number (n) of seats, the seller 1) finds the n best seats that are available, 2) marks those n seats as unavailable, and 3) deals with payment options for the customer (e.g. getting credit or debit card number, or sending the tickets to the Will Call window for pickup).

Suppose we have more than one seller working at the same time. What problems might we see, and how might we avoid those problems?

[Lewandowski et al., 2007]
Q: Can people develop natural language solutions to programming problems?

<table>
<thead>
<tr>
<th>Accomplishment</th>
<th>Percent of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problems identified:</td>
<td></td>
</tr>
<tr>
<td>• Sell ticket more than once</td>
<td>97%</td>
</tr>
<tr>
<td>• Other</td>
<td>41%</td>
</tr>
<tr>
<td>Provided “reasonable” solutions to concurrency problems</td>
<td>71%</td>
</tr>
</tbody>
</table>

- 66 CS1 students across 6 schools with no prior experience

(table from Making Software, 2011)
Q: Can people develop natural language solutions to programming problems?

Reservation information from each of the computers would have to cross-pollinate to each of the other computers as soon as the seats changed status at all, to either of the three states. This introduces the problem of crossed signals. If seller A and seller B both book seats 145 - 160 at the exact same time, or within milliseconds of one another, the instructions for reserving those seats on each of the other computers would cross mid-stream, introducing a problematic double-booking, or even worse, no booking at all. [ID417]
Q: Can people develop **natural language solutions** to programming problems?

Children (aged 11 and 12) played a short 3D role-playing game and were asked to describe the rules of the game.

[Figure 2. Errors in triggers and outcomes]

[Good et al., 2010]
Q: Can people develop natural language solutions to programming problems?

- Yes, but…
  - Lots of imprecision and underspecification
  - Novices assume that instructee will interpret instructions intuitively.
Intuitions about programming language constructs

- Twelve fifth graders in a Pittsburgh public elementary school
- Equally divided amongst boys and girls
- No prior experience programming
- “The participants received no reward other than the opportunity to leave their normal classroom for half an hour and the opportunity to play a computer game for a few minutes.” 😊

[Pane et al., 2001]
Intuitions about programming language constructs

Programming Style

- **54%** - production rules or event-based, beginning with *when*, *if* or *after*.
  - *When PacMan eats all the dots, he goes to the next level.*
- **18%** - global constraints
  - *PacMan cannot go through a wall*
- **16%** - declarations/other
  - *There are 4 monsters.*
- **12%** - imperative
  - *Play this sound. Display this string.*

Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

[Pane et al., 2001]
Intuitions about programming language constructs

Modifying State

- **61%** - behaviors were built into the entity, e.g. OO
  - *Get the big dot and the ghost will turn colors…*
- **20%** - direct modification of properties
  - *After eating a large dot, change the ghosts from original color to blue.*
- **18%** - other

[Pane et al., 2001]
Intuitions about programming language constructs

OR

- 63% - boolean disjunction
  - To make PacMan go up or down, you push the up or down arrow key
- 20% - clarifying or restating the prior item
  - When PacMan hits a ghost or a monster, he loses his life.
- 18% - meaning otherwise
- 5% - other

[Pane et al., 2001]
Intuitions about programming language constructs

Iteration or looping constructs

- **73%** - implicit, where only a terminating condition is specified
  - *Make PacMan go left until a dead end*
- **20%** - explicit, with keywords such as *repeat, while, and so on*, etc.
- **7%** - other

- Loops are hotspots of errors for novice programmers.
- Often expect terminating condition to be checked continuously.

[du Boulay, 1989]

[Pane et al., 2001]
Question 5A
- Describe in detail what the computer should do to obtain these results.

<table>
<thead>
<tr>
<th>No.</th>
<th>First name</th>
<th>Last name</th>
<th>Average score</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sandra</td>
<td>Bullock</td>
<td>3000</td>
<td>Fine</td>
</tr>
<tr>
<td>2</td>
<td>Bill</td>
<td>Clinton</td>
<td>60000</td>
<td>Extraordinary</td>
</tr>
<tr>
<td>3</td>
<td>Cindy</td>
<td>Crawford</td>
<td>500</td>
<td>Poor</td>
</tr>
<tr>
<td>4</td>
<td>Tom</td>
<td>Cruise</td>
<td>5000</td>
<td>Fine</td>
</tr>
<tr>
<td>5</td>
<td>Bill</td>
<td>Gates</td>
<td>6000</td>
<td>Fine</td>
</tr>
<tr>
<td>6</td>
<td>Whitney</td>
<td>Houston</td>
<td>4000</td>
<td>Fine</td>
</tr>
<tr>
<td>7</td>
<td>Michael</td>
<td>Jordan</td>
<td>20000</td>
<td>Extraordinary</td>
</tr>
<tr>
<td>8</td>
<td>Jay</td>
<td>Leno</td>
<td>50000</td>
<td>Extraordinary</td>
</tr>
<tr>
<td>9</td>
<td>David</td>
<td>Lettermen</td>
<td>700</td>
<td>Poor</td>
</tr>
<tr>
<td>10</td>
<td>Will</td>
<td>Smith</td>
<td>9000</td>
<td>Poor</td>
</tr>
</tbody>
</table>
Intuitions about programming language constructs

AND

- **47%** - boolean conjunction
  - *Erase Bill Clinton and Jay Leno*
- **76%** - incorrect
  - *Everybody whose name starts with the letter G and L...*
    - *If you score 90 and above*
- **43%** - sequencing
  - *Crossed out the highest score and added the lower scores*
- **4%** - specify a range
  - *Fine is between 3000 and 20,000*
- **5%** - other

[J. F. Pane et al., 2001]
Intuitions about programming language constructs

Specifying open intervals

- **36%** - words such as *below, greater than* were intended to be exclusive
  - The performance of the person with the average score below 1000 is considered as *poor* (assigned *good* for 1000)
- **22%** - …inclusive
  - *Poor would be below 999* (assigned *poor* for 999)
- **22%** - used powers of 10 for ranges
  - *If your score is in the hundred’s your performance is poor.*
- **5%** - mathematical notation
- **15%** - other [Pane et al., 2001]

Adults were **100% successful** when using mathematical notation.
Intuitions about programming language constructs

Insertion into a data structure

- **75%** - no mention of making room for new element
  - *Put Elton John in the records in alphabetical order*

- **16%** - make room for element before inserting it
  - *Use the cursor and push it down a little and then type Elton John in the free space*

- **6%** - make room for element after inserting it

- **4%** - other

[Pane et al., 2001]
Natural Language Programming?

- A **difficult proposition** – natural language is complex and imprecise
  - Computer and programmer do not have a shared context [Nardi, 1993]; programmers cannot use rules of cooperative conversation [Grice, 1975]
  - Not obvious where the computer’s limits are

- Novices **can use formal languages** if designed carefully [Bruckman and Edwards, 1999]
  - Describing the instructee as a naïve alien increases precision of instructions [Galotti, 1985]
  - Anthropomorphizing computers is counterproductive [du Boulay, 1989]
Natural Language Programming?

- A difficult proposition – natural language is complex and imprecise.
- Computer and programmer do not have a shared context [Nardi, 1993]; programmers cannot use rules of cooperative conversation [Grice, 1975].
- Not obvious where the computer's limits are.
- Novices can use formal languages if designed carefully [Bruckman and Edwards, 1999].
- Describing the instructee as a naïve alien increases precision of instructions [Galotti, 1985].
- Anthropomorphizing computers is counterproductive [du Boulay, 1989].
Principles

5-4. **Closeness of mapping**

“The programming is the process of transforming a mental plan into one that is compatible with the computer.”
— Jean-Michel Hoc

- The translation process from a plan to a program should be minimal. The **expressiveness** of a language.
  - **Direct Manipulation** [Shneiderman, 1983; Hutchins et al, 1986]
  - Users have difficulty with low-level primitives [Hoc, 1990; Nardi, 1993; Lewis, 1987]
  - **Domain-specific languages** are behind many successful end-user systems

[Pane and Myers, 1996]
Principles

Models and Metaphors

Figure 3. A concrete model of the computer for a file management language.

Table 9. Proportion of correct answers on transfer test for model and control groups—file management language.

<table>
<thead>
<tr>
<th>Group</th>
<th>Sort-1</th>
<th>Sort-2</th>
<th>Count</th>
<th>Compute-1</th>
<th>Compute-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>0.66</td>
<td>0.66</td>
<td>0.63</td>
<td>0.58</td>
<td>0.45</td>
</tr>
<tr>
<td>Control</td>
<td>0.63</td>
<td>0.44</td>
<td>0.43</td>
<td>0.33</td>
<td>0.22</td>
</tr>
</tbody>
</table>

*Adapted from Maye80a.
*Note. 20 subjects per group; group × problem-type interaction, p < .07.
Principles

Contextualizing for Motivation

Figure 1: A program that replaces the background of one image with that of another, using the chromakey technique common in weather forecasts and special effects.

In the following weeks, these concepts are revisited and expanded on in the context of sound and text. Students learn to increase and decrease pitch and volume, to search text, to generate HTML, and to manipulate directories and networks. Finally, animation is explored and students use what they learned about pictures to modify animations and create elementary special effects.

During the final few weeks of the semester, after they have become proficient in the computational manipulation of media, students are introduced to browser CS concepts:

- Why is Photoshop faster than the code they write?
- What are interpreters and compilers?
- What is object-oriented programming?
- What do other programming languages look like?

Students are expected to turn in six homework assignments over the course of the semester, all of which involve programming and entail the creation of their own, original media. Three in-class exams assess students' comprehension of conceptual material and ability to read and create code. Two take-home exams assess programming proficiency.

A collaborative website (CoWeb) enables students to post questions, discuss problems, and display their work if they choose to do so.

```python
def chromakey2(source, bg):
    for p in pixels(source):
        if (getRed(p)+getGreen(p) < getBlue(p)):
            setColor(p,getColor(getPixel(bg,x(p),y(p))))
    return source
```

Figure 2: Examples of student collages

Interviewer: What do you think about the homework galleries on the CoWeb?

Student A: I don't ever look at it until after I'm done, I have a thing about not wanting to copy someone else's ideas. I just wish I had more time to play around with that and make neat effects. But JES will be on my computer forever, so that's the nice thing about this class is that you could go as deep into the homework as you wanted. So, I'd turn it in and then my roommate would do more after to see what we could do with it.

Interviewer: Have you ever written code outside of assignments?


Do students find media computation relevant and/or useful?

One indicator of students' attitudes toward a class is the rate at which they drop the course. By drop day, only three students had dropped media computation—2.5% of 120 students. By the end of the semester, the overall WFD rate had only reached 11.5%. This indicates that students' attitudes toward media computation were generally positive.

<table>
<thead>
<tr>
<th></th>
<th>Drop Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Computation</td>
<td>2.5%</td>
</tr>
<tr>
<td>Traditional Intro to CS</td>
<td>10.1%</td>
</tr>
</tbody>
</table>

Table 1: Spring 2003 drop and withdrawal, failure and D-grade (WFD) rates for Introductory CS at Georgia Tech.

Many students reported that they found the content of the course useful. When asked what they liked best about the course at midterm, approximately 20% of media computation students indicated that they enjoyed the content, while about 12% named the things covered same material using media (audio/visual) tasks.

- Decrease in drop rate validated for both CS0.5 and CS1 at several institutions [Tew et al, 2005; Sloan and Troy 2008; Simon et al, 2010]

- Learn different things but do skills transfer later? [Tew et al, 2005]
  - Initial positive result, cannot be replicated
Principles

7. Consistency and Standards

“Users should not have to wonder whether different words, situations or actions mean the same thing.” [Nielson, 1994]

- Notation should abide by suggestions that can be derived from other places in the language, to facilitate transfer of knowledge [Green, 1996].

- Users get confused when there are two different syntaxes to accomplish the same effect [Eisenberg, 1987]

- The meaning of keywords should be context-independent.
  - Novices focus on surface features [McKeithen, 1981]
  - The keyword static in C++ has many meanings depending on context.

[Pane and Myers, 1996]
Principles

4.4. Beware of Misleading Appearances

4.5. Avoid Subtle Distinctions in Syntax

- [Fitter, 1979] cites the principle of restriction: the syntax prohibits the creation of code that could easily be confused with other closely-related forms.
- Common typos and cognitive slips should be caught [Green, 1996]
- `if (a = 0)` vs. `if (a == 0)` in C
Principles

4.4. *Principle of Conciseness*

- [Cordy, 1992] argues against redundant symbols, preambles, punctuation, declarations and annotations.
- Also argues for intelligent defaults.
- Conciseness is *not economy* (a minimal set of primitives).
  - Early versions of Prolog did subtraction by inverse addition [Green, 1990].
- APL takes conciseness to the extreme, leading to too many cryptic primitives.

[Pane and Myers, 1996]
Goal: Gentle Slope Systems

- Swing
- Backend Programming
- ActionScript
- Basic
- OO Programming
- Java
- Visual Basic
- Flash

Difficulty of Use

Program Complexity and Sophistication

Low Threshold

High Ceiling

Copyright © 2009 – Brad A. Myers
Historical Context

- Long History of study with other names
  - Original HCI!
    - 1973 “Psychology of Programming” (PoP)
  - “Software Psychology”
    - Ben Shneiderman book, 1980
  - “Empirical Studies of Programming” (ESP)
    - Workshops from 1986 through 1999
  - “Psychology of Programming”
    - Psychology of Programming Interest Group (PPIG)
      - from 1987 and PPIG’10 = 22th workshop
  - “Empirical Software Engineering”

- Much of the early CSCW research as well
  - Computer-Supported Cooperative Work
References


References


References


References


“In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use.... The human and computer parts of programming languages have developed in radical asymmetry.”

Allen Newell and Stuart Card 1985