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]

```
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.

> 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]

- **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**.

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 7-2. Number of solutions and problems identified by students (n=66), from [Lewandowski et al. 2007]

Accomplishment	**Percent of students**
Problems identified:	
• Sell ticket more than once	97%
• Other	41%
Provided “reasonable” solutions to concurrency problems	71%

- **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]

[Lewandowski et al., 2007]

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.

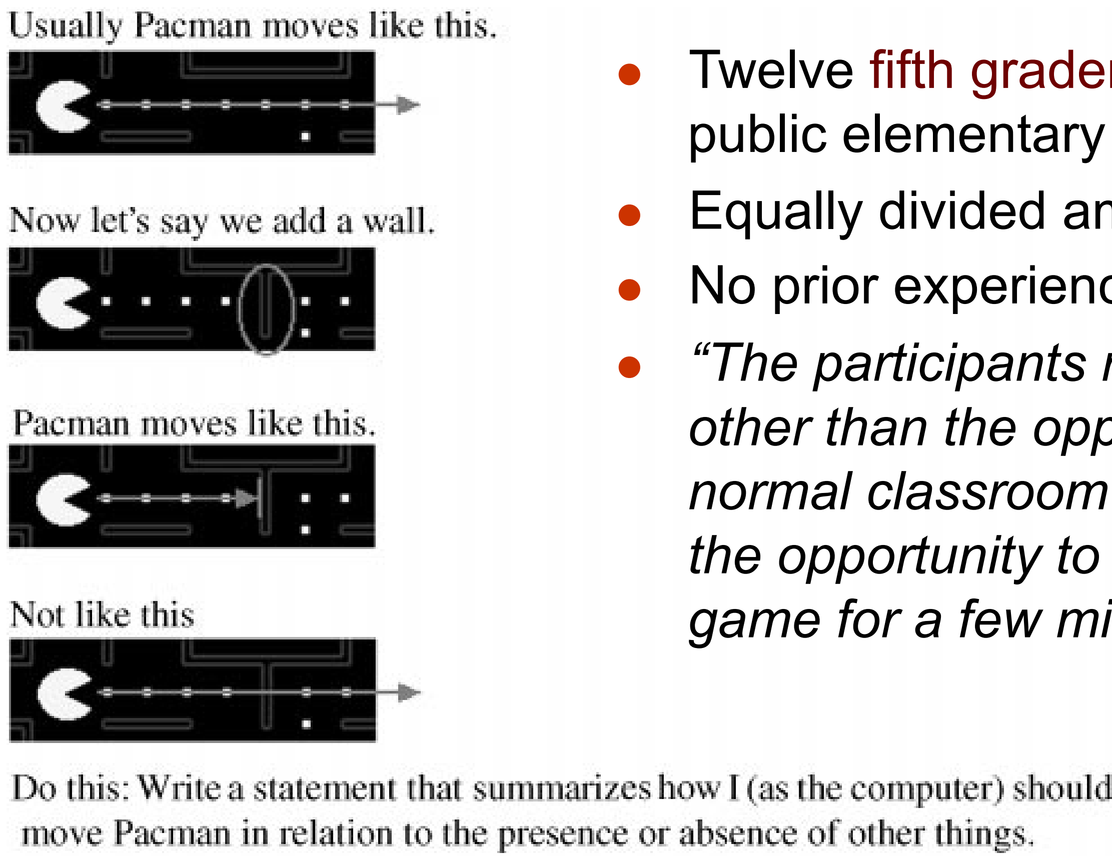
Figure 2. Errors in triggers and outcomes

[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]

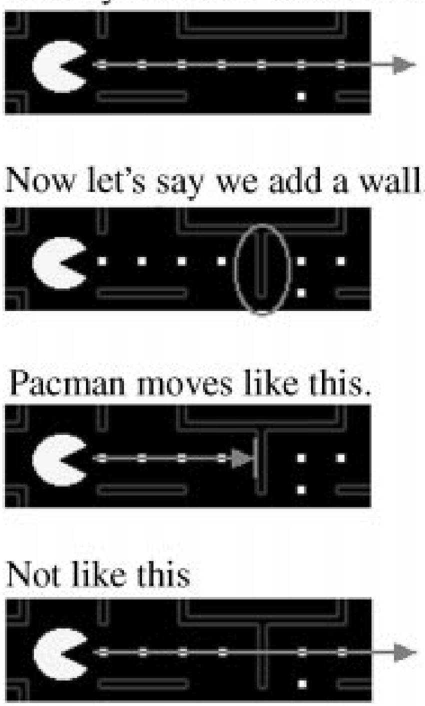
Intuitions about programming language constructs

Usually Pacman moves like this.

Now let's say we add a wall.

Pacman moves like this.

Not like this



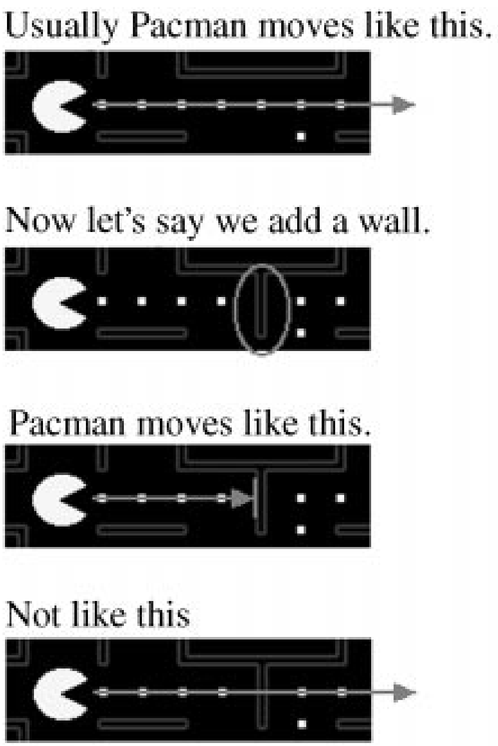
Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

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.*

[Pane et al., 2001]

Intuitions about programming language constructs



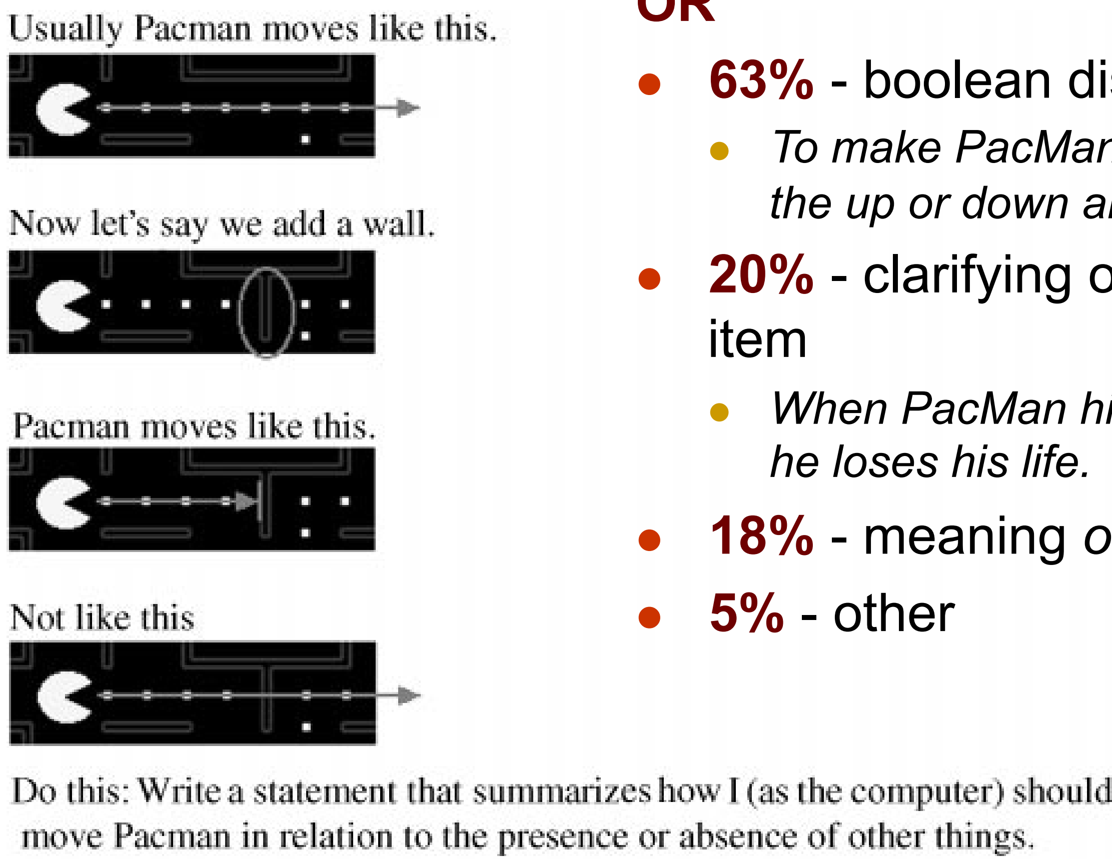
Do this: Write a statement that summarizes how I (as the computer) should move Pacman in relation to the presence or absence of other things.

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

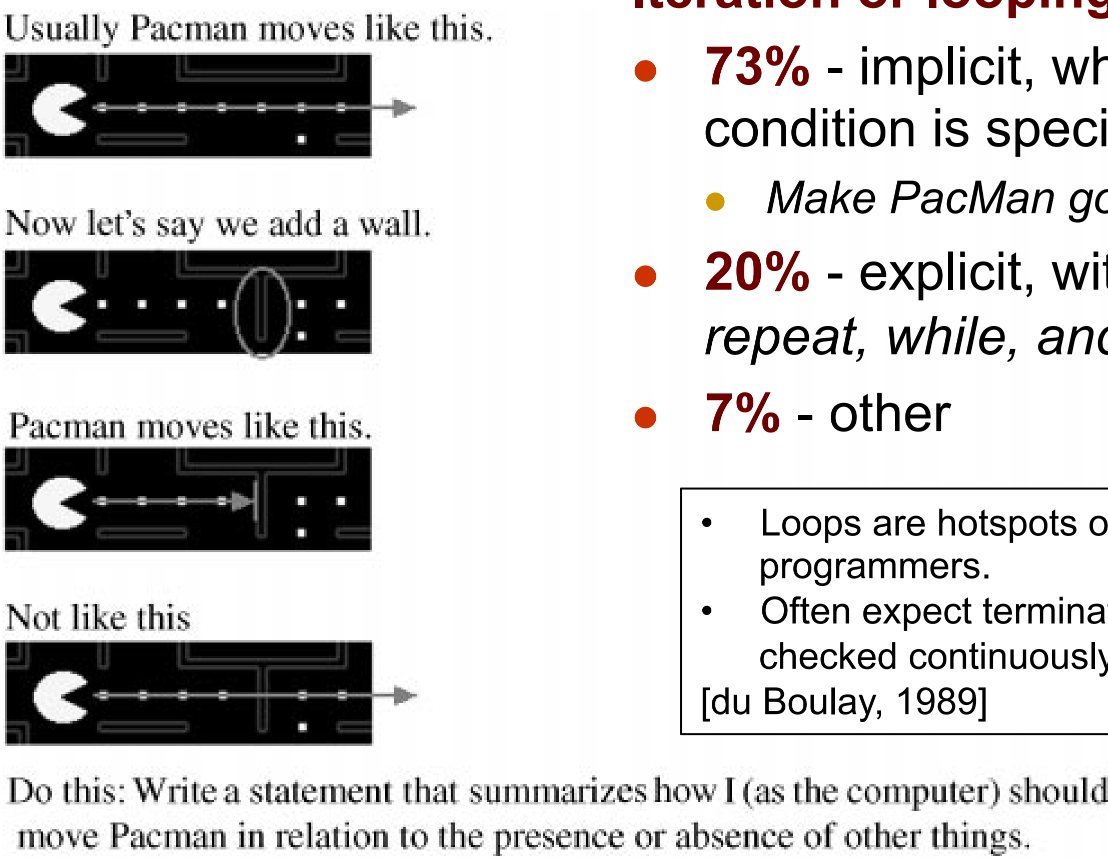


OR

- **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]

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	
2	Bill	Clinton	60 000	
3	Cindy	Crawford	500	
4	Tom	Cruise	5000	
5	Bill	Gates	6000	
6	Whitney	Houston	4000	
7	Michael	Jordan	20 000	
8	Jay	Leno	50 000	
9	David	Lettermen	700	
10	Will	Smith	9000	

Question 5A

- Describe in detail what the computer should do to obtain these results.

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	Fine
2	Bill	Clinton	60 000	Extraordinary
3	Cindy	Crawford	500	Poor
4	Tom	Cruise	5000	Fine
5	Bill	Gates	6000	Fine
6	Whitney	Houston	4000	Fine
7	Michael	Jordan	20 000	Extraordinary
8	Jay	Leno	50 000	Extraordinary
9	David	Lettermen	700	Poor
10	Will	Smith	9000	Poor

FIGURE 3. Depiction of a problem scenario in study two.

Population: Kids from same population + a few adults from CMU who had no programming experience.

[Pane et al., 2001]

Intuitions about programming language constructs

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	
2	Bill	Clinton	60 000	
3	Cindy	Crawford	500	
4	Tom	Cruise	5000	
5	Bill	Gates	6000	
6	Whitney	Houston	4000	
7	Michael	Jordan	20 000	
8	Jay	Leno	50 000	
9	David	Lettermen	700	
10	Will	Smith	9000	

Question 5A
- Describe in detail what the computer should do to obtain these results.

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	Fine
2	Bill	Clinton	60 000	Extraordinary
3	Cindy	Crawford	500	Poor
4	Tom	Cruise	5000	Fine
5	Bill	Gates	6000	Fine
6	Whitney	Houston	4000	Fine
7	Michael	Jordan	20 000	Extraordinary
8	Jay	Leno	50 000	Extraordinary
9	David	Lettermen	700	Poor
10	Will	Smith	9000	Poor

FIGURE 3. Depiction of a problem scenario in study two.

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

[Pane et al., 2001]

Intuitions about programming language constructs

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	
2	Bill	Clinton	60 000	
3	Cindy	Crawford	500	
4	Tom	Cruise	5000	
5	Bill	Gates	6000	
6	Whitney	Houston	4000	
7	Michael	Jordan	20 000	
8	Jay	Leno	50 000	
9	David	Lettermen	700	
10	Will	Smith	9000	

Question 5A

- Describe in detail what the computer should do to obtain these results.

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	Fine
2	Bill	Clinton	60 000	Extraordinary
3	Cindy	Crawford	500	Poor
4	Tom	Cruise	5000	Fine
5	Bill	Gates	6000	Fine
6	Whitney	Houston	4000	Fine
7	Michael	Jordan	20 000	Extraordinary
8	Jay	Leno	50 000	Extraordinary
9	David	Lettermen	700	Poor
10	Will	Smith	9000	Poor

FIGURE 3. Depiction of a problem scenario in study two.

Adults were **100% successful** when using mathematical notation.

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]

Intuitions about programming language constructs

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	
2	Bill	Clinton	60 000	
3	Cindy	Crawford	500	
4	Tom	Cruise	5000	
5	Bill	Gates	6000	
6	Whitney	Houston	4000	
7	Michael	Jordan	20 000	
8	Jay	Leno	50 000	
9	David	Lettermen	700	
10	Will	Smith	9000	

Question 5A

• Describe in detail what the computer should do to obtain these results.

No.	First name	Last name	Average score	Performance
1	Sandra	Bullock	3000	Fine
2	Bill	Clinton	60 000	Extraordinary
3	Cindy	Crawford	500	Poor
4	Tom	Cruise	5000	Fine
5	Bill	Gates	6000	Fine
6	Whitney	Houston	4000	Fine
7	Michael	Jordan	20 000	Extraordinary
8	Jay	Leno	50 000	Extraordinary
9	David	Lettermen	700	Poor
10	Will	Smith	9000	Poor

FIGURE 3. Depiction of a problem scenario in study two.

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]

(Popular Science)

Principles

5-4. *Closeness of mapping*

> "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

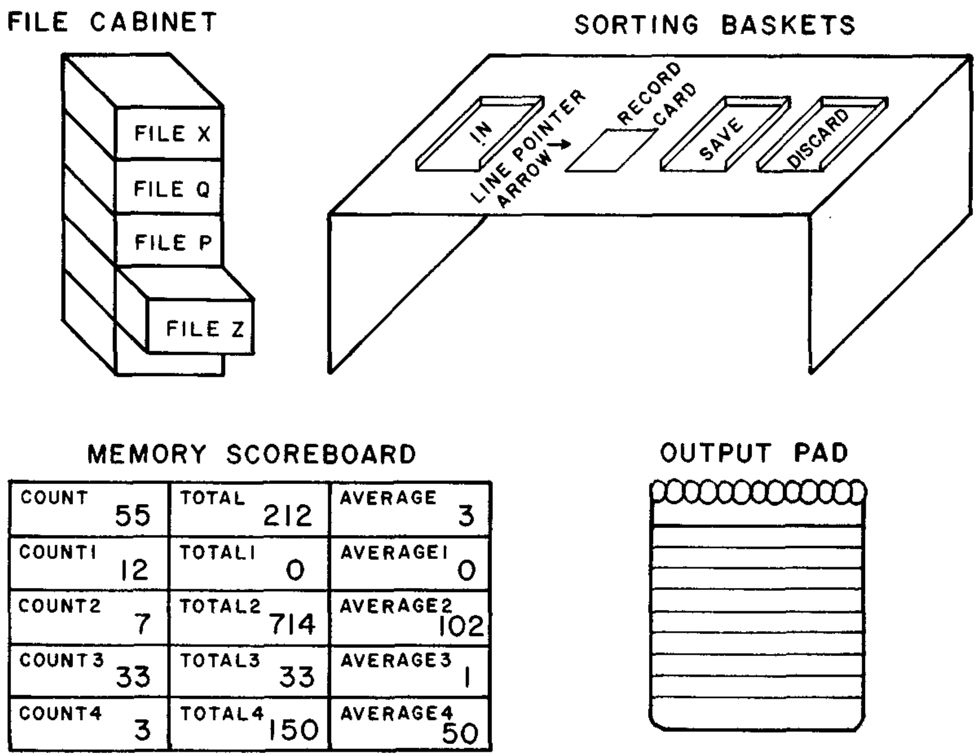


FIGURE 3. A concrete model of the computer for a file management language.

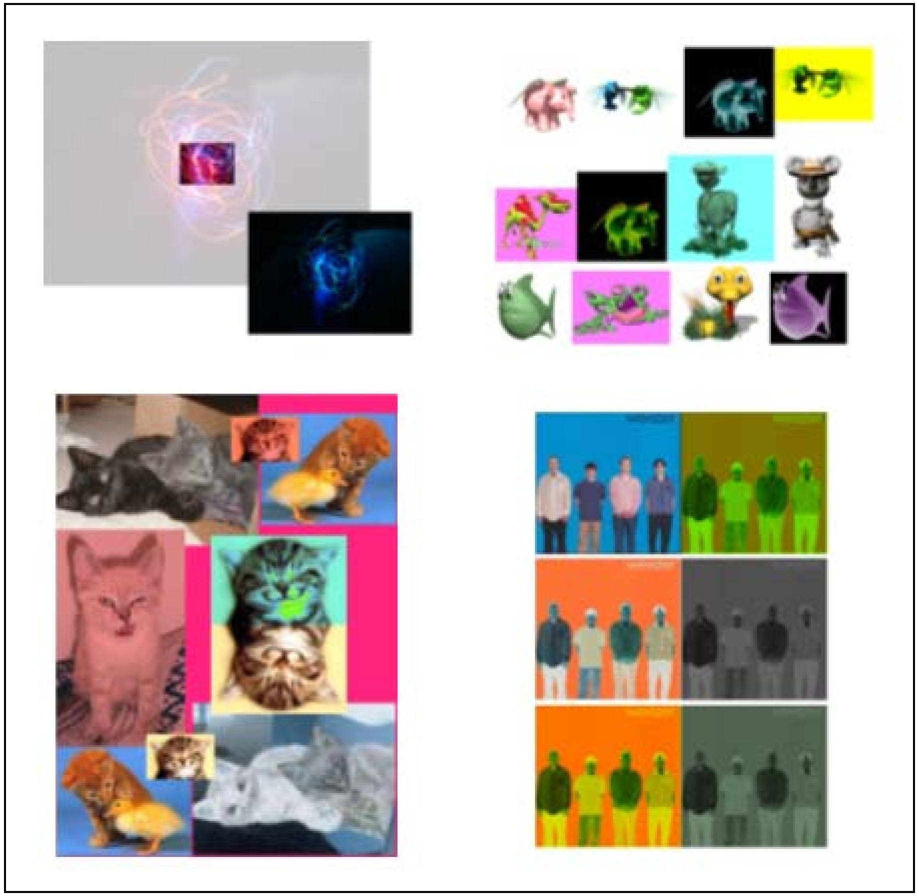
TABLE 9. PROPORTION OF CORRECT ANSWERS ON TRANSFER TEST FOR MODEL AND CONTROL GROUPS—FILE MANAGEMENT LANGUAGE[a]

Group	*Type of Test Problem*				
	Sort-1	*Sort-2*	*Count*	*Com-pute-1*	*Com-pute-2*
Model	.66	.66	.63	.58	.45
Control	.63	.44	.43	.33	.22

[a] Adapated from MAYE80a.
Note. 20 subjects per group; group × problem-type interaction, $p < .07$.

Principles

Contextualizing for Motivation



```
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
```

	Drop Rate
Media Computation	2.5%
Traditional Intro to CS	10.1%

- 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

[Pane and Myers, 1996]

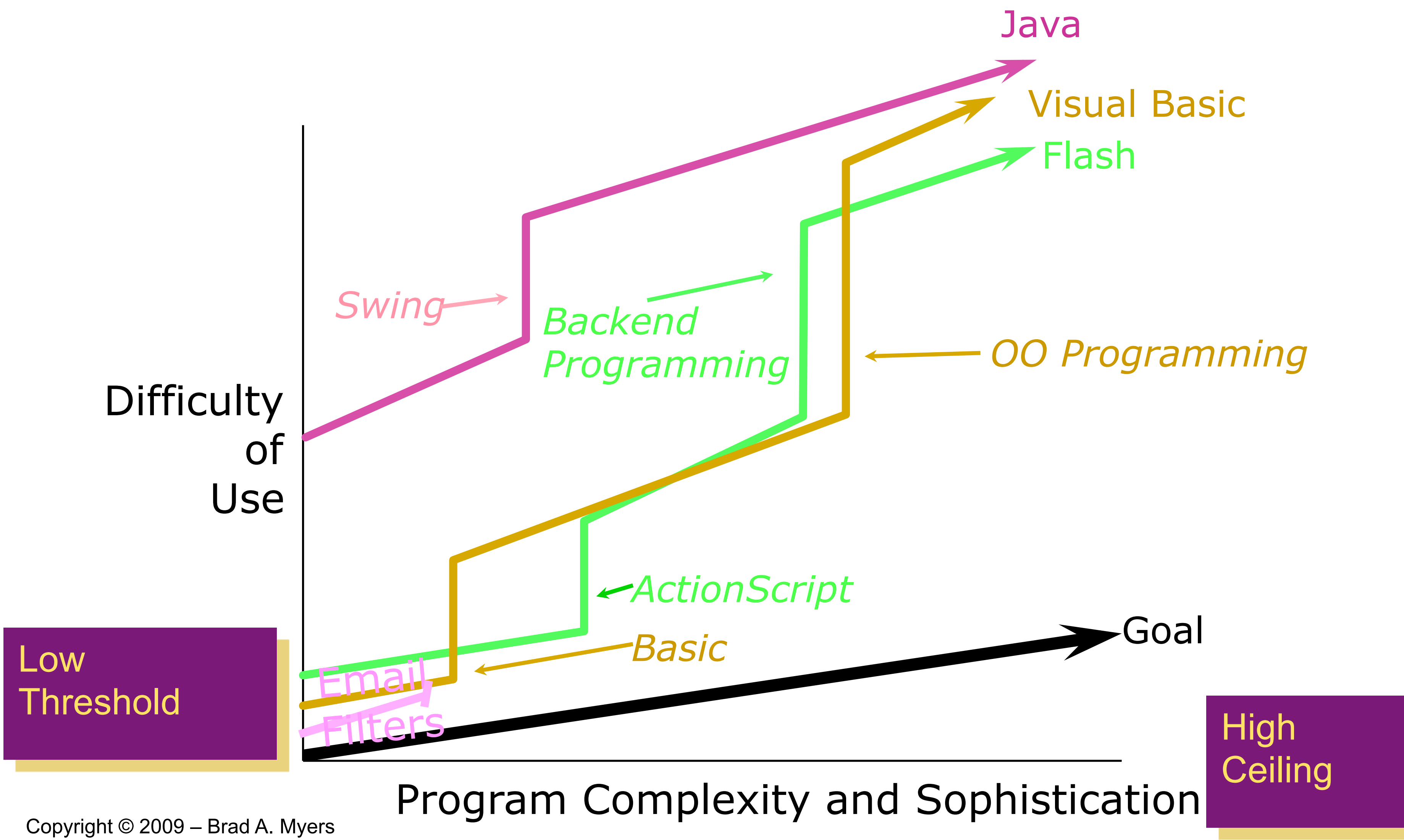
Principles

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**

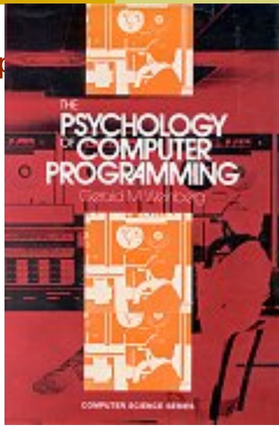
[Pane and Myers, 1996]

Goal: Gentle Slope Systems



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

Bennedsen, J., and M.E. Caspersen. 2007. Failure rates in introductory programming. SIGCSE Bull. 39(2): 32–36.

Bonar, J., and Soloway, E.. 1983. Uncovering principles of novice programming. In Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '83). ACM, New York, NY, USA, 10-13.

Bruckman, A. and Edwards, E. 1999. Should We Leverage Natural-Language Knowledge? Proceedings of CHI 99. New York: ACM Press, pp. 207-214.

Cordy, J.R. 1992. Hints on the Design of User Interface Language Features – Lessons from the Design of Turing. Languages for Developing User Interfaces. B. A. Myers. Boston, Jones and Bartlett Publishers: 329-340.

du Boulay, B. 1989. Some difficulties of learning to program. In E. Soloway & J. C. Spohrer, Eds., Studying the Novice Programmer, pp. 283-299. Hillsdale, NJ: Lawrence Erlbaum Associates.

Eisenberg, M., M. Resnick and F. Turbak. 1987. Understanding Proce- dures as Objects. Empirical Studies of Programmers: Second Workshop. G. M. Olson, S. Shepard and E. Soloway. Norwood, NJ, Ablex: 14-32.

Fitter, M.J. and T.R.G. Green. 1979. “When Do Diagrams Make Good Computer Languages?” International Journal of Man-Machine Studies 11: 235-261.

Galotti, K.M. and W.F. Ganong, III. 1985. What Non-Programmers Know About Programming: Natural Language Procedure Specification. International Journal of Man-Machine Studies 22: 1-10.

Good, J., Howland, K., and Nicholson, K. 2010. Young People's Descriptions of Computational Rules in Role-Playing Games: An Empirical Study. pp. 67-74, 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, 2010.

References

Green, T.R.G. 1990. The Nature of Programming. Psychology of Programming. J.-M. Hoc, T. R. G. Green, R. Samurçay and D. J. Gilmore. London, Academic Press: 21-44.

Green, T.R.G. and M. Petre. 1996. “Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework.” Journal of Visual Languages and Computing 7(2): 131-174.

Grice, H.P. 1975. Logic and Conversation. Syntax and Semantics III: Speech Acts. P. Cole and J. Morgan. New York, Academic Press.

Guzdial, M. 2011. Why is it so hard to learn to program? In Making Software: What really works and why we believe it, Andy Oram and Greg Wilson (eds), 111-121.

Hutchins, E. L., Hollan, J. D. and Norman, D. A. 1986. Direct Manipulation Interfaces. Hillsdale, NJ: Lawrence Erlbaum Associates.

Hoc, J.-M. and A. Nguyen-Xuan. 1990. Language Semantics, Mental Models and Analogy. Psychology of Programming. J.-M. Hoc, T. R. G. Green, R. Samurçay and D. J. Gilmore. London, Academic Press: 139-156.

Lewandowski, G., Bouvier, D. J., McCartney, R., Sanders, K. and Simon, B. 2007. Commonsense computing (episode 3): concurrency and concert tickets”. In Proceedings of the third international workshop on Computing education research (ICER '07). ACM, New York, NY, USA, 133-144.

Lewis, C. and G.M. Olson. 1987. Can Principles of Cognition Lower the Barriers to Programming? Empirical Studies of Programmers: Second Workshop. G. M. Olson, S. Sheppard and E. Soloway. Norwood, NJ, Ablex: 248-263.

Mayer, R. E. 1981. The Psychology of How Novices Learn Computer Programming. ACM Comput. Surv. 13, 1 (March 1981), 121-141.

References

McKeithen, K.B. 1981. “Knowledge Organization and Skill Differences in Computer Programmers.” Cognitive Psychology 13: 307-325.

Miller, L. A. 1981. "Natural language programming: Styles, strategies, and contrasts." IBM Systems Journal 29(2): 184–215.

Nardi, B.A. 1993. A Small Matter of Programming: Perspectives on End User Computing. Cambridge, MA, The MIT Press.

Newell, A. and S. K. Card. 1985. "The Prospects for Psychological Science in Human-Computer Interaction." Human-Computer Interaction. 1(3): 209-242.

Nielsen, J. 1994. Heuristic Evaluation. Usability Inspection Methods. J. Nielsen and R. L. Mack. New York, John Wiley & Sons: 25-62.

Pane, J. and Myers, B. 1996. Usability Issues in the Design of Novice Programming Systems, Carnegie Mellon University School of Computer Science Technical Report CMU-CS-96-132. and Human Computer Interaction Institute Technical Report CMU-HCII-96-101, August, 1996.

Pane, J. F., Ratanamahatana, C. A., and Myers, B. A. 2001. "Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems", International Journal of Human-Computer Studies (IJHCS). Special Issue on Empirical Studies of Programmers, vol. 54, no. 2, February 2001, pp. 237-264.

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. IEEE Computer, 16, 57}69.

Simon, B., P. Kinnunen, et al. 2010. Experience Report: CS1 for Majors with Media Computation. Paper presented at ACM Innovation and Technology in Computer Science Education Conference, June 26–30, in Ankara, Turkey.

References

Sloan, R.H., and P. Troy. 2008. CS 0.5: A better approach to introductory computer science for majors. Proceedings of the 39th SIGCSE technical symposium on computer science education: 271–275.

Soloway, E., J. Bonar, et al. 1983. Cognitive strategies and looping constructs: An empirical study. Communications of the ACM 26(11): 853–860.

Tew, A.E., W.M. McCracken, et al. 2005. Impact of alternative introductory courses on programming concept understanding. Proceedings of the first international workshop on computing education research: 25–35.

> "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