Learning Programming

SWE 795, Fall 2019
Software Engineering Environments
Today

• Part 1 (Lecture)(~40 mins)

• Part 2 (2 HW4 presentations)(36 mins)

• Break!

• Part 3 (3 HW4 presentations)(54 mins)

• Part 4 (Feedback and course evals)(20 mins)
Overview

• What makes learning programming hard?

• Tools & languages for learning programming
  • Simplify typing code
  • Understand program execution
  • Offer context-specific help
  • Motivate learning programming
What makes learning programming hard?

• What makes programming hard?
  • Is the challenge thinking computationally?
  • Or in understanding how to formally express computation in a programming language?

Slides partially adapted from Human Aspects of Software Development, Spring 2011, Lecture 11: How do people naturally think about computation? (Cyrus Omar)
Programming is difficult

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

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

Adapted from
Why is this hard?

• Conceiving a solution?
  • Q: Can people develop natural language solutions to programming problems?

• Formalizing the solution?
  • Q: Are languages & APIs intuitive?
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]
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*.
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.

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.” 😊

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

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

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

- 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

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

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]*
Is natural language programming a solution?

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]
Goal: Gentle Slope Systems

Minimalist Learning Theory

• Choose an action-oriented approach
  • Provide an immediate opportunity to act, encourage self-directed exploration & innovation, prioritize user’s goals over delivery of information

• Anchor the learning tool in the task domain
  • Use real tasks as instruction, organize instruction around task steps

• Support error recognition & recovery
  • Prevent mistakes when possible, provide error information that offers not only detection but ‘on-the-spot’ diagnosis & recovery

• Support reading to do, study, locate
  • Make instructions brief & self-contained to support different levels of engagement

Problem frames

• Developers approaching messy problem interpret it with a frame
• Imposes boundaries on what learners will consider
Simplify typing code

• If key barrier is syntax, reduce challenge of working with syntax
  • Reduce constructs in programming language
  • Simplify constructs in programming language
  • Eliminate possibility of syntax errors
Beginners All-Purpose Symbolic Instruction Code (BASIC, 1963)

- Support a subset of instructions & remove unnecessary syntax
- Offer rapid feedback through interpreted language
- Offer simplified statements w/ 3 parts: line number, operator, operands

---

<table>
<thead>
<tr>
<th>FORTRAN:</th>
<th>BASIC:</th>
</tr>
</thead>
<tbody>
<tr>
<td>do 30 i = 1, 10</td>
<td>100 FOR I = 1 TO 10</td>
</tr>
<tr>
<td>m = m + I</td>
<td>110 LET S = S + I</td>
</tr>
<tr>
<td>30 continue</td>
<td>120 NEXT I</td>
</tr>
</tbody>
</table>

Figure 2. A for loop to compute the sum of the numbers from 1 to 10 written in FORTRAN and BASIC.

J.G. Kemeny and T. Kurtz, Dartmouth College, 1963
LOGO (1967)

• Supports manipulating turtle to draw pictures
  • Move forward 10 spaces
  • Turn left 90 degrees
• Offers dialect of LISP with less punctuation
• Supports creating music, translating languages, and much more

Seymour Papert, MIT, 1967
Interacting with objects

- Enable users to create objects & rules on how objects behave

Figure 4. A view of the My Magic Castle courtyard. The user is creating the rule “Nicky should dance when it meets the horse.”

Structured editors

Alice 2, 2002
Understand program execution

- Execution of program is hidden
  - Forces novices to simulate execution of program
  - Novices may simulate execution incorrectly
- Offer novice programmers visibility into the current execution state of programs
ATARI 2600 BASIC (1979)

- Stack: displays expressions as evaluated, updating as cursor changes
- Variables: displays variables and values
Make programming concrete through micro-worlds: Karel

- Actors can only perform a few actions
- Include simulations that allow students to watch progress of actors
- Enables students to gain familiarity with control structures like conditionals & loops

Offer context-specific help

• Learners experience breakdowns & barriers that prevent progress on tasks
• Offer specific actions learners can take to make progress when they experience these
Stencils-based tutorials

- Compared to paper tutorials, enable students to complete tutorials 26% faster w/ fewer errors & less human assistance

Idea Garden

Fig. 2. Overview of our Codechella system, which is built upon the Online Python Tutor program visualization tool [10]. Here is a typical use case: a.) The user writes code in an ordinary Web browser, b.) runs their code and steps forward and backward through execution points, c.) sees a visualization of stack frames, variables, data structures, and pointers at each execution point, d.) clicks the “Start a Codechella session” button and sends a unique URL to a tutor or friend, and then e.) chats with other participants in the Codechella session while navigating the visualization and writing code together in-sync.

https://www.youtube.com/watch?v=CKdUMVNWkYM


Fig. 1. The OverCode user interface. The top-left panel shows the number of clusters, called stacks, and the total number of solutions visualized. The next panel down in the first column shows the largest stack, whereas the second column shows the remaining stacks. The third column shows the lines of code occurring in the cleaned solutions of the stacks together with their frequencies.

https://www.youtube.com/watch?v=6ov_82nxpbQ

Motivating novice programmers

- Typical intro CS courses have assignments about numeric tasks and data structures
- As novices begin to learn programming, unclear why it matters or what is possible
- How can novices be motivated to invest the effort necessary to learn programming?
Storytelling Alice

- Formative study of middle school girls
- Offer high-level animations & support of multiple scenes
- Offer characters & scenery that spark story ideas
- Offer story-based tutorial

<table>
<thead>
<tr>
<th>Storytelling Alice</th>
<th>Generic Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Say, think</td>
<td>Move</td>
</tr>
<tr>
<td>Play sound</td>
<td>Turn</td>
</tr>
<tr>
<td>Walk to, walk offscreen, walk</td>
<td>Roll</td>
</tr>
<tr>
<td>Move</td>
<td>Resize</td>
</tr>
<tr>
<td>Sit on, lie on</td>
<td>Play sound</td>
</tr>
<tr>
<td>Kneel</td>
<td>Move to</td>
</tr>
<tr>
<td>Fall down</td>
<td>Move toward</td>
</tr>
<tr>
<td>Stand up</td>
<td>Move away from</td>
</tr>
<tr>
<td>Straighten</td>
<td>Orient to</td>
</tr>
<tr>
<td>Look at, Look</td>
<td>Point at</td>
</tr>
<tr>
<td>Turn to face, turn away from</td>
<td>Set point of view to</td>
</tr>
<tr>
<td>Turn</td>
<td>Set pose</td>
</tr>
<tr>
<td>Touch, Keep Touching</td>
<td>Move at speed, turn at speed, roll at speed</td>
</tr>
</tbody>
</table>

Games: Gidget

Figure 1. The Gidget game, where learners first help a damaged robot fix its programs by debugging its code (shown above), then create their own programs after completing all the levels.

http://www.helpgidget.org/

Communities: Scratch

https://scratch.mit.edu/