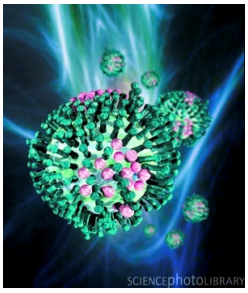
Lecture 27: Viruses and Object-Oriented Programming



Avian Flu Virus

CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

From Paul Graham's "Undergraduation":

My friend Robert learned a lot by writing network software when he was an undergrad. One of his projects was to connect Harvard to the Arpanet; it had been one of the original nodes, but by 1984 the connection had died. Not only was this work not for a class, but because he spent all his time on it and neglected his studies, he was kicked out of school for a year.
... When Robert got kicked out of grad school for writing the Internet worm of 1988, I envied him enormously for finding a way out without the stigma of failure.
... It all evened out in the end, and now he's a professor at MIT. But you'll probably be happier if you don't go to that extreme; it caused him a lot of worry at the time.

3 years of probation, 400 hours of community service, $10,000+ fine

Morris Internet Worm (1988)

- *P* = fingerd
  - Program used to query user status
  - Worm also attacked other programs
- *I* = "nop^{400} pushl $68732f pushl $6e69622f movl sp,r10 pushl $0 pushl $0 pushl r10 pushl $3 movl sp,ap chmk $3b"
  (is-worm? '(P I)) should evaluate to #t
- Worm infected several thousand computers (~10% of Internet in 1988)

Worm Detection Problem

Input: A program *P* and input *I*

Output: **true** if evaluating (P I) would cause a remote computer to be "infected".

Virus Detection Problem

Input: A program specification *P*

Output: **true** if evaluating (P) would cause a file on the host computer to be "infected".

Uncomputability Proof

Suppose we could define is-virus? Then:

```
(define (halts? P)
  (is-virus?
    '(lambda ()
       (begin ((remove-infects P))
              (infect-files)))))
```


Uncomputability Proof

```
(define (halts? P)
  (is-virus?
    '(lambda ()
       (begin ((remove-infects P))
              (infect-files)))))
```

#t: Since it *is* a virus, we know (infect-files) was evaluated, and *P* must halt.

Can we make ***remove-infects***?

#f: The (infect-files) would not evaluate, so *P* must not halt.

Yes, just remove all file writes.

"Solving" Undecidable Problems

- No perfect solution exists:
  - Undecidable means there is no procedure that:
    - Always gives the correct answer
    - Always terminates
- Must give up one of these to "solve" undecidable problems
  - Giving up #2 is not acceptable in most cases
  - Must give up #1
- Or change the problem: e.g., detect file infections during an execution

Conclusion?

- Anti-Virus programs cannot exist!

Actual is-virus? Programs

- Give the wrong answer sometimes
  - "False positive": say P is a virus when it isn't
  - "False negative": say P is safe when it is
- Database of known viruses: if P matches one of these, it is a virus
- Clever virus authors can make viruses that change each time they propagate
  - Emulate program for a limited number of steps; if it doesn't do anything bad, assume it is safe

Proof Recap

- If we had is-virus? we could define halts?
- We know halts? is undecidable
- Hence, we can't have is-virus?
- Thus, we know is-virus? is undecidable

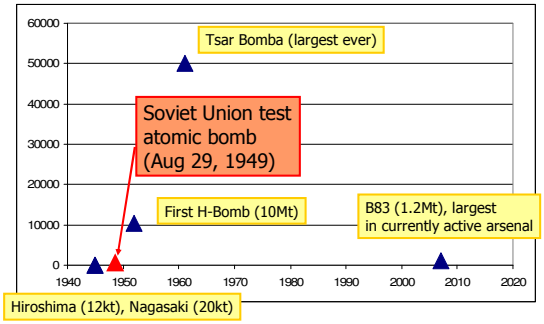
History of Object-Oriented Programming

Pre-History: MIT's Project Whirlwind (1947-1960s)

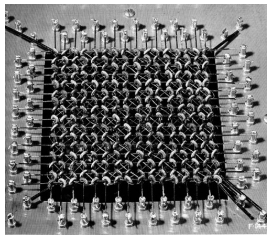
Jay Forrester

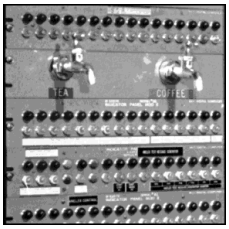
Why Whirlwind?

from Class 2...



Whirlwind Innovations





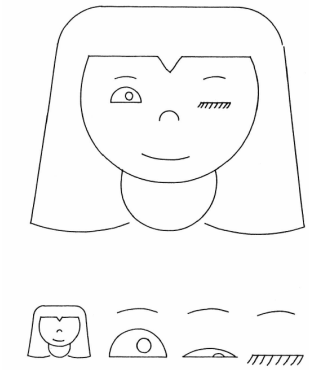
Magnetic Core Memory
(first version used vacuum tubes)

Sketchpad

- Ivan Sutherland, 1963 (PhD thesis supervised by Claude Shannon)
- Interactive drawing program
- Light pen

http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-574.pdf

Components in Sketchpad



Objects in Sketchpad

In the process of making the Sketchpad system operate, a few very general functions were developed which make no reference at all to the specific types of entities on which they operate. These general functions give the Sketchpad system the ability to operate on a wide range of problems. The motivation for making the functions as general as possible came from the desire to get as much result as possible from the programming effort involved. For example, the general function for expanding instances makes it possible for Sketchpad to handle *any* fixed geometry subpicture. The rewards that come from implementing general functions are so great that the author has become reluctant to write any programs for specific jobs.

Each of the general functions implemented in the Sketchpad system abstracts, in some sense, some common property of pictures independent of the specific subject matter of the pictures themselves.

Ivan Sutherland, *Sketchpad: a Man-Machine Graphical Communication System*, 1963 (major influence on Alan Kay developoing OOP in 1970s)

Simula

- Considered the first "object-oriented" programming language
- Language designed for *simula*tion by Kristen Nygaard and Ole-Johan Dahl (Norway, 1962)
- Had special syntax for defining classes that packages state and procedures together

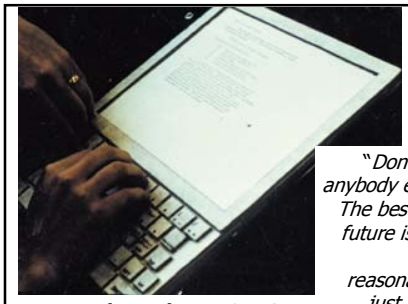
Counter in Simula

```
class counter;
  integer count;
  begin
    procedure reset(); count := 0; end;
    procedure next();
        count := count + 1; end;
    integer procedure current();
        current := count; end;
  end
```


XEROX Palo Alto Research Center (PARC)

1970s:

- Bitmapped display
- Graphical User Interface
  - Steve Jobs paid $1M to visit and PARC, and returned to make Apple Lisa/Mac
- Ethernet
- First personal computer (Alto)
- PostScript Printers
- **Object-Oriented Programming**



Dynabook, 1972

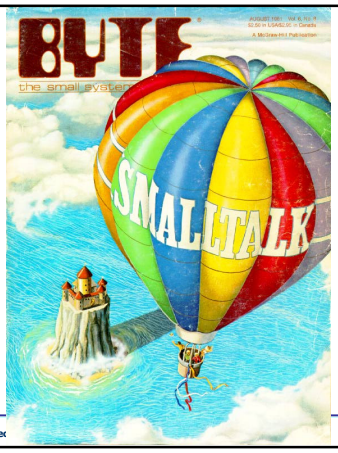
(Just a model)

"Don't worry about what anybody else is going to do… The best way to predict the future is to invent it. Really smart people with reasonable funding can do just about anything that doesn't violate too many of Newton's Laws!"

— Alan Kay, 1971

Dynabook 1972

- Tablet computer
- Intended as tool for learning
- Kay wanted children to program it also
- Hallway argument, Kay claims you could define "the most powerful language in the world in a page of code"
- Proof: Smalltalk
  - Scheme is as powerful, but takes two pages
  - Before the end of the class, we will see an equally powerful language that fits in ¼ page



BYTE Magazine, August 1981

Smalltalk

- Everything is an *object*
- Objects communicate by sending and receiving *messages*
- Objects have their own state (which may contain other objects)
- How do you do 3 + 4?

  send the object **3** the message **"+ 4"**

Counter in Smalltalk

```
class name counter
    instance variable names count
    new count <- 0
    next count <- count + 1
    current ^ count
```


Counter in Python

```
class counter:
    def __init__(self): self._count = 0
    def rest(self): self._count = 0
    def next(self): self._count = self._count + 1
    def current(self): return self._count
```

counter() creates a new counter using the __init__ method
count is the instance variable (is just a naming convention)

Who was the first object-oriented programmer?

By the word operation, we mean any process which alters the mutual relation of two or more things, be this relation of what kind it may. This is the most general definition, and would include all subjects in the universe. Again, it might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine... Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent.

Ada, Countess of Lovelace, around 1843

Charge

- Chapter 12 and PS7 out now: start reading Chapter 12 before next class
- The statement "There will be a surprise quiz some day this week" may still be true
- Wednesday: one more computability problem ("the AliG problem"); interpreters and Python