

history of programming, part II

thursday 10 august 2000

lecture overview:

- John McCarthy (1979)
- David Parnas (1972)
- Edgar Codd (1970)
- THE assignment

1

NATO 1968 conference

- came from panic in response to “software crisis”
- inability to design and deliver clean code on time in large projects
- ordinary techniques (of the day) failed to scale up
 - flow charts
 - functional subroutines
- caused people working in the field to rethink the standard models
- brought about software engineering movement (later in the term)
- today: Lisp, OOP, RDBMS

2

John McCarthy

- inventor of Lisp
- MIT, then Stanford University
- “History of Lisp” (1979)
- referenced by Backus (as one of the good guys)
- work began in the 1950's at IBM as FLPL (FORTRAN List Processing Language)
- implemented in 1962
- Lisp is the second oldest programming language in widespread use today (FORTRAN is oldest)

3

McCarthy: characterization of Lisp

- computes with symbolic expressions rather than numbers
- represents symbolic expressions and other information by list structure in the memory of a computer
- represents information in external media mostly by multi-level lists and sometimes by s-expressions
- contains a small set of selector and constructor operations expressed as functions
- allows composition of functions as a tool for forming more complex functions
- uses conditional expressions for getting branching into function definitions
- uses recursive conditional expressions as a sufficient tool for building computable functions
- uses λ -expressions for naming functions
- represents Lisp programs as Lisp data
- allows conditional expression interpretation of Boolean connectives
- contains *eval* function that serves both as a formal definition of the language and as an interpreter
- provides garbage collection for erasure

4

Review of Backus' criticisms of von Neumann languages

- primitive word-at-a-time style of programming
- close coupling of semantics to state transitions
- division of programming into a world of expressions and a world of statements
- inability to effectively use powerful combining forms for building new programs from existing ones
- lack of useful mathematical properties for reasoning about programs

5

How does Lisp hold up to Backus' criticisms?

- applicative models:
 - foundations: concise and useful
 - history sensitivity: no storage, not history sensitive
 - type of semantics: reduction semantics, no states
 - program clarity: programs can be clear and conceptually useful
- Lisp is an applicative model (good)
- but, "pure Lisp is often buried in large extensions with many von Neumann features" (Backus)

6

Moving right along...

and the next major shift in programming methodology was...

⇒ object-oriented programming (OOP)

- began with Parnas (1972)
- attempting to answer question: why is Edgar Dijkstra's code so different (better)?
- popular OOP languages today:
 - C++
 - Java

7

OOP methodology

two ingredients:

- modularity
- inheritance

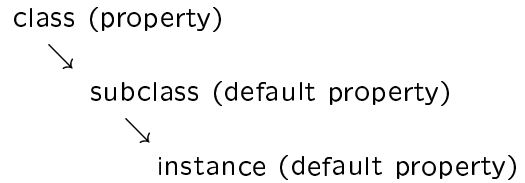
8

OOP: modularity

- traditional code modules
 - based on sequential chunks of processing
 - i.e., flow of control = flow charts
- Parnas' model
 - based on “information hiding”
 - ⇒ *data abstraction*
 - each module hides its design decisions
 - especially, each data structure hides:
 - ◊ representation syntax
 - ◊ access details
 - ◊ processing sequences

9

OOP: inheritance



- origin:
 - AI: Minsky's frames
 - simulation languages

10

Here's another way of thinking about it

if programming is comprised of NOUNS and VERBS...

- old style:
 - define VERBS to operate on NOUNS
- OOP:
 - define NOUNS, with a set of VERBS to operate on them

NOUNS ≡ OBJECTS

VERBS ≡ METHODS

11

Parnas: A Tale of Two Modularizations

- task: KWIC index production system

12

Parnas: changeability

- input format
- data storage decisions
- indexing decisions
- algorithmic decisions

13

Parnas: criteria

- a data structure, its internal linkings, accessing procedures and modifying procedures are part of a single module
- the sequence of instructions necessary to call a given routine and the routine itself are part of the same module
- the formats of control blocks used in queues in operating systems, etc. must be hidden within a control block module
- character codes, alphabetic orderings, etc. should be hidden in a module for greatest flexibility
- processing sequences should be hidden within a single module

14

Parnas: efficiency

- be careful!

15

Edgar Codd (1970)

- page 378 is out of order!!
(in green book)
- introduced RDBMS
(relational database management system)
- worked at IBM
- SQL also developed at IBM
- with RDBMS, again:
 - separate the NOUNS (OBJECTS)
 - get away from the FLOW CHART
 - *promote data independence*
- other models:
 - hierarchical
 - spreadsheet

16

Codd: achieving true data independence

- data description tables
 - a major advance
- need to be able to change data representation without necessarily changing the programs that access that data

in other words:

- independent ordering
- independent indexing
- independent access paths

“The universality of the data sublanguage lies in its descriptive ability (not its computing ability).”

17

Codd: relational model

- relation R
- domain
- degree
- relationship
- primary key
- foreign key
- exploitation

18

Codd: operations

- normalization
- permutation
- projection
- join

19

for next lecture:

there are 4 HANDOUTS today!!!

- Hertz, J., Krogh, A. and Palmer, R. (1991) Introduction to the Theory of Neural Computation. Sydney: Addison-Wesley Publishing Company. *Selected sections from chapters 1 and 5.* (handout #1)

- read Braitenberg, V. (1984) Vehicles: Experiments in Synthetic Psychology. Cambridge, MA: The MIT Press. *Introduction, vehicles 1 and 2.* (handout #2)

- read Turing, A. (1950) Computing Machinery and Intelligence. *Mind*, vol. 59, pp. 433-460. (in green book)

- read Azimov, A. (1950) I, Robot. London: Voyager. *Introduction and Robbie.* (handout #3)

- the assignment (handout #4)

20