open.michigan

Author(s): August E. Evrard, PhD. 2010

Citation Key

for more information see: http://open.umich.edu/wiki/CitationPolicy

Use + Share + Adapt

{ Content the copyright holder, author, or law permits you to use, share and adapt. }

PD-GOV **Public Domain – Government**: Works that are produced by the U.S. Government. (17 USC § 105)

PD-EXP **Public Domain – Expired**: Works that are no longer protected due to an expired copyright term.

PD-SELF **Public Domain – Self Dedicated**: Works that a copyright holder has dedicated to the public domain.

ZERO **Creative Commons – Zero Waiver**

BY **Creative Commons – Attribution License**

BY-SA **Creative Commons – Attribution Share Alike License**

BY-NC **Creative Commons – Attribution Noncommercial License**

BY-NC-SA **Creative Commons – Attribution Noncommercial Share Alike License**

GNU-FDL **GNU – Free Documentation License**

Make Your Own Assessment

{ Content Open.Michigan believes can be used, shared, and adapted because it is ineligible for copyright. }

PD-INEL **Public Domain – Ineligible**: Works that are ineligible for copyright protection in the U.S. (17 USC § 102(b)) *laws in your jurisdiction may differ

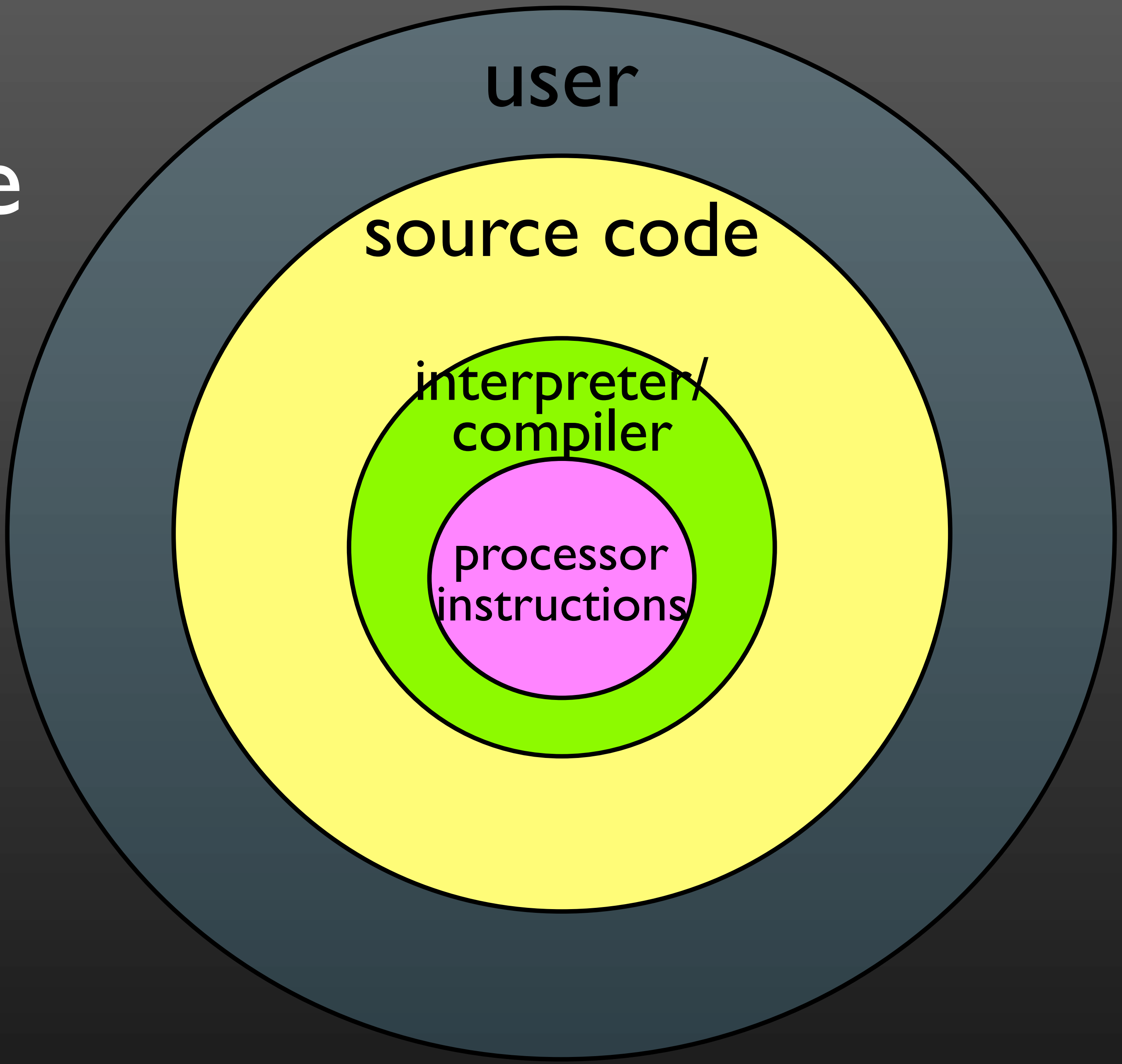
{ Content Open.Michigan has used under a Fair Use determination. }

FAIR USE **Fair Use**: Use of works that is determined to be Fair consistent with the U.S. Copyright Act. (17 USC § 107) *laws in your jurisdiction may differ

Our determination **DOES NOT** mean that all uses of this 3rd-party content are Fair Uses and we **DO NOT** guarantee that your use of the content is Fair.

To use this content you should **do your own independent analysis** to determine whether or not your use will be Fair.

Cyberscience: Computational Science and the Rise of the Fourth Paradigm



Honors 352, Class #0.10

August E. (Gus) Evrard, PhD

Fall 2010

University of Michigan

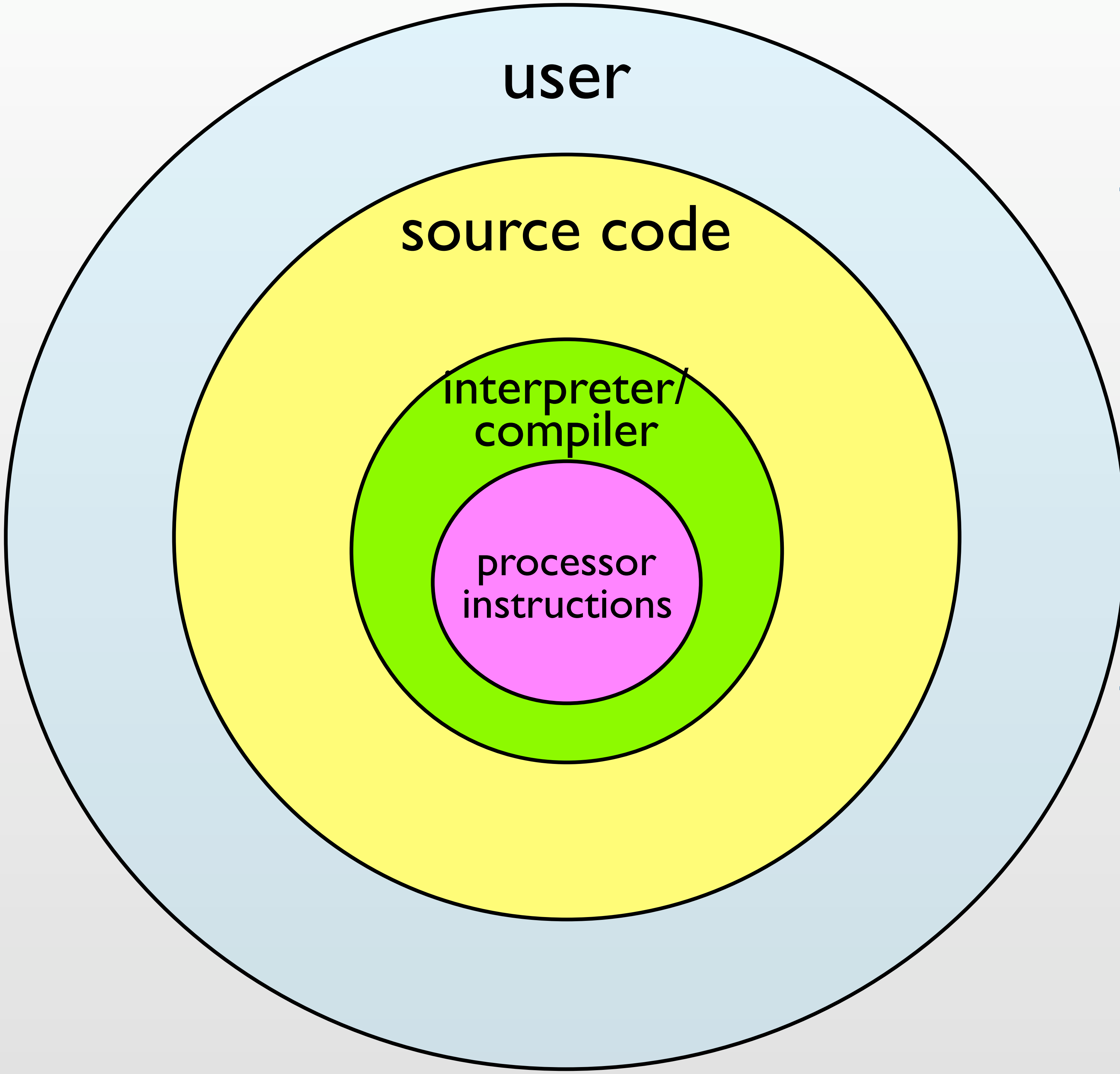
today

* lecture: a brief history of programming

* processing lab review + voting

* reading quiz on-line (timed 30 min, open 9am-11:59pm)

software as human-cpu interface



– user needs to instruct cpu

1940's: write processor (assembly) code directly

– `high-level' codes emerge to offer layers

1950's: FORTRAN, BASIC

compiler translates

human-readable code into machine instructions

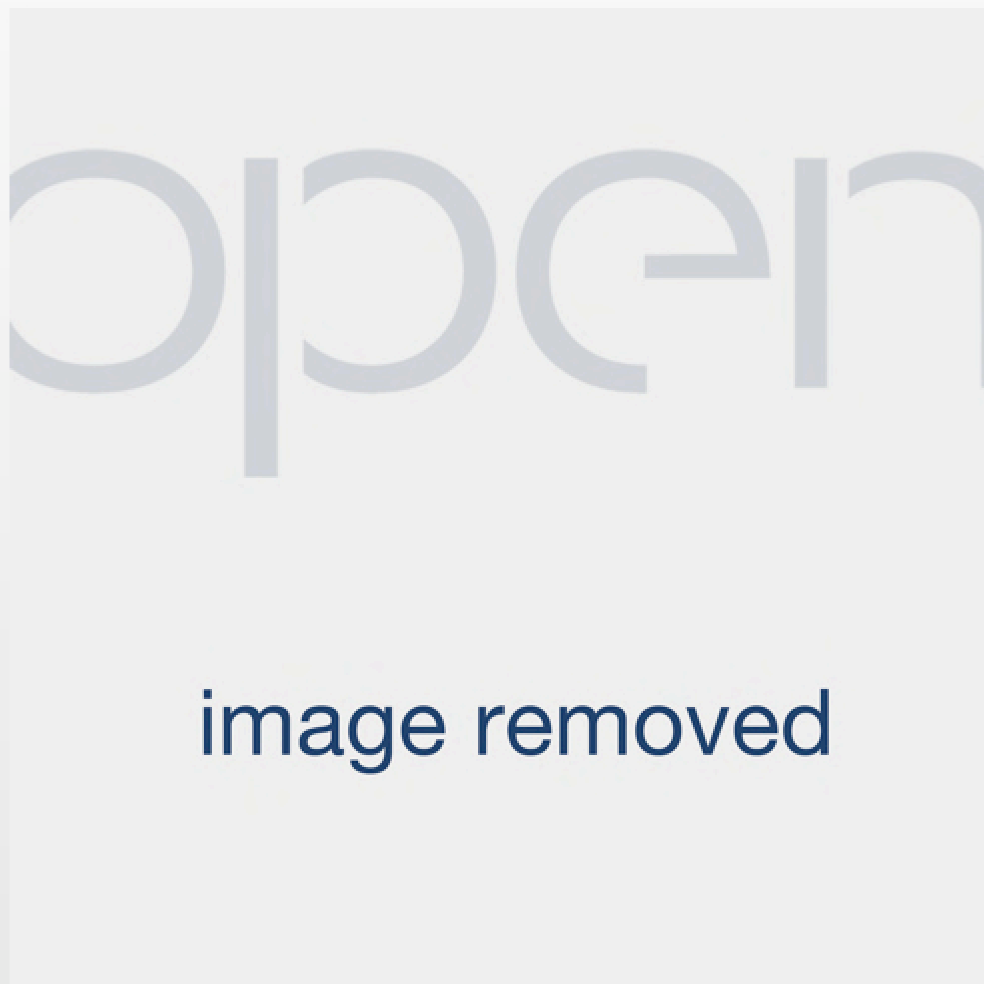
– time sharing /multi-tasking

1960's: MIT's CTSS

interleave tasks from multiple users

memory and i/o device management

1962: Michigan's MAD compiler



Please see original quote regarding the Michigan Algorithm Decoder at http://www.multicians.org/thvv/7094.html.

unix is born

* 1964: Multics (Multiplexed Information and Computing Service) is overambitious failure

* K. Thomson and Dennis Ritchie (Bell Labs) build UNICS (Uniplexed Information and Computing Services) with a decidedly anti-bloat perspective,

"build small neat things instead of grandiose ones."

four original elements: *kernel, shell, editor, assembler*

Please see original image of Bell logos at
http://www.porticus.org/bell/bell_logos.html

* early 1970's: Unix spreads within AT&T

* founding philosophy

- Write programs that do one thing and do it well.
- Write programs that work together.
- Write programs that handle text streams because that is a universal interface.

Success of Open Source, p. 26-28

linux and open source software

* 1978: Bill Joy (Sun Microsystems founder) packages Berkeley Software Distribution (BSD) unix
  – enhanced editor (*ex*) and Pascal compiler
  – sends 30 free copies to other universities, labs

News spread around the Unix community about Bill Joy's Pascal system. As requests for the software arrived at Berkeley, Joy put together a package of tools and utilities he called the Berkeley Software Distribution, which later came to be known as BSD. Joy sent out in 1978 about thirty free copies of the software, which included the Pascal system and the ex text editor. As Peter Salus says, the essential elements of a collaborative culture as well as a primitive mechanism for software sharing and creation were now in place:

> Something was created at BTL. It was distributed in source form. A user in the UK created something from it. Another user in California improved on both the original and the UK version. It was distributed to the community at cost. The improved version was incorporated into the next BTL release.

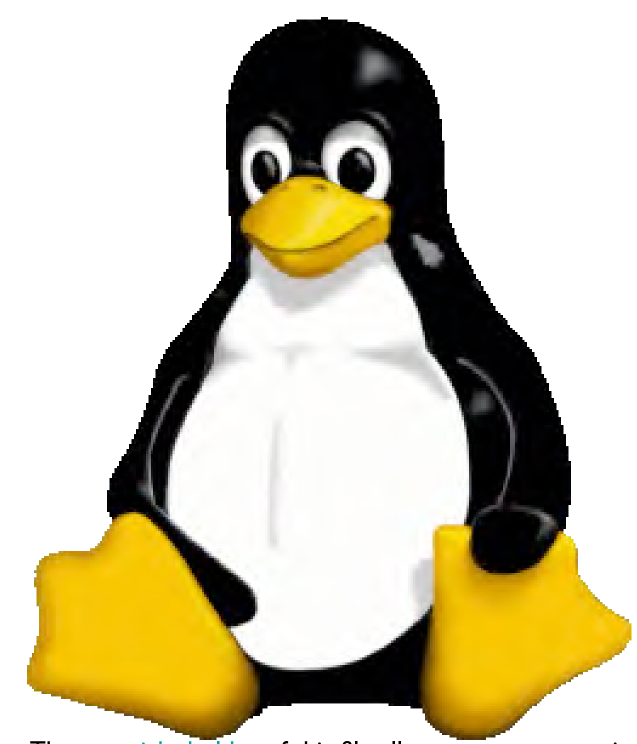
 Success of Open Source, pg 31.

* 1980's: AT&T vs. BSD unix
  $100,000 annual license vs. "free" ???
* 1988: Open Software Foundation (OSF)
  fails to overcome AT&T monopoly but seeds cooperative era
* 1991: Linus Torvalds releases **linux**

shared programming development model

voluntary participation and voluntary selection of tasks

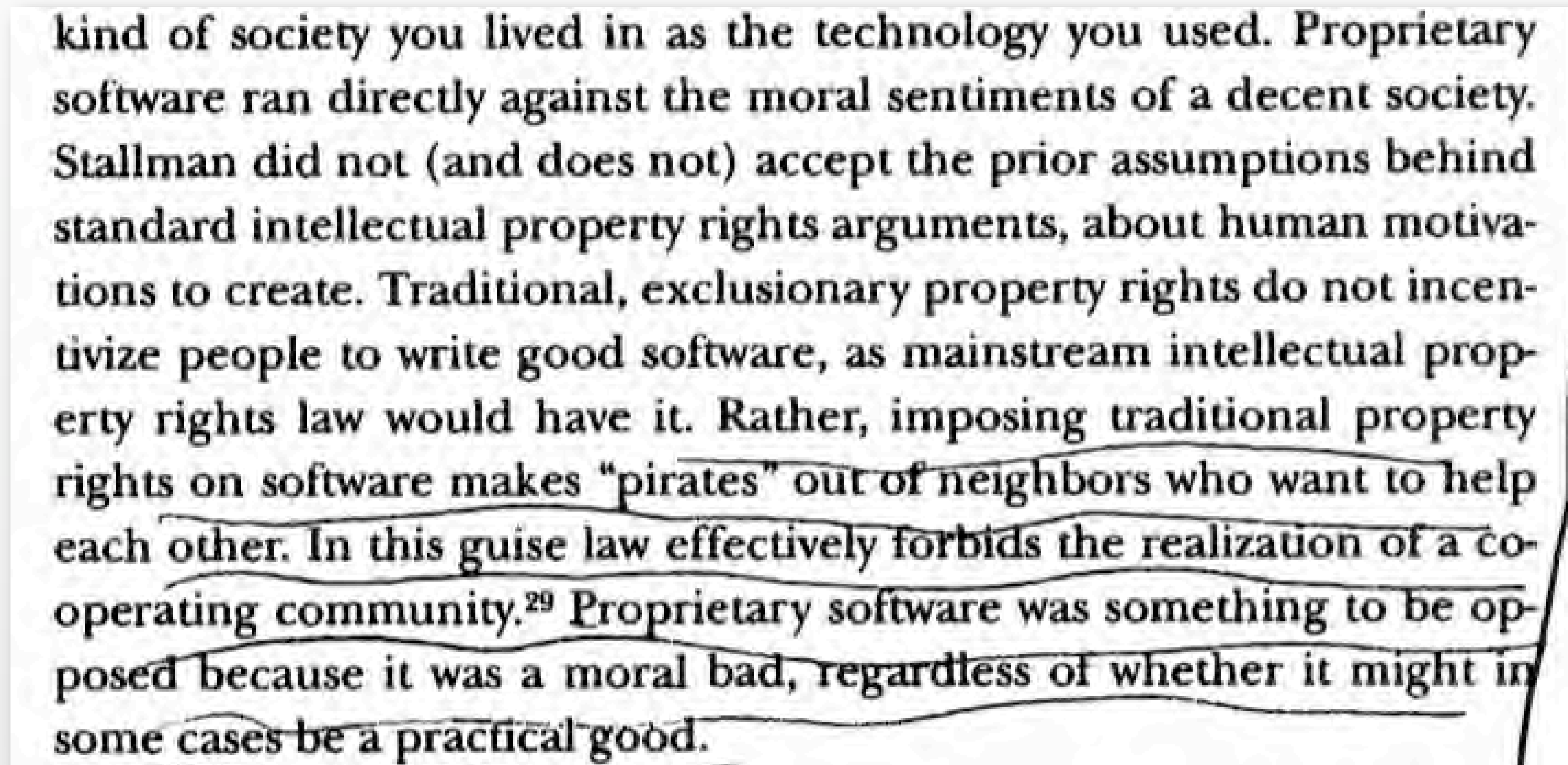
1994: v1.0 release (today: v2.6.36 http://www.linux.org/)



free software foundation and GPL

* 1984: Richard Stallman resigns from MIT, partly over inability to fix a XEROX printer

  – starts backlash against proprietary software



kind of society you lived in as the technology you used. Proprietary software ran directly against the moral sentiments of a decent society. Stallman did not (and does not) accept the prior assumptions behind standard intellectual property rights arguments, about human motivations to create. Traditional, exclusionary property rights do not incentivize people to write good software, as mainstream intellectual property rights law would have it. Rather, imposing traditional property rights on software makes "pirates" out of neighbors who want to help each other. In this guise law effectively forbids the realization of a cooperating community.[29] Proprietary software was something to be opposed because it was a moral bad, regardless of whether it might in some cases be a practical good.

 Please see original quote from the book, Success of Open Source, pg. 47.

* Stallman establishes Free Software Foundation

  builds GNU = GNU's Not Unix

* GNU Public License (GPL)

  copyleft : codes derived from GPL'ed code must also be GPL'ed

  free software `infects' other software with its licensing terms

GNU meaning of “free”

The Free Software Definition

We maintain this free software definition to show clearly what must be true about a particular software program for it to be considered free software. From time to time we revise this definition to clarify it. If you would like to review the changes we've made, please see the History section below for more information.

“Free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.”

Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it means that the program's users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

software encodes algorithms

* scientific computing has many common tasks

    sorting

    matrix inversion

    special function evaluation

    solving roots of equations

    spectral analysis (Fast Fourier Transform)

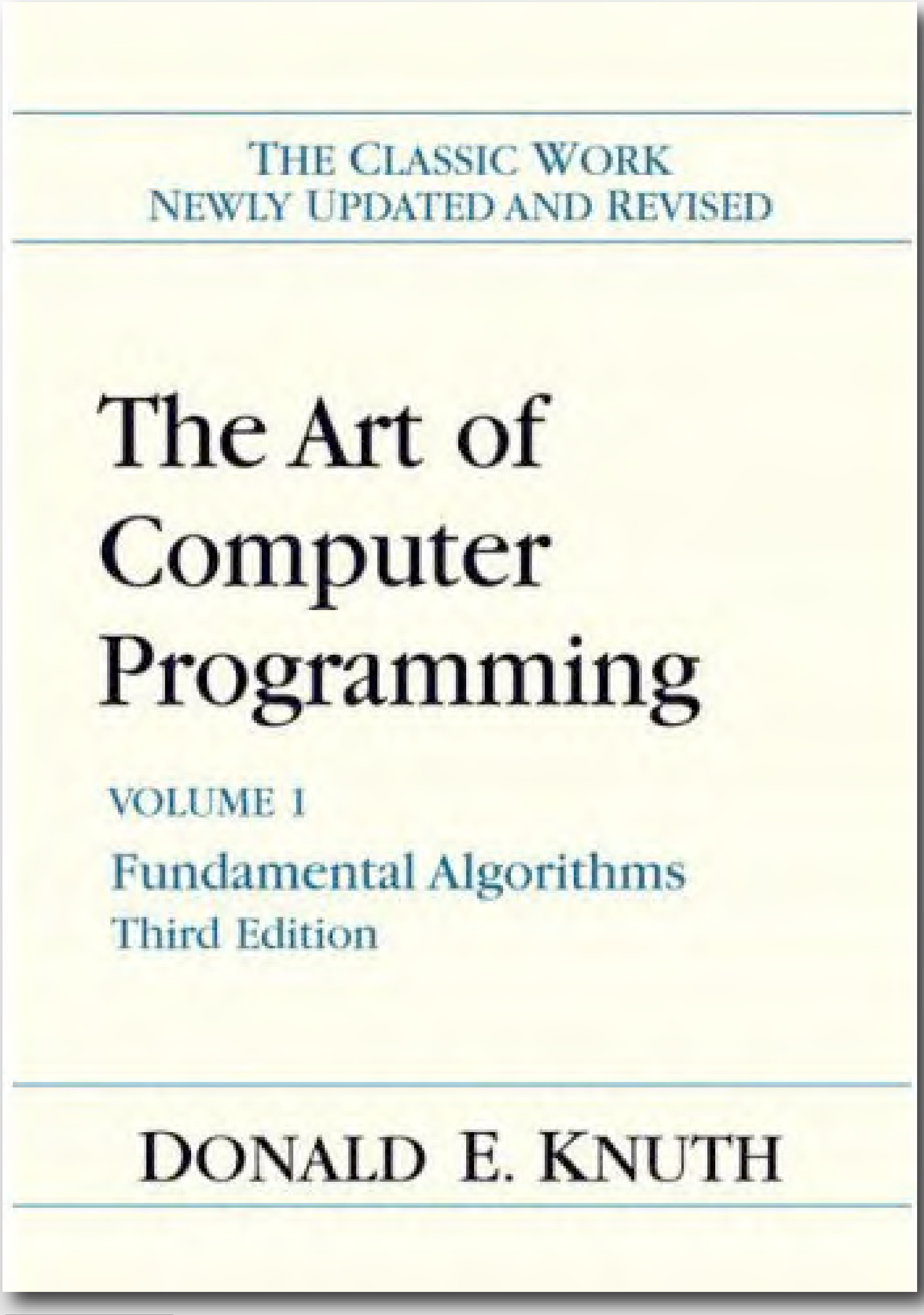
    ...

good software encodes these functions in a modular (flexible) manner

* practical implementations vary in style

    choice of programming language

    speed vs. accuracy constraints



The Art of Programming is in mapping the algorithm to a particular situation in a manner that maximizes return while minimizing cost.

imperative vs. functional styles

Coding styles

[edit]

Imperative programs tend to emphasize the series of steps taken by a program in carrying out an action, while functional programs tend to emphasize the composition and arrangement of functions, often without specifying explicit *steps*. A simple example illustrates this with two solutions to the same programming goal (calculating Fibonacci numbers) using the same multi-paradigm language Python.

```
# Fibonacci numbers, imperative style
N=10

first = 0     # seed value fibonacci(0)
second = 1    # seed value fibonacci(1)
fib_number = first + second     # calculate fibonacci(2)
for position in range(N-2):     # iterate N-2 times to give Fibonacci number N (for N > 2)
    first = second              # update the value of the two 'previous' variables
    second = fib_number
    fib_number = first + second # update the result value to fibonacci(position)
print fib_number
```

A functional version has a different feel to it:

```
# Fibonacci numbers, functional style
def fibonacci(N):  # Fibonacci number N (for N >= 0)
    if N <= 1: return N    # base cases
    else: return fibonacci(N-1) + fibonacci(N-2)   # recursively calculate fibonacci(N)

print fibonacci(10)
```

The imperative style describes the intermediate steps involved in calculating `fibonacci(N)`, and places those steps inside a loop statement. In contrast, the functional style describes the mathematical equation that defines a `fibonacci(N)` number with respect previous numbers in the Fibonacci sequence, where intermediate calculation steps are calculated using recursion.

Table of Contents

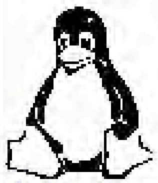
David A Rusling
3 Foxglove Close,
Wokingham,
Berkshire RG41 3NF,
United Kingdom

Show Frames, No Frames

david.rusling@arm.com

Table of Contents, Show Frames, No Frames

Chapter 2
Software Basics



A program is a set of computer instructions that perform a particular task. That program can be written in assembler, a very low level computer language, or in a high level, machine independent language such as the C programming language. An operating system is a special program which allows the user to run applications such as spreadsheets and word processors. This chapter introduces basic programming principles and gives an overview of the aims and functions of an operating system.

2.1 Computer Languages

2.1.1 Assembly Languages

The instructions that a CPU fetches from memory and executes are not at all understandable to human beings. They are machine codes which tell the computer precisely what to do. The hexadecimal number *0x89E5* is an Intel 80486 instruction which copies the contents of the ESP register to the EBP register. One of the first software tools invented for the earliest computers was an assembler, a program which takes a human readable source file and assembles it into machine code. Assembly languages explicitly handle registers and operations on data and they are specific to a particular microprocessor. The assembly language for an Intel X86 microprocessor is very different to the assembly language for an Alpha AXP microprocessor. The following Alpha AXP assembly code shows the sort of operations that a program can perform:

```
    ldr r16, (r15)    ; Line 1
    ldr r17, 4(r15)   ; Line 2
    beq r16,r17,100   ; Line 3
    str r17, (r15)    ; Line 4
100:                  ; Line 5
```

The first statement (on line 1) loads register 16 from the address held in register 15. The next instruction loads register 17

tiobe.com ranking of computer languages

Ratings

The ratings are calculated by counting hits of the most popular search engines. The search query that is used is

+"<language> programming"

This search query is executed for the top 6 websites of Alexa that meet the following conditions:

* The entry page of the site contains a search facility
* The result of querying the site contains an indication of the number of page hits

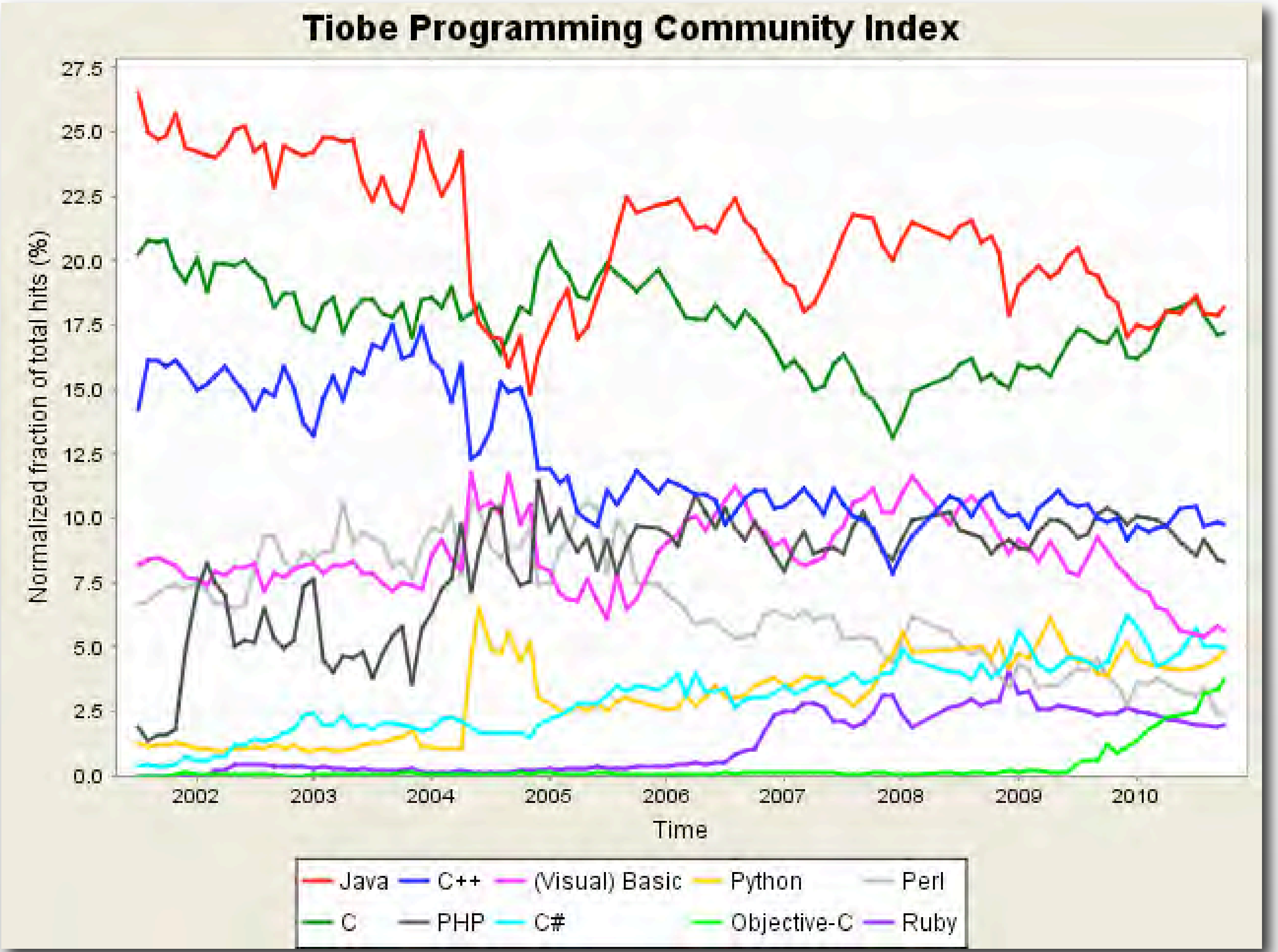
Based on these criteria currently Google, YouTube, Yahoo!, Live, Wikipedia and Blogger are used as search engines. Baidu should be part of this well but the TIOBE index calculator is not capable yet of dealing with Chinese characters. This facility will be added soon.

The number of hits determine the ratings of a language.

Position Oct 2010	Position Oct 2009	Delta in Position	Programming Language	Ratings Oct 2010	Delta Oct 2009	Status
1	1	=	Java	18.166%	-0.48%	A
2	2	=	C	17.177%	+0.33%	A
3	4	↑	C++	9.802%	-0.08%	A
4	3	↓	PHP	8.323%	-2.03%	A
5	5	=	(Visual) Basic	5.650%	-3.04%	A
6	6	=	C#	4.963%	+0.55%	A
7	7	=	Python	4.860%	+0.96%	A
8	12	↑↑↑↑	Objective-C	3.706%	+2.54%	A
9	8	↓	Perl	2.310%	-1.45%	A
10	10	=	Ruby	1.941%	-0.51%	A
11	9	↓↓	JavaScript	1.659%	-1.37%	A
12	11	↓	Delphi	1.558%	-0.58%	A
13	17	↑↑↑↑	Lisp	1.084%	+0.48%	A-
14	24	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	0.820%	+0.42%	A-
15	15	=	Pascal	0.771%	+0.10%	A-
16	18	↑↑	RPG (OS/400)	0.708%	+0.12%	A-
17	29	↑↑↑↑↑↑↑↑↑↑	Ada	0.704%	+0.40%	A--
18	14	↓↓↓↓	SAS	0.664%	-0.14%	B
19	19	=	MATLAB	0.627%	+0.05%	B
20	-	↑↑↑↑↑↑↑↑↑↑	Go	0.626%	+0.63%	B

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

tiobe.com rank trends



http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

growth of open source is ~exponential

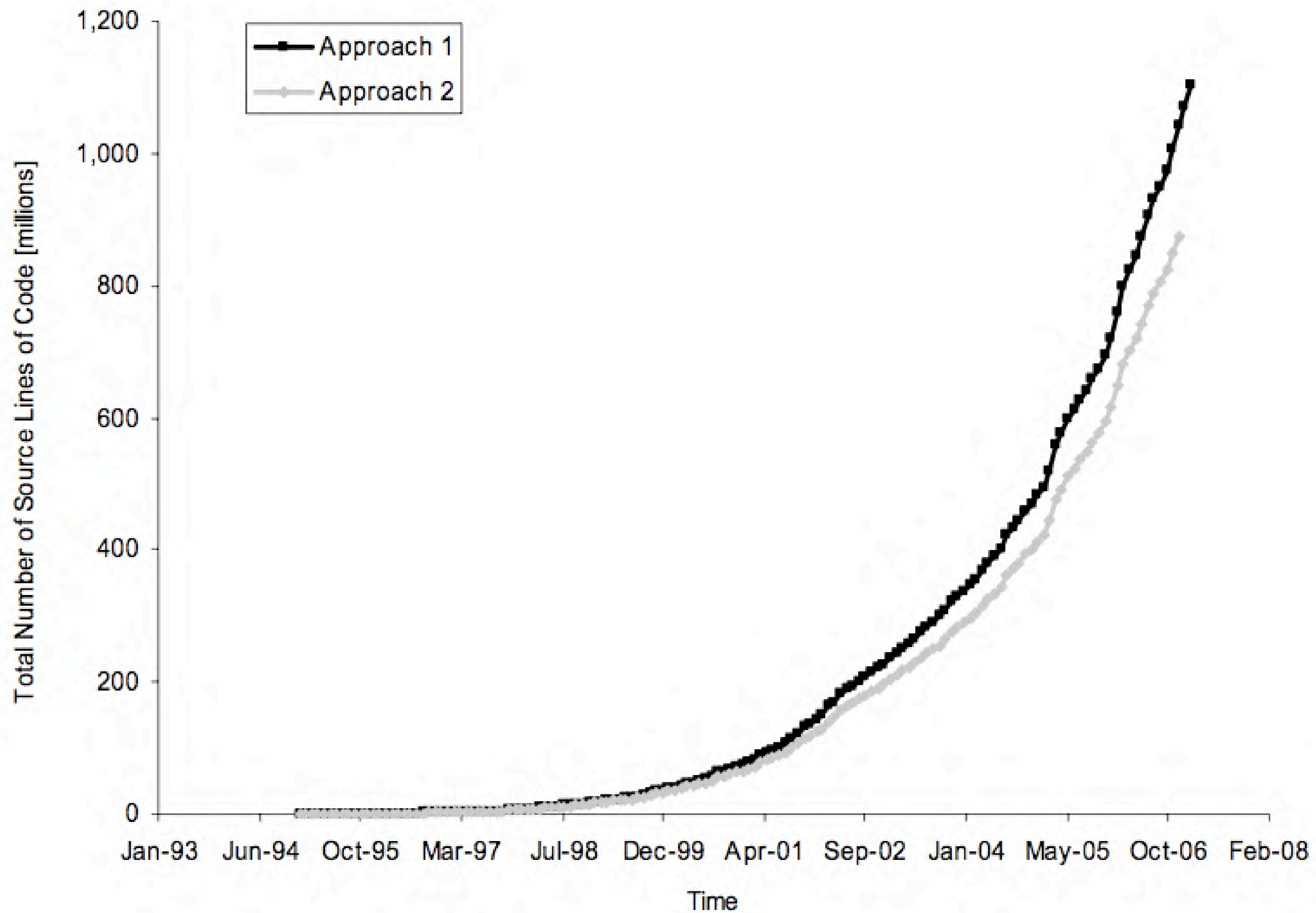


Figure 3: Graph of total source lines of code [millions] (both approaches)

 Amit Deshpande, Dirk Riehle, dirkriehle.com

~1999 list from UM CIS 400

 http://groups.engin.umd.umich.edu/CIS/course.des/cis400/

Additional Source Information

for more information see: http://open.umich.edu/wiki/CitationPolicy

Slide 3: A. E. Evrard, University of Michigan

Slide 5: A. E. Evrard, University of Michigan

Slide 6: Please see original quote regarding the Michigan Algorithm Decoder at http://www.multicians.org/thvv/7094.html.

Slide 7: Please see original image of Bell logos at http://www.porticus.org/bell/bell_logos.html

Slide 8, Quote (top): Success of Open Source, by Steve Weber, Harvard University Press, 2004
Slide 8, Image (bottom): LINUX, http://www.isc.tamu.edu/~lewing/linux/. The copyright holder of this file allows anyone to use it for any purpose, provided that one acknowledges lewing@isc.tamu.edu and The GIMP

Slide 9, Image 1 (top): Please see original quote from the book, Success of Open Source, by Steve Weber, Harvard University Press, 2004, pg. 47.
Slide 9, Image 2 (bottom): “GNU License Logos,” GNU.org, http://www.gnu.org/graphics/license-logos.html, CC: BY-ND 3.0, http://creativecommons.org/licenses/by-nd/3.0/us/

Slide 10: "The Free Software Definition," GNU.org, http://www.gnu.org/philosophy/free-sw.html, CC: BY-ND 3.0, http://creativecommons.org/licenses/by-nd/3.0/us/.

Slide 11: The Art of Computer Programming, by Donald Knuth

Slide 12: “Functional Programming,” Wikipedia, http://en.wikipedia.org/wiki/Functional_programming, CC: BY-SA 3.0, http://creativecommons.org/licenses/by-sa/3.0/.

Slide 13: Please see original image of screenshot of the Linux Kernel website at http://en.tldp.org/LDP/tlk/tlk.html.

Slide 14: “TIOBE Programming Community Index for September 2011,” Tiobe, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Slide 15: “TIOBE Programming Community Index for September 2011,” Tiobe, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Slide 16: Amit Deshpande and Dirk Riehle, “The Total Growth of Open Source,” http://dirkriehle.com/2008/03/14/the-total-growth-of-open-source/

Slide 17: “The Language Guide,” http://groups.engin.umd.umich.edu/CIS/course.des/cis400/