SOLID is Solid

Enterprise principles in OOP architectural design-phase pattern-level constructs

Hillel Wayne

SOLID

- Single responsibility
- Open–closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

Why SOLID?

Why SOLID?

Why SOLID?

Robert Martin

☆ If I had to write commandments, these would be candidates.

1. Software entities (classes, modules, etc) should be open for extension, but closed for modification. (The open/closed principle -- Bertrand Meyer)

2. Derived classes must usable through the base class interface without the need for the user to know the difference. (The Liskov Substitution Principle)

3. Details should depend upon abstractions. Abstractions should not depend upon details. (Principle of Dependency Inversion)

4. The granule of reuse is the same as the granule of release. Only components that are released through a tracking system can be effectively reused.

5. Classes within a released component should share common closure. That is, if one needs to be changed, they all are likely to need to be changed. What affects one, affects all.

6. Classes within a released componen should be reused together. That is, it is impossible to separate the components from each other in order to reuse less than the total.

7. The dependency structure for released components must be a DAG. There can be no cycles.

8. Dependencies between released components must run in the direction of stability. The dependee must be more stable than the depender.

9. The more stable a released component is, the more it must consist of abstract classes. A completely stable component should consist of nothing but abstract classes.

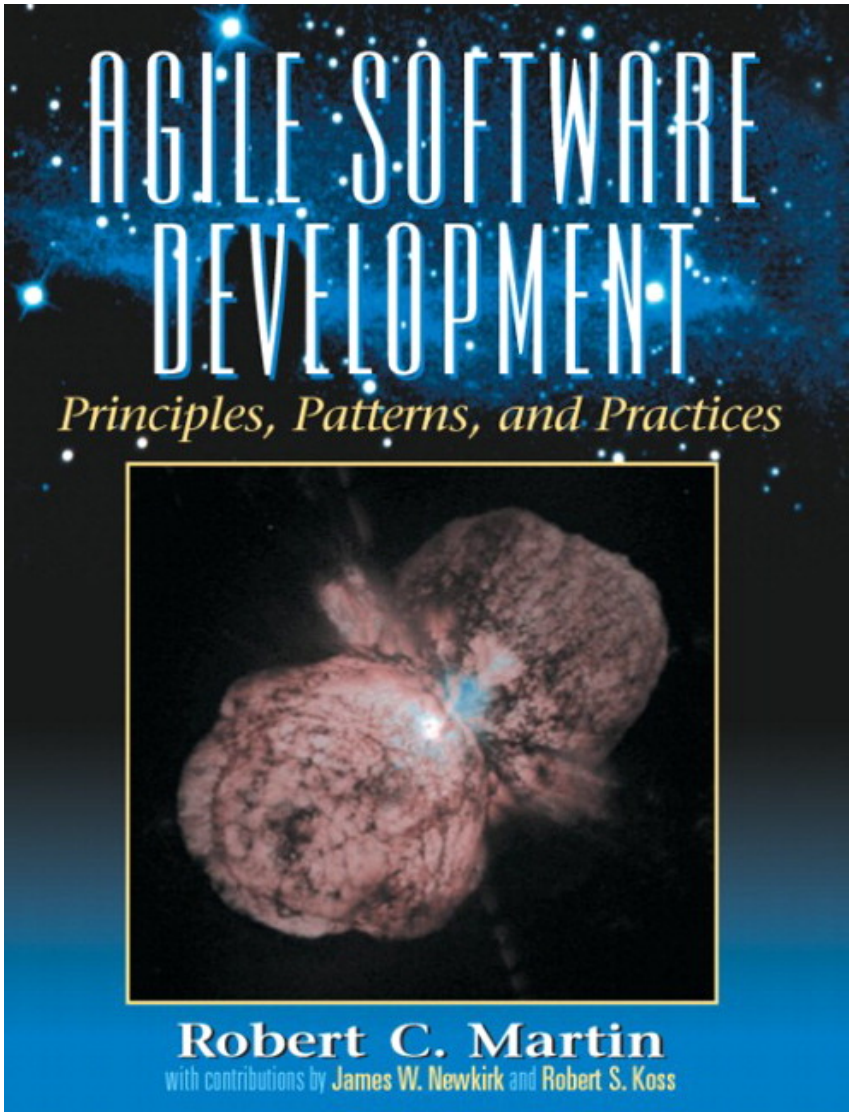
10. Where possible, use proven patterns to solve design problems.

11. When crossing between two different paradigms, build an interface layer that separates the two. Don't pollute one side with the paradigm of the other.

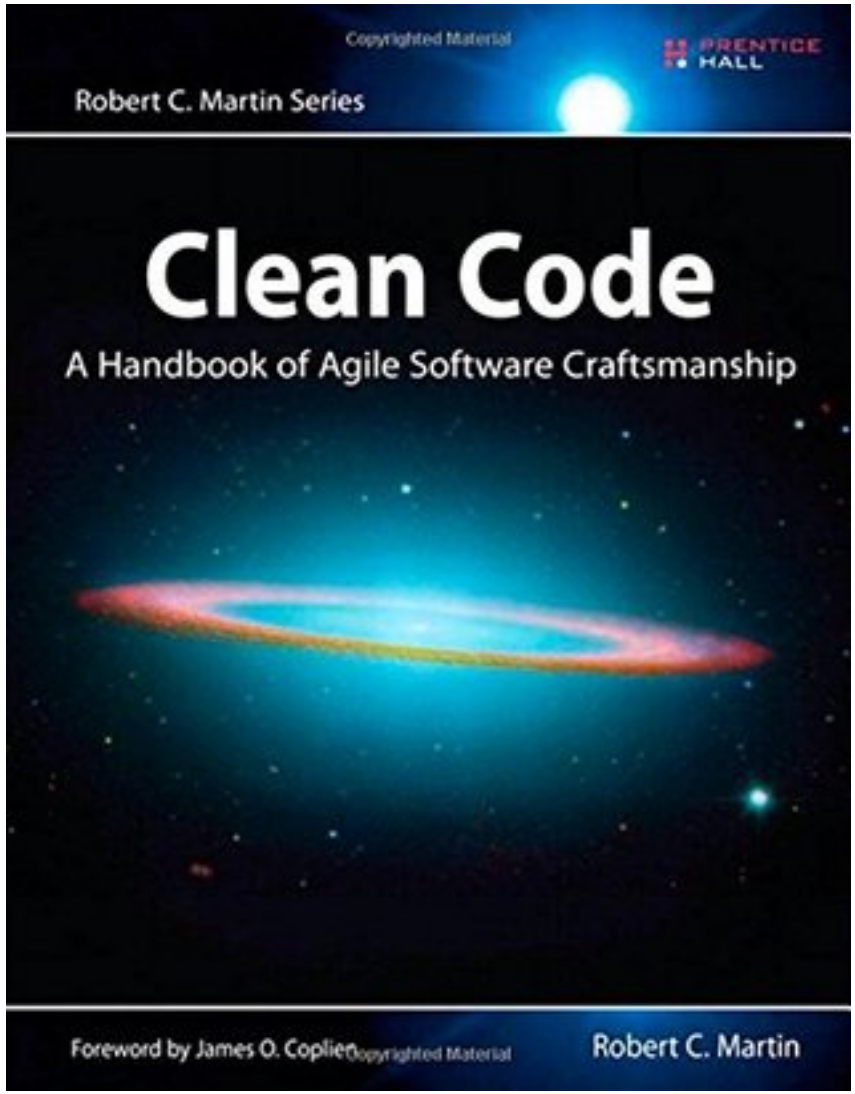
--
Robert Martin | Design Consulting | Training courses offered:
Object Mentor Assoc.| rma...@rcmcon.com | Object Oriented Analysis
2080 Cranbrook Rd. | Tel: (708) 918-1004 | Object Oriented Design
Green Oaks IL 60048 | Fax: (708) 918-1023 | C++

@hillelogram

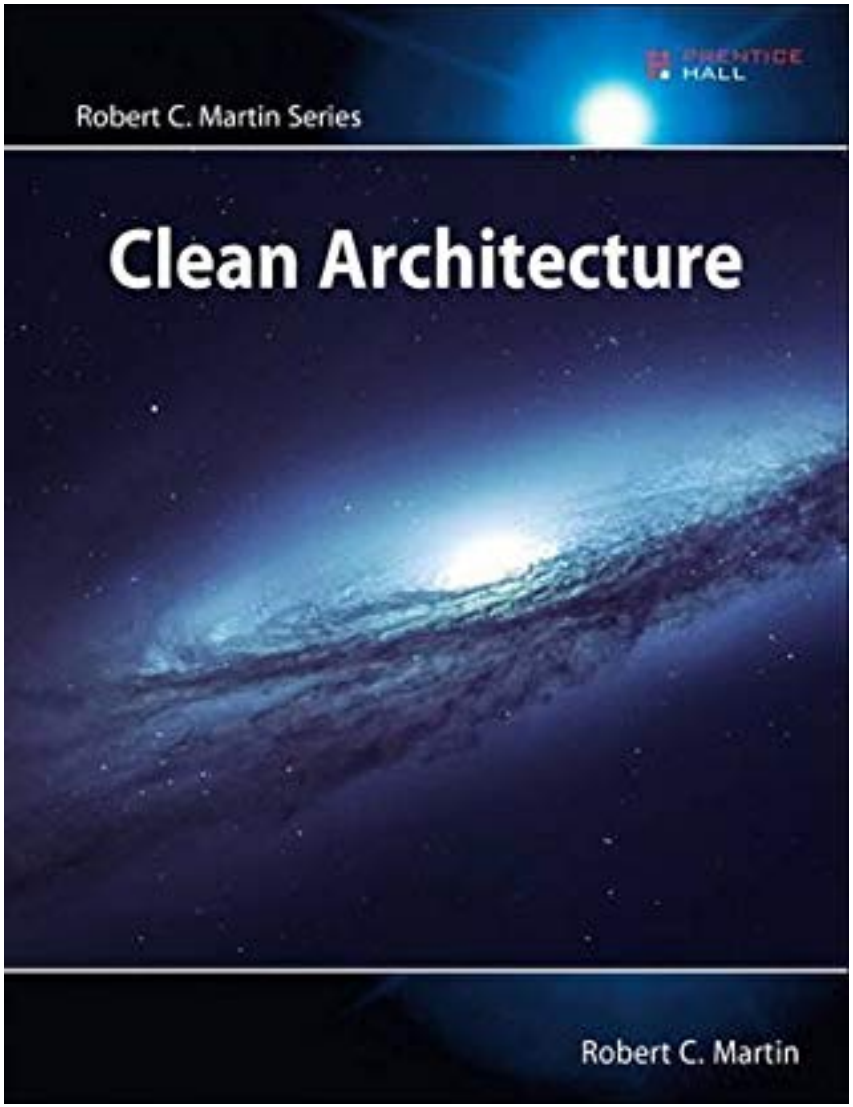
Why SOLID?


AGILE SOFTWARE DEVELOPMENT
Principles, Patterns, and Practices
Robert C. Martin
with contributions by James W. Newkirk and Robert S. Koss

Why SOLID?


Copyrighted Material
PRENTICE HALL
Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Copyrighted Material
Robert C. Martin

Why SOLID?


Robert C. Martin Series
PRENTICE HALL
Clean Architecture
Robert C. Martin

What We Can Learn From Software History

Hillel Wayne
hillelwayne.com
@hillelogram

www.hillelwayne.com/talks/software-history

When you create the new classes, make sure you properly test them and create them using SOLID principles so they will be easier to change in the future.

If it's too complex, it's violating a ton of SOLID principles.

Don't be STUPID: GRASP SOLID!

The Process

Reverse a linked list

Given pointer to the head node of a linked list, the task is to reverse the linked list. We need to reverse the list by changing links between nodes.

Examples:

Input: *Head of following linked list*
1->2->3->4->NULL
Output: *Linked list should be changed to,*
4->3->2->1->NULL

Input: *Head of following linked list*
1->2->3->4->5->NULL
Output: *Linked list should be changed to,*
5->4->3->2->1->NULL

Input: *NULL*
Output: *NULL*

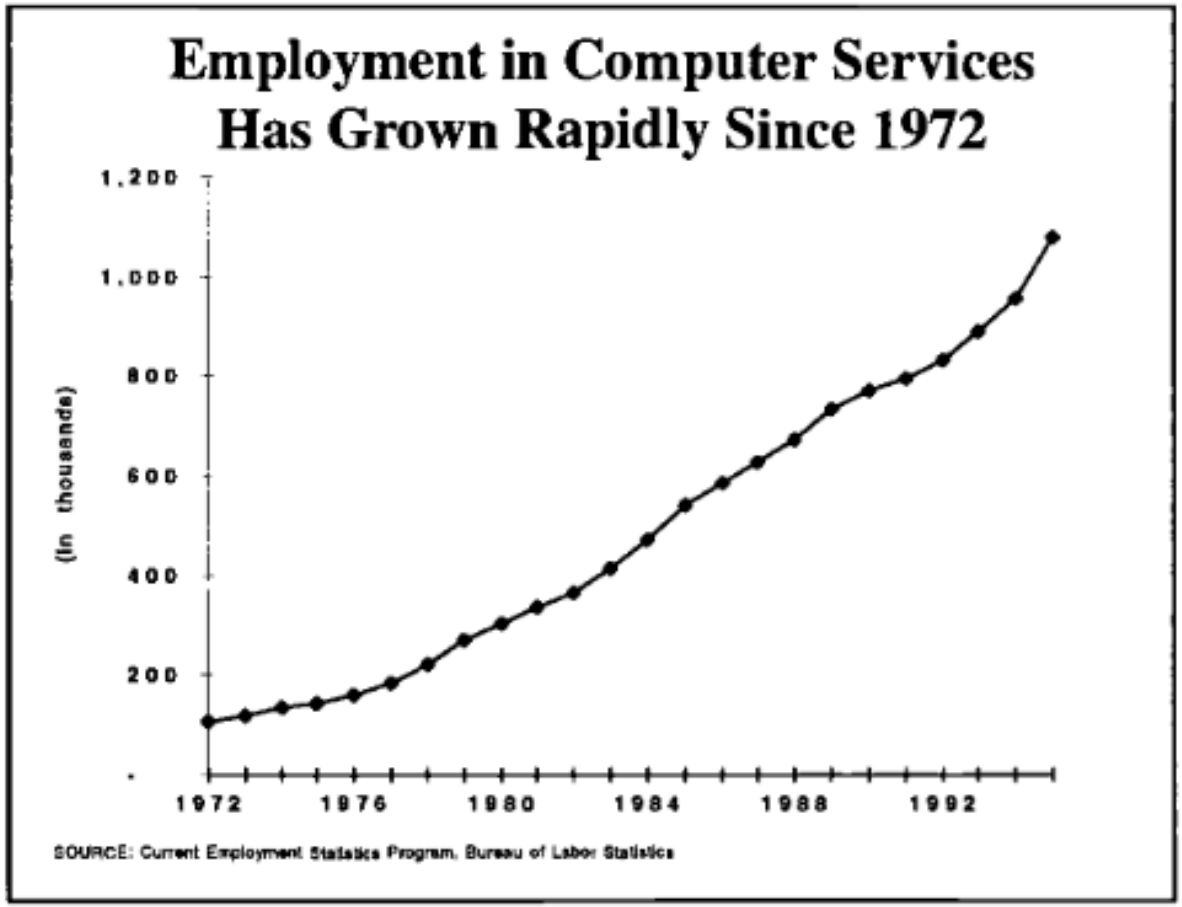
Input: *1->NULL*
Output: *1->NULL*

Do you know CS?

Do you know CS?

Can you think quickly?

1. When did it happen?


Employment in Computer Services
Has Grown Rapidly Since 1972
1,200
1,000
800
600
400
200
-
(in thousands)
1972
1976
1980
1984
1988
1992
SOURCE: Current Employment Statistics Program, Bureau of Labor Statistics

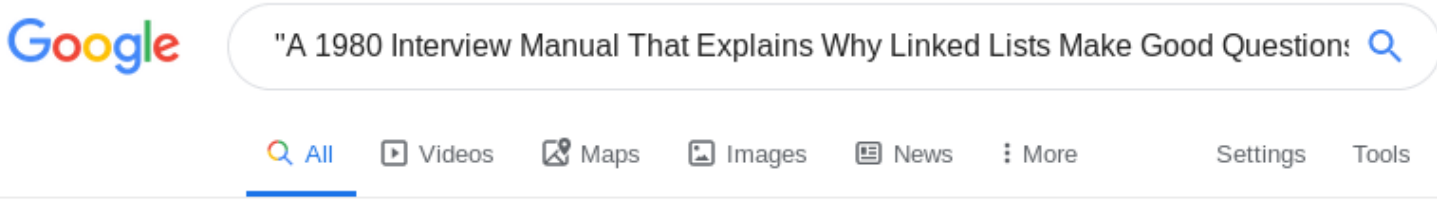
2. What was the context?

Primary Source

Artifacts and information from that time

Secondary Source

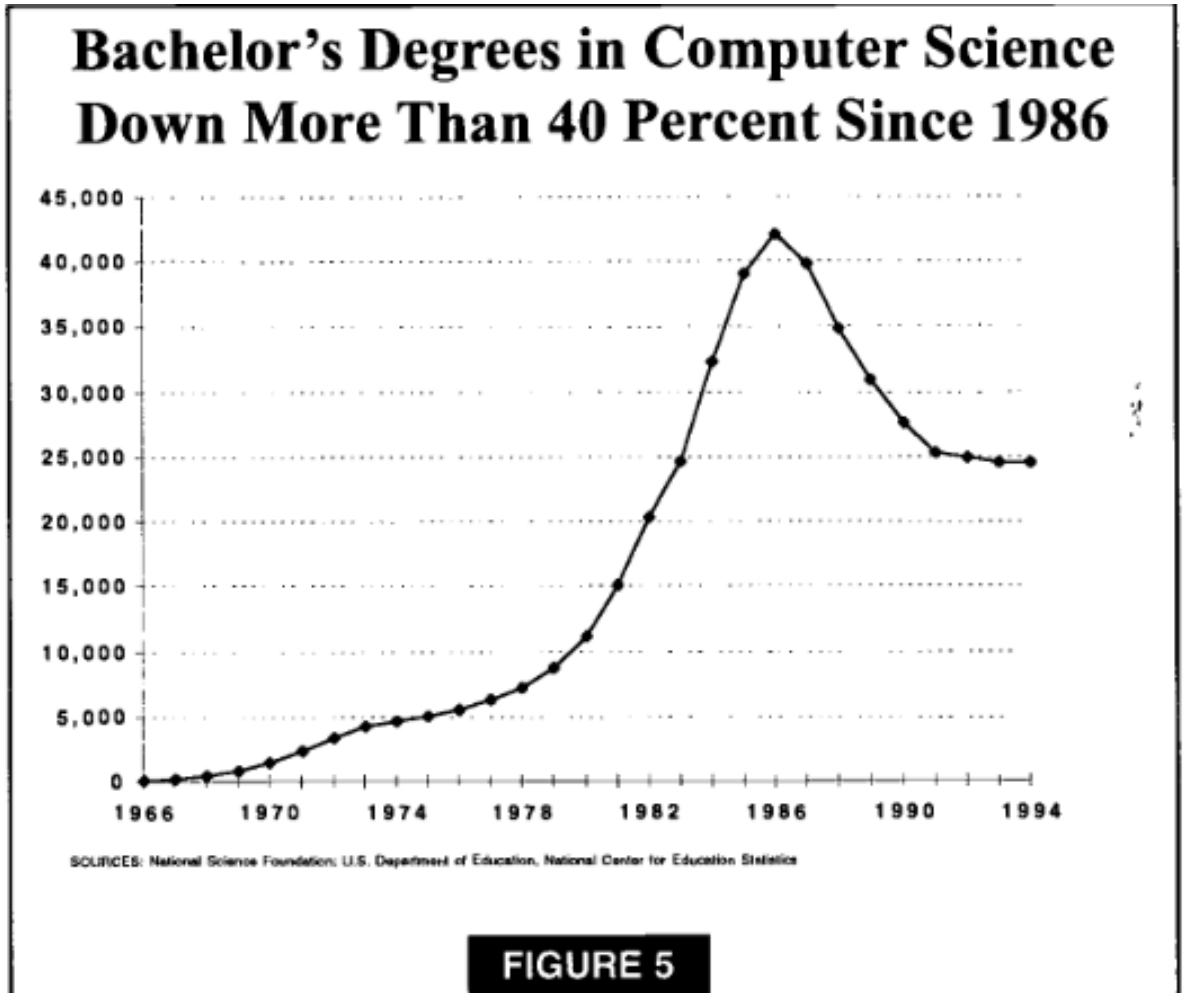
Information produced after the fact



Your search - **"A 1980 Interview Manual That Explains Why Linked Lists Make Good Questions"** - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.


Bachelor's Degrees in Computer Science
Down More Than 40 Percent Since 1986
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
1966
1970
1974
1978
1982
1986
1990
1994
SOURCES: National Science Foundation; U.S. Department of Education, National Center for Education Statistics
FIGURE 5

~~Do you know CS?~~

Can you think quickly?

3. What were the reasons?

Dynamic allocation

Dynamic allocation

Fortran Nope

Dynamic allocation

Fortran	Nope
Lisps	just `cons` it
Smalltalk	`copyWith: newElement`

Dynamic allocation

Fortran	Nope
Lisps	just `cons` it
Smalltalk	`copyWith: newElement`
C	Manual memory manipulation

~~Do you know CS?~~

~~Do you know CS?~~

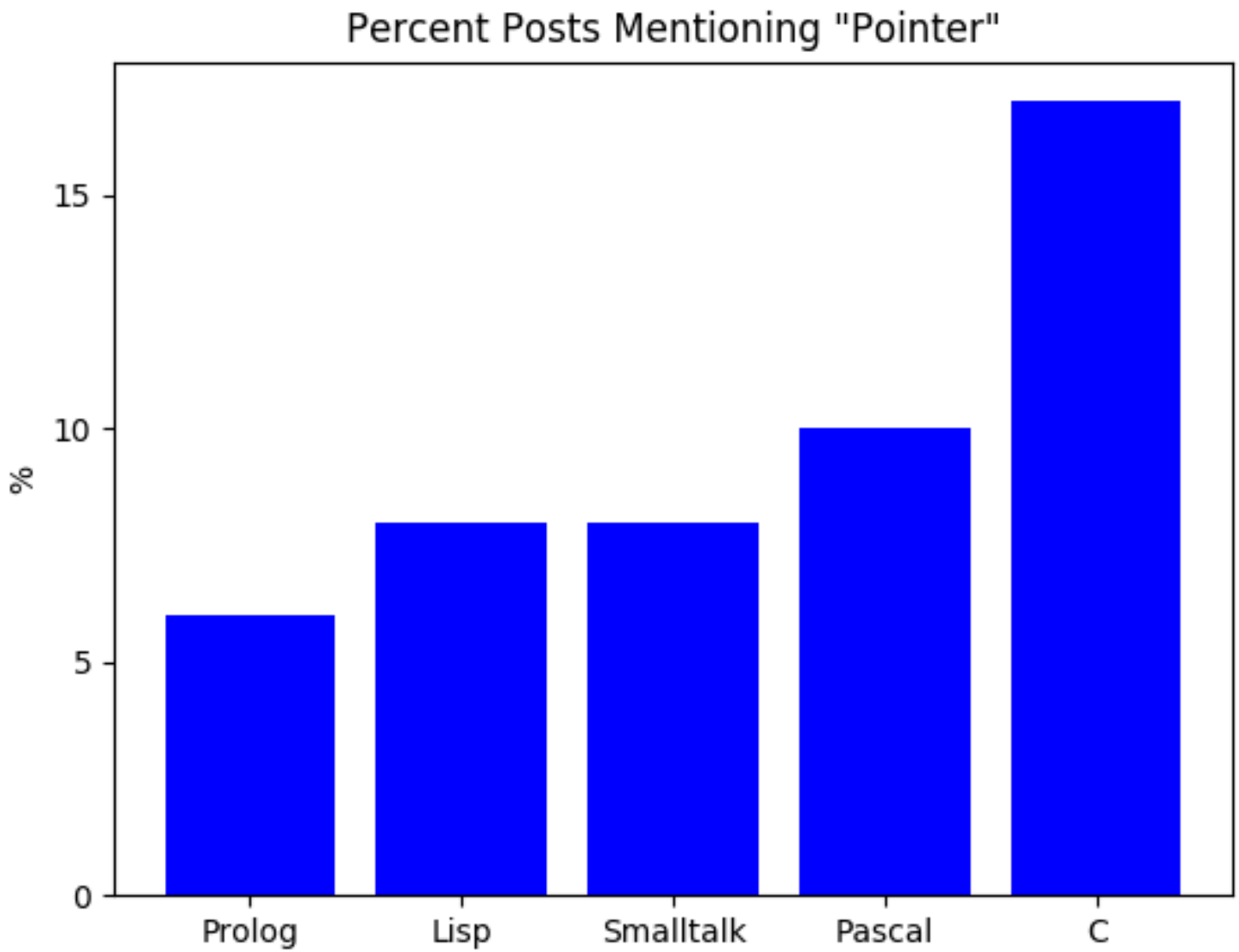
~~Can you think quickly?~~

~~Do you know CS?~~

~~Can you think quickly?~~

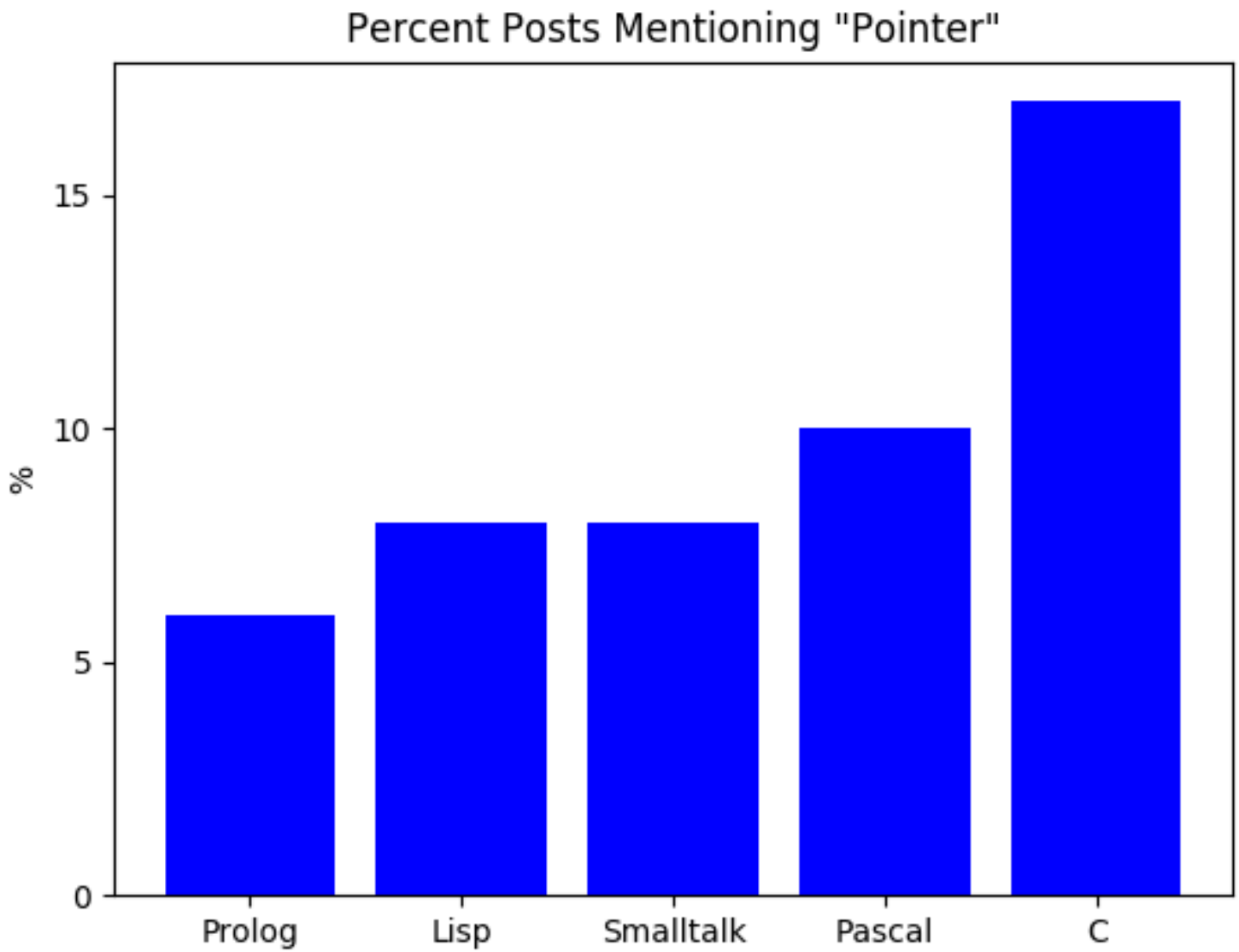
Have you used C?

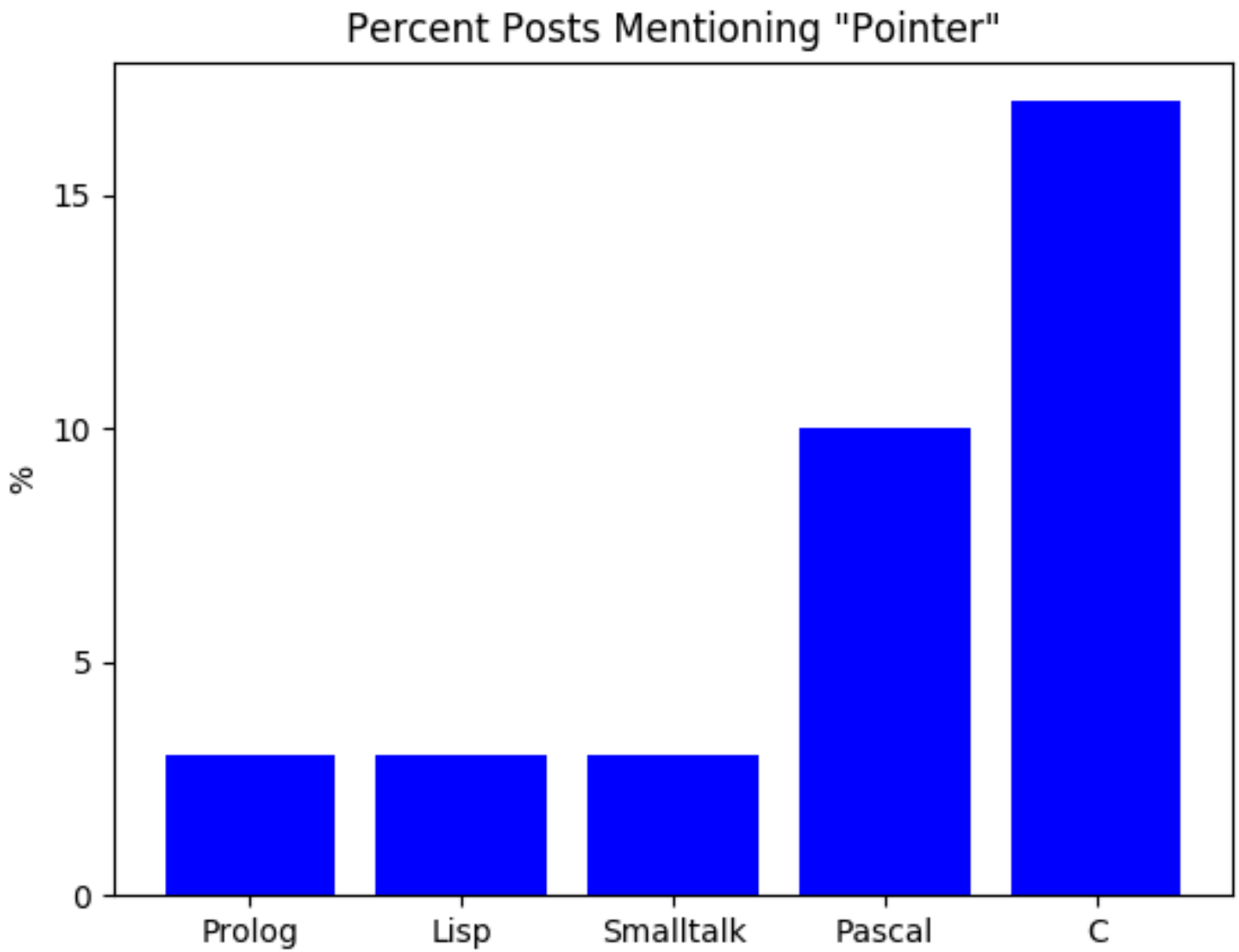
4. What does this predict?

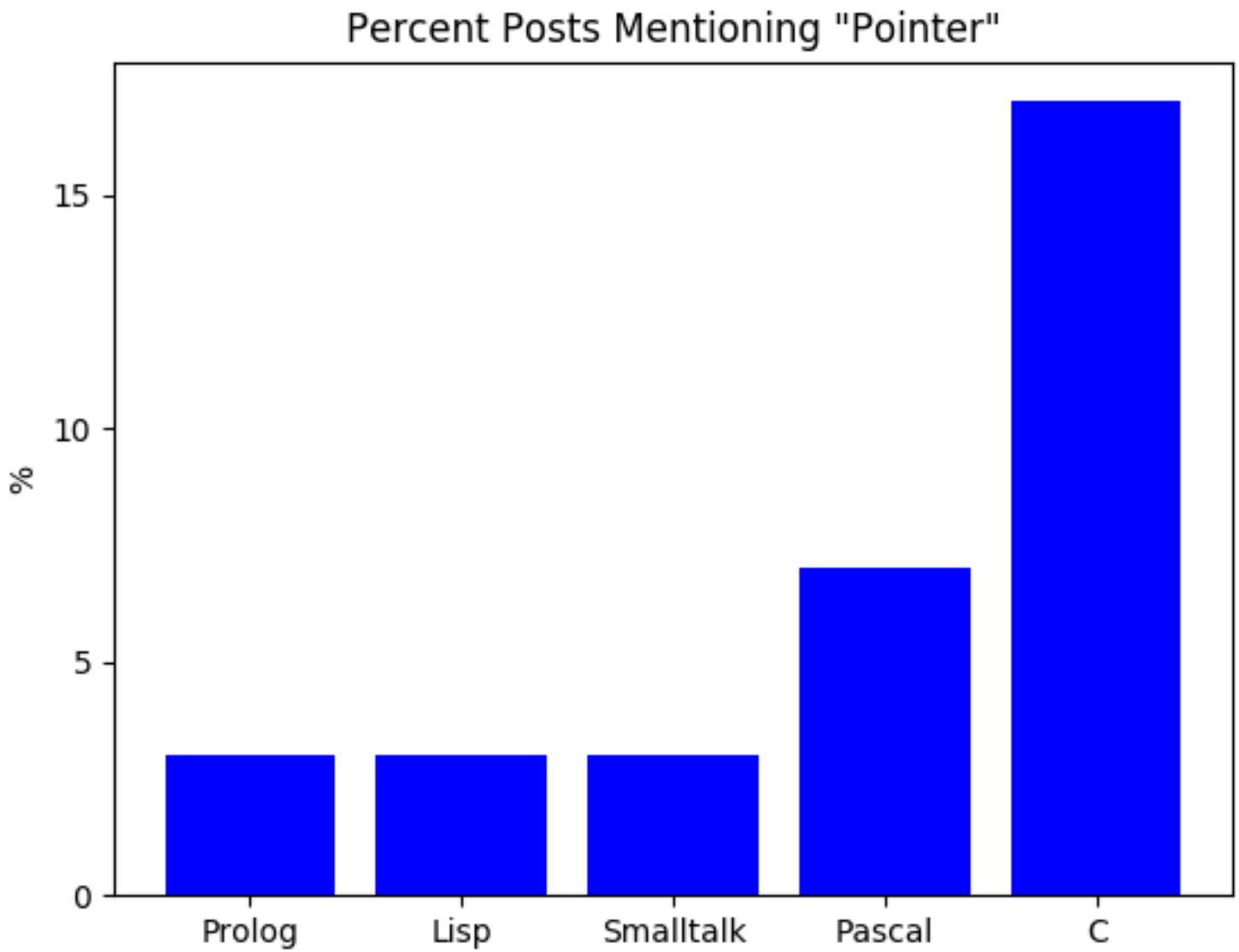

Percent Posts Mentioning "Pointer"
%
0
5
10
15
Prolog
Lisp
Smalltalk
Pascal
C

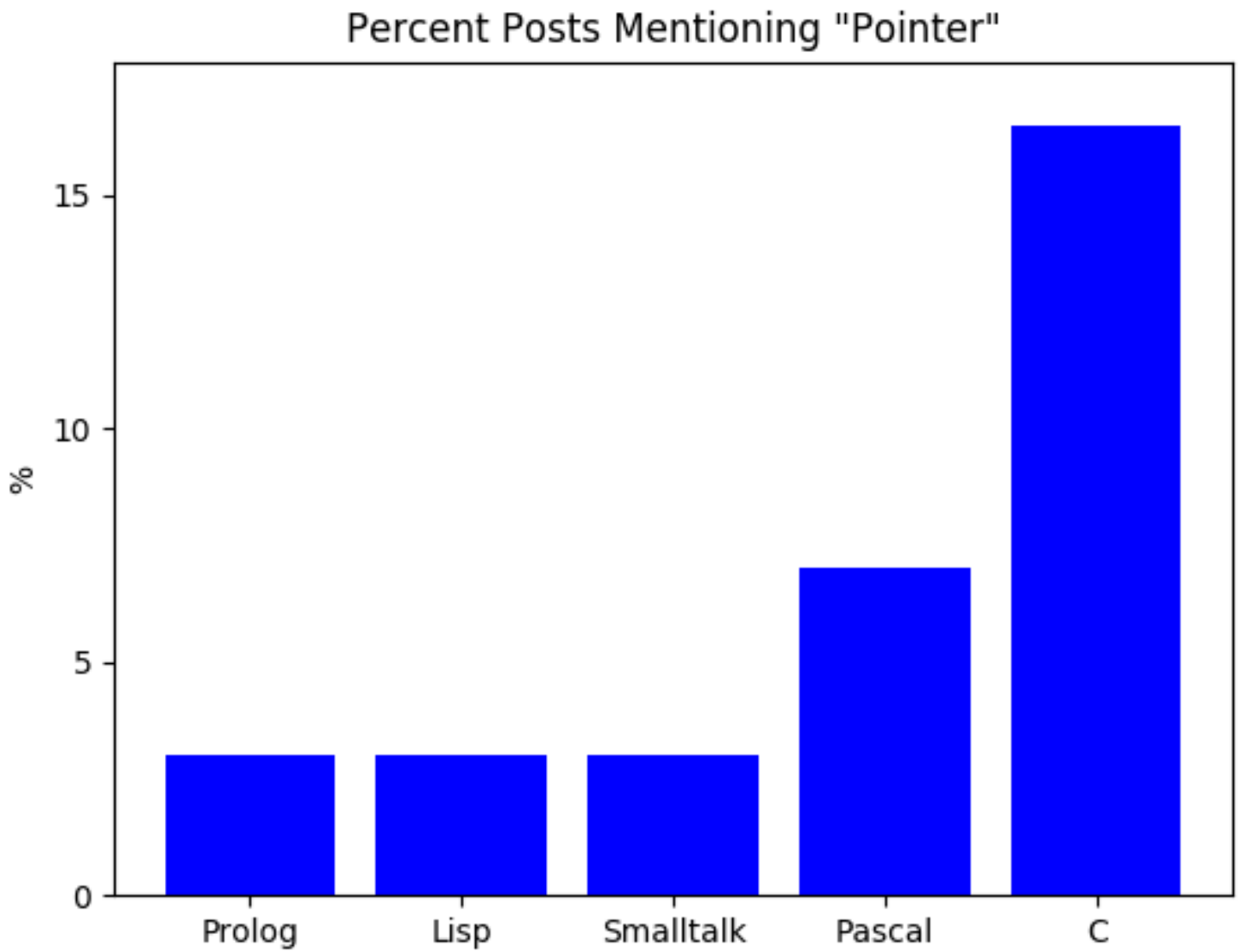
Does anyone know of a version of Smalltalk that runs on Apollo workstations? [...] i've heard rumors that such a program exists at Utah or Brown but have no firm pointers. only full Smalltalk-80 please – not little Smalltalk.

Does anyone know of a version of Smalltalk that runs on Apollo workstations? [...] i've heard rumors that such a program exists at Utah or Brown but have no firm pointers. only full Smalltalk-80 please – not little Smalltalk.

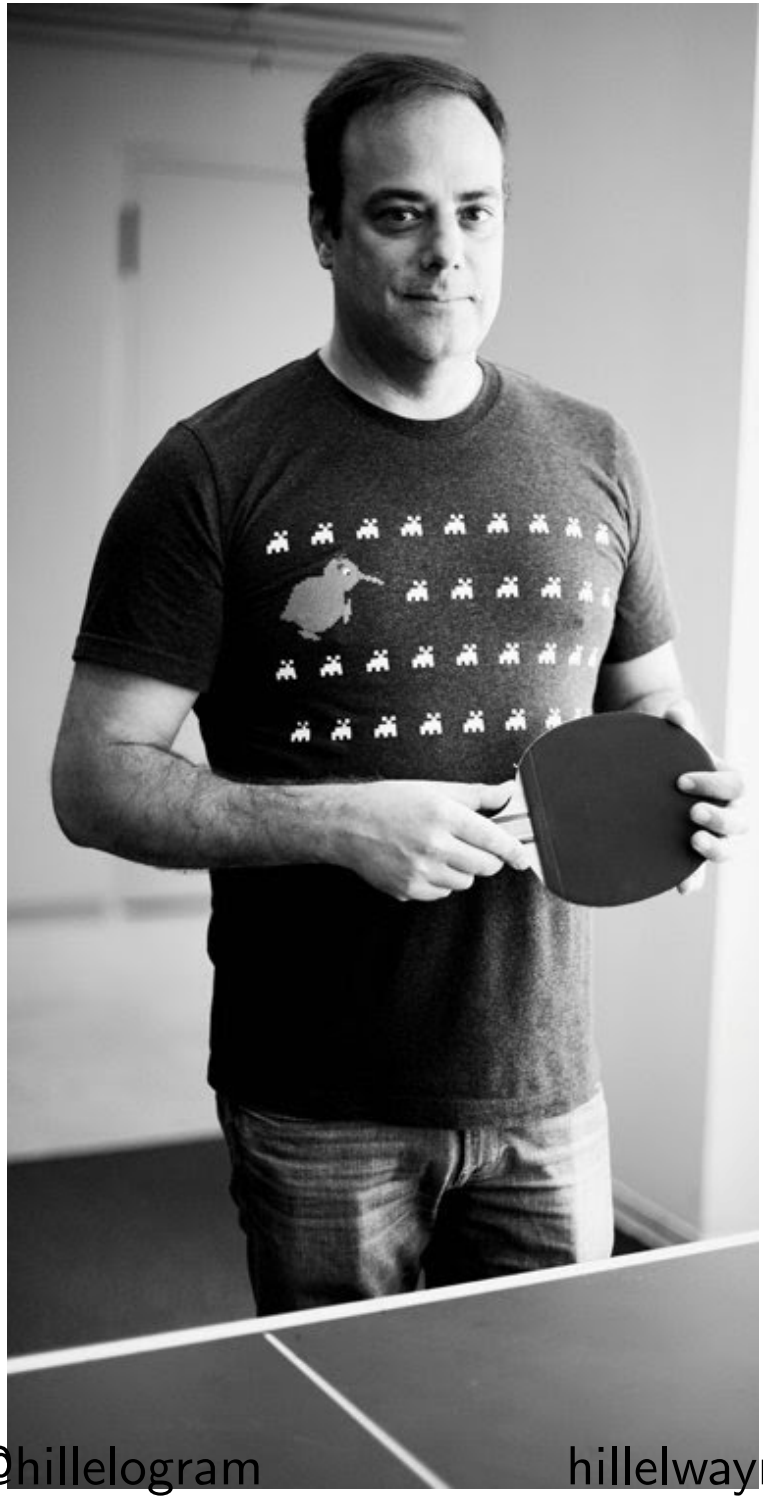

Percent Posts Mentioning "Pointer"
%
0
5
10
15
Prolog
Lisp
Smalltalk
Pascal
C


Percent Posts Mentioning "Pointer"
%
0
5
10
15
Prolog
Lisp
Smalltalk
Pascal
C


Percent Posts Mentioning "Pointer"
%
0
5
10
15
Prolog
Lisp
Smalltalk
Pascal
C


Percent Posts Mentioning "Pointer"
%
0
5
10
15
Prolog
Lisp
Smalltalk
Pascal
C

5. What are the loose ends?

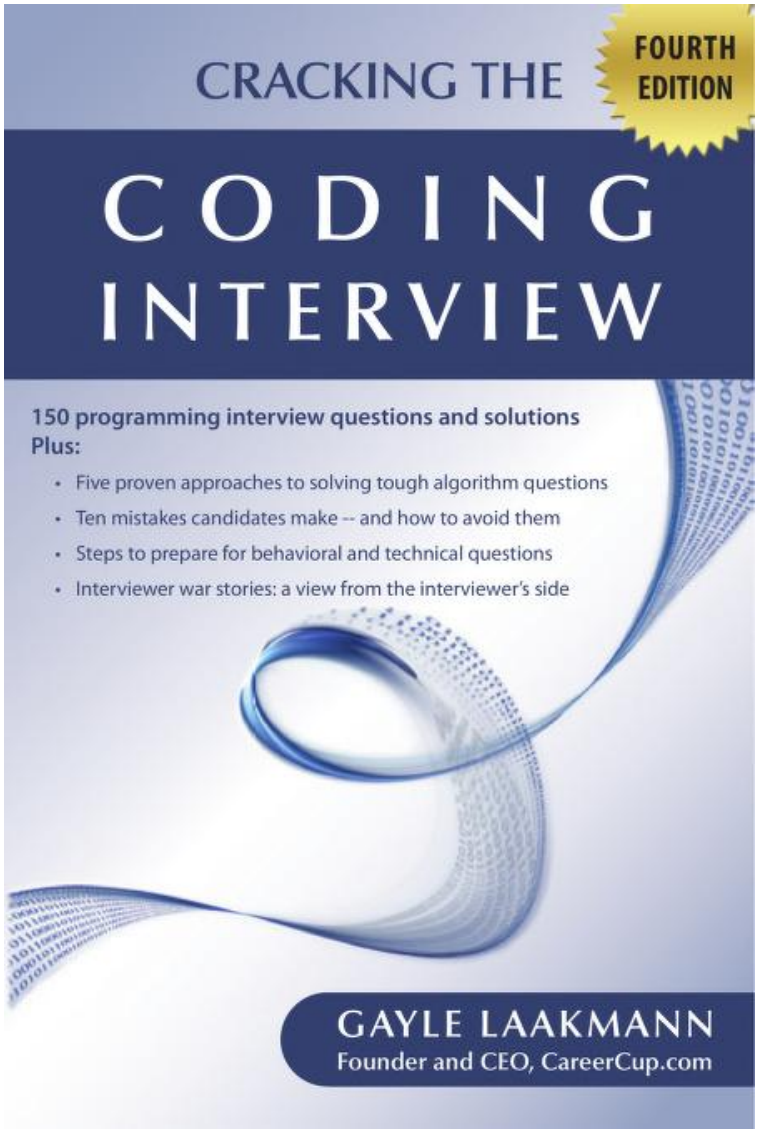



SMART
& GETS
THINGS DONE
JOEL SPOLSKY's CONCISE GUIDE TO
FINDING THE BEST TECHNICAL TALENT

Pointers require a complex form of doubly-indirected thinking that some people just can't do, and it's pretty crucial to good programming. A lot of the "script jocks" who started programming by copying JavaScript snippets into their web pages and went on to learn Perl never learned about pointers, and they can never quite produce code of the quality you need.

Pointers require a complex form of doubly-indirected thinking that some people just can't do, and it's pretty crucial to good programming. A lot of the "script jocks" who started programming by copying JavaScript snippets into their web pages and went on to learn Perl never learned about pointers, and they can never quite produce code of the quality you need.

That's the source of all these famous interview questions you hear about, like "reversing a linked list" or "detect loops in a tree structure."


CRACKING THE
FOURTH EDITION
CODING INTERVIEW
150 programming interview questions and solutions
Plus:
• Five proven approaches to solving tough algorithm questions
• Ten mistakes candidates make -- and how to avoid them
• Steps to prepare for behavioral and technical questions
• Interviewer war stories: a view from the interviewer's side
GAYLE LAAKMANN
Founder and CEO, CareerCup.com

Conclusion

www.hillelwayne.com/talks/software-history