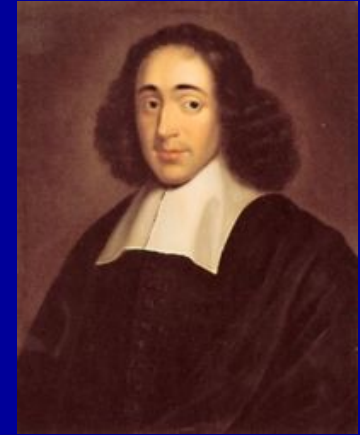


Minimizing Understanding in the Construction and Maintenance of Software Systems

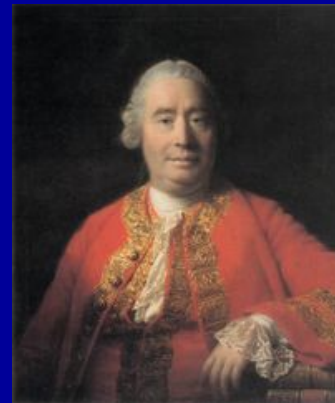
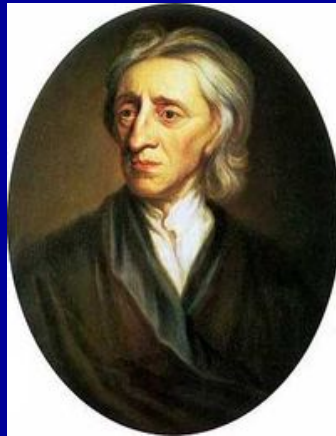
Martin Rinard

Department of Electrical Engineering and Computer Science
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

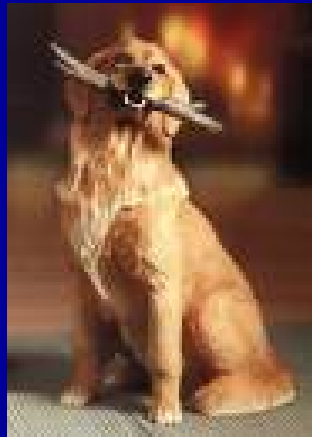
Rationalism



Empiricism



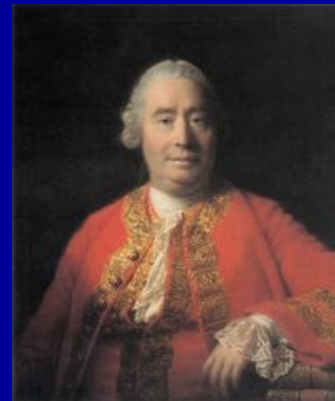
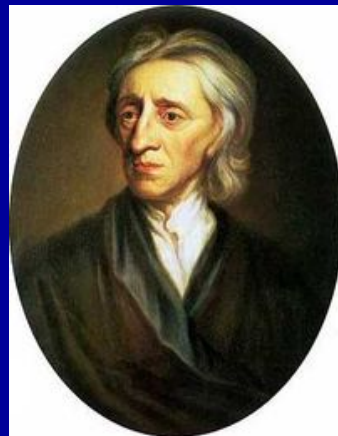
Cluelessness



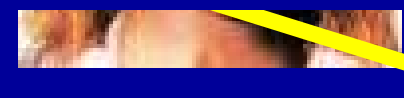
Rationalism



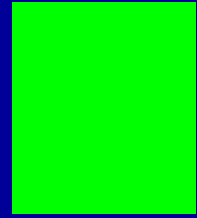
Empiricism



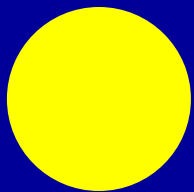
Selective Cluelessness



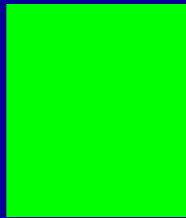
Basic Message



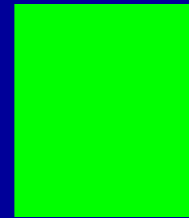
To Maximize Functionality
Minimize Understanding
Subject to Buildability



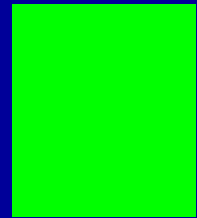
Your Cognitive
Capability



System You
Understand



System You
 $\frac{1}{2}$ Understand



System You
 $\frac{1}{4}$ Understand

Rest of the Talk

- Three phases in computer science
 - Climbing away from the machine
 - Search for truth, elegance, and understanding
 - Age of gigantic building blocks and functionality overkill
- Summary of where we are now
- Where we go from here
(ways to understand even less)

A Discovery Of The Problem

"As soon as we started programming, we found to our surprise that **it wasn't as easy to get programs right as we had thought**. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."



Maurice Wilkes, 1949

Response To The Problem

- Subroutines
- Libraries

Consequences

- Specialization
- Reuse
- Potential
 - Loss of Efficiency
 - Deskilling



David Wheeler

FORTRAN

Basic premise

- You have formulas you wish to evaluate
- Using the computer is your problem



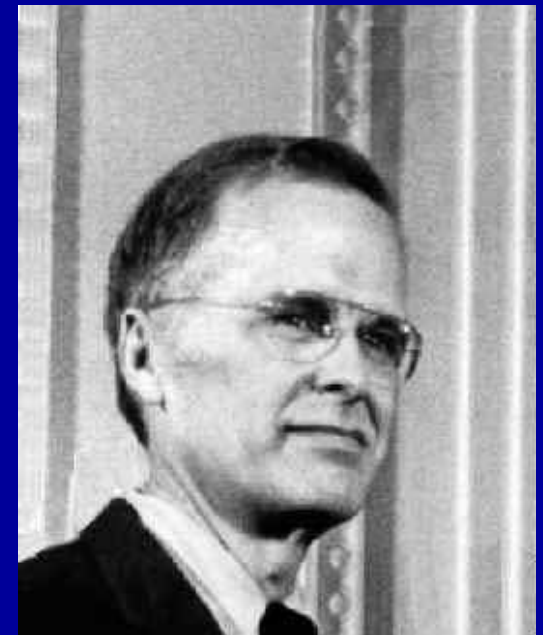
Formulas



FORTRAN
Compiler



Executable
Code



John Backus

Expectation

FORTRAN will solve your problem

“Since FORTRAN will virtually eliminate coding and debugging ...”

(Specifications for the IBM FORMula TRANslating System FORTRAN, November 10, 1954)

- Solve problems for
 - Less than 1/2 the cost
 - Less than 1/4 the elapsed time
- Double machine time spent on useful problem solving

FORTRAN and Understanding

“...the amount of knowledge necessary to utilize the 704 effectively by means of FORTRAN is far less than the knowledge required to make effective use of the 704 by direct coding.”

(Specifications for the IBM FORMula TRANslating System FORTRAN, November 10, 1954)

COBOL



Grace Hopper

Goals

Portability (Common Business-Oriented Language)

“The need is for a programming language that is easier to use even if somewhat less powerful.”

“We need to broaden the base of those who can state problems to the computer.”

(The Early History of COBOL, The First ACM SIGPLAN Conference on the History of Programming Languages)

Expectations

“In summary, a well-conducted four-week COBOL programming course should enable the graduate to contribute immediately to the company’s programming efforts.”

(Guides to Teaching COBOL, Communications of the ACM, May 1962)

“It was certainly intended (and expected) that the language could be used by novice programmers and read by management.”

(The Early History of COBOL, The First ACM SIGPLAN Conference on the History of Programming Languages)

LISP

Initially set up as language to support Advice Taker natural language processing and reasoning system

"... it became clear that this combination of ideas made an elegant mathematical system as well as a practical programming language. Then mathematical neatness became a goal ..."

(History of LISP, The First ACM SIGPLAN Conference on the History of Programming Languages)



John McCarthy

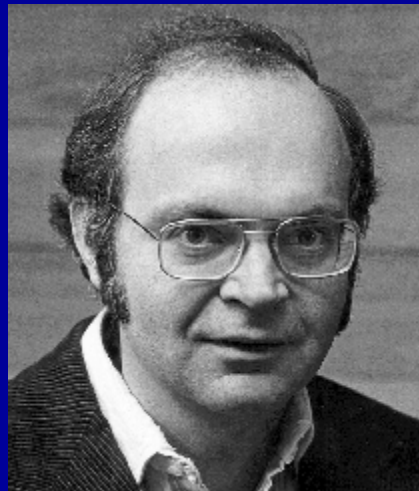
Search for Truth, Elegance, and Understanding

People realized programming was

- Important
- Difficult
- Unforgiving
(single error could have drastic consequences)
- Intellectually engaging

Responses

"Some programs are elegant, some are exquisite, some are sparkling. My claim is that it is possible to write *grand* programs, *noble* programs, truly *magnificent* ones!"
(Don Knuth, ACM Turing Award Lecture)



Don Knuth

Simula

“To program is to understand”
(Kristen Nygaard,
moral philosopher)



Ole-Johan Dahl Kristen Nygaard

- Original goal:
 - Computer as means for understanding real world
 - Concepts inside computer match concepts in real world
- Eventually applied approach to all kinds of programming
- Big advantage is that it reduces thinking
 - Conceptual framework ready to go
 - Builds on concepts from physical world

Smalltalk

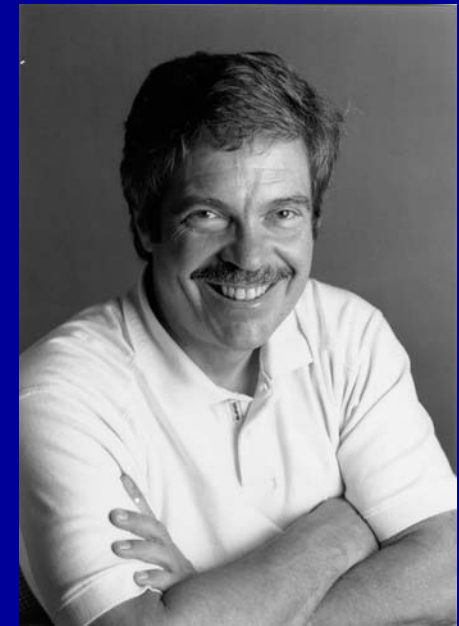
- Make a new world inside computer
- Live inside that world
 - World should be elegant
 - Build on physical world you know
- Primary goal: power



Adele Goldberg



Dan Ingalls



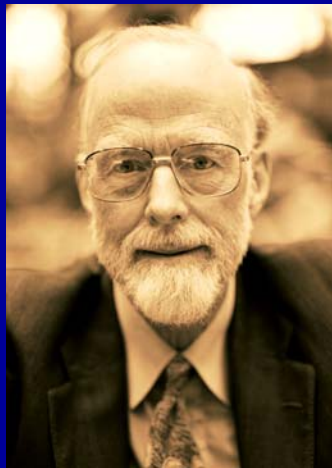
Alan Kay

Reasoning from Premises

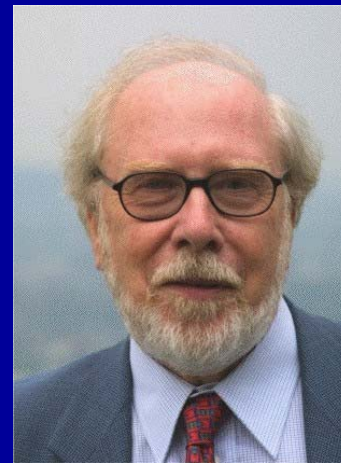
- Programs are unforgiving, therefore they must be correct
- To make program correct, must completely understand problem and the program
- Programming is difficult, therefore simplicity and elegance are keys to success



Edsger Dijkstra



Tony Hoare



Niklaus Wirth



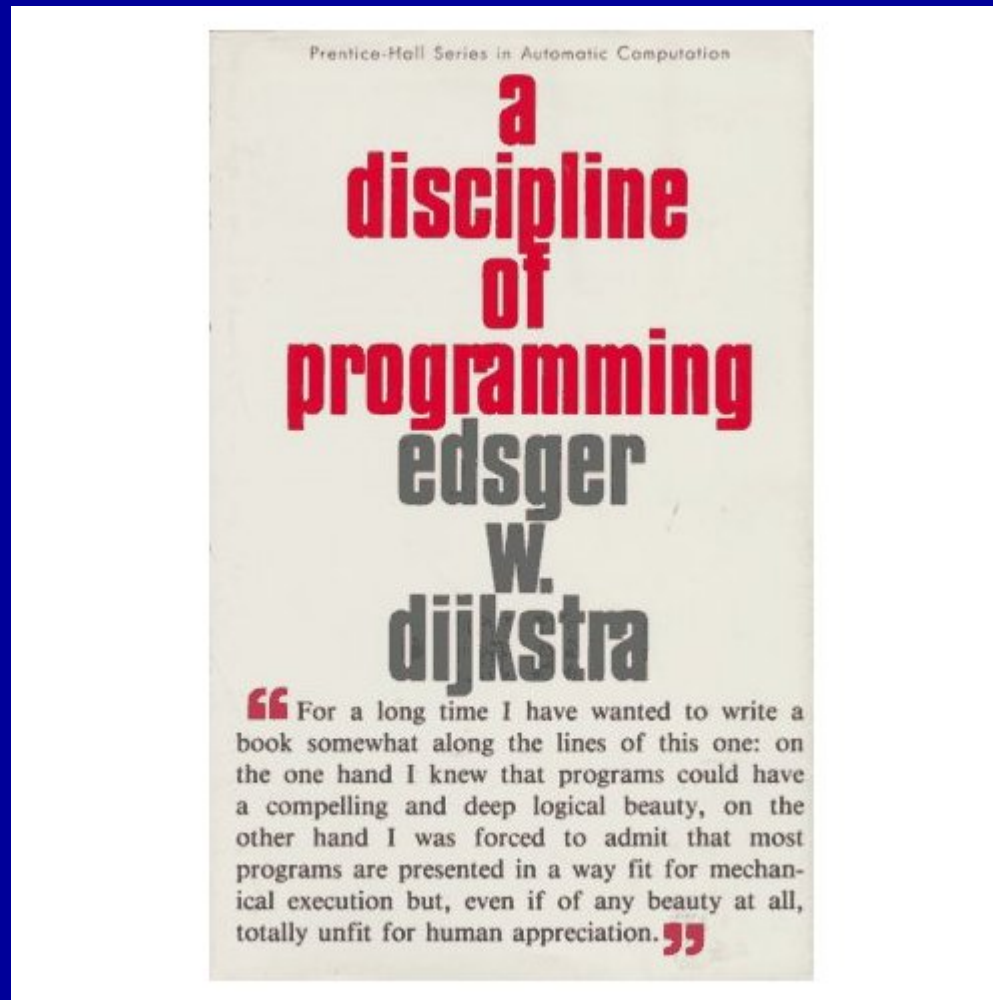
Ole-Johan Dahl

Unfortunately, Simplicity and Elegance Are Hard to Come By...

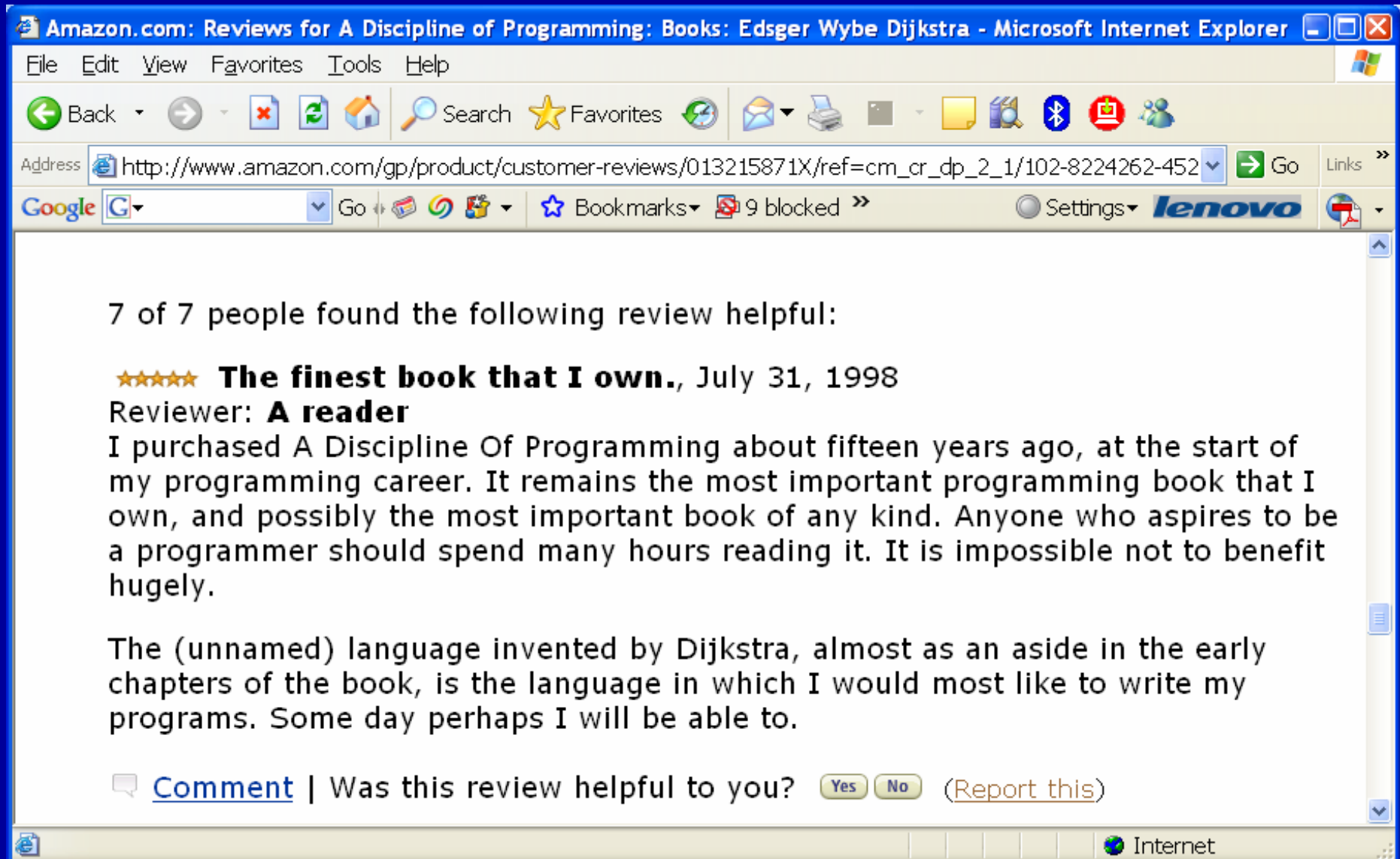
“Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated.”
(Edsger Dijkstra, EWD 1234, “The Next 50 Years”)

“Simple, elegant solutions are more effective, but they are harder to find than complex ones, and they require more time, which we too often believe to be unaffordable.”
(Niklaus Wirth, ACM Turing Award Lecture)

A Discipline of Programming



Programmer Reaction



Amazon.com: Reviews for A Discipline of Programming: Books: Edsger Wybe Dijkstra - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://www.amazon.com/gp/product/customer-reviews/013215871X/ref=cm_cr_dp_2_1/102-8224262-452 Go Links

Google G Go Bookmarks 9 blocked Settings lenovo

7 of 7 people found the following review helpful:

★★★★★ **The finest book that I own.**, July 31, 1998
Reviewer: **A reader**

I purchased A Discipline Of Programming about fifteen years ago, at the start of my programming career. It remains the most important programming book that I own, and possibly the most important book of any kind. Anyone who aspires to be a programmer should spend many hours reading it. It is impossible not to benefit hugely.

The (unnamed) language invented by Dijkstra, almost as an aside in the early chapters of the book, is the language in which I would most like to write my programs. Some day perhaps I will be able to.

[Comment](#) | Was this review helpful to you? ([Report this](#))

Internet

Programmer Reaction

Amazon.com: Reviews for A Discipline of Programming: Books: Edsger Wybe Dijkstra - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://www.amazon.com/gp/product/customer-reviews/013215871X/ref=cm_cr_dp_2_1/102-8224262-452 Go Links

Google Go Bookmarks 9 blocked Settings lenovo

7 of 11 people found the following review helpful:

★★★★★ **Nice place to visit, wouldn't want to live there**, July 26, 2002
Reviewer: **A reader**

I really wanted to get my hands on this book and now that i have (via interlibrary loan) i want to warn folks that this is not light reading. I found a majority of this book very boring and all but impenetrable. I like Dijkstra's English prose, but when he embarks on the math I wish he'd state the point of each set of formulae above them. It would have also helped if he stressed practical uses of his insights vis-a-vis an actual programming language. This "just shows how much I know" I'm sure, but I suspect many people will feel similarly. FYI: My background is Bachelor's in C.S. with a C.S. GPA of 3.87/4.0. A depressing indictment of U.S. education, Dijkstra would say :)

[Comment](#) | Was this review helpful to you? ([Report this](#))

Internet

Programmer Reaction

Amazon.com: Reviews for A Discipline of Programming: Books: Edsger Wybe Dijkstra - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites

Address http://www.amazon.com/gp/product/customer-reviews/013215871X/ref=cm_cr_dp_2_1/102-8224262-452 Go Links

Google Go Bookmarks 9 blocked Settings lenovo

7 of 11 people found the following review helpful:

★★★★☆ **Nice place to visit, wouldn't want to live there**, July 26, 2002
Reviewer: **A reader**

I really wanted to get my hands on this book and now that i have (via interlibrary loan) i want to warn folks that this is not light reading. I found a majority of this book **very boring and all but impenetrable.** I like Dijkstra's English prose, but when he embarks on the math i wish he'd state the point of each set of formulae above them. It would have also helped if he stressed practical uses of his insights vis-a-vis an actual programming language. This "just shows how much I know" I'm sure, but I suspect many people will feel similarly. FYI: My background is Bachelor's in C.S. with a C.S. GPA of 3.87/4.0. A depressing indictment of U.S. education, Dijkstra would say :)

[Comment](#) | Was this review helpful to you? ([Report this](#))

Internet

Programmer Reaction

Amazon.com: Reviews for A Discipline of Programming: Books: Edsger Wybe Dijkstra - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites

Address http://www.amazon.com/gp/product/customer-reviews/013215871X/ref=cm_cr_dp_2_1/102-8224262-452 Go Links

Google Go Bookmarks 9 blocked Settings lenovo

7 of 11 people found the following review helpful:

★★★★☆ **Nice place to visit, wouldn't want to live there**, July 26, 2002
Reviewer: **A reader**

I really wanted to get my hands on this book and now that i have (via interlibrary loan) i want to warn folks that this is not light reading. I found a majority of this book very boring and all but impenetrable. I like Dijkstra's English prose, but when he embarks on the math I wish he'd state the point of each set of formulae above them. It would have also helped if he stressed practical uses of his insights vis-a-vis an actual programming language. This "just shows how much I know"

I'm sure, but I suspect many people will feel similarly. FYI: My background is Bachelor's in C.S. with a C.S. GPA of 3.87/4.0. A depressing indictment of U.S. education, Dijkstra would say :)

[Comment](#) | Was this review helpful to you? ([Report this](#))

Internet

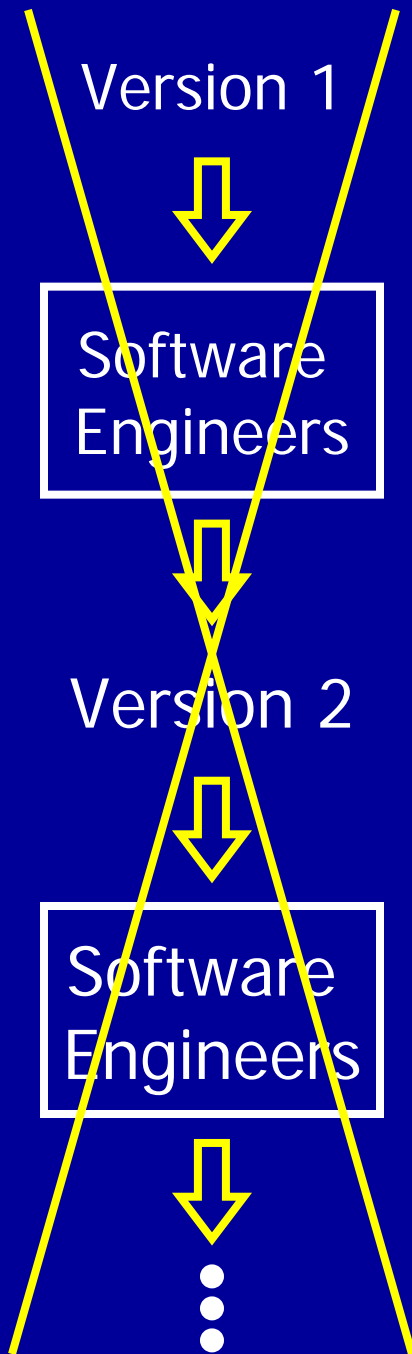
Dijkstra's Unpleasant Truths

"When, in the late sixties, it became abundantly clear that we did not know how to program well enough, people concerned with Programming Methodology tried to figure out what a competent programmer's education should encompass. As a result of that effort **programming emerged as a tough engineering discipline with a strong mathematical flavour.**"

Implications

- "good programming is probably **beyond the intellectual capabilities** of today's 'average programmer'"
- "to do, hic et nunc, **the job well** with today's army of practitioners, many of whom have been lured into a profession beyond their intellectual abilities, **is an insoluble problem**"

(Edsger Dijkstra, EWD 611 "The Atlantic Ocean Has Two Sides")



New Product!

**Fewer Bugs!
More Functionality!**

**Fewer Bugs!
More Functionality!**

No Mas!



Gigantic Building Blocks and Functionality Overkill



Gigantic Building Blocks and Functionality Overkill



What Have We Learned?

- Insatiable demand for functionality
- Minimizing understanding is the way to maximize functionality
- Programming can be
 - Intellectually challenging
 - Practically difficult
- Programs can be beautiful, elegant, and simple
- Simplicity is a nonstarter
- Elegance is largely irrelevant in practice
- Can achieve previously inconceivable levels of cluelessness (and therefore functionality) in successful deployed systems

How Are We Doing?

Great!

“There's an old story about the person who wished his computer were as easy to use as his telephone. That wish has come true, since I no longer know how to use my telephone.”

(Bjarne Stroustrup, inventor of C++)

How Can We Improve?

- Program Verification
 - Data structures, algorithms
 - Operating systems, compilers, virtual machines
 - Small real-time systems
- System Engineering
- Living With Errors

New Assumption Basis

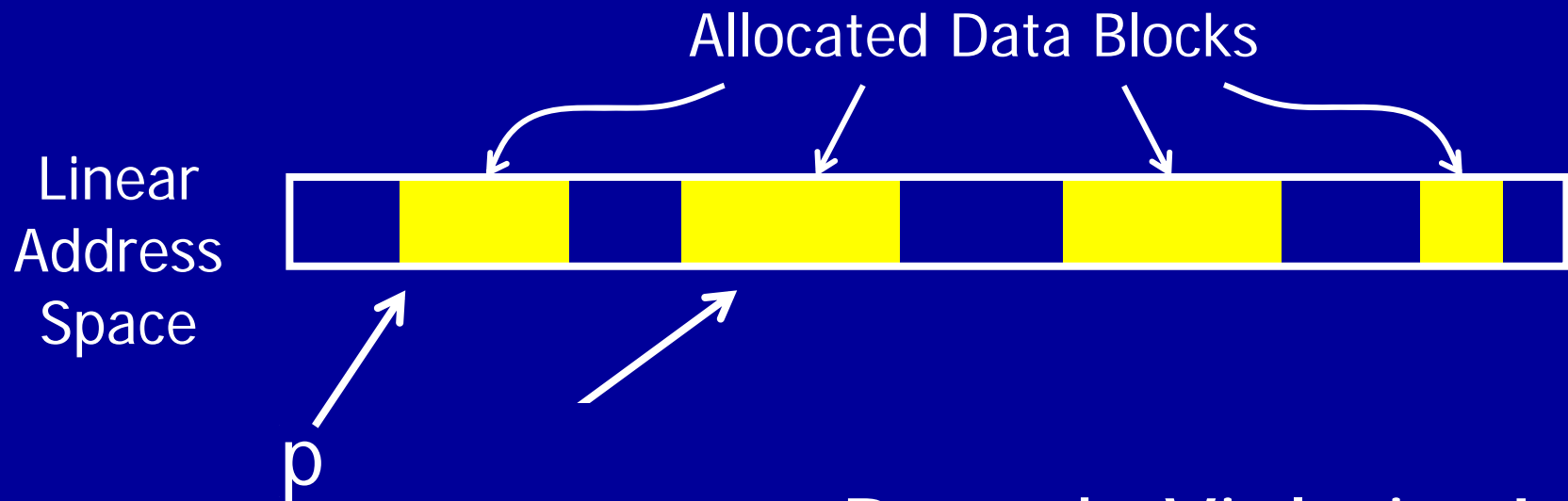
- Software should be acceptable, not correct
 - Acceptability depends on context
 - Software usually part of a larger system
 - Larger system can often tolerate errors
- Cost and difficulty of developing software is roughly proportional to amount of correctness

New Assumption Basis

- Software should be acceptable, not correct
 - Acceptability depends on context
 - Software usually part of a larger system
 - Larger system can often tolerate errors
- Cost and difficulty of developing software is roughly proportional to amount of correctness
- **Obvious conclusion**
 - **If you want to reduce cost and difficulty of producing acceptable software**
 - **Make more errors acceptable**
 - **Leave more errors in system**

Addressing Errors

Out of Bounds Accesses

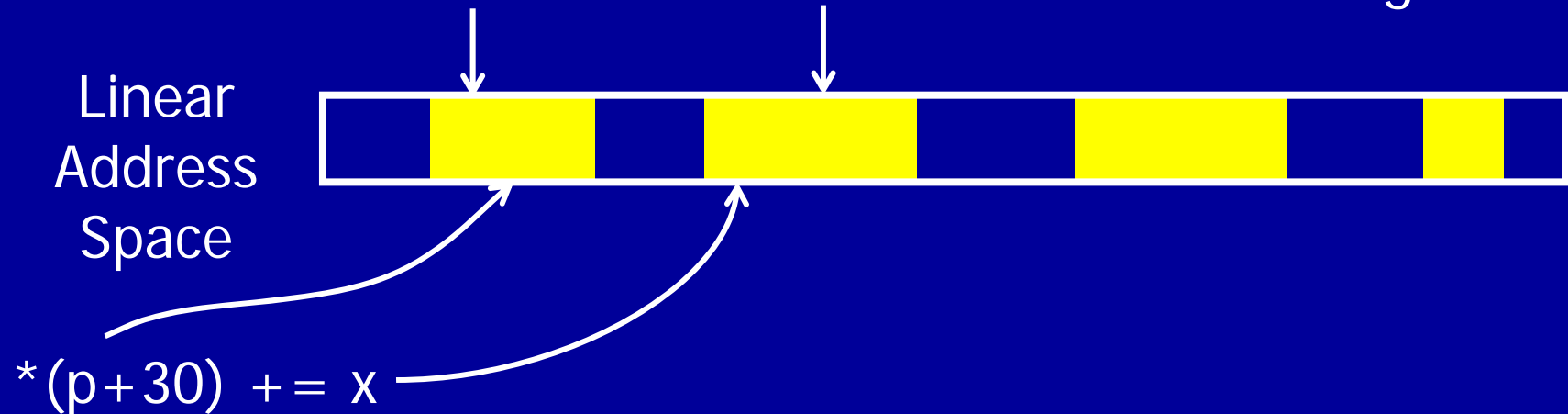


Bounds Violation!

- Data corruption...
- Segmentation violation...
- Security Vulnerability

Bounds Checked C Programming Model

Base Data Block \neq Accessed Data Block \Rightarrow Illegal Access!



- Track base data block for each pointer
- Dynamically check that each access falls within the bounds of the base data block
- If not, access is illegal

Jones&Kelly IWAD 1997, Ruwase&Lam NDSS 2004

Traditional Bounds Check Philosophy

- Bounds violation (illegal access) is irrefutable evidence of an error in the program
- Unsafe to continue because program is outside its anticipated execution envelope
- Two reasonable alternatives
 - Terminate computation
 - Throw exception

Our Philosophy

- Should be able to ignore addressing errors
 - Perform dynamic bounds checks
 - Discard out of bounds writes
 - Manufacture values for out of bounds reads
 - Continue executing
- Called *failure-oblivious computing*

Consequences of Failure-Oblivious Computing

- No corruption of other data blocks
- No segmentation violation
- No abnormal termination
- No addressing exceptions
- No security vulnerabilities
(from out of bounds writes)

Consequences of Failure-Oblivious Computing

- No corruption of other data blocks
- No segmentation violation
- No abnormal termination
- No addressing exceptions
- No security vulnerabilities
(from out of bounds writes)

But what about errors in continued execution ?!?!

Experiment

- Implemented compiler that generates failure-oblivious code (based on Ruwase & Lam's CRED compiler)
- Acquired programs (servers)
 - Pine, Mutt (mail user agent)
 - Apache (web server)
 - Sendmail (mail transfer agent)
 - Midnight Commander (file manager)
- Found bounds violation errors
 - Potential security vulnerabilities
 - Vulnerability-tracking web sites

Experiment

- Generated three versions of each program
 - SC – standard compilation
 - BC – bounds check compilation
(terminates program on bounds violations)
 - FO – failure-oblivious compilation
(continues through bounds violations)
- Ran each version on workload containing inputs that attempted to exploit vulnerability

Results

	Secure?			Initializes?			Continues Correctly?			Correct for Attack Input?		
	SC	BC	FO	SC	BC	FO	SC	BC	FO	SC	BC	FO
Pine	✗	●	●	◆	◆	●	✗	✗	●	✗	✗	●
Mutt	✗	●	●	◆	◆	●	✗	✗	●	✗	✗	✗
Sendmail	✗	●	●	●	✗	●	✗	—	●	✗	—	◆
Apache	✗	●	●	●	●	●	●	●	●	✗	✗	●
Midnight	✗	●	●	●	◆	●	✗	✗	●	✗	✗	◆

✗ No

● Yes

◆ Maybe

— Not Applicable

Why?

Let's take a look at some errors

Pine Error

Send mail message

Carefully crafted FROM field



```
To: john.doe@cs.uni.edu  
From: <code>\\"\\\"\\\"\\\"<addr>
```

Pine Error

Send mail message

Carefully crafted FROM field

To: john.doe@cs.uni.edu
From: <code>\\"\\\"\\\"\\\"<addr>

Mail Folder

To: john.doe@cs.uni.edu
From: <code>\\"\\\"\\\"\\\"<addr>

Pine Vulnerability

Send mail message

Carefully crafted FROM field

```
To: john.doe@cs.uni.edu  
From: <code>\\"\"\"\"\"<addr>
```

Mail Folder

```
To: john.doe@cs.uni.edu  
From: <code>\\"\"\"\"\"<addr>
```

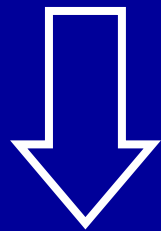
Pine

- Pine reads message
- Processes FROM field
- Overflows buffer

Pine



FROM field



rfc822_cat



Buffer

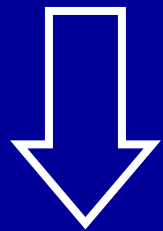
Two Quoting Rules

- 1) \ replaced by \\
 - 2) " replaced by \"
- FROM field can double in size
 - Buffer is not twice size of FROM field

Pine

a	\	"	\	"	\	"	\
---	---	---	---	---	---	---	---

FROM field



rfc822_cat

a	\	\	\	"	\	\	\	"	\	\	\	"	\	\
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

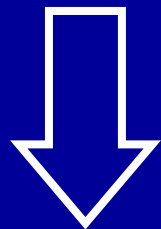
Two Quoting Rules

- 1) \ replaced by \\
 - 2) " replaced by \"
- FROM field can double in size
 - Buffer is not twice size of FROM field

Pine and Failure-Oblivious Computing

a	\	"	\	"	\	"	\
---	---	---	---	---	---	---	---

FROM field



rfc822_cat

a	\	\	\	"	\	\	\
---	---	---	---	---	---	---	---

Buffer

- Writes beyond end of buffer discarded
- FROM field effectively truncated

```
leut@cycleserv2:/home/safec/pinedev/pine4.44-fo
PINE 4.44 MESSAGE INDEX Folder: crash Message 12 of 13
+ 1 May 20 "\"\"\"\"\"\"\"\"\"\" (593) Exploit for pine 4.44
+ 2 May 19 "\"\"\"\"\"\"\"\"\"\" (1563) Exploit for pine 4.44
+ 3 May 20 "\"\"\"\"\"\"\"\"\"\" (593) Exploit for pine 4.44
4 Sep 10 "\"\"\"\"\"\"\"\"\"\" (703) Exploit for pine 4.44
5 Sep 13 "\"\"\"\"\"\"\"\"\"\" (1690) Exploit for pine 4.44
6 Sep 13 "\"\"\"\"\"\"\"\"\"\" (1690) Exploit for pine 4.44
7 Sep 13 "\"\"\"\"\"\"\"\"\"\" (1690) Exploit for pine 4.44
8 Sep 13 "\"\"\"\"\"\"\"\"\"\" (1690) Exploit for pine 4.44
+ 9 Dec 3 "\"abdracasfkshkfs (1681) Exploit for pine 4.44
+ 10 Dec 3 "\"abdracasfkshkfs (1681) Exploit for pine 4.44
+ 11 Dec 3 "\"abdracasfkshkfs (1681) Exploit for pine 4.44
+ 12 Dec 3 "\"abdracasfkshkfs (1681) Exploit for pine 4.44
+ 13 Dec 3 "\"abdracasfkshkfs (1448) Exploit for pine 4.44

? Help      FldrList  PreVMsg    PreVPage  Delete     Reply
OTHER CMDS  [ViewMsg] NextMsg    Spc NextPage  Undelete   Forward
```

FROM fields that overflow buffer
Truncated at 18 characters in user interface

Apache

To redirect all gif files to corresponding jpg files on another server, use this RedirectMatch command:

```
RedirectMatch (.*)\.gif$ http://www.images.com$1.jpg
```

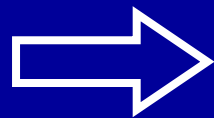


Capture regular
expression here



Reference regular
expression here

/u/blue.gif



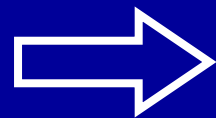
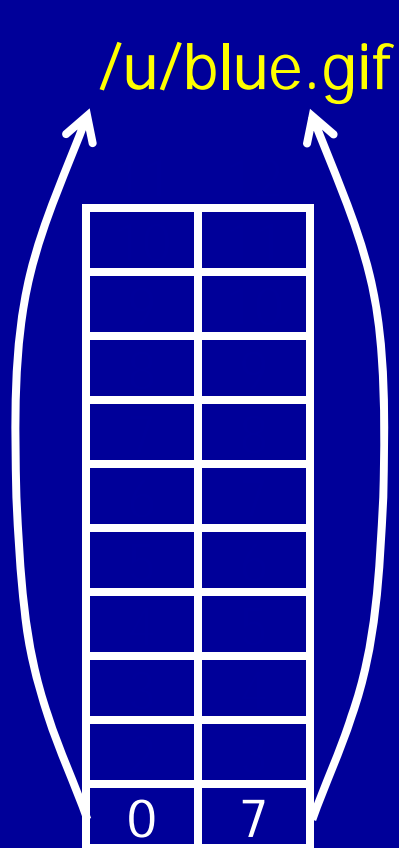
http://www.images.com/u/blue.jpg

Captured regular
expression

Captured regular
expression

Storing Captured Regular Expression Information

RedirectMatch (.*)\.gif\$ http://www.images.com\$1.jpg



`http://www.images.com/u/blue.jpg`

- Array of structures that stores captured regular expressions
 - Start offset
 - End offset
- Array has ten elements
- Space for ten captured regular expressions

Storing Captured Regular Expression Information

RedirectMatch C(0*)(1*)(2*)(3*)(4*)(5*)(6*)(7*)(8*)(9*)(A*)
index.html?input=\$11

C0123456789A  index.html?input=01

11	12
10	11
9	10
8	9
7	8
6	7
5	6
4	5
3	4
2	3
1	2

 Offsets for eleventh captured regular expression written beyond array bounds

Apache and Failure-Oblivious Computing

RedirectMatch C(0*)(1*)(2*)(3*)(4*)(5*)(6*)(7*)(8*)(9*)(A*)
index.html?input=\$11

C0123456789A  index.html?input=01

 Offsets for eleventh captured regular expression discarded

10	11
9	10
8	9
7	8
6	7
5	6
4	5
3	4
2	3
1	2

Apache and Failure-Oblivious Computing

RedirectMatch C(0*)(1*)(2*)(3*)(4*)(5*)(6*)(7*)(8*)(9*)(A*)
index.html?input=\$11

C0123456789A  index.html?input=01

 Offsets for eleventh captured regular expression discarded

10	11
9	10
8	9
7	8
6	7
5	6
4	5
3	4
2	3
1	2

- What about referencing eleventh, ..., captured regular expressions?
- Can't reference - names are \$1-\$9
- Same fix as developers applied later...

Mutt

```
static char *  
utf8_to_utf7 (const char *u8, int u8len) {  
    char *buf = malloc(2*u8len+1);  
    ...  
    return buf;  
}
```

utf8_to_utf7 called to translate folder names
from utf8 to utf7 format

Mutt

```
static char *  
utf8_to_utf7 (const char *u8, int u8len) {  
    char *buf = malloc(2*u8len+1);  
    ...  
    return buf;  
}
```

↑
Too small

Mutt

```
static char *
```

```
utf8_to_utf7 (const char *u8, int u8len) {
```

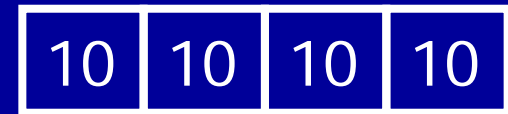
```
    char *buf = malloc(2*u8len+1);
```

```
    ...
```

```
    return buf;
```

```
}
```

↑
Too small



u8



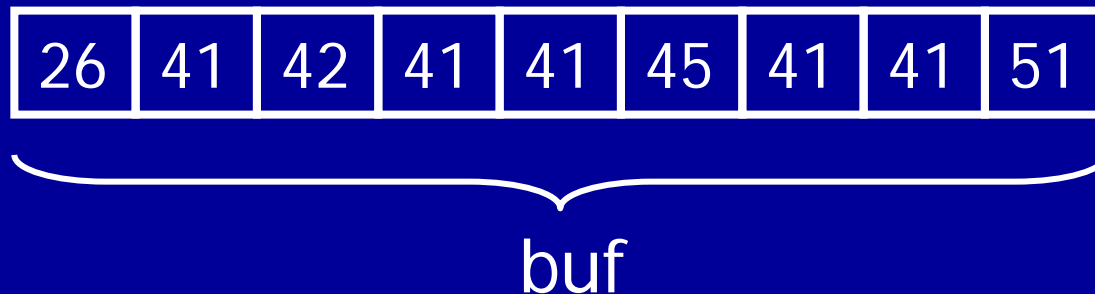
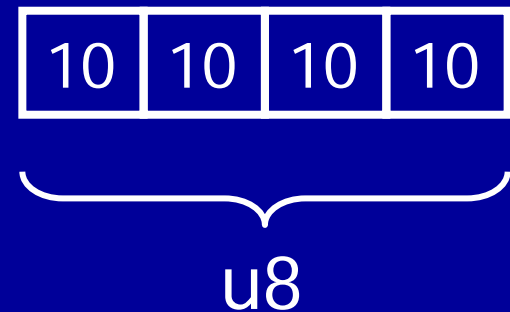
buf

41 42 41 2d 00

Out of bounds writes

Mutt and Failure-Oblivious Computing

- Out of bounds writes discarded
- Converted string is effectively truncated
- Mailbox lookup fails (anticipated error case)
- Mutt remains usable



Summary

- Out of bounds reads and writes access data irrelevant to final result of computation
 - **Pine** - accessed data truncated in user interface
 - **Apache** - inaccessible data
 - **Sendmail** - FROM field fails later length check
 - **Midnight Commander** - same (incorrect) result regardless of content of written data
- Attacks converted into anticipated error cases
 - **Mutt** - folder not found
 - **Sendmail** - FROM field fails later length check
 - **Midnight Commander** - file lookup fails

Expectations for Other Servers

- Effect of failure-oblivious computing
 - Discarding out of bounds writes eliminates global data structure (call stack) corruption
 - Tends keep errors localized
 - Server continues on to subsequent requests
- Servers have short error propagation distances
 - Localized errors in one request
 - Tend not to propagate to next request
- Subsequent requests serviced without errors

Memory Leaks

```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    → int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```

Allocation site

Memory Leaks

```
int *f(int n, int v) {
```

```
    int i, s;
```

```
    s = v * sizeof(int);
```

```
    → int *t = malloc(s); ●
```

```
    for (i = 0; i < n; i++)
```

```
        t[i] = v;
```

```
    return t;
```

```
}
```

Memory Leaks

```
int *f(int n, int v) {
```

```
    int i, s;
```

```
    s = v * sizeof(int);
```

```
    → int *t = malloc(s); ● ●
```

```
    for (i = 0; i < n; i++)
```

```
        t[i] = v;
```

```
    return t;
```

```
}
```

Memory Leaks

```
int *f(int n, int v) {
```

```
    int i, s;
```

```
    s = v * sizeof(int);
```

```
    → int *t = malloc(s); ● ● ●
```

```
    for (i = 0; i < n; i++)
```

```
        t[i] = v;
```

```
    return t;
```

```
}
```

Memory Leaks

```
int *f(int n, int v) {
```

```
    int i, s;
```

```
    s = v * sizeof(int);
```

```
    → int *t = malloc(s); ● ● ● ●
```

```
    for (i = 0; i < n; i++)
```

```
        t[i] = v;
```

```
    return t;
```

```
}
```

Memory Leaks

```
int *f(int n, int v) {
```

```
    int i, s;
```

```
    s = v * sizeof(int);
```

```
    → int *t = malloc(s); ● ● ● ● ● ...
```

```
    for (i = 0; i < n; i++)
```

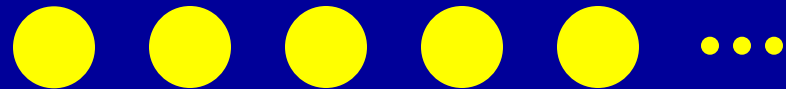
```
        t[i] = v;
```

```
    return t;
```

```
}
```

Memory Leaks

```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    → int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Memory Leak Conditions

- Unbounded allocation
- Unbounded number of dead objects
- Dead objects not reclaimed
 - C – explicit free not called
 - Java – dead objects remain reachable

Memory Leak Issues

- Wastes resources, bogs down system
- Exhaust address space, program fails
- Similar problems with other resources
 - File handle leaks
 - Thread leaks
 - Process leaks

Memory Leaks

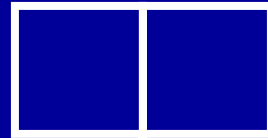
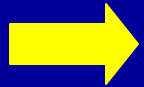
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    → int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Program
Accesses At
Most Last **k**
Allocated
Objects

Cyclic Memory Allocation

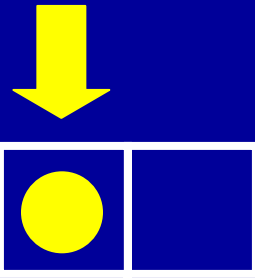
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer

Cyclic Memory Allocation

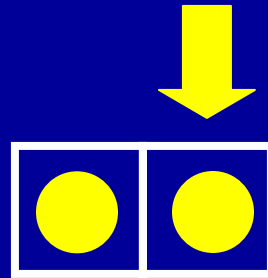
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    → int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer

Cyclic Memory Allocation

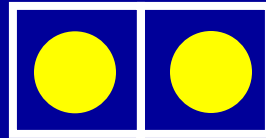
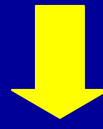
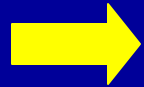
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    → int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer

Cyclic Memory Allocation

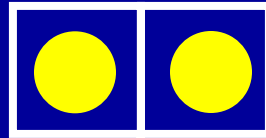
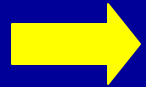
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer

Cyclic Memory Allocation

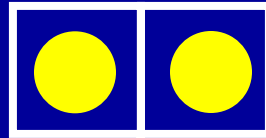
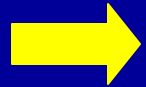
```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer
Frees turn into no-ops

Cyclic Memory Allocation

```
int *f(int n, int v) {  
    int i, s;  
    s = v * sizeof(int);  
    int *t = malloc(s);  
    for (i = 0; i < n; i++)  
        t[i] = v;  
    return t;  
}
```



Preallocate buffer with **k** slots
Cyclically allocate objects out
of buffer
Frees turn into no-ops

Result – no more memory leak!

Cyclic Memory Allocation Options

- **Option 1:** use cyclic allocation for all sites with k
- **Option 2:**
 - Start out using normal allocation
 - Watch for signs of triggered memory leak
 - Count number of outstanding objects at each site with a k
 - Leak signaled when number $\gg k$
 - Switch when leak signaled

How Do We Obtain k ?

- Instrument allocation sites, reads, writes
- Run program on test inputs
- For each allocation site and each test input
 - Observe how far back in allocation stream reads and writes access data
 - Compute k from observations

Standard Response

You didn't run program on ALL inputs.
Isn't it possible for k to be too small for
some inputs you didn't test?
Won't you overlay live data?

Standard Response

You didn't run program on ALL inputs.
Isn't it possible for k to be too small for
some inputs you didn't test?

Won't you overlay live data?

Yes

Two Experimental Questions

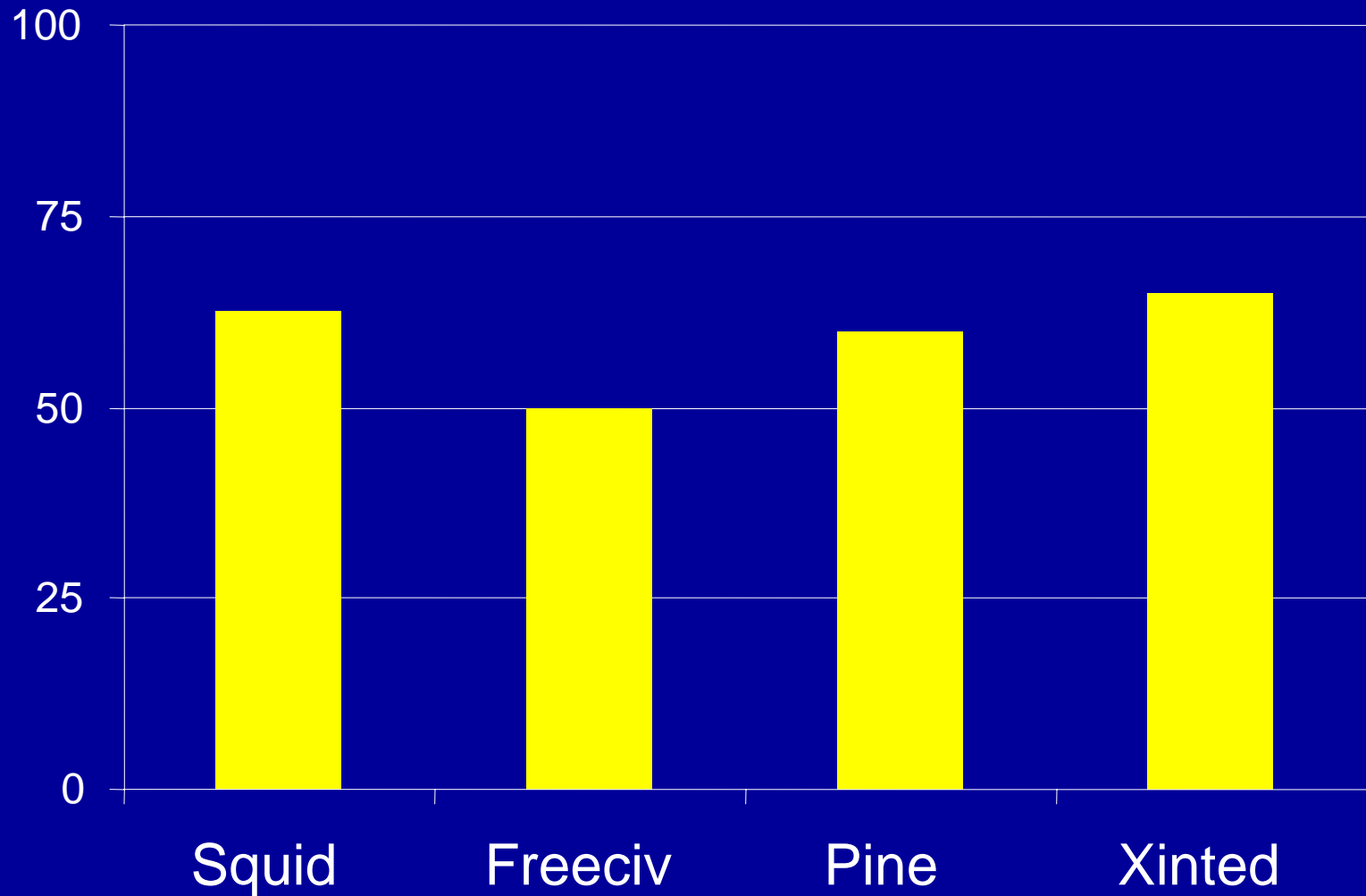
How often is k too small?

What happens when you overlay live data?

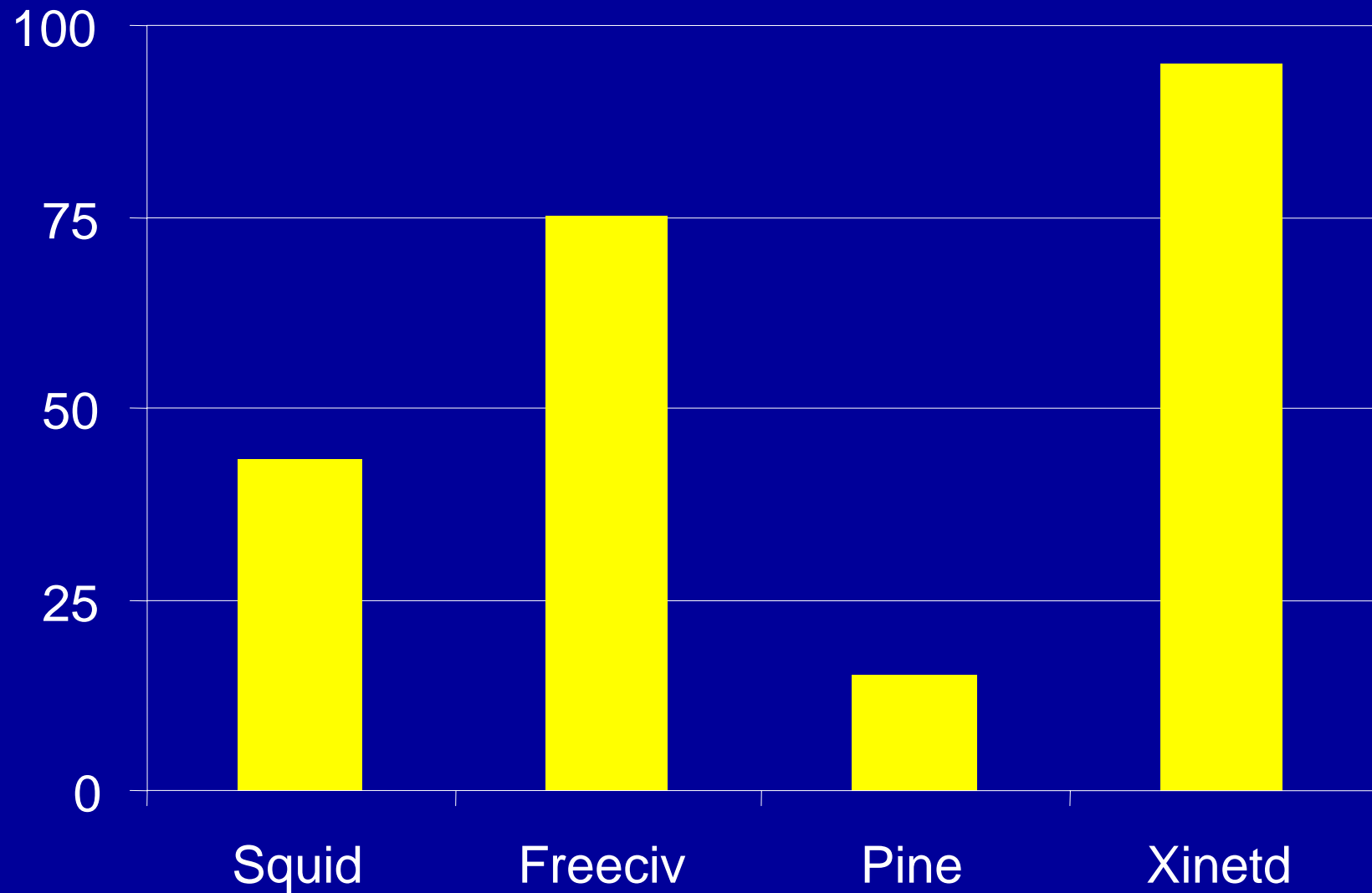
Methodology

- Acquired applications
 - Squid – web proxy cache
 - Xinetd – manages connections, requests
 - Freeciv – interactive multiple player game
 - Pine – mail client
- Training runs to find allocation sites with a **k**
- Validation runs to see if **k** too small

Percentage of Allocation Sites With a k



Percentage of Memory Allocated at Sites With a k



Memory Leaks?

- Squid has memory leak in SNMP module, vulnerable to denial of service attack
- Xinetd leaks memory whenever it rejects a connection, vulnerable to denial of service attack
- Freeciv leaks boolean array storing presence or absence of threats from ocean

Memory Leaks?

- Squid has memory leak in SNMP module, vulnerable to denial of service attack
- Xinetd leaks memory whenever it rejects a connection, vulnerable to denial of service attack
- Freeciv leaks boolean array storing presence or absence of threats from ocean

All of these leaks occur at sites with a k
Our technique eliminates them all

Memory Leaks?

- Squid has memory leak in SNMP module, vulnerable to denial of service attack
- Xinetd leaks memory whenever it rejects a connection, vulnerable to denial of service attack
- Options
 - Developer finds and fixes memory leak (service is unavailable until leak fixed)
 - Apply cyclic memory allocation automatically (can find and fix leak immediately) (no service interruption at all)

Memory Leaks?

What??

- Squid has memory leak in **SNMP module**, vulnerable to denial of service attack
- Xinetd leaks memory whenever it rejects a connection, vulnerable to denial of service attack
- Options
 - Developer finds and fixes memory leak (service is unavailable until leak fixed)
 - Apply cyclic memory allocation automatically (can find and fix leak immediately) (no service interruption at all)



© 2003 ESA - CNES - AIRBUSPACE / Photo Service Optique Vidéo CSG



Any k too Small?

- 160 allocation sites have **k** during training runs
- 1 site has **k** too small during validation runs
 - Objects implement circular doubly linked list of status messages
 - Overlaying causes Pine to dereference null pointer

Conflict Runs

- Take all allocation sites with $k > 1$
- Replace k by $\lceil k/2 \rceil$
- Observe effects
- 8 allocation sites with $k > 2$
 - Infinite loop for 1 of 8 sites
 - Segmentation fault for 2 of 8 sites
 - Functionality impairment for 2 of 8 sites
 - Squid – incorrect SNMP query response
 - Squid – can't process SNMP queries at all
 - Fully functional for 3 of 8 sites

Getting Rid of Infinite Loops

- Record number of times n each loop executes
 - Initialize with training runs
 - Update during production runs
- Transform code so that each loop executes at most $10^3 * n$ iterations
(Don't update n if executes $10^3 * n$ times)
- Result
 - No infinite loop
 - Pine fully functional
 - Some garbage in HTML documents

Effect of Failure-Oblivious Computing

- Pine
 - Execution continues beyond null pointer error

New Trade Off

Traded off correctness

In return for memory leak elimination

- Play a little differently
- Result
 - Programs all survive
 - Fully functional for 6 of 8 sites

Where Are We?

- Increased robustness of existing systems
 - No infinite loops, memory errors, leaks
 - Software will **NEVER** crash
 - Survival preserves desirable behavior in systems with multiple components
- Potential ways to exploit this flexibility
 - Leave more errors in systems
 - Preserve structure of system
 - Eliminate introduction of more errors
 - Less maintenance cost
 - More aggressive development, releases

How You Can Use These Ideas Today

- You have something you need (but don't have)
 - A consistent data structure
 - An output every ten milliseconds
 - Server to survive a given input
 - Eliminate a null pointer dereference
- Find an easy fix to get it
 - Apply fix only where you have problem
 - See if you are comfortable using it

Acknowledgements

Dick Gabriel

- Butler Lampson
- Gerry Sussman
- Saman Amarasinghe
- Viktor Kuncak
- Karen Zee
- Rob Seator
- Arvind
- Cristian Cadar
- Daniel Dumitran
- Dan Roy
- Wes Beebee
- Tudor Leu
- Huu Hai Nguyen
- Brian Demsky

DARPA

Lee Badger

Suggested Reading

- **Acceptability-Oriented Computing**, Martin Rinard (OOPSLA Onwards! 2003)
- **Probabilistic Accuracy Bounds for Fault-Tolerant Computations That Discard Tasks**, Martin Rinard, (ICS 2006)
- **Enhancing Server Availability and Security Through Failure-Oblivious Computing**, Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and William S. Beebee, Jr. (OSDI 2006)
- **Automatic Inference and Enforcement of Data Structure Consistency Specifications**, Brian Demsky, Michael Ernst, Philip Guo, Stephen McCamant, Jeff Perkins, and Martin Rinard (ISSTA 2006)
- **A Dynamic Mechanism for Recovering from Buffer Overflow Attacks**, Stelios Sidiroglou, Giannis Giovanidis, and Angelos D. Keromytis (ISC 2005)

- **DieHard: Probabilistic Memory Safety for Unsafe Languages**, Emery Berger and Benjamin Zorn (PLDI 2006)
- **Microreboot - A Technique for Cheap Recovery**, George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, Armando Fox (OSDI 2004)
- **Rx: Treating Bugs as Allergies – A safe method for surviving software failures**, Feng Qin, Joseph Tucek, Jagadessan Sundaresan, Yuanyuan Zhou (SOSP 2005)

Post Talk Comments

- Development is different from deployment
 - Want lots of checks during development to find bugs early
 - Can then turn checks off during deployment if your goal is to have a survivable system
- Be sure you understand the context in which your software is going to be used before you decide whether you want to continue or not