It Depends...
novice
advanced beginner

novice
proficient
competent
advanced beginner
novice
expert
proficient
competent
advanced beginner
novice
expert
proficient
competent
advanced beginner
novice
expert
proficient
competent
advanced beginner
novice
intuitive
analytical
expert   context-sensitive
proficient
competent
advanced beginner
novice   context-free
expert  “it depends...”
proficient
competent
advanced beginner
novice    “always/never...”
shuhari
shu·ha·ri
守破離
imitate
守破離 innovate
What do I think?
This code sucks.

Teedy Deigh
The Way of the Consultant
What do I think?

Well... it’s not all bad! Nothing that some aggressive, merciless and inconsiderate refactoring couldn’t solve.

Teedy Deigh
The Way of the Consultant
What do I think?

Although there are aspects of the system’s design that are sound, the solution as a whole may be better aligned with the needs of the business by leveraging the synergies of complementary solution paths. The resulting amelioration of quality will be further enhanced by the displacement of vestigial solution components extant from the status quo.

Teedy Deigh
The Way of the Consultant
What do I think?
It depends.

Teedy Deigh
The Way of the Consultant
Beware the Share

Collective Wisdom from the Experts

97 Things Every Programmer Should Know

Udi Dahan
As I worked through my first feature, I took extra care to put in place everything I had learned — commenting, logging, pulling out shared code into libraries where possible, the works.

The code review that I had felt so ready for came as a rude awakening — reuse was frowned upon!

Udi Dahan
How could this be? Throughout college, reuse was held up as the epitome of quality software engineering. All the articles I had read, the textbooks, the seasoned software professionals who taught me — was it all wrong?
It turns out that I was missing something critical.
Context.
The fact that two wildly different parts of the system performed some logic in the same way meant less than I thought.

Up until I had pulled out those libraries of shared code, these parts were not dependent on each other. Each could evolve independently.
directives
principles
patterns
context
conflicting forces
problem
configuration
consequences
patterns
design
code
There’s no such thing as bad weather, only unsuitable clothing.
mutex
bottleneck
scoped_lock
synchronized
String
Immutability Changes Everything

We need it, we can afford it, and the time is now.

BY PAT MELLAND

Lasch has become harder; last latency loses lots of opportunities. Keeping copies of lots of data is now realistic, and one payoff is reduced access challenges.

Storage is increasing as the cost per terabyte of disk keeps dropping. It means a lot of data can be kept in the long run. Distribution is increasing as more and more data are spread across a great distance. Data within a data center seems away. Data within a many-terabyte system may seem "far away." Analyzing increasing when trying to reason with systems that are far away—stuff has happened since you heard the news. Can you take any with incomplete knowledge? Can wait for enough knowledge?

Turtles all the way down. serious technological impacts.
There’s no such thing as thread-unsafe code, only unsuitable threading.
StringBuffer
StringBuilder
static
using the equivalent of the following algorithm.

```c
char *asctime(const struct tm *timeptr)
{
    static const char wday_name[7][3] = {
        "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static const char mon_name[12][3] = {
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];
    sprintf(result,
            "%3s %3s%3d %2d:%2d:%2d %d\n",
            wday_name[timeptr->tm_wday],
            mon_name[timeptr->tm_mon],
            timeptr->tm_mday, timeptr->tm_hour,
            timeptr->tm_min, timeptr->tm_sec,
            1900 + timeptr->tm_year);
    return result;
}
```

Returns

The `asctime` function returns a pointer to the string.

7.23.3.2 The `ctime` function

Synopsis

```c
#include "time.h"
struct tm *ctime(const char *s);
```

Description

The `ctime` function converts a null-terminated `char` string containing a
broken-down time, expressed in the local time zone

Returns

The `ctime` function returns a pointer to the structure

7.23.3.4 The `localtime` function

Synopsis

```c
#include "time.h"
struct tm *localtime(const time_t *t);
```

Description

The `localtime` function converts a broken-down time,

Returns

The `localtime` function returns a pointer to the structure
gets
char * gets(char * s);
puts
gets
void example(void)
{
    char s[32];
    puts("What is your full name?");
    gets(s);
    ...
}
void example(void)
{
    char s[32];
    puts("What is your full name?");
    gets(s);
    ...
}

S-Programs
P-Programs
E-Programs

Meir M Lehman
“Programs, Life Cycles, and Laws of Software Evolution”
S-Programs

P-Programs

E-Programs

Programs whose function is formally defined by and derivable from a specification.

Meir M Lehman

“Programs, Life Cycles, and Laws of Software Evolution”
WRITING CORRECT PROGRAMS

In the late 1960s people were talking about the promise of programs that verify the correctness of other programs. Unfortunately, it is now the middle of the 1980s, and, with precious few exceptions, there is still little more than talk about automated verification systems. Despite unrealized expectations, however, the research on program verification has given us something far more valuable than a black box that gobbles programs and flashes “good” or “bad”—we now have a fundamental understanding of computer programming.

The purpose of this column is to show how that fundamental understanding can help programmers write correct programs. But before we get to the subject itself, we must keep it in perspective. Coding skill is just one small part of writing correct programs. The majority of the task is the subject of the three previous columns: problem definition, algorithm design, and data structure selection. If you perform those tasks well, then writing correct code is usually easy.

The Challenge of Binary Search

In most of the past few columns, we made the assumption that the user would supply a sorted list to the program. In this case, the binary search is quite simple. But what happens if the list is not sorted? Can the binary search be made to work with any list? I’ve given this problem as an in-class assignment in courses at Bell Labs and IBM. The professional programmers had one hour (sometimes more) to convert the above description into a program in the language of their choice; a high-level pseudo-code was fine. At the end of the specified time, almost all the programmers reported that they had correct code for the task. We would then take 30 minutes to examine their code, which the programmers did with test cases. In many different classes and with over a hundred programmers, the results varied little: 90 percent of the programmers found bugs in their code (and I wasn’t always convinced of the correctness of the code in which no bugs were found).

I found this amazing: only about 10 percent of professional programmers were able to get this small program right. But they aren’t the only ones to find this task difficult. In the history in Section 6.2.1 of his Sorting and Searching, Knuth points out that while the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962.
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }

    return -(low + 1);  // key not found.
}
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1);  // key not found.
}
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = low + ((high - low) / 2);
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }

    return -(low + 1); // key not found.
}
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }

    return -(low + 1); // key not found.
}
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }

    return -(low + 1); // key not found.
}

Probably faster, and arguably as clear
Probably faster but may be obscure to most Java developers (including me)

Alberto Savoia
If the programmer can simulate a construct faster than the compiler can implement the construct itself, then the compiler writer has blown it badly.

Guy L Steele, Jr
Simple Testing Can Prevent Most Critical Failures

An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U Jain & Michael Stumm

usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf
A majority of the production failures (77%) can be reproduced by a unit test.
The general lesson that I take away from this bug is humility: It is hard to write even the smallest piece of code correctly, and our whole world runs on big, complex pieces of code.

Joshua Bloch

ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html
The acceptability of a solution is determined by the environment in which it is embedded.
THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
AI is characterized by output that isn’t strictly dependent on the input or on the algorithm: the output of an AI system depends critically on a training process, in which the program learns how to perform its task. Training differentiates AI from traditional software applications and data analysis.
Explanations must be wrong. They cannot have perfect fidelity with respect to the original model. If the explanation was completely faithful to what the original model computes, the explanation would equal the original model, and one would not need the original model in the first place, only the explanation.

Cynthia Rudin

“Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”
“Machine learning” is a fancy way of saying “finding patterns in data”.

Laurie Penny

theguardian.com/commentisfree/2017/apr/20/robots-racist-sexist-people-machines-ai-language
Of course, as Lydia Nicholas [...] explains, all this data “has to have been collected in the past, and since society changes, you can end up with patterns that reflect the past. If those patterns are used to make decisions that affect people’s lives you end up with unacceptable discrimination.”

Laurie Penny

theguardian.com/commentisfree/2017/apr/20/robots-racist-sexist-people-machines-ai-language
Programs that mechanize a human or societal activity. The program has become a part of the world it models, it is embedded in it.
The Making of a Fly: The Genetics of Animal Design (Paperback)
by Peter A. Lawrence

Always pay through Amazon.com’s Shopping Cart or 1-Click.
Learn more about Safe Online Shopping and our safe buying guarantee.

New 1-2 of 2 offers

<table>
<thead>
<tr>
<th>Price + Shipping</th>
<th>Condition</th>
<th>Seller Information</th>
<th>Buying Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1,730,045.91</td>
<td>New</td>
<td>profnath</td>
<td>Add to Cart</td>
</tr>
<tr>
<td>$2,198,177.95</td>
<td>New</td>
<td>bordeebook</td>
<td>Add to Cart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Seller Rating: 5 Star: 93% positive over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!</td>
<td></td>
</tr>
</tbody>
</table>

Price at a Glance
- List: $70.00
- Used: from $35.54
- New: from $1,730,045.91

Have one to sell? Sell yours here
$1,730,045.91
$2,198,177.95
<table>
<thead>
<tr>
<th>Seller</th>
<th>Rating</th>
<th>Condition</th>
<th>Price + Shipping</th>
<th>Seller Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>profath</em></td>
<td>⭐⭐⭐⭐⭐</td>
<td>New</td>
<td>$1,730,045.91</td>
<td>In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.</td>
</tr>
<tr>
<td><em>bordebook</em></td>
<td>⭐⭐⭐⭐⭐</td>
<td>New</td>
<td>$2,198,177.95</td>
<td>In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!</td>
</tr>
</tbody>
</table>

Price at a Glance
- List: $70.00
- Used: from $35.54
- New: from $1,730,045.91

Have one to sell? Sell here.
<table>
<thead>
<tr>
<th></th>
<th>profnath</th>
<th>bordeebook</th>
<th>profnath over previous bordeebook</th>
<th>bordeebook over profnath</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-Apr</td>
<td>$1,730,045.91</td>
<td>$2,198,177.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9-Apr</td>
<td>$2,194,443.04</td>
<td>$2,788,233.00</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>10-Apr</td>
<td>$2,783,493.00</td>
<td>$3,536,675.57</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>11-Apr</td>
<td>$3,530,663.65</td>
<td>$4,486,021.69</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>12-Apr</td>
<td>$4,478,395.76</td>
<td>$5,690,199.43</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>13-Apr</td>
<td>$5,680,526.66</td>
<td>$7,217,612.38</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>Date</td>
<td>Profnath</td>
<td>Bodedebook</td>
<td>Profnath over previous Bodedebook</td>
<td>Bodedebook over Profnath</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>------------</td>
<td>----------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>8-Apr</td>
<td>$1,730,049.91</td>
<td>$2,998,201.26</td>
<td>0.998300</td>
<td>1.7059</td>
</tr>
<tr>
<td>9-Apr</td>
<td>$2,194,441.04</td>
<td>$2,788,723.00</td>
<td>0.998300</td>
<td>1.27059</td>
</tr>
<tr>
<td>10-Apr</td>
<td>$2,783,493.00</td>
<td>$3,559,075.50</td>
<td>0.998300</td>
<td>1.27059</td>
</tr>
<tr>
<td>11-Apr</td>
<td>$3,530,663.65</td>
<td>$4,486,021.69</td>
<td>0.998300</td>
<td>1.27059</td>
</tr>
<tr>
<td>12-Apr</td>
<td>$4,478,395.76</td>
<td>$5,690,199.43</td>
<td>0.998300</td>
<td>1.27059</td>
</tr>
<tr>
<td>13-Apr</td>
<td>$5,680,526.66</td>
<td>$7,217,612.38</td>
<td>0.998300</td>
<td>1.27059</td>
</tr>
<tr>
<td>Date</td>
<td>Profnath</td>
<td>Bardeebook</td>
<td>Profnath over previous</td>
<td>Bordeebook over nominal</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>------------</td>
<td>------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>8-Apr</td>
<td>$1,730,045.9</td>
<td>$2,198,777.95</td>
<td>1.27509</td>
<td>0.99830</td>
</tr>
<tr>
<td>9-Apr</td>
<td>$2,194,443.0</td>
<td>$2,724,233.00</td>
<td>1.27509</td>
<td>1.27059</td>
</tr>
<tr>
<td>10-Apr</td>
<td>$2,783,493.0</td>
<td>$3,350,663.65</td>
<td>1.27509</td>
<td>1.27059</td>
</tr>
<tr>
<td>11-Apr</td>
<td>$3,530,663.65</td>
<td>$4,486,021.69</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>12-Apr</td>
<td>$4,478,395.76</td>
<td>$5,690,199.43</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
<tr>
<td>13-Apr</td>
<td>$5,680,526.66</td>
<td>$7,217,612.38</td>
<td>0.99830</td>
<td>1.27059</td>
</tr>
</tbody>
</table>
Always design a thing by considering it in its next larger context.

Eliel Saarinen
THE HITCHHIKER'S GUIDE TO THE GALAXY

DOUGLAS ADAMS

Based on the famous Radio series
There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.

There is another theory which states that this has already happened.
Sterling flash crash

£/$, 6-7 October

The pound has dived on Asian markets with automated trading being blamed for the volatility.
Digital devices tune out small errors while creating opportunities for large errors.

Earl Wiener
S-Programs
P-Programs
E-Programs

Meir M Lehman
“Programs, Life Cycles, and Laws of Software Evolution”
Defined
 Undefined
 Undefined
 Undefined
Definable  Definable  Undefinable
To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge.

Grace Hopper
0. lack of ignorance
1. lack of knowledge
2. lack of awareness
3. lack of process
4. meta-ignorance
known knowns
known unknowns
unknown unknowns
unknowable unknowns
known knowns
known unknowns
unknown unknowns
unknowable unknowns
known knowns
known unknowns
unknown unknowns
unknowable unknowns
known knowns
known unknowns
unknown unknowns
unknowable unknowns
known knowns
known unknowns
unknown unknowns
unknowable unknowns
I know that I know nothing.

Socrates * Possibly
Ultra-Large-Scale Systems
Unknowable
Decentralised
Evolving
Heterogeneous
Failing
A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.

Leslie Lamport
Fire in California, can’t read your ebook in Pennsylvania

Well, this is awkward...

Due to the fires and power outages in California, oreilly.com is unavailable.
It is a feature of a distributed system that it may not be in a consistent state, but it is a bug for a client to contradict itself.
Brewer’s theorem
CAP theorem
Consistency
Availability
Partition tolerance
Consistency
Availability
Partition tolerance
Consistency
Availability
Partition tolerance
Consistency
Availability
Partition tolerance
\[ \Delta x \Delta p \geq \frac{\hbar}{2} \]
THE HITCHHIKER'S GUIDE TO THE GALAXY

DOUGLAS ADAMS

Based on the famous Radio series
We demand rigidly defined areas of doubt and uncertainty!
On Formally Undecidable Propositions Of Principia Mathematica And Related Systems

KURT GÖDEL

Translated by
B. MELTZER

Introduction by
R B. BRAITHWAITE
In 1911 Russell & Whitehead published Principia Mathematica, with the goal of providing a solid foundation for all of mathematics.

In 1931 Gödel’s Incompleteness Theorem shattered the dream, showing that for any consistent axiomatic system there will always be theorems that cannot be proven within the system.

Adrian Colyer
blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/
All consistent axiomatic formulations of number theory include undecidable propositions.
How long is a piece of string?
size_t strlen(const char * s) {
    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
size_t strlen(const char * s)
{
    assert(s != NULL);
    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
size_t strlen(const char * s) {
    assert(s != NULL);
    assert(∃n (s[n] == '\0'));
    assert(∀i∈0..n (s+i is a valid pointer));
    size_t n = 0;
    while (s[n] != '\0')
        ++n;
    return n;
}
void well_defined(void)
{
    char s[] = "Be excellent to each other";
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
void insufficient_space(void)
{
    char s[5] = "Bogus";
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
void undefined_pointer_value(void)
{
    char * s;
    printf("\"%s\" -> %zu\n", s, strlen(s));
}
One premise of many models of fairness in machine learning is that you can measure (‘prove’) fairness of a machine learning model from within the system – i.e. from properties of the model itself and perhaps the data it is trained on.

To show that a machine learning model is fair, you need information from outside of the system.

Adrian Colyer

blog.acolyer.org/2020/02/03/measure-mismeasure-fairness/
AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.¹

By Alonzo Church.

1. Introduction. There is a class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function $f$ of $n$ positive integers, such that $f(x_1, x_2, \cdots, x_n) = 2^2$ is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving $x_1, x_2, \cdots, x_n$ as free variables.

An example of such a problem is the problem to find a means of determining of any given positive integer $n$ whether or not there exist positive integers $x, y, z$, such that $x^n + y^n = z^n$. For this may be interpreted, required to find an effectively calculable function $f$, such that $f(n)$ is equal to 2 if and only if there exist positive integers $x, y, z$, such that $x^n + y^n = z^n$. Clearly
push := make(chan string)
pop := make(chan string)
go Stack(push, pop)
push<- "ACCU"
push<- "2021"
Println(<-pop)
Println(<-pop)
push := make(chan string)
pop := make(chan string)
go Stack(push, pop)
Println(<-pop)
Non-Empty

Empty

push

pop [#push = #pop]

push

pop [#push > #pop]
This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

CR Categories: 4.20, 4.22, 4.32
This form of failure is known as a deadlock.

This paper suggests that input and output are basic primitives of programming and that parallel computing and controlling sequential processes in systems is an interesting method. When used with the notion of a guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays

CR Categories: 4.20, 4.22, 4.32
func Stack(push <-chan string, pop chan<- string) {
    var items [] string
    for {
        if depth := len(items); depth == 0 {
            items = append(items, <-push)
        } else {
            select {
                case newTop := <-push:
                    items = append(items, newTop)
                case pop<- items[depth - 1]:
                    items = items[:depth - 1]
            }
        }
    }
}
To iterate is human, to recurse divine.

L Peter Deutsch
func Stack(push <-chan string, pop chan<- string) {
    for {
        nonEmptyStack(push, pop, <-push)
    }
}

func nonEmptyStack(push <-chan string, pop chan<- string, top string) {
    for {
        select {
            case newTop := <-push:
                nonEmptyStack(push, pop, newTop)
            case pop<- top:
                return
        }
    }
}
Program testing can be used to show the presence of bugs, but never to show their absence!

Edsger W. Dijkstra

Notes on Structured Programming
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
  case _ = <-pop:
    test.Errorf("empty stack can never be popped")
  case <-time.After(???):
}
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
    case _ = <-pop:
        test.Errorf("empty stack can never be popped")
    case <-time.After(time.Eternity):
}
push, pop := make(chan int), make(chan int)
go Stack(push, pop)

select {
case _ = <-pop:
    test.Errorf("empty stack can never be popped")
case <-time.After(time.Second):
}
Halting Problem
Halting Problem
Prediction is very difficult, especially about the future.

Niels Bohr?
prioritise by business value
prioritise by estimated business value
Humans are allergic to change. They love to say, “We’ve always done it this way.” I try to fight that. That’s why I have a clock on my wall that runs counter-clockwise.

Grace Hopper
Epistemologically speaking, assumptions are the barefoot-trodden Lego bricks in the dark of knowledge. You don't know they're there until you know that they're there. And even if you know there are some there, you don't know exactly where and you'll still end up stepping on some.
The connections between modules are the assumptions which the modules make about each other.

David Parnas
Code you own

Market
Customers, product requirements, domain, governance, etc.

Platform
Programming languages, operating systems, middleware, services, etc.

Code you don't
It’s often not the direct dependencies of your project that bite you, but the dependencies of your dependencies, all the way on down to transitive closure.

Adrian Colyer

blog.acolyer.org/2020/09/21/watchman/
How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

23 Mar 2016 at 01:24, Chris Williams

Updated: Programmers were left staring at broken builds and failed installations on Tuesday after someone toppled the Jenga tower of JavaScript.

A couple of hours ago, Azer Koçulu unpublished more than 250 of his modules from NPM, which is a popular package manager used by JavaScript projects to install dependencies.
When we try to pick out anything by itself, we find it hitched to everything else in the universe.

John Muir
It Depends...