CS130: Software Engineering

Lecture 11: The Art of Readable Code

https://forms.gle/BaKfdVJJGk8Np1pcA

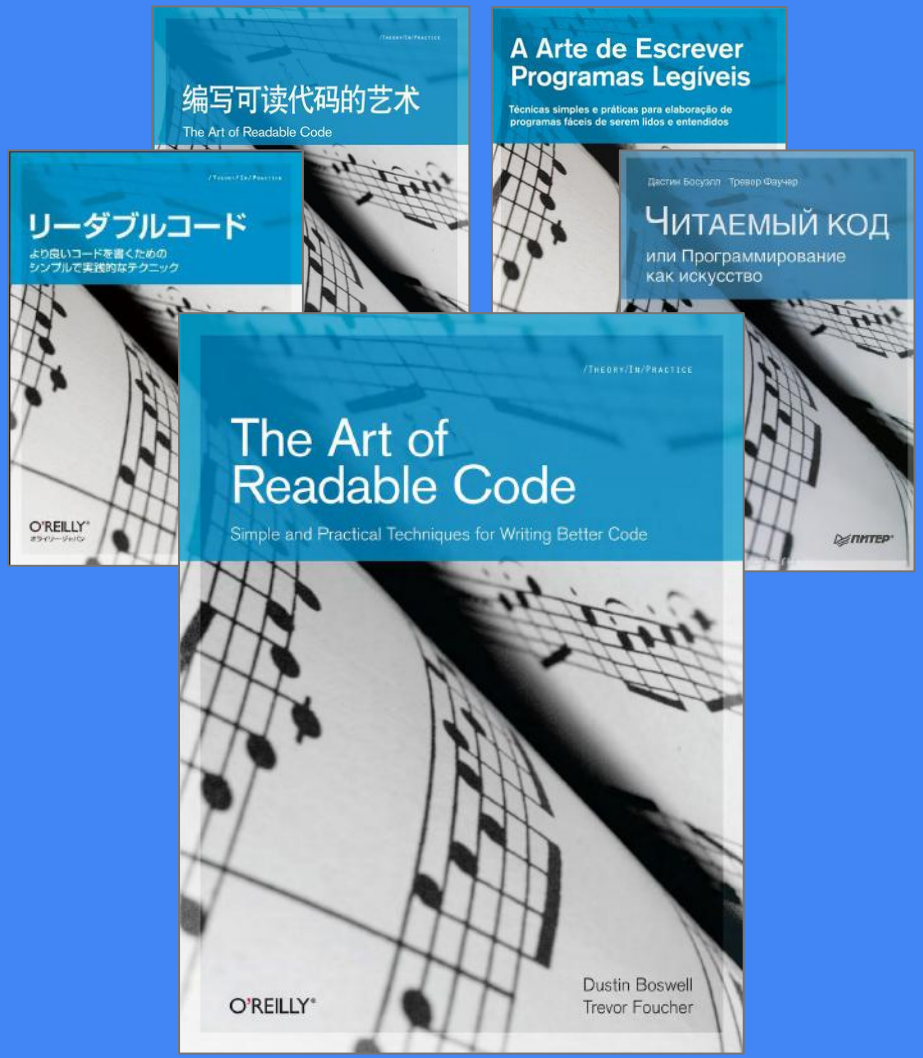
A word: What's the best playlist title in your Spotify/Apple Music/YouTube Music

A tweet: What makes your favorite refactor better than the others?

The Art of Readable Code

The Art of Readable Code

- Book by Dustin Boswell, Trevor Foucher
- Slides by Dustin Boswell



Outline

- Why code readability
- The Fundamental Theorem of Readability
- Code length
- Naming
- Commenting
- Comparison expressions
- Unnecessary variables
- Avoiding complexity

Why Code Readability?

Yes, this actually compiles and runs

```
#define E  printf(                                  """\033["
main(i,f,x,y,L,B,F                                  ,s,t,Y,c,
h){f=-5,B=0;while(                                  !B){for(y
=0;y<16;y++){for(L                                  =3,i=0;c=
"ldldiktljbip"[i];i++){h=c>>3&3;if(h>=L){B?E"%d;3%d;4%dm",F>>3&
1,s?B:F&7,s?0:B):E"0m") ;for(x=10;x--;putchar(B?s?(x+1)%5>1?95:
32:35:32));B=0;}L=h;if((t=f-2*i)>=-20){Y=t*(t+40);Y=(Y>0?0:Y/10
)+4*h+3;y<Y+5&&y>=Y?s=y==Y,F="FNLNIKNFB@I"[i],B=c&7:0;}}E"0m\n"
                                        );}usleep(
                                        50000);B=f
                                        ++<20?E"16"
                                        "A"),0:1;}}
```

Code readability: Why bother?

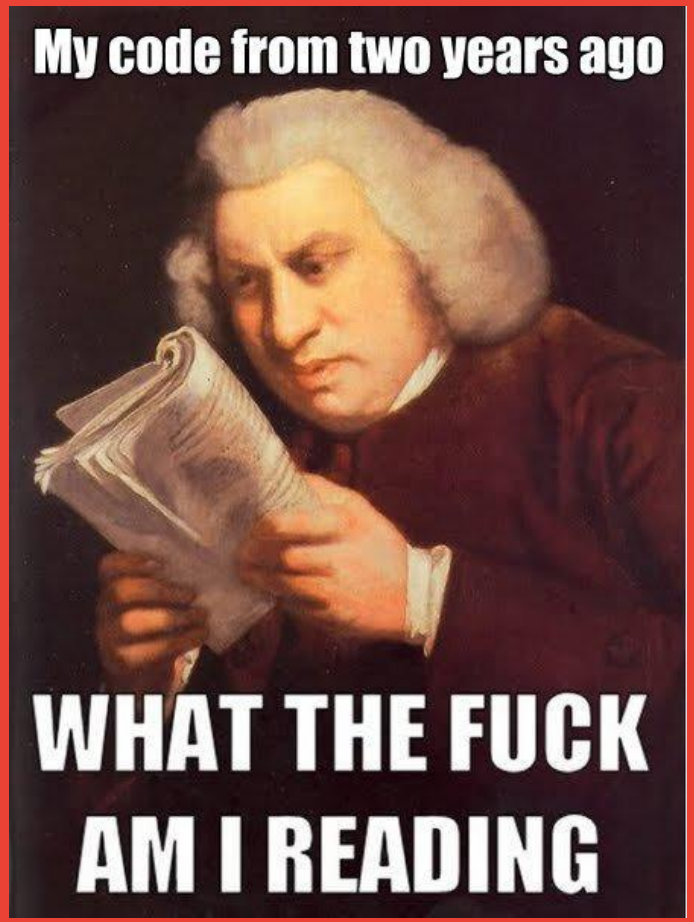
- Makes it easier for your teammates
  - and hopefully they'll reciprocate

Code readability: Why bother?

- Makes it easier for your teammates
  - and hopefully they'll reciprocate
- Makes it easier for **you** later on
  - "Woa. I wrote this code?"

Code readability: Why bother?

- Makes it easier for your teammates
  - and hopefully they'll reciprocate
- Makes it easier for **you** later on
  - "Woa. I wrote this code?"
- Helps you right now *(not so obvious)*
  - Fewer bugs, less headache developing


My code from two years ago
WHAT THE FUCK
AM I READING

Readability helps avoid this (real) situation :

This CL is actually really worrying. Do you understand how this shits all over code quality and what a nightmare it would be to deal with this?

Reviewer

Can you please explain more clearly? I would like to understand.

VP who saw the comment!

Many code quality mantras

Which should you follow?

Use as few lines as possible

Code should be modular.

...

A function should fit on a screen.

Law of Demeter (1 'dot' per expression)

The Fundamental Theorem of Code Readability

Fundamental theorem

Code should be written to minimize the **time** it would take for **someone else** to **understand** it.

Fundamental theorem

Code should be written to minimize the **time** it would take for **someone else** to **understand** it.

- **time** - concrete way to capture difficulty & size

Fundamental theorem

Code should be written to minimize the **time** it would take for **someone else** to **understand** it.

- **time** - concrete way to capture difficulty & size
- **someone else** - someone who didn't just write your code

Fundamental theorem

Code should be written to minimize the **time** it would take for **someone else** to **understand** it.

- **time** - concrete way to capture difficulty & size
- **someone else** - someone who didn't just write your code
- **understand** - able to find bugs, reuse, make changes

Code length

Code size

- Fewer lines of code is usually better,
  - because it takes less time to understand it (duh).

Consider:

```
if (bucket.contains(obj)) {
  return true;
} else {
  return false;
}
```

Versus:

```
return bucket.contains(obj);
```

Code size

- Fewer lines of code is usually better,
  - because it takes less time to understand it (duh).
- But not always!

Consider:

```
assert((!(bucket = FindBucket(key))) || !bucket->IsOccupied());
```

Code size

- Fewer lines of code is usually better,
  - because it takes less time to understand it (duh).
- But not always!

Consider:

```
assert((!(bucket = FindBucket(key))) || !bucket->IsOccupied());
```

Versus:

```
bucket = FindBucket(key);
if (bucket != NULL) {
  assert(!bucket->IsOccupied());
}
```

Function length

- Some coders say
  - "a function should be at most 1 screen of code"
- Well-intentioned advice, often right
  - But somewhat imprecise / arbitrary rule
- Difficult to obey this 100% of the time
- Lots of false-positives and false-negatives.

Function length

- Some coders say
  - "a function should be at most 1 screen of code"
- Well-intentioned advice, often right
  - But somewhat imprecise / arbitrary rule
- Difficult to obey this 100% of the time
- Lots of false-positives and false-negatives.

```
Manager::save(Context ctx, Object obj) {
  return store.save(ctx, obj);
}

Store::save(Context ctx, Object obj) {
  return database.save(ctx, obj);
}

Database::save(Context ctx, Object obj) {
  return db.write(ctx, obj.toString());
}
```

Again, just ask yourself

"Is the function easy to understand?"

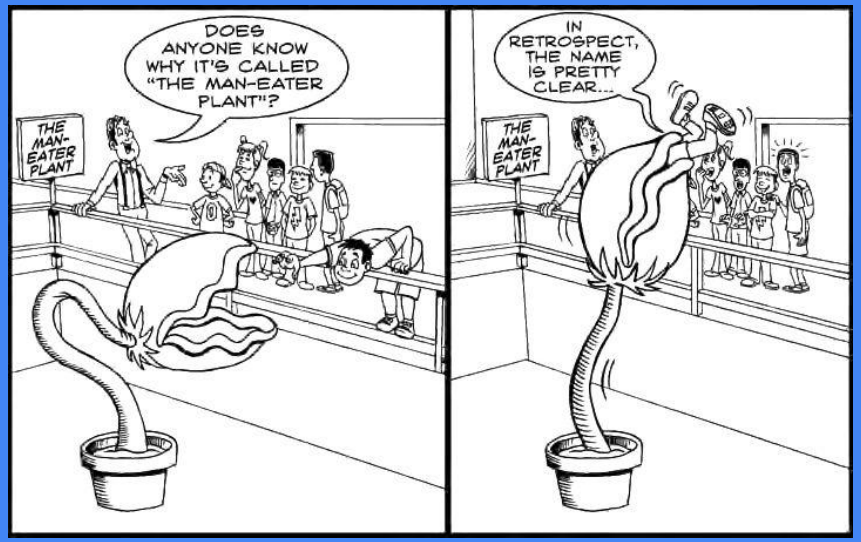
That's more important.

Naming

Naming variables

Put **units** in quantity names

```
float rate =
  1000 * size / elapsed_time;
```

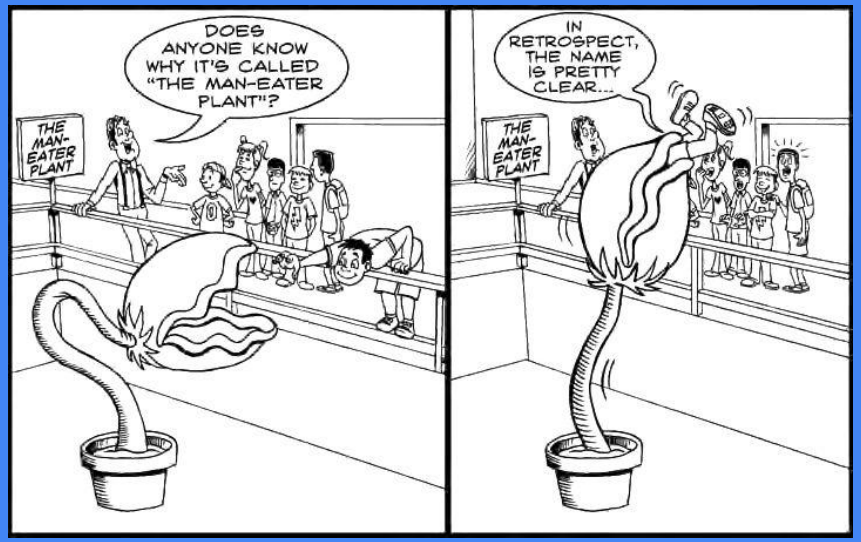


Naming variables

Put **units** in quantity names

LESS IDEAL:

```
// Bits per second

float rate =
  1000 * size / elapsed_time;
```



Naming variables

Put **units** in quantity names

MORE IDEAL:

```
float rate =
  1000 * size / elapsed_time;

float bits_per_sec =
  1000 * size_kbits / elapsed_secs;
```

Naming variables

Put **units** in quantity names

```
float rate =
  1000 * size / elapsed_time;

float bits_per_sec =
  1000 * size_kbits / elapsed_secs;
```

Other examples for function parameters:

- `Rotate(float angle)`
  - `angle -> degrees_cw`
- `StartServer(int count)`
  - `count -> num_threads`

Other "units"

- Depending on the context, there might be other important attributes of a variable.
- If there's potential confusion, add a prefix or suffix to clarify.

Example:

- Data that has been converted to UTF-8
  - `html -> html_utf8`
- A user message that will be displayed somewhere
  - `message -> unescaped_message`

Use the type-system (boost::units)

```
// quantity of length
quantity<length> L = 2.0*meters;

// quantity of energy
quantity<energy> E =
    kilograms*pow<2>(L/seconds);
```

(Java)

```
Duration offerDuration =
    Duration.ofWeeks(2);

LocalDate customerBirthday =
  loadBirthday(database, customer);
LocalDate today = LocalDate.now();
if (customerBirthday.equals(today)) {
  LocalDate offerDate  =
      today.plus(offerDuration)
      .with(next(FRIDAY));
  sendOffer(customer, offerDate);
}
```

Obscure Name

A simple variable:

```
int d; // elapsed time in days
```

A name should be:

- searchable
- distinct
- pronounceable
- meaningful

Clear Name

A better variable:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Obscure Name

A simple variable:

```
int d; // elapsed time in days
```

A name should be:

- searchable
- distinct
- pronounceable
- meaningful

(Not just for variable names!)

Clear Name

A better variable:

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Embedded Logic

Here's a complicated comparison:

```
public String compact(String message)
{
  if (expected == null
      || actual == null
      || areStringsEqual())

    return Assert.format(message,
expected, actual);

  … // code to compact

}
```

Give it a Name

Better comparison:

```
public String compact(String message)
{
  if (!canCompact()) {

    return Assert.format(...)

  }

  … // code to compact
}
```

Embedded Logic

Here's another complicated comparison:

```
// check if eligible for full benefits

if ((employee.flags & HOURLY_FLAG)
    && (employee.age > 65)) {...}
```

Here's a simple loop:

```
while (idx < CONTAINER_SIZE) {...}

for (int i = 0; i < l.size(); i++)
{...}
```

Give it a Name

Clear comparison:

```
if (employee.isEligbleForFullBenefits())
```

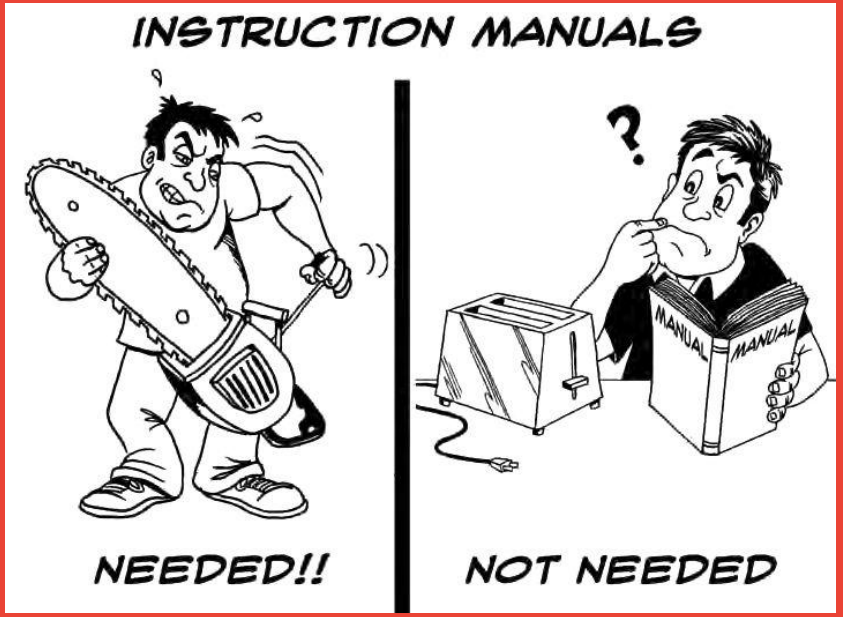
More readable?

```
while (!atEnd(idx)) {...}

// Really?

for (int i = 0; !atEnd(i); i++) {...}
```

Commenting

What to comment

- Comments take up space on the screen, take time to read, and can grow stale.
- Some people say "code should be self-documenting".
- So what should you comment?

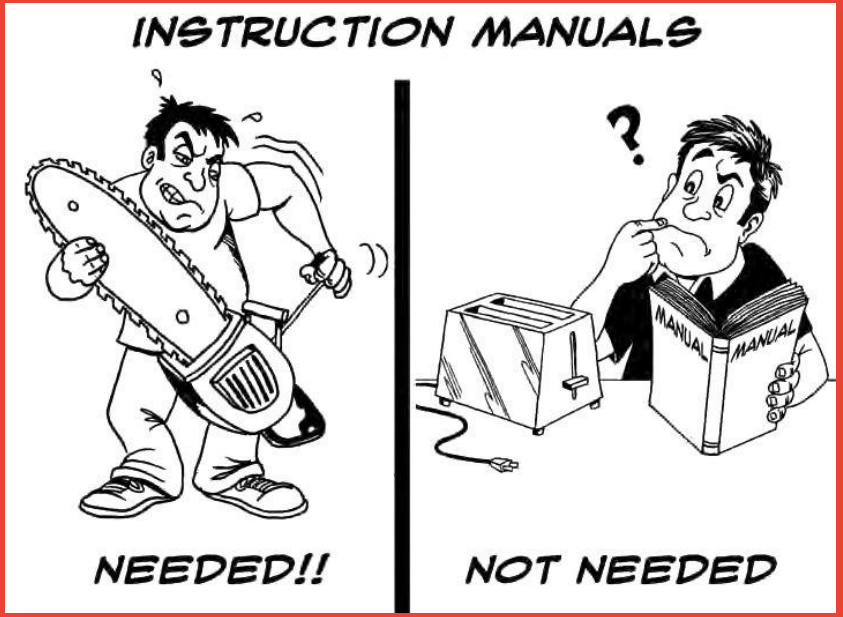


What to comment

- Comments take up space on the screen, take time to read, and can grow stale.
- Some people say "code should be self-documenting".
- So what should you comment?

Imagine the code with & without the comment.

Which would take less time to understand?



What to comment

- Comments take up space on the screen, take time to read, and can grow stale.
- Some people say "code should be self-documenting".
- So what should you comment?

Imagine the code with & without the comment.

Which would take less time to understand?

Bad Comment

```
// constructor for LocationRecorder
public LocationRecorder() { ...
```

What to comment

- Comments take up space on the screen, take time to read, and can grow stale.
- Some people say "code should be self-documenting".
- So what should you comment?

Imagine the code with & without the comment.

Which would take less time to understand?

Bad Comment

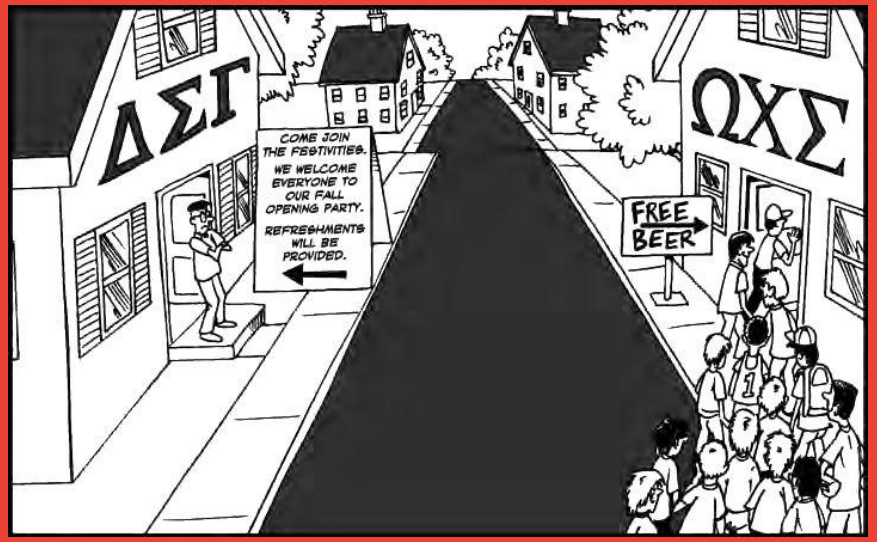
```
// constructor for LocationRecorder
public LocationRecorder() { ...
```

Good Comment

```
// fast version of
// "hash = (65599 * hash) + c"
hash = (hash << 6) + (hash << 16) -
  hash + c;
```

Be concise

Write comments with a high information/space ratio



Be concise

Consider this comment:

```
// Depending on whether we've
// already crawled this URL before,
// give it a different priority.
```

Be concise

Consider this comment:

```
// Depending on whether we've
// already crawled this URL before,
// give it a different priority.
```

Compared to this new version:

```
// Give higher priority to URLs
// we've never crawled before.
```

Be robust

Here's a simple comment:

```
// Return the number of lines in
// this file.
int CountLines(String filename)
{...}
```

Be robust

Here's a simple comment:

```
// Return the number of lines in
// this file.
int CountLines(String filename)
{...}
```

Edge cases to think about:

- "" (0 or 1 line?)
- "hello" (0 or 1 line?)
- "hello\n" (1 or 2 lines?)
- "hello\n world\r" (1 or 2 or 3 lines?)

Be robust

Here's a simple comment:

```
// Return the number of lines in
// this file.
int CountLines(String filename)
{...}
```

Edge cases to think about:

- `""` (0 or 1 line?)
- `"hello"` (0 or 1 line?)
- `"hello\n"` (1 or 2 lines?)
- `"hello\n world\r"`
  (1 or 2 or 3 lines?)

Better comment:

```
// Count how many newlines ('\n')
// in the file.
int CountLines(String filename)
{...}
```

What Not To Do

A story by Philo Juang

```
// For each feeditem we receive inside the list, try to standardize on
// external_video_id.  Extract action type if possible.
for (const auto& fi : feed_items) {
  YouTubeId external_video_id;
  // ContentId is a fucking bullshit ID format that we have to fucking
  // put up with despite it making no sense whatsoever.
```

Angry comment by pjuang@

```
// For each feeditem we receive inside the list, try to standardize on
// external_video_id.  Extract action type if possible.
for (const auto& fi : feed_items) {
  YouTubeId external_video_id;
  // ContentId is a fucking bullshit ID format that we have to fucking
  // put up with despite it making no sense whatsoever.
```

Angry comment by pjuang@

16 months later

```
// For each feeditem we receive inside the list, try to standardize on
// external_video_id.  Extract action type if possible.
for (const auto& fi : feed_items) {
  YouTubeId external_video_id;
  // ContentId is a fucking bullshit ID format that we have to fucking
  // put up with despite it making no sense whatsoever.
```

Angry comment by pjuang@

16 months later

CL #123456789 by [redacted]

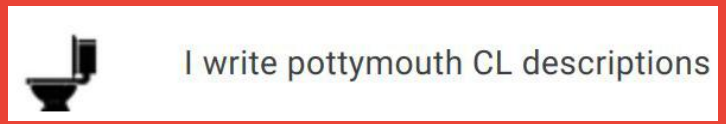
Please remove unnecessarily offensive comment.

Engineer reading my code

	author	num_cls	ratio
1	[redacted]	60	0.0151
2	[redacted]	53	0.0133
3	[redacted]	46	0.0116
3	[redacted]	46	0.0116
5	[redacted]	43	0.0108
6	[redacted]	38	0.0095
7	[redacted]	35	0.0088
7	[redacted]	35	0.0088
9	[redacted]	33	0.0083
10	[redacted]	31	[illegible]
11	[redacted]	30	
12	[redacted]	29	
13	[redacted]	28	0.0070
14	[redacted]	27	0.0068
15	[redacted]	26	0.0065
16	**pjuang**	23	0.0058
16	[redacted]	23	0.0058
18	[redacted]	22	0.0055
18	[redacted]	22	0.0055
20	[redacted]	21	0.0053
20	[redacted]	21	0.0053

There's a leaderboard for people who write foul mouthed CLs!

You get a prize on your internal profile page :



Comparison Expressions

Arranging "a < b"

- You can write comparisons in either direction
  - `if (length > 10) …`
  - `if (10 < length) …`

Arranging "a < b"

- You can write comparisons in either direction
  - `if (length > 10) …`
  - `if (10 < length) …`
- Or how about:
  - `while (bytes_received <= bytes_expected) …`
  - `while (bytes_expected >= bytes_received) …`
- Which is better? How do you know in general?

Left hand side

The expression
being interrogated
whose value is
more in flux

<
>
==

Right hand side

The expression
being compared against
whose value is
more constant

This matches English usage:

"If you are at least 18 years old."
"If 18 years is less than or equal to your age."

Arranging "a < b"

One last example...

```
def is_expired(self):
    return self.expiration_date < now()
```

Arranging "a < b"

One last example...

```
def is_expired(self):
    return self.expiration_date < now()
```

`now()` is more in flux, it's the star of the show

`expiration_date` is acting as a constant

```
def is_expired(self):
    return now() > self.expiration_date
```

Unnecessary Variables

Unnecessary variables

```
function remove_one(array, value_to_rm) {
  var index_to_rm = null;
  for (var i = 0; i < array.length; i++) {
    if (array[i] === value_to_rm) {
      index_to_rm = i;
      break;
    }
  }
  if (index_to_rm !== null) {
    array.splice(index_to_rm, 1);
  }
}
```

Unnecessary variables

```
function remove_one(array, value_to_rm) {
  var index_to_rm = null;
  for (var i = 0; i < array.length; i++) {
    if (array[i] === value_to_rm) {
      index_to_rm = i;
      break;
    }
  }
  if (index_to_rm !== null) {
    array.splice(index_to_rm, 1);
  }
}
```

```
function remove_one(array, value_to_rm) {

  for (var i = 0; i < array.length; i++) {
    if (array[i] === value_to_rm) {
      array.splice(i, 1);
      return;
    }
  }



}
```

Unnecessary variables

```
var hbts = [10000, 30000, 50000];
var hbti = 0;
function heartbeat() {
 $.get('/heartbeat/…');
 hbti++;
 if (hbti < hbts.length) {
   window.setTimeout(heartbeat,
       hbts[hbti]);
 }
}
window.setTimeout(heartbeat, hbts[0]);
```

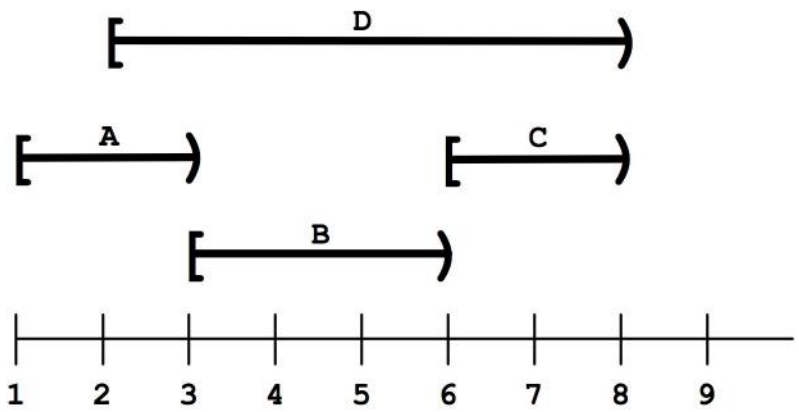
Unnecessary variables

```
var hbts = [10000, 30000, 50000];
var hbti = 0;
function heartbeat() {
 $.get('/heartbeat/…');
 hbti++;
 if (hbti < hbts.length) {
   window.setTimeout(heartbeat,
       hbts[hbti]);
 }
}
window.setTimeout(heartbeat, hbts[0]);
```

```
function heartbeat() {
 $.get('/heartbeat/…');
}

window.setTimeout(heartbeat, 10000);
window.setTimeout(heartbeat, 40000);
window.setTimeout(heartbeat, 90000);
```

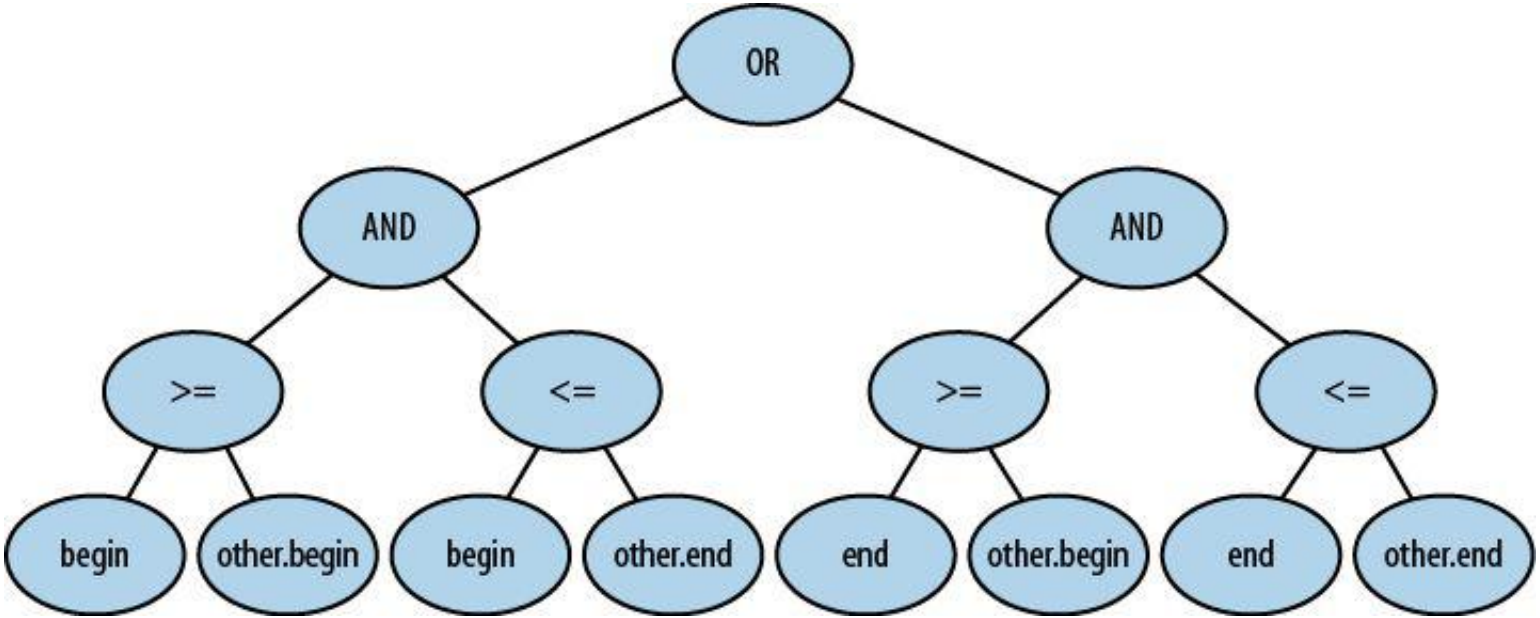
Avoiding Complexity

Range overlap: the problem




```
public class Range {
  public int begin; // Inclusive
  public int end;   // Not inclusive

  // For example, [0,5) overlaps with [3,8)
  public boolean OverlapsWith(Range other)
  {...}
}
```

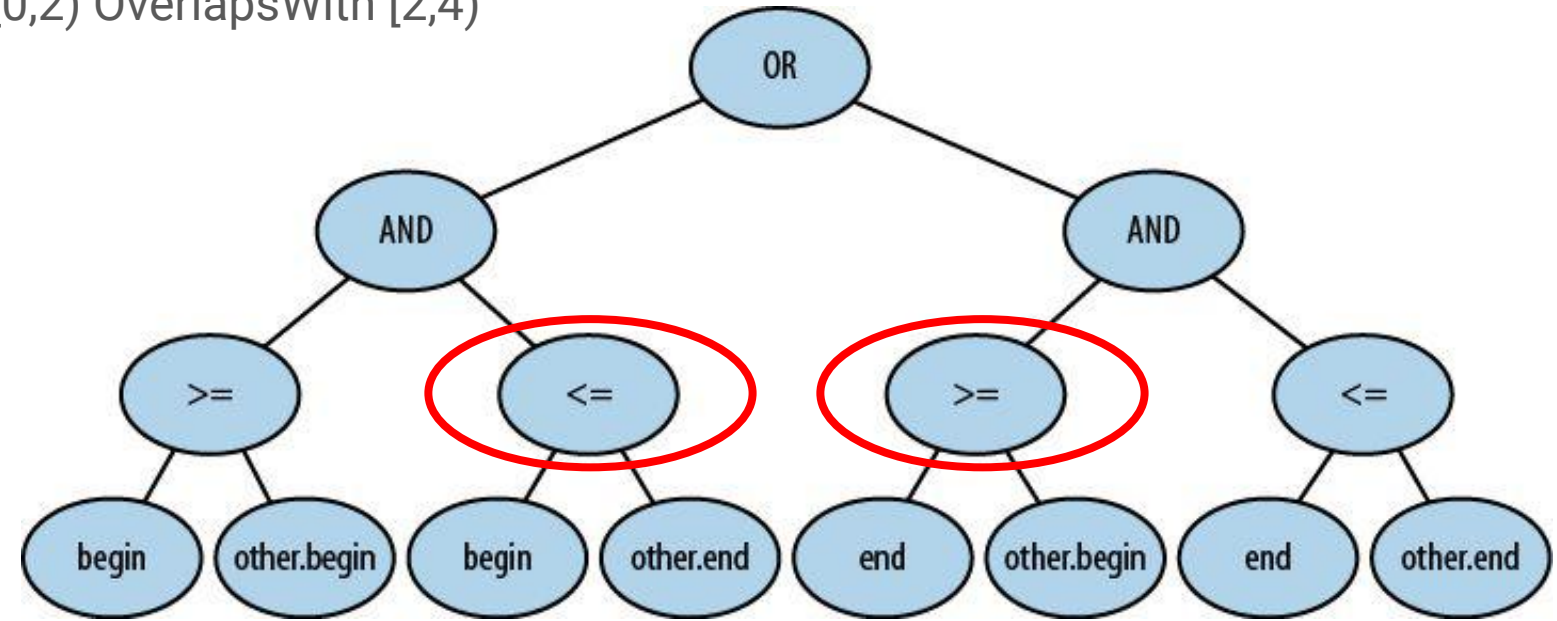
```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin <= other.end) ||
         (end >= other.begin && end <= other.end);
}
```



Range overlap: the solution?

```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin <= other.end) ||
         (end >= other.begin && end <= other.end);
}
```

Bug: [0,2) OverlapsWith [2,4)



Range overlap: the bug

Range overlap: the bug fix

```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin < other.end) ||
         (end > other.begin && end <= other.end);
}
```

Range overlap: the bug fix

```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin < other.end) ||
         (end > other.begin && end <= other.end);
}
```

But wait! What if `other` is inside this range? e.g. [0, 10) OverlapsWith [4, 6)

Range overlap: the bug fix

```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin < other.end) ||
         (end > other.begin && end <= other.end);
}
```

But wait! What if `other` is inside this range? e.g. [0, 10) OverlapsWith [4, 6)

```
public boolean OverlapsWith(Range other) {
  // Check if 'begin' or 'end' falls inside 'other'
  return (begin >= other.begin && begin < other.end) ||
         (end > other.begin && end <= other.end) ||
         (begin <= other.begin && end >= other.end);
}
```

How about now? Is it correct? Hard to tell...

Range overlap: re-approach

- Solution is getting too complex (imagine the logic tree)
- Let's try to solve it a simpler way

Range overlap: re-approach

- Solution is getting too complex (imagine the logic tree)
- Let's try to solve it a simpler way
- Common CS trick:
  solve the opposite problem
  - Check if two ranges **don't overlap**
  - The other range must be completely before/after the first.

Range overlap: re-approach

- Solution is getting too complex (imagine the logic tree)
- Let's try to solve it a simpler way
- Common CS trick: **solve the opposite problem**
  - Check if two ranges **don't overlap**
  - The other range must be completely before/after the first.

```
public boolean OverlapsWith(Range other) {
  // case 1
  if (other.end <= begin) return false;
  // case 2
  if (other.begin >= end) return false;

  // Only possibility left: overlap
  return true;
}
```

Range overlap: summary

Lessons learned:

- Avoid large logic trees
- Trick: solve the opposite problem
- Get a sense of "is this solution too complicated?"

Avoid Nesting

Nested Logic

```
public boolean ProcessDocument(Path path) {
  if (customer.canAccess(path)) {
    if (path.isValid()) {
        for (Row row : db.read(path) {
            // process rows
            // in the document
        }
    } else {
        throw new InvalidLocation(path);
    }
  } else {
    throw new AccessDenied(customer, path);
  }
}
```

Nested Logic

```
public boolean ProcessDocument(Path path) {
  if (customer.canAccess(path)) {
    if (path.isValid()) {
        for (Row row : db.read(path) {
            // process rows
            // in the document
        }
    } else {
        throw new InvalidLocation(path);
    }
  } else {
    throw new AccessDenied(customer, path);
  }
}
```

Guard with Early Return

```
public boolean ProcessDocument(Path path) {
  if (!customer.canAccess(path)) {
    throw new AccessDenied(customer, path);
  }
  if (!path.isValid()) {
    throw new InvalidLocation(path);
  }
  for (Row row : db.read(path) {
    // process rows in the document
  }

}
```

Reminder

Code should be written to minimize the ***time*** *it would take for* ***someone else*** *to* ***understand*** *it.*

https://bit.ly/3w6NALU

You've got some refactoring to do!

A tweet: Suggest an anti-readability change!

A word: Are you looking forward to changing another server?