The ratio of time spent reading vs. writing is well over 10:1. We are constantly reading old code as part of the effort to write new code.

Robert C. Martin
(Uncle Bob)

Clean Code
Consequences:
- Program comprehension is important
- Program comprehension is a major factor in software development costs

But…
- Program comprehension is hard
Why is software hard to understand?
On the Menu:

- Code comprehension
- Eye tracking
- Methodology
- Experiments
- Results
- Ideas
- Code comprehension
Code Complexity

• The property of code that makes it hard to comprehend
  – Using goto, pointers, recursion
  – Code smells
  – Just having a lot of it

• Identified and measured by **complexity metrics**
  – Static analysis
  – Overly complex code = warning that code is “bad”
    = danger of bugs
McCabe Cyclomatic Complexity (MCC)

• Given a strongly connected directed graph $G$ with $n$ nodes, $e$ edges, $p$ connected components the cyclomatic number of the graph is

$$v(G) = e - n + p$$

• Calculate this on a module’s control flow graph

• Shortcut: in planar (structured) graph count branch points + 1

• Suggestion: keep MCC $\leq 10$
BUT...

• It is reasonable to give the same weight to all branching constructs?
  – while loop
  – for loop
  – if-then-else
  – case in switch

• Is control flow the only thing that matters?

• It’s so 1970s...
We Can Do Better

Measure the effect of different code elements
• Express the same functionality in different ways
• Conduct controlled experiments
• Compare the results
  – Time to comprehend
  – Probability to make an error

Example 1: Loop Idioms

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

<table>
<thead>
<tr>
<th>version</th>
<th>init</th>
<th>cmp</th>
<th>end</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td>lp0</td>
<td>0</td>
<td>&lt;</td>
<td>len</td>
<td>++</td>
</tr>
<tr>
<td>lp1</td>
<td>0</td>
<td>&lt;=</td>
<td>len-1</td>
<td>++</td>
</tr>
<tr>
<td>lp2</td>
<td>0</td>
<td>&lt;</td>
<td>len-1</td>
<td>++</td>
</tr>
<tr>
<td>lp3</td>
<td>1</td>
<td>&lt;</td>
<td>len-1</td>
<td>++</td>
</tr>
<tr>
<td>lp4</td>
<td>1</td>
<td>&lt;</td>
<td>len</td>
<td>++</td>
</tr>
<tr>
<td>lp5</td>
<td>len-1</td>
<td>&gt;=</td>
<td>0</td>
<td>--</td>
</tr>
<tr>
<td>lp6</td>
<td>len-1</td>
<td>&gt;</td>
<td>0</td>
<td>--</td>
</tr>
</tbody>
</table>

- Loops counting down take more time
- Abnormal versions cause more errors
Example 1: Loop Idioms

Variations on for (i = 0; i < n; i++)

- Loops counting down take more time
- Abnormal versions cause more errors

Code complexity depends on expectations!
Complexity and Expectations

• “Code complexity depends on expectations”
• Is this really a form of “complexity”?
• Actually, misleading code causes more errors
Example 2: Code Regularity

Regularity = the same structures repeated time after time
Example 2: Code Regularity

Regularity = the same structures repeated time after time

Hypothesis: Figuring out the basic repeated element makes it easier to understand additional instances
Example 2: Code Regularity

Regularity = the same structures repeated time after time.

Hypothesis: Figuring out the basic element makes it easier to understand additional instances.

Code complexity depends on context!
Example 2: Code Regularity

• Same style of controlled experiment
• Additional metric: gaze pattern
  – Distribution of fixations
  – Total time of fixations
• Measure time devoted to repeated elements

void func(int **mat, int rs, int cs) {
    int i, j, k; x, y, z, d[8];
    for (i = 0; i < rs; i++) {
        for (j = 0; j < cs; j++) {
            if (j > i) mat[j][i] = mat[i][j-1];
            else mat[0][0] = 0;
            if (j > i) mat[1][i] = mat[1-1][j-1];
            else mat[1][0] = 0;
            if (j > i) mat[2][i] = mat[2-1][j-1];
            else mat[2][0] = 0;
            if (j > i) mat[3][i] = mat[3-1][j-1];
            else mat[3][0] = 0;
            if (j > i) mat[4][i] = mat[4-1][j-1];
            else mat[4][0] = 0;
            if (j > i) mat[5][i] = mat[5-1][j-1];
            else mat[5][0] = 0;
            if (j > i) mat[6][i] = mat[6-1][j-1];
            else mat[6][0] = 0;
            if (j > i) mat[7][i] = mat[7-1][j-1];
            else mat[7][0] = 0;
            mat[8][i] = mat[1][j];
            for (j = 0; j < cs; j++) {
                if (mat[j][j] > mat[i][j-1]) {
                    mat[i][j] = mat[i][j-1];
                    mat[i][j] = mat[i][j-1];
                    mat[i][j] = x;
                }
                printf("%d", mat[i][j]);
            }
            printf("\n");
        }
    }
}
Exponential model:
Time reduced by 40% in each additional instance
1. We can perform detailed measurements
   • Empirically quantify code complexity
2. Code complexity ≠ code “misleadingness”
   • Prefer time over errors as metric
3. We can use eye tracking data to parameterize models
   • e.g. “effective complexity” based on instance number
However...

• The model that comprehension depends on code complexity is too simplistic

• Comprehension more realistically depends on the combination of 3 factors:
  1. The code’s complexity
  2. The developer’s skill and domain knowledge
  3. The circumstances (fatigue, priming)
Eye tracking can help
But need careful consideration of code and task
Single Function Code

• Used in most code comprehensions studies
• Used in most eye tracking studies
• Fits on one page / screen
• Somewhat artificial situation
  – When do you consider a function in isolation?
1. Non-linear order of reading
   Unlike mostly linear order for normal text
2. Use of recurring patterns
   Linear scan of body of code (general overview)
   Jump back (to declarations?)
   Jump forward (to return value?)
   Hover (focus on a certain statement)
3. Huge differences between subjects
EMIP 2013 Data
EMIP 2013 Data
Full System

What does it mean to comprehend a system?
• More about structure than statements
  – The components comprising the system
  – Their interactions
  – Data flow
• Largely detached from the code itself
• Related to the system’s architecture
• Eye tracking not very relevant

O. Levy and D. G. Feitelson, “Understanding large-scale software -- A hierarchical view”. In 27th Intl. Conf. Program Comprehension, May 2019
In Between

• More realistic developer workflow
  – Functions in the context of classes and packages
  – Follow control flow from one function to another

• Using an IDE
  – Sequence of jumps between modules
  – What is read in each one

• Also other tools (issue tracking, Google)

• Eye tracking tools just beginning to emerge
  – iTrace by Maletic and Sharif
  – Enabler of new data collection
What Does “Reading” Mean?

How do you read
• A WhatsApp conversation
• A novel
• A newspaper story
• A poem
• A scientific paper
• Code

Do you read every word, or just skim most of it?
Do you fully understand it?
Levels of Understanding

• How do I use this function?
  – Black box / information hiding
  – Very common (libraries etc.)

• How does this function work?
  – White box
  – Superficial: given arguments, what does it print?
  – Deep: how does it compute its output?
Interpretation vs. Comprehension

• Interpretation: follow control flow and interpret commands in sequence
  – Technical process

• Comprehension: articulate the essence of what the code does
  – Deep understanding
int sum=0;
for (int i=1; i<=100; i++) {
    sum += i;
}
print( sum );

Comprehension can be easier than interpretation!
Performing a task is proof of a certain level of comprehension for different tasks.
Answer a trivia question

E.g. does the code include a variable “fooBar”

• Need to scan the code
• Eye tracking will probably reflect scanning
• Does not reflect understanding
What does this code print?

- Need to trace and interpret the code
- Eye tracking will reflect tracing of control flow
- Perhaps with shortcuts
  (e.g. don’t repeat loops)
How do I use this code?

• Treat code as black box
• Depends on API and usage examples
• Eye tracking will probably show focus on signature
I need to fix a bug in the code

• Simple case: just trace the code
e.g. to find a null pointer reference
• Hard case: need to really understand it
• Same as other hard scenarios
As in the context of a code review

• Do you need to really understand it?
• Can you just scan to search for problem indicators?
• Eye tracking may reflect linear scan
Modify, fix, or document

- Need to really understand the code (white box)
- Eye tracking will reflect the effort
- Scan the code, jump forward, jump back
- Different patterns for different people
PEOPLE
In addition to the code and the task how code is read may depend on WHO is reading it
Gender

• Do women read code differently from men?
• Few studies
• Some differences
  – Women perhaps more methodical
    e.g. start with problem statement
  – Men tend to jump ahead
Novice vs. Expert

• Do expert developers read code differently from novices?

• Some studies

• Some differences
  – Experts identify and focus on the important parts
  – Experts read code less linearly
The BIG Question for Eye Tracking:

What are they actually doing?
Humans

• Developers’ goal is to “get the job done”
• They want to do this efficiently
• So may actually try to avoid comprehension, because comprehension is hard
• Eye tracking results may reflect activity to **circumvent** comprehension rather than activity to **achieve** comprehension
EMIP Coding Scheme for Strategies

- AttentionToDetail
- DataFlow
- Debugging
- LinearHorizontal and LinearVertical
- Deductive
- DesignAtOnce
- FlowCycle
- Inductive
- InterproceduralControlFlow
- IntraproceduralControlFlow
- StrayGlance
- TestHypothesis
- Touchstone
- Trial&Error
- Wandering

“It is a capital mistake to theorize before one has data.”
—Sherlock Holmes
Collecting Lots of Data

- Needed for small effects and many confounding factors
- Identify common vs. uncommon cases
- Analyze effects and interactions
- Increase confidence via reproducibility
- EMIP dataset example
- Is this “Big Data”?
Wrapping Up

Goal: scientific basis for software engineering
Wrapping Up

Goal: scientific basis for software engineering

Subgoal: understand program comprehension
Wrapping Up

Goal: scientific basis for software engineering
Subgoal: understand program comprehension
Means: controlled experiments
Wrapping Up

Goal: scientific basis for software engineering
Subgoal: understand program comprehension
Means: controlled experiments
Prerequisites:
Define tasks, metrics, variables
Wrapping Up

Goal: scientific basis for software engineering
Subgoal: understand program comprehension
Means: controlled experiments
Prerequisites:
Define tasks, metrics, variables
Tools: eye tracking
Wrapping Up

Goal: scientific basis for software engineering
Subgoal: understand program comprehension
Means: controlled experiments
Prerequisites:
Define tasks, metrics, variables
Tools: eye tracking
Observation: we know so little
Wrapping Up

Goal: scientific basis for software engineering
Subgoal: understand program comprehension
Means: controlled experiments
Prerequisites:
Define tasks, metrics, variables
Tools: eye tracking
Observation: we know so little
consequence: there’s so much to do!