How to Write Quality Code
For Fun and Profit
Who am I?
Why should you listen to me?
What should you get out of this presentation?
Resources:

- Effective Java, 3rd edition
- Effective C++, 3rd edition
- Google Style guides
- Stack Exchange
- Other books: Clean Code, Implementation Patterns, Code Complete
- https://testing.googleblog.com/ Google Testing
- Anything that explains WHAT and WHY
How would you describe quality code?
How would you describe your code?
Is this good code?
Answer: It depends (but yes!)
$ ./mines.pl

A helluva, helluva, helluva, helluva, helluva engineer,
A helluva, helluva, helluva, helluva, helluva, helluva engineer,
Like every honest fellow, I take my whisky clear,
I'm a rambling wreck from Golden Tech, a helluva engineer.
Common properties of good code

- Compiles
- Does what it is supposed to (and not what it isn’t supposed to)
- Testable
- Extendable
- Efficient
- Easy to write
- Easy to read
- Easy to modify
- Consistent formatting
- Doesn’t reinvent the wheel
- Well documented where needed
Attributes not typically associated with quality code

- Clever
- Complex
- Compact
- Overly performant
- Vague
- Nondeterministic
“I bet I can do this with recursion and 10 lines of code”
TLDR (TLDW? TLDPA?)

- Quality code is the result of a lot of small decisions, not a few big ones
  - Many are independent, but not all
- These decisions might be clever, but they are not complex
- They are prioritized around the properties of the project
- Writing quality code is not a binary operation, it’s a constant learning process
- Quality code starts by asking the right questions
- Those questions are asked at various levels in the project, all of them can contribute to quality code
- There are hundreds of ways to improve code, far more than I can cover in one talk (even a semester would be pushing it, plus I don’t even know ½ of them.)
## How coding at school is different from coding at work

<table>
<thead>
<tr>
<th>School</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborative</td>
<td>Collaborative</td>
</tr>
<tr>
<td>You own every part of everything</td>
<td>You own lots of smaller things</td>
</tr>
<tr>
<td>Write the night before</td>
<td>Smaller changes over time</td>
</tr>
<tr>
<td>Never see again</td>
<td>Come back to 6 months later</td>
</tr>
<tr>
<td>Start from scratch</td>
<td>Start from someone’s code/design</td>
</tr>
<tr>
<td>Told exactly what to do</td>
<td>Just given a problem</td>
</tr>
<tr>
<td>Reuse is cheating</td>
<td>Reuse is encouraged</td>
</tr>
<tr>
<td>Document because you have to</td>
<td>Document because you want to</td>
</tr>
</tbody>
</table>
Alternative Viewpoint - Quality Code Minimizes Cost

- SWEs are more expensive than hardware
  - 1 SWE might be worth 5000 cores, 30 TB RAM & 20 PiB of storage
  - Optimize the algorithm first
  - Opportunity cost is a huge factor
- Bugs that hit prod are REALLY expensive
  - Case study: Knight Capital
- Debugging is expensive
- Learning a codebase is expensive
- Deprecating a codebase is expensive
- Maintaining a codebase is expensive
- Quality code minimizes the cost of the above
The questions that are asked

- Does this make it harder or easier for bugs or unintended behavior to exist?
- Does this make it harder or easier to understand what it does?
- Does this make it harder or easier to maintain or change?
- Does the performance of this matter?
- What happens if someone I’m unaware of uses this code?
Priorities according to a senior Googler

1. Correctness
2. Maintainability
3. Performance
These decisions are made at all levels in a system

- Individual lines
- Individual functions/methods
- Classes/Objects
- Modules
- Services
- Jobs
- Systems

Note: The more senior an engineer is, the more they focus on the bottom of the list.
Why is writing quality code difficult?

- New coders aren’t generally aware of what makes a successful software engineer
  - It isn’t even a question typically asked until someone encourages you to
  - It isn’t just more CS knowledge and fewer bugs with higher output
- It is not always obvious to know what code quality code looks like until you see it
- Writing code for a course != writing code for a company
- Becoming a good coder requires learning from experienced software engineers
- Being able to write good code requires knowledge and a mindset
- It takes time and practice to learn
  - There are $O(\text{dozens})$ of language specific tips
    - Not all language or library features are helpful!
  - $O(\text{dozens})$ of software design patterns
    - All are situational
Nothing I’ve pointed out so far is language specific!
Identify the strengths, weaknesses & tradeoffs

● Ask yourself “In what situations would method A be better than B or C?”
● Read Style Guides
  ○ Google has good ones!
  ○ So do others!
  ○ These contain more than “only use 80 characters and name your variables well”
● There are still “generally better” options that in everyday use cases are superior
● Read books - Effective [Java|C++]!
  ○ Ask “why?” and really try to understand the answer
● Don’t follow rules blindly, understand why they exist and in what context they are appropriate
Understand the goals and objectives of your scenario

- What is important?
- *Why* are those important (and not important!)
- Recognize class experience != real world experience when it comes to projects
- Default to putting yourself in the scenario of a “real world” class experience
- Programmer time is expensive, computers are cheap
  - Write simple code, optimize later
- Some rules are fairly universal
  - Bugs are expensive
  - Clever code and pre-optimizing cause bugs
  - Good design is easier to write
Pick the option that best suits those goals

- Do it the way that makes the most sense
- Feel free to go back and change things
- Analyze afterward, note where expectations ≠ reality
  - Be honest with yourself
  - Learn from mistakes
  - Try to leave ego out of it
Practical Tips & Examples!
Make everything immutable if it can be!

Const and Final are your friends.
Narrow the scope!

- Default to *private*
- Only pass in what is needed
- Only expose what is needed
- Have a strictly defined role and contract
- If it feels wrong analyze your design
- Avoid side-effects
  - if you use “and” you probably want two functions
Bugs are **EXPENSIVE**!

- Error check as you go
- Fail loudly and early when developing
- Tests save time
  - Especially if you are going to use this code again
- Expect bugs always, no matter how confident you are
  - You are a human, humans make mistakes
- Preventing bugs almost always pays off. In the long run it pays off 10x (at least).
- Compiler writers are smart - listen to their warnings
- Know your tools
  - Debuggers, profilers, version control, bug tracking
  - printf debugging is fine too!
Don’t reinvent the wheel!

Use what is available: ArrayList/std::vector instead of []
**General Naming Rules**

Names should be descriptive; eschew abbreviation.

Give as descriptive a name as possible, within reason. Do not worry about saving horizontal space as it is far more important to make your code immediately understandable by a new reader. Do not use abbreviations that are ambiguous or unfamiliar to readers outside your project, and do not abbreviate by deleting letters within a word.

```c
int price_count_reader;  // No abbreviation.
int num_errors;          // "num" is a widespread convention.
int num_dns_connections; // Most people know what "DNS" stands for.
```

```c
int n;                   // Meaningless.
int nerr;                // Ambiguous abbreviation.
int n_comp_conns;        // Ambiguous abbreviation.
int wgc_connections;     // Only your group knows what this stands for.
int pc_reader;           // Lots of things can be abbreviated "pc".
int cstmr_id;            // Deletes internal letters.
```
Which is better?

// Option 1
for(int i=0; i < a.size(); i++) {
    // Do stuff
}

// Option 2
for(int customerIndex = 0; customerIndex < customerList.size(); i++) {
    // Do Stuff
}
Comment When You Can’t Explain with Code

<table>
<thead>
<tr>
<th>Introduce an explaining variable.</th>
</tr>
</thead>
</table>
| ```java
// Subtract discount from price.
finalPrice = (numItems * itemPrice) - min(5, numItems) * itemPrice * 0.1;
``` |
| ```java
price = numItems * itemPrice;
discount = min(5, numItems) * itemPrice;
finalPrice = price - discount;
``` |

<table>
<thead>
<tr>
<th>Extract a method.</th>
</tr>
</thead>
</table>
| ```java
// Filter offensive words.
for (String word : words) { ... }
``` |
| ```java
filterOffensiveWords(words);
``` |

<table>
<thead>
<tr>
<th>Use a more descriptive identifier name.</th>
</tr>
</thead>
</table>
| ```java
int width = ...; // Width in pixels.
``` |
| ```java
int widthInPixels = ...;
``` |

<table>
<thead>
<tr>
<th>Add a check in case your code has assumptions.</th>
</tr>
</thead>
</table>
| ```java
// Safe since height is always > 0.
return width / height;
``` |
| ```java
checkArgument(height > 0);
``` |
### Useful Comments

Reveal your intent: explain *why* the code does something (as opposed to *what* it does).

```c
// Compute once because it’s expensive.
```

Protect a well-meaning future editor from mistakenly “fixing” your code.

```c
// Create a new Foo instance because Foo is not thread-safe.
```

Clarification: a question that came up during code review or that readers of the code might have.

```c
// Note that order matters because...
```

Explain your rationale for what looks like a bad software engineering practice.

```c
@SuppressWarnings("unchecked") // The cast is safe because...
```
Builder Pattern

```java
public class User {
    private final String firstName;  //required
    private final String lastName;  //required
    private final int age;          //optional
    private final String phone;     //optional
    private final String address;   //optional

    // Constructor
    public User(String firstName, String lastName, String phone, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.phone = phone;
        this.address = address;
    }

    public User(String firstName, String lastName) {
        this(firstName, lastName,"");
    }

    public User(String firstName, String lastName, int age) {
        this(firstName, lastName, age,"");
    }
}
```
public static class UserBuilder {
    private final String firstName;
    private final String lastName;
    private int age;
    private String phone;
    private String address;

    public UserBuilder(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public UserBuilder age(int age) {
        this.age = age;
        return this;
    }

    public UserBuilder phone(String phone) {
        this.phone = phone;
        return this;
    }

    public UserBuilder address(String address) {
        this.address = address;
        return this;
    }

    public User build() {
        return new User(this);
    }
}
public User getUser() {
    return new User.UserBuilder("John", "Doe")
        .age(30)
        .phone("1234567")
        .address("Fake Address 1234")
        .build();
}
Is this quality?

```c
Resource getUserResource(string resourcePath, bool *isAuthorized) {
    if(isAuthorized) {
        // ...
    }
}
```
Now what does it do?

```c
Resource getUserResource(string resourcePath, bool *isAuthorized) {
    if(isAuthorized == NULL) {
        // ...
    }
}
```
Effective Java Item 28: Prefer List to Arrays

```java
// Fails at runtime!
Object[] objectArray = new Long[1];
objectArray[0] = "I don't fit in"; // Throws ArrayStoreException

// Won't Compile
List<Object> ol = new ArrayList<Long>(); // Incompatible types
ol.add("I don't fit in");
```
Effective Java Item 50: Make Defensive Copies

```java
public final class DatePeriod {
    private final Date start;
    private final Date end;
    public Period(Date start, Date, end) {
        this.start = start;
        this.end = end;
    }

    public Date start() {
        return start;
    }
}

Date start = new Date();
Date end = new Date();
Period p = new Period(start, end);
end.setYear(78); // Modifies internals of p!
```
public Period(Date start, Date end) {
    this.start = new Date(start.getTime());
    this.end = new Date(end.getTime());
}

p1.getStart().setYear(78); // Still broken!

// Fix it
public Date getStart() {
    return new Date(start.getTime());
}
Tools like `@AutoValue`

```java
import com.google.auto.value.AutoValue;

@AutoValue
abstract class Animal {
    abstract String name();
    abstract int numberOfLegs();

    static Builder builder() {
        return new AutoValue_Animal.Builder();
    }

    @AutoValue.Builder
    abstract static class Builder {
        abstract Builder setName(String value);
        abstract Builder setNumberOfLegs(int value);
        abstract Animal build();
    }
}
```
With a test!

```java
public void testAnimal() {
    Animal dog = Animal.builder().setName("dog").setNumberOfLegs(4).build();
    assertEquals("dog", dog.name());
    assertEquals(4, dog.numberofLegs());

    // You probably don't need to write assertions like these; just illustrating.
    assertTrue(
        Animal.builder().setName("dog").setNumberOfLegs(4).build().equals(dog));
    assertFalse(
        Animal.builder().setName("cat").setNumberOfLegs(4).build().equals(dog));
    assertFalse(
        Animal.builder().setName("dog").setNumberOfLegs(2).build().equals(dog));
    assertEquals("Animal{name=dog, numberOfLegs=4}", dog.toString());
}
```
Effective Java Item 55: Return Optionals Judiciously

```
Optional<String> getFriend() {
    // Complex logic
    return Optional.of(name);
    // else
    return Optional.empty();
}

Optional<String> friend = getFriend()
if(friend.isPresent()) {
    billy.addFriend(optional.get());
}
```
# Reduce Nesting

<table>
<thead>
<tr>
<th>Code with too much nesting</th>
<th>Code with less nesting</th>
</tr>
</thead>
<tbody>
<tr>
<td>response = server.Call(request)</td>
<td>response = server.Call(request)</td>
</tr>
<tr>
<td>if response.GetStatus() == RPC.OK:</td>
<td>if response.GetStatus() != RPC.OK:</td>
</tr>
<tr>
<td>if response.GetAuthorizedUser():</td>
<td>raise RpcError(response.GetStatus())</td>
</tr>
<tr>
<td>if response.GetEnc() == 'utf-8':</td>
<td>if not response.GetAuthorizedUser():</td>
</tr>
<tr>
<td>if response.GetRows():</td>
<td>raise ValueError('wrong encoding')</td>
</tr>
<tr>
<td>vals = [ParseRow(r) for r in response.GetRows()]</td>
<td>if response.GetEnc() != 'utf-8':</td>
</tr>
<tr>
<td>avg = sum(vals) / len(vals)</td>
<td>raise AuthError('unauthorized')</td>
</tr>
<tr>
<td>return avg, vals</td>
<td>vals = [ParseRow(r) for r in response.GetRows()]</td>
</tr>
<tr>
<td>else:</td>
<td>avg = sum(vals) / len(vals)</td>
</tr>
<tr>
<td>raise AuthError('unauthorized')</td>
<td>return avg, vals</td>
</tr>
<tr>
<td>else:</td>
<td>if not response.GetRows():</td>
</tr>
<tr>
<td>raise ValueError('wrong encoding')</td>
<td>raise AuthError('unauthorized')</td>
</tr>
<tr>
<td>else:</td>
<td>vals = [ParseRow(r) for r in response.GetRows()]</td>
</tr>
<tr>
<td>raise ValueError('wrong encoding')</td>
<td>avg = sum(vals) / len(vals)</td>
</tr>
<tr>
<td>else:</td>
<td>return avg, vals</td>
</tr>
<tr>
<td>else:</td>
<td>raise EmptyError()</td>
</tr>
<tr>
<td>raise AuthError('unauthorized')</td>
<td>raise EmptyError()</td>
</tr>
<tr>
<td>else:</td>
<td>raise ValueError('wrong encoding')</td>
</tr>
<tr>
<td>raise ValueError('wrong encoding')</td>
<td>raise AuthError('unauthorized')</td>
</tr>
<tr>
<td>else:</td>
<td>raise EmptyError()</td>
</tr>
</tbody>
</table>
Code Review
Example: CostManager.java

public class CostManager {
    public static final int GroundPriceLevel10 = 30;
    public static final int GroundPriceLevel11 = 50;
    public static final int GroundPriceLevel12 = 120;
    public static final int GroundPriceLevel13 = 300;
    public static final int AirPriceLevel10 = 25;
    public static final int AirPriceLevel11 = 60;
    public static final int AirPriceLevel12 = 150;
    public static final int AirPriceLevel13 = 400;
    public static final int FreezePriceLevel10 = 45;
    public static final int FreezePriceLevel11 = 75;
    public static final int FreezePriceLevel12 = 160;
    public static final int FreezePriceLevel13 = 450;
}

public enum CostManager {
    LEVEL0(30, 25, 45),
    LEVEL1(50, 60, 75),
    LEVEL2(120, 150, 160),
    LEVEL3(300, 400, 450);

    private final int groundPrice;
    private final int airPrice;
    private final int freezePrice;

    CostManager(int groundPrice, int airPrice, int freezePrice) {
        this.groundPrice = groundPrice;
        this.airPrice = airPrice;
        this.freezePrice = freezePrice;
    }

    public int getGroundPrice() { return groundPrice; }
    public int getAirPrice() { return airPrice; }
    public int getFreezePrice() { return freezePrice; }
}
Read a Style Guide for your preferred language(s)

- Style guides aren’t just about formatting
- Good ones include defensive coding
- Stress language features that create quality code, discourage usage of ones that don’t
- Google has many, other companies do to.
- Use a linter and auto-formatter!
Random Tips

- Learn your tools
- Read other people’s code & have them read yours
- Read a style guide
  - Pick parts to follow!
- Read Books
- Ask “why?”
- Make a conscious effort to improve
  - Go back and read your old code
Questions?