CLEAN CODE FUNDAMENTALS

TOPICS

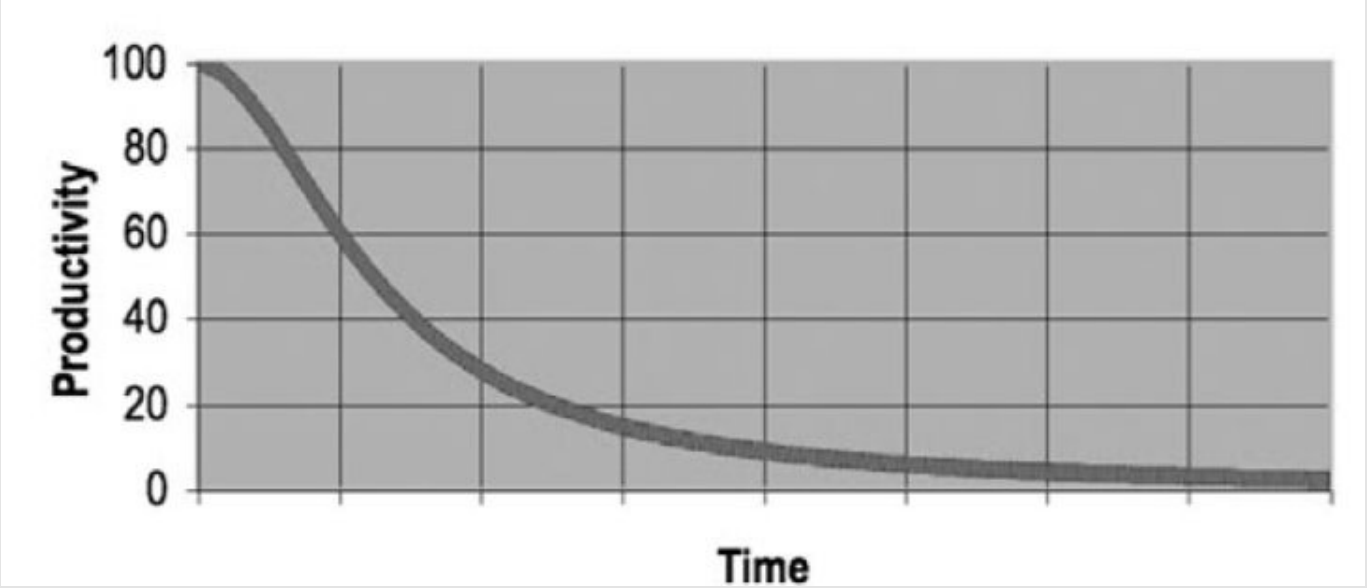
- Why Code Quality Matters
- Good Code Vs Bad Code
- Some Fundamental Principles
- Formatting Matters
- What's in a Name

TOPICS (CONTINUED)

- Functions
- Classes
- Don't Repeat Yourself… Ever
- Comments
- Error Handling
- References and Further Reading

WHY DOES CODE QUALITY MATTER?

- Bad Code Slows Programmers Down



- Productivity decreases asymptotically to zero
- "Bad Code" has brought entire companies down
- "Bad Code" makes programmers lives miserable

SO WHY IS BAD CODE WRITTEN?

- The overall entropy of the code will increase or decrease with momentum
- Good programmers can write bad code
- Programmers are rushing
- Programmers misunderstanding the design of the code they are working on
- Trying to do too much at once
- Overcomplicating requirements for fantasy scenarios

GOOD CODE VS BAD CODE

- What are the measurements of code quality?
  - Suggestions?

GOOD CODE VS BAD CODE

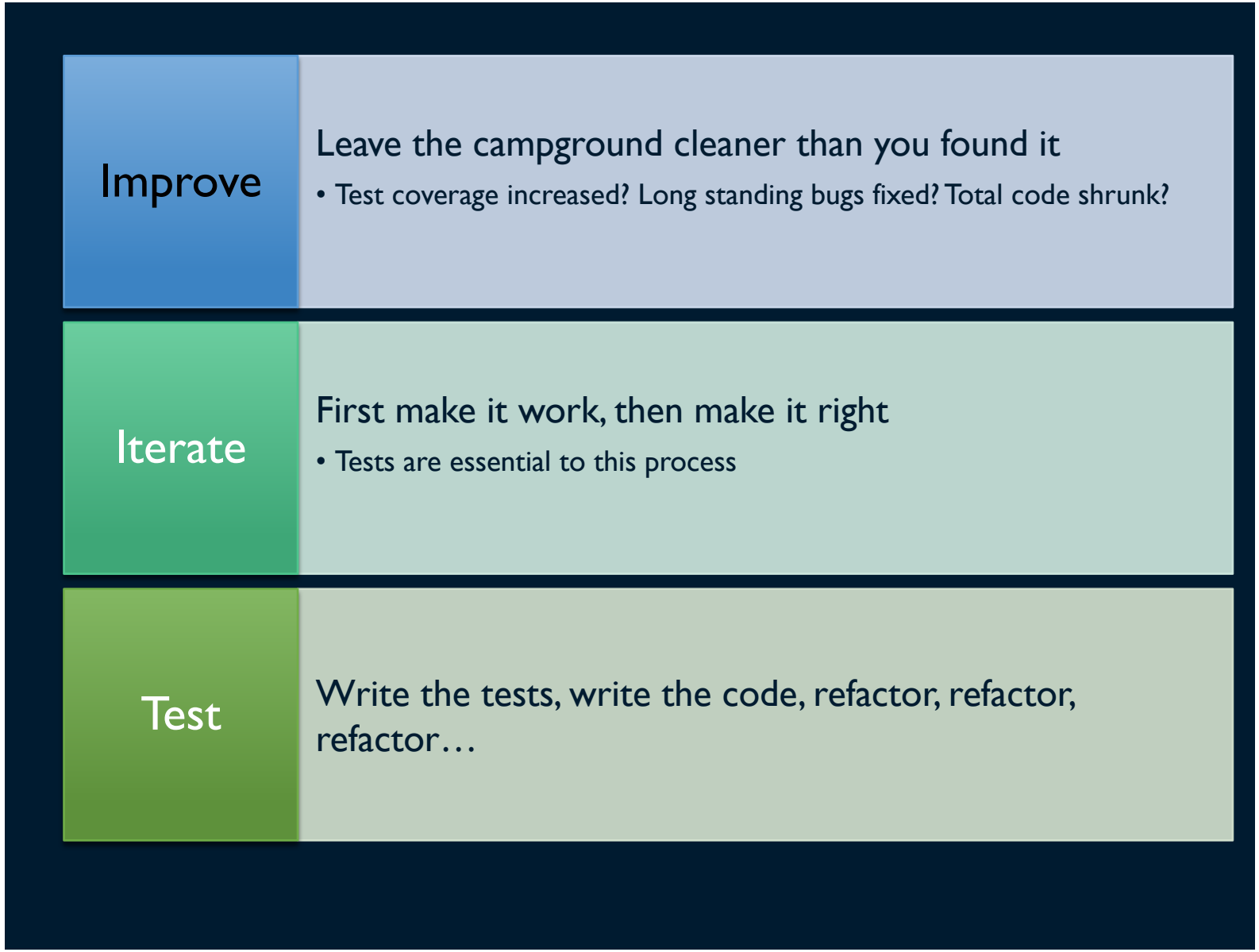
Can be difficult to define, some commonly used phrases

- Readable – code is meant to be read! (Harold Abelson)
- Elegant, Efficient, Simple, Minimal
- Logic should be straightforward, easy to understand
- Don't "tempt" bad practices
- Must have tests
- The designer's intent is clear
- Consistency in all things – be predictable

KENT BECK'S 4 RULES OF SOFTWARE DESIGN

1. Passes all the tests (meets requirements)
2. Reveals intention (should be easy to understand)
3. No duplication (DRY)
4. Fewest elements (Remove anything that doesn't serve the three previous rules)

COMMONLY SHARED WISDOM



FORMATTING MATTERS

Some studies say that indentation is one of the most statistically significant indicators of bug density.

Show whitespace in your editor

Linting or Checkstyle plugin. Install it and learn how to use it.

Tabs Vs spaces….

Vertical formatting – a logical order to things

Blank lines – not too many, not too few

Horizontal formatting – line length (80 chars is common)

The art of indenting broken lines

NAMING CONVENTIONS

- Be consistent – **pay attention** to the naming convention in the code you are working on and stick with it.
- Using good names is important, continuously refactor
  - Modules, packages, classes, functions, variables….
- Names should reveal intention
  - int d / int elapsedTime / int elapsedTimeInDays
  - theList, currVal, thisVal
- Don't tempt fate with subtle differences
  - employeeIdNumber Vs. employeeIpNumber
- Use pronounceable names
  - genymdhms Vs. generatedTimestamp

FUNCTIONS (1)

- Functions should be small
  - Less than 10 lines is commonly preferable
  - If you're indenting a lot then your function is probably too big
- Functions should do one thing (SRP)
- Consider your arguments carefully
  - Long argument lists make code hard to understand
  - Writing tests will make you appreciate short argument lists
  - Avoid using 'flag' arguments to change function behaviour

FUNCTIONS (2)

- Functions should be easy to test, this is another by-product of "doing one thing".
- Functions shouldn't have side effects, or unexpected behaviour
- In general exceptions are preferable to returning error codes
- Error handling is one thing
  - catch blocks should often contain just a single function call

CLASSES

- Classes should be small in terms of responsibilities
  - Watch out for classes with a lot of seemingly unrelated functions
- Classes should do be responsible for one thing (SRP)
- Classes should be “cohesive”
  - Does each method access one or more class variables
- Classes should not depend on the implementation details of objects they use

- Constantly refactor to avoid repetition
- If a function is getting too long (doing more than one thing) break it up
- If a class has too many responsibilities, then break it up
- If classes are doing similar things, then consider a common base class
- Write libraries to avoid having the same code in multiple modules
  - Maven, Gradle, Pip are all vital as they help with sharing libraries and therefore prevent repeating code

DON'T REPEAT YOURSELF... EVER!

A COMMENT ON COMMENTS

- Good comments can be extremely helpful
- Old out of date comments can be disastrous for understanding
- Be wary of adding long comments that will not be maintained
- Can you explain what's happening without comments through better naming or clearer logic?
- However, explaining why you're doing something unexpected can be useful (...but **why** are you doing something unexpected!!)

MORE COMMENTS

Sometimes TODO comments make sense, but ask yourself why are you not doing it now!

Keeping commented out code is not good. Your source control system still has the old version. If code isn't being used then delete it.

ERROR HANDLING 1

- Consider exception handling from the caller's perspective
  - Pass a sensible message in your exceptions that will make sense in a "catch" block
  - Think carefully about when a single exception class will suffice instead of multiple different exceptions. Save the caller from becoming cluttered with "catch" blocks.
- Avoid returning null or None, and don't pass in null as an argument
  - Code becomes cluttered with checks for null
  - End up with null pointer exceptions

ERROR HANDLING 2

- Consider exception handling from the caller's perspective
  - Pass a sensible message in your exceptions that will make sense in a "catch" block
  - Think carefully about when a single exception class will suffice instead of multiple different exceptions. Save the caller from becoming cluttered with "catch" blocks.
- Avoid returning null or None, and don't pass in null as an argument
  - Code becomes cluttered with checks for null
  - End up with null pointer exceptions

SOLID PRINCIPLES – A BASIS FOR GOOD OO DESIGN

- "Uncle Bob" summarizes many of the clean code guidelines in 5 principles, remembered with the acronym "SOLID"

- The SOLID principles are not rules. They are not laws. They are not perfect truths.

- *"An apple a day keeps the doctor away."* This is a good principle, it is good advice, but it's not a pure truth, nor is it a rule.

SOLID PRINCIPLES

- S: Single Responsibility Principle (SRP) - Do one thing well!

- O: Open Closed Principle (OCP). Open for extension, Closed for modification.
  - A class should be extendable without modifying the class itself.

- L: Liskov substitution principle: "Every subclass or derived class should be substitutable for their base or parent class."

SOLID PRINCIPLES (2)

- I: The Interface segregation principle: "Many client-specific interfaces are better than one general-purpose interface."
  - A client should never be forced to implement an interface that it doesn't use, or clients shouldn't be forced to depend on methods they do not use

- D: The Dependency inversion principle: "Depend upon abstractions, [not] concretions."
  - This principle allows for decoupling.

SOLID PRINCIPLES

- Detailed Examples:
  - https://www.baeldung.com/solid-principles
  - https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

SMELLS

- Detailed Examples:
  - https://refactoring.guru

REFERENCES AND FURTHER READING

- Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
- Martin, Robert C. *The clean coder: a code of conduct for professional programmers*. Pearson Education, 2011.
- Hunt, Andrew, and David Thomas. "The pragmatic programmer: from journeyman to master." Addison Wesley 1999.
- Beck, Kent, et al. "Manifesto for agile software development." (2001): 2006.
- Gamma, Erich. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

JAVA CODE REVIEW EXERCISE

- As a group, take a look at the Java code examples here:
  - https://github.com/framaz/The_Game_No_Name/tree/master/app/src/main/java/com/example/framaz1/myapplication
- You may alternatively choose another source of either good or bad code.
- Pick a particular file to focus on and review it critically
- Try to identify specific aspects that are good and bad
- In the case of bad code, what further issues might it cause later
- We'll get specific groups to explain their findings when we come back together

THANK YOU