Clean Code

Écrire du code propre & maintenable

Outline

- Outcomes
- What does it mean to be clean?
- What is Clean Code?
- Bad Code Dumpster Dive
- Names
- Functions
- Demonstration: Cleaning Code
- Comments
- Formatting
- Practice
- Exercises
- Some tools

Outcomes

After this class students will be able to:

1. recognize the elements of bad code, including within functions, classes,arguments, and comments.
2. review code with an eye towards cleanliness and design.
3. plan and execute code cleanup without risk to the application.
4. incrementally improve bad design choices.
5. be much more aware of the quality of the code they, and their team-mates are writing.

What does it mean to be clean?

- The business cost of bad code.
  - The Productivity Roller-Coaster.
  - Race to the Bottom.
  - The Grand-Redesign Myth.
- Why Does Code Rot?
- The Only Way to go Fast

What is Clean Code?

Elegance

I like my code to be elegant and efficient

Clean code does one thing well

Bjarne Stroustrup

Simple, Direct & Prose

Clean code is simple and direct

Clean code reads like well-written prose

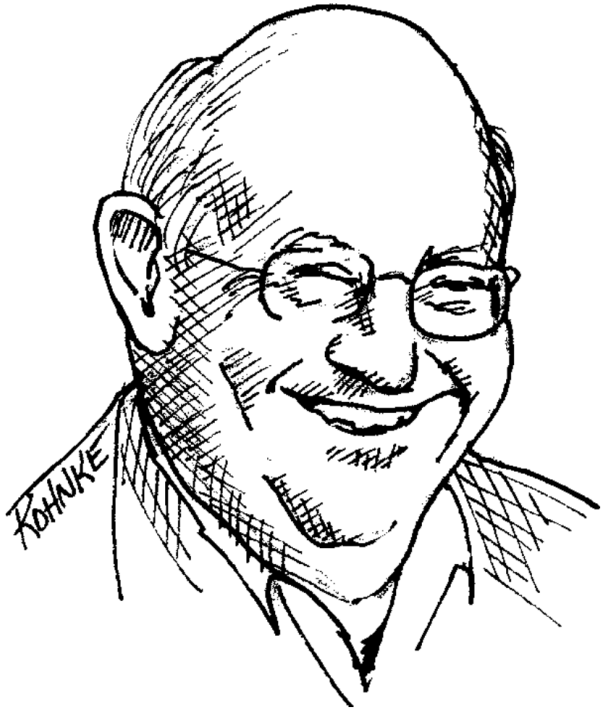
Grady Booch

Literate

Clean code can be read

Clean code should be literate

Dave Thomas



Care

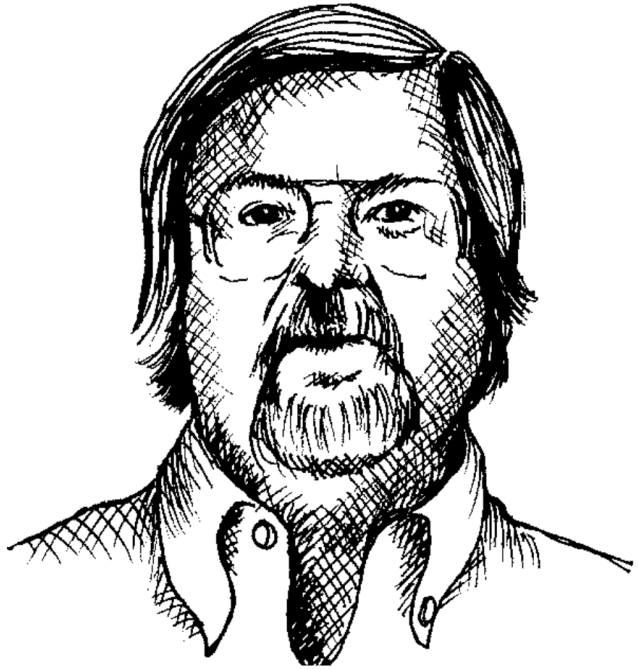
Clean code always looks like it was written by someone who cares

Michael Feathers

Small, expressive, simple

Reduced duplication, high expressiveness, and early building of, simple abstractions

Ron Jeffries



What you expected

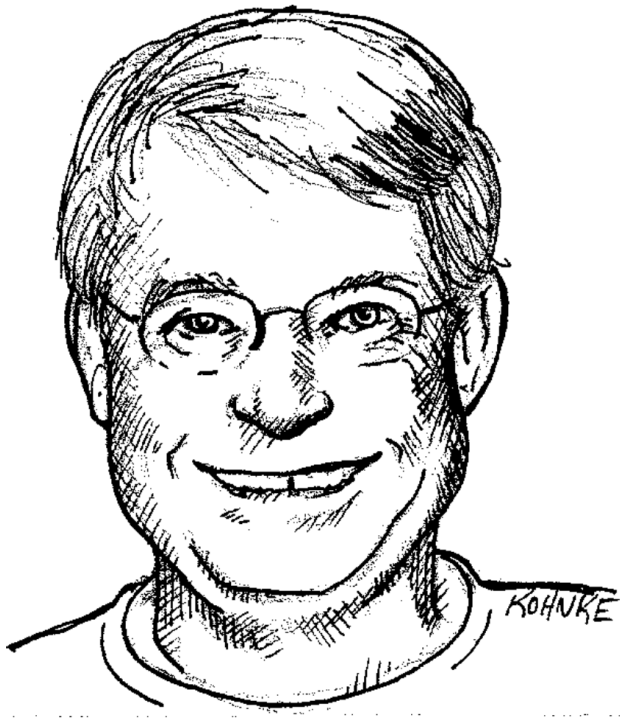
You know you are working on clean code
when each routine you read turns out to
be pretty much what you expected

Ward Cunningham

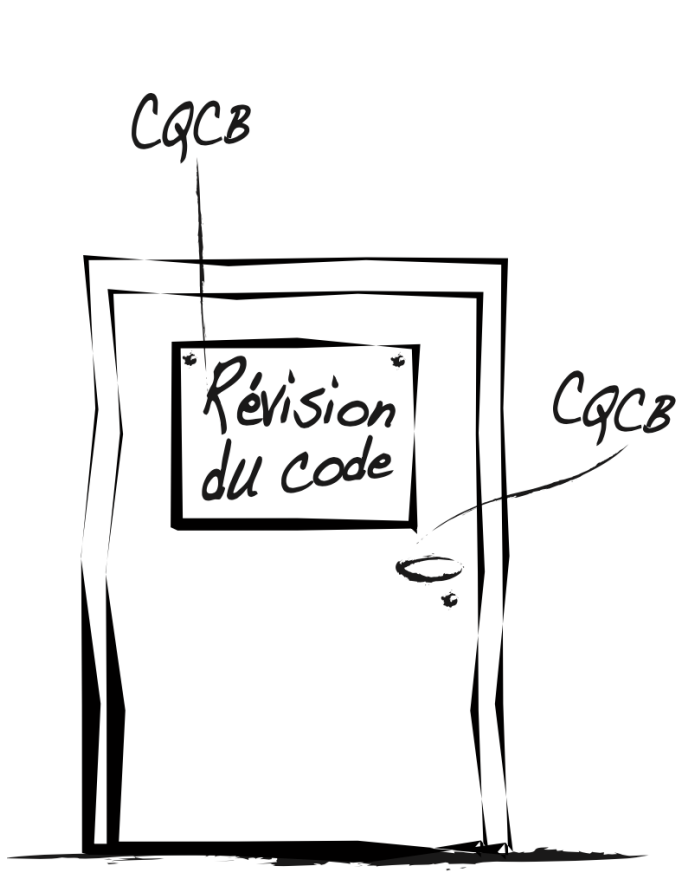
They Boy Scout Rule

Check the code in cleaner than you checked it out.

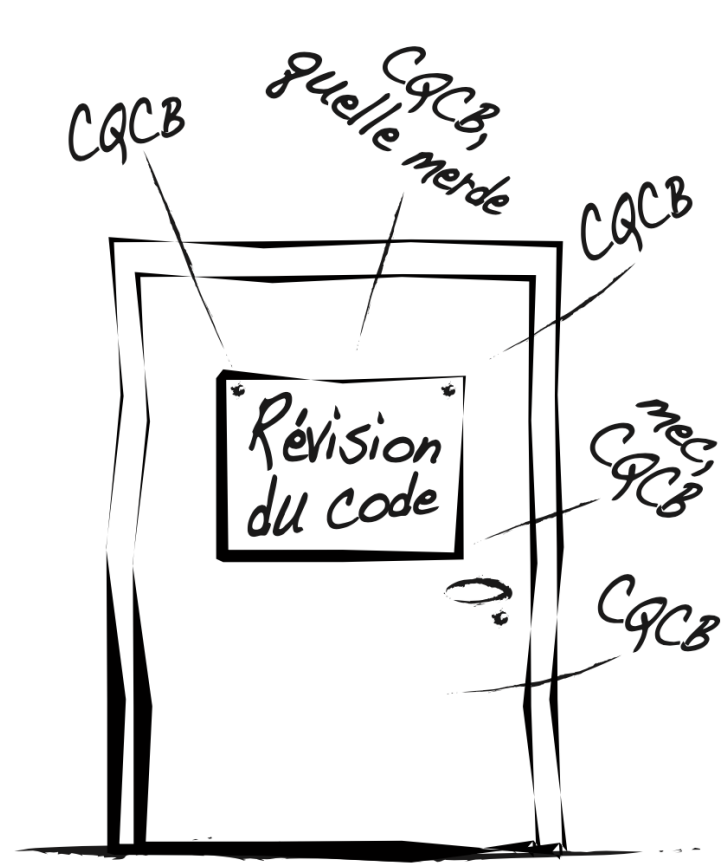
Robert C. Martin "Uncle Bob"



La SEULE mesure valide de la QUALITÉ du code :
nombre de CQCB par minute



Bon code



Mauvais code

CQCB : C'est quoi ce bordel

Bad Code Dumpster Dive

```
function testableHtml(pageData, includeSuiteSetup) {
  const wikiPage = pageData.getWikiPage();
  let buffer = new StringBuffer();
  if (pageData.hasAttribute("Test")) {
    if (includeSuiteSetup) {
      const suiteSetup = PageCrawlerImpl.getInheritedPage(
        SuiteResponder.SUITE_SETUP_NAME,
        wikiPage
      );
      if (suiteSetup != null) {
        const pagePath = suiteSetup.getPageCrawler().getFullPath(suiteSetup);
        const pagePathName = PathParser.render(pagePath);
        buffer.append("!include -setup .").append(pagePathName).append("\n");
      }
    }
    const setup = PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
    if (setup != null) {
      const setupPath = wikiPage.getPageCrawler().getFullPath(setup);
      const setupPathName = PathParser.render(setupPath);
      buffer.append("!include -setup .").append(setupPathName).append("\n");
    }
  }
  buffer.append(pageData.getContent());
  if (pageData.hasAttribute("Test")) {
    const teardown = PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
    if (teardown != null) {
      const tearDownPath = wikiPage.getPageCrawler().getFullPath(teardown);
      const tearDownPathName = PathParser.render(tearDownPath);
      buffer
        .append("\n")
        .append("!include -teardown .")
        .append(tearDownPathName)
        .append("\n");
    }
  }
}
```

Bad Code Dumpster Dive

```
function renderPageWithSetupsAndTeardowns(pageData, isSuite) {
  const isTestPage = pageData.hasAttribute("Test");
  if (isTestPage) {
    const testPage = pageData.getWikiPage();
    let newPageContent = new StringBuffer();
    includeSetupPages(testPage, newPageContent, isSuite);
    newPageContent.append(pageData.getContent());
    includeTeardownPages(testPage, newPageContent, isSuite);
    pageData.setContent(newPageContent.toString());
  }
  return pageData.getHtml();
}
```

Meaningful Names

Meaningful Names

Use Intention-Revealing Names

```
function getThem(theList) {
  const list1 = [];
  for (let x of theList) {
    if (x[0] == 4) {
      list1.push(x)
    }
  };
  return list1;
}
```

Meaningful Names

Use Intention-Revealing Names

```
function getFlaggedCells(gameBoard) {
  const flaggedCells = [];
  for (let cell of gameBoard) {
    if (cell.isFlagged()) {
      flaggedCells.push(cell);
    }
  }
  return flaggedCells;
}
```

Meaningful Names

Avoid Disinformation

```
let a = l;
if (O == l)
  a = O1;
else
  l = 01;
```

Make Meaningful Distinctions

```
function copyChars(a1, a2) {
  for (let i = 0; i < a1.length; i++) {
    a2[i] = a1[i];
  }
 }
```

Meaningful Names

Use Pronounceable Names

```
class DtaRcrd102 {
  constructor(genymdhms, modymdhms) {
    this.genymdhms = genymdhms;
    this.modymdhms= modymdhms;
  }
  pszqint = "102"; /* ... */
};
```

Meaningful Names

Use Pronounceable Names

```
class Customer {
  constructor(generationTimestamp, modificationTimestamp) {
    this.generationTimestamp = generationTimestamp;
    this.modificationTimestamp = modificationTimestamp;
  }
  recordId = "102"; /* ... */
};
```

Meaningful Names

Use Searchable Names

```
    for (let j = 0; j < 34; j++) {
      s += (t[j] * 4) / 5;
    }
```

Meaningful Names

Use Searchable Names

```
    const realDaysPerIdealDay = 4;
    const WORK_DAYS_PER_WEEK = 5;
    let sum = 0;
    for (let j = 0; j < NUMBER_OF_TASKS; j++) {
      const realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
      const realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
      sum += realTaskWeeks;
    }
```

Meaningful Names

Member Prefixes (Avoid encodings)

```
class Part {
  ...
  m_dsc; // The textual description
  ...
  setName(name) {
    m_dsc = name;
  }
}
```

Hungarian Notation (Avoid encodings)

```
PhoneNumber phoneString;
// name not changed when type changed!
```

Meaningful Names

Member Prefixes (Avoid encodings)

```
    class Part {
      ...
      description;
      ...
      setDescription(description) {
        this.description = description;
      }
    }
```

Hungarian Notation(Avoid encodings)

```
PhoneNumber phone;
```

Meaningful Names

Avoid Mental Mapping

```
    for (let a = 0; a < 10; a++)
      for (let b = 0; b < 10; b++)
```

Class Names

```
Manager, Processor, Data, Info;
```

Meaningful Names

Avoid Mental Mapping

```
  for (let i = 0; i < 10; i++)
    for (let j = 0; j < 10; j++)
```

Class Names

```
Customer, WikiPage, Account, AddressParser;
// a class name should not be a verb
```

Meaningful Names

Method Names

```
    postPayment, deletePage, save
    // methods should have verb or verb phrase names

    const name = employee.getName();
    customer.setName("mike");

    if (paycheck.isPosted())...

    const fulcrumPoint = Complex.fromRealNumber(23.0);
    // is generally better than
    const fulcrumPoint = new Complex(23.0);
```

Meaningful Names

Pick One Word per Concept

```
fetch, retrieve, get; // as equivalent methods

controller, manager, driver; // confusing
```

Don’t Pun

```
// avoid using the same word for two purposes
```

Meaningful Names

Use Solution Domain Names

```
AccountVisitor, JobQueue;
// people who read your code will be programmers
```

Add Meaningful Context

```
    firstName, lastName, street, city, state, zipcode

    // a better solution
    addrFirstName, addrLastName, addrState

    // a better solution
    Class Address
```

Meaningful Names

Don’t Add Gratuitous Context

```
Address
// is a fine name for a class
AccountAddress, CustomerAddress
// are fine names for instances of the class Address
// but could be poor names for classes

MAC addresses, port addresses, Web addresses

// a better solution
PostalAddress, MAC, URI
```

Functions

Small!

```
// rules of functions:
// 1. should be small
// 2. should be smaller than that
// < 150 characters per line
// < 20 lines
```

Do One Thing

```
// FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL.
// THEY SHOULD DO IT ONLY.
```

Functions

One Level of Abstraction per Function

```
// high level of abstraction
getHtml()

// intermediate level of abstraction
const pagePathName = PathParser.render(pagePath);

// remarkably low level
.append("\n")
```

Reading Code from Top to Bottom

```
// the Stepdown Rule
```

Functions

Switch Statements

```
class Employee...
  payAmount() {
    switch (getType()) {
    case EmployeeType.ENGINEER:
      return _monthlySalary;
    case EmployeeType.SALESMAN:
      return _monthlySalary + _commission;
    case EmployeeType.MANAGER:
      return _monthlySalary + _bonus;
    default:
      throw "Incorrect Employee";
    }
  }
```

Functions

Switch Statements

```
class EmployeeType...
  payAmount(employee)...

class Salesman extends EmployeeType...
  payAmount(employee) {
    return employee.getMonthlySalary() + employee.getCommission();
  }

class Manager extends EmployeeType...
  payAmount(employee) {
    return employee.getMonthlySalary() + employee.getBonus();
  }
```

Functions

Use Descriptive Names

```
(testableHtml) => includeSetupAndTeardownPages;

includeSetupAndTeardownPages,
  includeSetupPages,
  includeSuiteSetupPage,
  includeSetupPage;
// what happened to
includeTeardownPages, includeSuiteTeardownPage, includeTeardownPage;
```

Function Arguments

```
// the ideal number of arguments for a function is zero
```

Functions

Common Monadic Forms

```
// if a function is going to transform its input argument, // the transformation should appear as the return value

StringBuffer transform(in)
// is better than
void transform(out)
```

Flag Arguments

```
render(true);
```

Functions

Common Monadic Forms

```
// asking a question about that argument

fileExists(“MyFile”)
// operating on that argument, transforming and returning it
fileOpen(“MyFile”)
// event, use the argument to alter the state of the system
passwordAttemptFailedNtimes(attempts)
```

Flag Arguments

```
renderForSuite();
renderForSingleTest();
```

Functions

Dyadic Functions

```
writeField(name);
// is easier to understand than
writeField(outputStream, name);

// perfectly reasonable
const p = new Point(0, 0);
// problematic
assertEquals(expected, actual);
```

Triads

```
  assertEquals(message, expected, actual)
```

Functions

Argument Objects

```
makeCircle(x, y, radius);
makeCircle(center, radius);
```

Verbs and Keywords

```
write(name);
writeField(name);

assertEquals(expected, actual);
assertExpectedEqualsActual(expected, actual);
```

Functions

Have No Side Effects

```
// do something or answer something, but not both
set(attribute, value);

setAndCheckIfExists

if (attributeExists("username")) {
  setAttribute("username", "unclebob");
  ...
}
```

Functions

Don’t Repeat Yourself (DRY)

```
// duplication may be the root of all evil in software
```

Structured Programming

```
// Edsger Dijkstra’s rules
//  one entry
//  one exit

// functions small
// occasional multiple return, break, or continue statement
// can sometimes even be more expressive Dijkstra’s rules
```

Comments

Comments Do Not Make Up for Bad Code

```
// don’t comment bad code, rewrite it!
```

Explain Yourself in Code

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) && (employee.age > 65))
  ...

// better
if (employee.isEligibleForFullBenefits())
```

Comments (good)

Legal Comments

```
// Copyright (C) 2011 by Osoco. All rights reserved.
// Released under the terms of the GNU General Public License
// version 2 or later.
```

Informative Comments

```
// Returns an instance of the Responder being tested.
responderInstance();

// renaming the function: responderBeingTested
// format matched kk:mm:ss EEE, MMM dd, yyyy
const timeMatcher = Pattern.compile("\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

Comments (good)

Explanation of Intent

```
//This is our best attempt to get a race condition //by creating large number of threads.
for (let i = 0; i < 25000; i++) {
  const widgetBuilderThread = new WidgetBuilderThread(
    widgetBuilder,
    text,
    failFlag
  );
  const thread = new Thread(widgetBuilderThread);
  thread.start();
}
```

Clarification

```
assertTrue(a.compareTo(b) == -1); // a < b assertTrue(b.compareTo(a) == 1); // b > a
```

Comments (good)

Warning of Consequences

```
function makeStandardHttpDateFormat() {
  //SimpleDateFormat is not thread safe,
  //so we need to create each instance independently.
  const df = new SimpleDateFormat("dd MM yyyy");
  df.setTimeZone(TimeZone.getTimeZone("GMT"));
  return df;
}
```

TODO Comments

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
```

Comments (good)

Amplification

```
const listItemContent = match.group(3).trim();
// the trim is real important. It removes the starting
// spaces that could cause the item to be recognized
// as another list.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end()));
```

Comments (bad)

Mumbling

```
try {
  const propertiesPath = propertiesLocation + "/" + PROPERTIES_FILE;
  const propertiesStream = new FileInputStream(propertiesPath);
  loadedProperties.load(propertiesStream);
} catch(e) {
  // No properties files means all defaults are loaded
}
```

Comments (bad)

Redundant Comments

```
// Utility method that returns when this.closed is true.
// Throws an exception if the timeout is reached.
function waitForClose(timeoutMillis) {
  if (!closed) {
    wait(timeoutMillis);
    if (!closed) {
      throw "MockResponseSender could not be closed";
    }
  }
}
```

Comments (bad)

Redundant Comments

```
/**
* The processor delay for this component. */
protected backgroundProcessorDelay = -1;
/**
* The lifecycle event support for this component. */
protected LifecycleSupport lifecycle = new LifecycleSupport(this);
/**
* The container event listeners for this Container. */
protected ArrayList listeners = new ArrayList();
```

Comments (bad)

Mandated Comments

```
/**
* @param title The title of the CD
* @param author The author of the CD
* @param tracks The number of tracks on the CD
* @param durationInMinutes The duration of the CD in minutes */
void addCD(title, author,
tracks, durationInMinutes) {
CD cd = new CD();
cd.title = title;
cd.author = author;
cd.tracks = tracks;
cd.duration = durationInMinutes;
 }
```

Comments (bad)

Journal Comments

```
/*
* Changes (from 11-Oct-2001) * -------------------------
* 11-Oct-2001 : * Re-organised the class and moved it to new package com.jrefinery.date (DG);
* 05-Nov-2001 : * Added a getDescription() method, and eliminated NotableDate class (DG);
* 12-Nov-2001 : * IBD requires setDescription() method, now that NotableDate class is gone (DG)
* 05-Dec-2001 : * Fixed bug in SpreadsheetDate class (DG);
```

Comments (bad)

Noise Comments

```
/** constructor. **/
constructor() { }
/** The day of the month. */
const dayOfMonth;
/* Returns the day of the month. * @return the day of the month. */
function getDayOfMonth() {
  return dayOfMonth;
}
```

Comments (bad)

Scary Noise

```
/** The name. */
let name;
/** The version. */
let version;
/** The licenceName. */
let licenceName;
/** The version. */
let info;
```

Comments (bad)

Don’t Use a Comment When You Can Use a Function or a Variable

```
// does the module from the global list <mod> depend on the
// subsystem we are part of?
if (smodule.getDependSubsystems().contains(subSysMod.getSubSystem()))

// this could be rephrased without the comment as
const moduleDependees = smodule.getDependSubsystems();
ourSubSystem = subSysMod.getSubSystem();
if (moduleDependees.contains(ourSubSystem))
```

Comments (bad)

Position Markers

```
// Actions ////////////////////////////////////
```

Closing Brace Comments

```
while ((line = in.readLine()) != null) {
  lineCount++;
  charCount += line.length();
  words = line.split("\\W");
  wordCount += words.length;
} //while
```

Comments (bad)

Attributions and Bylines

```
/* Added by Rick */
```

Commented-Out Code

```
const response = new InputStreamResponse();
response.setBody(formatter.getResultStream(), formatter.getByteCount());
// const resultsStream = formatter.getResultStream();
// const reader = new StreamReader(resultsStream);
// response.setContent(reader.read(formatter.getByteCount()));
```

Comments (bad)

HTML Comments

```
/**
* Task to run fit tests.
* This task runs fitnesse tests and publishes the results.
* <p/>
* <pre>
* Usage:
* &lt;taskdef name=&quot;execute-fitnesse-tests&quot;
* classname=&quot;fitnesse.ant.ExecuteFitnesseTestsTask&quot * classpathref=&quot;classpath&quot; /&gt;
* OR
* &lt;taskdef classpathref=&quot;classpath&quot;
* resource=&quot;tasks.properties&quot; /&gt;
* <p/>
* &lt;execute-fitnesse-tests
 ;
```

Comments (bad)

Nonlocal Information

```
/*
- Port on which fitnesse would run. Defaults to <b>8082</b>. \*
- @param fitnessePort
*/
setFitnessePort(fitnessePort) {
  this.fitnessePort = fitnessePort;
}
```

Comments (bad)

Too Much Information

```
/*
RFC 2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies section 6.8. Base64 Content-Transfer-Encoding
The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8-bit input groups.
These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet.
When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most- significant-bit first.
*/
```

Comments (bad)

Inobvious Connection

```
/*
start with an array that is big enough to hold all the pixels
(plus filter bytes), and an extra 200 bytes for header info
*/
this.pngBytes = new byte[((this.width + 1) * this.height * 3) + 200];
```

Function Headers

```
// short functions don’t need much description
```

Formatting

The Purpose of Formatting

```
// communication
```

The Newspaper Metaphor

```
// high-level -> details
```

Vertical Openness Between Concepts

```
// each blank line is a visual cue
// that identifies a new and separate concept
```

Formatting

Vertical Distance

```
// variables
// should be declared as close to their usage as possible

// instance variables
// should be declared at the top of the class

// dependent functions
// if one function calls another, they should be vertically
// close, and the caller should be above the called

// conceptual affinity
// certain bits of code want to be near other bits
```

Formatting

Horizontal Openness and Density

```
function measureLine(line) {
  lineCount++;
  const lineSize = line.length();
  totalChars += lineSize;
  lineWidthHistogram.addLine(lineSize, lineCount);
  recordWidestLine(lineSize);
}

function root2(a, b, c) {
  const determinant = determinant(a, b, c);
  return (-b - Math.sqrt(determinant)) / (2 * a);
}
```

Formatting

Horizontal Alignment

```
class FitNesseExpediter extends ResponseSender {
  ...
  let Socket           socket;
  let InputStream      input;
  let OutputStream     output;
  let Request          request;
  let Response         response;
  let FitNesseContext context;
  ...
 }
```

Formatting

Horizontal Alignment

```
class FitNesseExpediter extends ResponseSender {
  ...
  let Socket socket;
  let InputStream input;
  let OutputStream output;
  let Request request;
  let Response response;
  let FitNesseContext context;
  ...
 }
```

Formatting

Breaking Indentation

```
class CommentWidget extends TextWidget {
  const REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";
  CommentWidget(text) { super(text); }
  render() { return ""; }
}
```

Formatting

Breaking Indentation

```
class CommentWidget extends TextWidget {
  const REGEXP = "^#[^\r\n]\*(?:(?:\r\n)|\n|\r)?";
  CommentWidget(text) {
  super(text);
  }
  render() {
    return "";
  }
}
```

Formatting

Team Rules

```
// every programmer has his own favorite formatting rules
// but if he works in a team
// then the team rules
```

Error Handling

Prefer Exceptions to Returning Error Codes

```
    if (deletePage(page) == E_OK) {
      if (registry.deleteReference(page.name) == E_OK) {
        if (configKeys.deleteKey(page.name.makeKey()) == E_OK) {
          logger.log("page deleted");
        } else {
          logger.log("configKey not deleted");
        }
      } else {
        logger.log("deleteReference from registry failed");
      }
    } else {
      logger.log("delete failed");
      return E_ERROR;
    }
```

Error Handling

Prefer Exceptions to Returning Error Codes

```
try {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
}
catch (e) {
  logger.log(e.getMessage());
}
```

Error Handling

Extract Try/Catch Blocks

```
function delete(page) {
  try {
    deletePageAndAllReferences(page);
  } catch (e) {
    logError(e);
  }
}

function deletePageAndAllReferences(page) {
  deletePage(page);
  registry.deleteReference(page.name);
  configKeys.deleteKey(page.name.makeKey());
}

function logError(e) {
  logger.log(e.getMessage());
}
```

Error Handling

Error Handling Is One Thing

```
// functions should do one thing
// error handing is one thing

// if the keyword try exists in a function
// it should be the very first word in the function and that
// there should be nothing after the catch/finally blocks
```

Error Handling

Define the Normal Flow

```
try {
  const expenses = expenseReportDAO.getMeals(employee.getID());
  m_total += expenses.getTotal();
} catch(MealExpensesNotFound e) {
  m_total += getMealPerDiem();
}
```

Error Handling

Define the Normal Flow

```
MealExpenses expenses = expenseReportDAO.getMeals(employee.getID());
m_total += expenses.getTotal();
```

Error Handling

Don’t Return Null

```
const employees = getEmployees();
if (employees != null) {
  for(let e of employees) {
    totalPay += e.getPay();
  }
}
```

Error Handling

Don’t Return Null

```
const employees = getEmployees();
for(let e of employees) {
  totalPay += e.getPay();
}

function getEmployees() {
  if( .. there are no employees .. )
    return [];
}
```

Error Handling

Don’t Pass Null

```
function xProjection(p1, p2) {
  return (p2.x – p1.x) * 1.5;
}

function xProjection(p1, p2) {
  if (p1 == null || p2 == null) {
    throw "Invalid argument for MetricsCalculator.xProjection";
  }
  return (p2.x – p1.x) * 1.5;
}
```

Unit Tests

The Three Laws of TDD

```
// first law
// you may not write production code until
// you have written a failing unit test
// second law
// you may not write more of a unit test
// than is sufficient to fail, and not compiling is failing
// third law
// you may not write more production code
// than is sufficient to pass the currently failing test
```

Unit Tests

Keeping Tests Clean

```
// test code is just as important as production code
```

Clean Tests

```
// what makes a clean test? three things
// readability, readability, and readability
```

Unit Tests

One Assert per Test

```
    // tests come to a single conclusion
    // that is quick and easy to understand
```

Single Concept per Test

```
    // the best rule is that you should
    // minimize the number of asserts per concept and
    // test just one concept per test function
```

Unit Tests

F.I.R.S.T.

```
// Fast
// Independent
// Repeatable
// Self-validating // Timely
```

Classes

Class Organization

```
// constants
// private variables
// private instance variables
// functions
// private utilities called by a function right after
```

Classes Should Be Small!

```
      // the first rule is that they should be small
      // the second rule is that they should be smaller than that
```

Classes

The Single Responsibility Principle (SRP)

```
// a class or module should have one, and only one, // reason to change
// SRP is one of the more important concept in OO design
```

Cohesion

```
// maintaining cohesion results in many small classes
```

Emergence

```
Simple Design Rule 1: Runs All the Tests
Simple Design Rules 2: No Duplication
Simple Design Rules 3: Expressive
Simple Design Rules 4: Minimal Classes and Methods
```

Quelques outils

Prettier

- Prettier is an opinionated code **formatter**.
- Formatter = a tools that scans your files for style issues and automatically reformats your code to ensure consistent rules
- **Formating rules**: e.g. `max-len`, `no-mixed-spaces-and-tabs`, `keyword-spacing`,etc.

```
foo(reallyLongArg(),omgSoManyParameters(),IShouldRefactorThis(),isThereSeriouslyAnotherOne()
);
```

```
foo(
  reallyLongArg(),
  omgSoManyParameters(),
  IShouldRefactorThis(),
  isThereSeriouslyAnotherOne()
);
```

ESLint

- ESLint is the most popular JavaScript **linter**
- Linter = a tool that scans code for errors and in some instances can fix them automatically
- Error can include:
  - Coding errors that lead to bugs
  - Stylistic errors
- **Code-quality rules**: eg `no-unused-vars`, `no-extra-bind`, `no-implicit-globals`, `prefer-promise-reject-errors`, etc.

Style Guides

Javascript

- JavaScript Standard Style
- Airbnb JavaScript Style Guide() {
- eslint-config-google

React

- ESLint-plugin-React
- Airbnb React/JSX Style Guide

API

- Google
- Microsoft

Rule(Rule Name)	Google	AirBnB	Standard
Semicolons (semi)	Required	Required	No
Trailing Commas (comma-dangle)	Required	Required	Not Allowed
Template Strings (prefer-template)	No Stance	Prefered	No Stance
Space Before Function Parentheses (space-before-function-paren)	No Space	No Space	Space Required
Import Extensions (import/extensions)	Allowed	Not Allowed	Allowed
Object Curly Spacing (object-curly-spacing)	No Space Allowed	Space Required	Space Required
Console Statements (no-console)	No Stance	None	No Stance
Arrow Functions Return Assignment (no-return-assign)	No Stance	No	No
React Prop Ordering (react/sort-prop-types)	N/A	No Stance	No Stance
React Prop Validation (react/prop-types)	N/A	Required	Not Required
Object Property Shorthand (object-shorthand)	No Stance	Prefer	No Stance
Object Destructuring (prefer-destructuring)	No Stance	Prefer	No Stance