writing code for other people

CCJS Edition!

Glen R. Goodwin

before we start…

writing good code is HARD

writing <u>GREAT</u> code is HARDER

good code is
written to be executed.

and

great code is
written to be read.

good code
gets the job done

good code works…
but great code teaches.

great code teaches others…

- to understand the problem
- to understand the solution
- to understand how
  to go beyond both

great code teaches others…

- to understand the problem
- to understand the solution
- to understand how
  to go beyond both

great code teaches others…

- to understand the problem
- to understand the solution
- to understand how
  to go beyond both

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

great code is

- readable
- contextual
- understandable
- usable

writing great code
is about writing code
for other people.

writing great code
is about writing code
for other people.

writing great code
is about writing code
for other people.

writing great code
is about writing code
for other people.

writing great code
is about writing code
for other people.

Readable

indentation conveys hierarchy

1/readable/indentation conveys hierarchy

```
const FS = require('fs');
const contents = FS.readFileSync(process.argv[2], 'utf8');
let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
line = line.toLowerCase();
line = line.replace(/[^\sA-Za-z0-9-]/g, '');
line = line.replace(/\s\s|\t/g, ' ');
return line;
});
lines = lines.map(line => line.split(/\s/));
const words = lines.reduce((words, line) => {
line.forEach(word => {
words[word] = words[word] + 1 || 1;
});
return words;
},{});
Object.keys(words).forEach(word => {
console.log(word + ' ' + words[word]);
});
```

1/readable/indentation conveys hierarchy

```
const FS = require('fs');
const contents = FS.readFileSync(process.argv[2], 'utf8');
let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));
const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
},{});
Object.keys(words).forEach(word => {
  console.log(word + ' ' + words[word]);
});
```

1/readable/indentation conveys hierarchy

```
// JS
function add(x,y) {
  return x + y;
}


// HTML
<div>
  <button>
    Click me!
  </button>
</div>


// CSS
div > button {
  background: red;
}
```

Readable

meaningful
whitespace

2/readable/meaningful whitespace

```
const FS = require('fs');
const contents = FS.readFileSync(process.argv[2], 'utf8');
let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));
const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
},{});
Object.keys(words).forEach(word => {
  console.log(word + ' ' + words[word]);
});
```

2/readable/meaningful whitespace

```
const FS = require('fs');

const contents = FS.readFileSync(process.argv[2], 'utf8');

let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));

const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
},{});

Object.keys(words).forEach(word => {
  console.log(word + ' ' + words[word]);
});
```

2/readable/meaningful whitespace

```
export default Component.extend({
  classNames: ['my-magic-dialog'],
  magic: null,
  magicSelect: null,
  makeMagic: task(function(magic) {
    this.set('magic', magic);
  }),
  clickMagic: action(function() {
    if (this.magicSelect) this.magicSelect(this.magic);
    if (this.closeMagic) this.closeMagic();
  })
});
```

2/readable/meaningful whitespace

```
export default Component.extend({
  classNames: ['my-magic-dialog'],

  magic: null,
  magicSelect: null,

  makeMagic: task(function(magic) {
    this.set('magic', magic);
  }),

  clickMagic: action(function() {
    if (this.magicSelect) this.magicSelect(this.magic);
    if (this.closeMagic) this.closeMagic();
  })
});
```

contextual

contextual
naming

3/contextual/contextual naming

```
function x(y) {
  return y < 1 ? 0
    : y <= 2 ? 1
    : x(y - 1) + x(y - 2);
}
```

3/contextual/contextual naming

```
function fibonacci(y) {
  return y < 1 ? 0
    : y <= 2 ? 1
    : fibonacci(y - 1) + fibonacci(y - 2);
}
```

3/contextual/contextual naming

```
const THIS_IS_MY_CONSTANT = 123;     // SNAKE_CASE with UPPERCASE

class ThisIsMyClass {};              // PascalCase

let thisIsMyVariable = 123;          // camelCase
const alsoThisVariable = 456;

class MyClass {
  static MyStaticMethod() { }        // PascalCase
}

class MyClass {
  myMethod() { }                     // camelCase
}
```

contextual

convey logic
through comments

4/contextual/convey logic through comments

```
const FS = require('fs');

const contents = FS.readFileSync(process.argv[2], 'utf8');

let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));

const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
},{});

Object.keys(words).forEach(word => {
  console.log(word + ' ' + words[word]);
});
```

4/contextual/convey logic through comments

```
// Bring in the standard filesystem library
const FS = require('fs');

// Read the file
const contents = FS.readFileSync(process.argv[2], 'utf8');

// Split our file content into lines and
// convert each line to an array of simple words
let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));

// Count the occurance of each word in all the content.
const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
...
```

4/contextual/convey logic through comments

```
// Bring in the standard filesystem library
const FS = require('fs');

// Read the file
const contents = FS.readFileSync(process.argv[2], 'utf8');

// Split our file content into lines and
// convert each line to an array of simple words
let lines = contents.split(/\r\n|\n/);
lines = lines.map(line => {
  line = line.toLowerCase();
  line = line.replace(/[^\sA-Za-z0-9-]/g, '');
  line = line.replace(/\s\s|\t/g, ' ');
  return line;
});
lines = lines.map(line => line.split(/\s/));

// Count the occurance of each word in all the content.
const words = lines.reduce((words, line) => {
  line.forEach(word => {
    words[word] = words[word] + 1 || 1;
  });
  return words;
...
```

understandable

organize your code

5/understandable/organize your code

```
// JS/TS file organization

// require libraries
const fs = require("fs");

// declare constants
const MY_CONSTANT = 123;

// declare variables
let myVariable = 456;

// declare classes
class Blah {}

// declare functions
Function someFunc() {}
```

5/understandable/organize your code

```
// JS/TS Class organization

Class Blah {
  // declare static members
  static MyStaticVariable = 123;

  // declate members
  myVariable = 456;

  // declare static methods
  static MyStaticMethod() {}

  // declare methods
  myMethod() {}
}
```

5/understandable/organize your code

```
// JS/TS Function organization

function blah(x,y,z) {
  // Argument Checking
  if (!x && !y && !z) return;

  // Variable declaration
  const abc = x * y;

  // especially before you use them
  let n = 10;
  while (n) n--;
}
```

5/understandable/organize your code

```
// NEVER RELY ON HOISTING

function blah(x,y,z) {
  let n = 10;
  while (n) n = decrement(n,1);

  function decrement(n,step) {
    return n - step;
  }
}
```

understandable

separation of concerns

6/understandable/separation of concerns

```
// MyComponent.html
<div id="my-component-1">
  This is my component
</div>

// MyComponent.css
#my-component-1 {
  display: grid;
  grid-template-columns: 100px;
  grid-template-rows: 100px;
  border: 10px solid red;
}

// MyComponent.js
document.querySelector("#my-component-1").addEventListener("click", () => {
  console.log("MyComponent was clicked!");
});
```

6/understandable/separation of concerns

```
// MyComponent.html
<div id="my-component-1">
  This is my component
</div>

// MyComponent.css
#my-component-1 {
  display: grid;
  grid-template-columns: 100px;
  grid-template-rows: 100px;
  border: 10px solid red;
}

// MyComponent.js
document.querySelector("#my-component-1").addEventListener("click", () => {
  console.log("MyComponent was clicked!");
});
```

6/understandable/separation of concerns

```
// MyComponent.html
<div id="my-component-1">
  This is my component
</div>

// MyComponent.css
#my-component-1 {
  display: grid;
  grid-template-columns: 100px;
  grid-template-rows: 100px;
  border: 10px solid red;
}

// MyComponent.js
document.querySelector("#my-component-1").addEventListener("click", () => {
  console.log("MyComponent was clicked!");
});
```

6/understandable/separation of concerns

```
// MyComponent.html
<div id="my-component-1">
  This is my component
</div>

// MyComponent.css
#my-component-1 {
  display: grid;
  grid-template-columns: 100px;
  grid-template-rows: 100px;
  border: 10px solid red;
}

// MyComponent.js
document.querySelector("#my-component-1").addEventListener("click", () => {
  console.log("MyComponent was clicked!");
});
```

6/understandable/separation of concerns

```
// MyComponent.html
<div id="my-component-1">
  This is my component
</div>

// MyComponent.css
#my-component-1 {
  display: grid;
  grid-template-columns: 100px;
  grid-template-rows: 100px;
  border: 10px solid red;
}

// MyComponent.js
document.querySelector("#my-component-1").addEventListener("click", () => {
  console.log("MyComponent was clicked!");
});
```

6/understandable/separation of concerns

```
// Coupling

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}

function even(n) {
  if (n < 1) return;
  !(n%2) && console.log(n + " is even.");
  odd(!(n%2) ? n-1 : n);
}

odd(10);
```

6/understandable/separation of concerns

```
// Coupling

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}

function even(n) {
  if (n < 1) return;
  !(n%2) && console.log(n + " is even.");
  odd(!(n%2) ? n-1 : n);
}

odd(10);
```

6/understandable/separation of concerns

```
// Coupling

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}

function even(n) {
  if (n < 1) return;
  !(n%2) && console.log(n + " is even.");
  odd(!(n%2) ? n-1 : n);
}

odd(10);
```

6/understandable/separation of concerns

```
// Cohesion

function isOdd(n) {
  return !!(n%2);
}

function isEven(n) {
  return !(n%2);
}

function isRectangle(r) {
  return r.x && r.y && r.width && r.height && r.width>0 && r.height>0;
}
```

6/understandable/separation of concerns

```
// Cohesion

function isOdd(n) {
  return !!(n%2);
}

function isEven(n) {
  return !(n%2);
}

function isRectangle(r) {
  return r.x && r.y && r.width && r.height && r.width>0 && r.height>0;
}
```

6/understandable/separation of concerns

```
// Cyclomatic complexity

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}
```

6/understandable/separation of concerns

```
// Cyclomatic complexity

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}
```

6/understandable/separation of concerns

```
// Cyclomatic complexity

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}
```

6/understandable/separation of concerns

```
// Cyclomatic complexity

function odd(n) {
   if (n < 1) return;
   n%2 && console.log(n + " is odd.");
   even(n%2 ? n-1 : n);
}
```

6/understandable/separation of concerns

```
// Cyclomatic complexity

function odd(n) {
  if (n < 1) return;
  n%2 && console.log(n + " is odd.");
  even(n%2 ? n-1 : n);
}
```

6/understandable/separation of concerns

```
// Coupling - Weak is Good

// Cohesion - High is Good

// Cyclomatic Complexity - Lower is better
```

usable

defensive coding

7/usable/defensive coding

```
function fibonacci(y) {
  return y < 1 ? 0
       : y <= 2 ? 1
       : fibonacci(y - 1) + fibonacci(y - 2);
}
```

7/usable/defensive coding

```
function fibonacci(y) {
   if (typeof y!=="number") throw new Error("Y must be a number!");
  return y < 1 ? 0
       : y <= 2 ? 1
       : fibonacci(y - 1) + fibonacci(y - 2);
}
```

7/usable/defensive coding

```
function fibonacci(y) {
  if (typeof y!=="number") throw new Error("Y must be a number!");
  return y < 1 ? 0
       : y <= 2 ? 1
       : fibonacci(y - 1) + fibonacci(y - 2);
}
```

7/usable/defensive coding

```
function fibonacci(y) {
  if (typeof y!=="number") throw new Error("Y must be a number!");
  return y < 1 ? 0
        : y <= 2 ? 1
        : fibonacci(y - 1) + fibonacci(y - 2);
}
```

usable

code is a
user interface

7/usable/defensive coding

Build everything with another user's experience in mind.

8/usable/code is a user interface

Build everything with another user's experience in mind.

8/usable/code is a user interface

No code is an island…

usable

documentation

9/usable/documentation

```
/*

   If you fail to communicate how to
   run your code, you might as well
   not have written the code at all.

*/
```

9/usable/documentation

```
// this is a comment
// use it to describe logic.


/*
   This is documentation.

   Use it to tell a user
   how to use your code.
*/
```

9/usable/documentation

```
/**
 * Move the wizard selection to the "first" section.
 *
 * @return {void}
 */
selectFirst() {
  const first = this.sections.find(section => section.startHere);
  this.select(first || this.sections[0] || null);
}
```

9/usable/documentation

```
/**
 * Move the wizard selection to the "first" section.
 * @return {void}
 */
selectFirst() {
  const first = this.sections.find(section => section.startHere);
  this.select(first || this.sections[0] || null);
}
```

9/usable/documentation

```
/**
 * Move the wizard selection to the "first" section.
 * @return {void}
 */
selectFirst() {
  const first = this.sections.find(section => section.startHere);
  this.select(first || this.sections[0] || null);
}
```

great code is

1/readable/indentation conveys hierarchy
2/readable/meaningful whitespace
3/contextual/contextual naming
4/contextual/convey logic through comments
5/understandable/organize your code
6/understandable/separation of concerns
7/usable/defensive coding
8/usable/code is a user interface
9/usable/documentation

great code is

1/readable/indentation conveys hierarchy
2/readable/meaningful whitespace
3/contextual/contextual naming
4/contextual/convey logic through comments
5/understandable/organize your code
6/understandable/separation of concerns
7/usable/defensive coding
8/usable/code is a user interface
9/usable/documentation

great code is

1/readable/indentation conveys hierarchy
2/readable/meaningful whitespace
3/contextual/contextual naming
4/contextual/convey logic through comments
5/understandable/organize your code
6/understandable/separation of concerns
7/usable/defensive coding
8/usable/code is a user interface
9/usable/documentation

writing great code
is about writing code
for other people.

great code is

- readable
- contextual
- understandable
- usable

good code works…
but great code teaches.

Glen R. Goodwin

whitebox.com

@areinet

arei.net

github.com/arei