CS 4500
Software Development

[Code, pt 2]

Ferdinand Vesely
(partially based on Clean Code by Robert C. Martin)

September 20, 2019

Functions

Best Practices, not Dogma

- Good to remember: these are principles, best practices
- Blindly applying them $\neq$ good code
- Always use good judgement
- Consider circumstances
- Think about readability

```java
public String testableHtml(PageData pageData, boolean includeSuiteSetup)
          throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();

    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup = PageCrawlerImpl.getInheritedPage(SuiteResponder.SUIT
            if (suiteSetup != null) {
                WikiPagePath pagePath = wikiPage.getPageCrawler().getFullPath(suiteSet
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .").append(pagePathName).append("\n");
            }
        }
        WikiPage setup = PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath = wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .").append(setupPathName).append("\n");
        }
    }
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown = PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath = wikiPage.getPageCrawler().getFullPath(teardown
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("!include -teardown .").append(tearDownPathName).append("\n"
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown = PageCrawlerImpl.getInheritedPage(SuiteResponder.SU
            if (suiteTeardown != null) {
                WikiPagePath pagePath = wikiPage.getPageCrawler().getFullPath(suiteTea
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .").append(pagePathName).append("\n"
            }
        }
    }

    pageData.setContent(buffer.toString());
    return pageData.getHtml();
}
```

Functions: Keep ’em Small

- No function should be longer than 20 lines
- Much shorter, actually – 3-4 lines long for many functions
- Old “one screenful rule”
  - “a function should fit onto a terminal screen”
- But: avoid, e.g., one-liners with verbose names
- Consider:

```
private int isShapeGreenOrYellowOrRed(Shape shape) {
    return shape.getColor() == GREEN
        || shape.getColor() == YELLOW
        || shape.getColor() == RED;
}
```

Is it improving readability?

Blocks and Indenting

- Blocks in `if-else`, `for`, `while` should be short (~1 line long)
- None or minimal nested structures
- Consequence: 1-2 levels of indentation

```
...
if (pageData.hasAttribute("Test")) {
    if (includeSuiteSetup) {
        ...
        if (suiteSetup != null) {
            ...
        }
    }
}
...
```

...if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program. — Linux kernel coding style

```java
public static String renderPage(PageData pageData, boolean isSuite)
        throws Exception {

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

One Thing Only Rule

- Each function: do one thing only (and do it well)
- What is one thing?

```java
public static String renderPage(PageData pageData, boolean isSuite)
    if (isTestPage(pageData))
        includeSetupAndTeardownPages(pageData, isSuite);
    return pageData.getHtml();
}
```

1. Check if test page
2. If yes, include setup and teardown
3. Render HTML

What is "One Thing"?

- In `renderPage`: the 3 items are one level of abstraction below
- We can write a "TO paragraph":

  *TO **renderPage**, check if the page is a test page. If yes, include setups and teardowns. Render page in HTML.*

One Level of Abstraction per Function

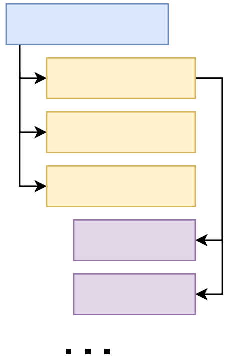
- Ensures that a function does one thing
- In the first example – multiple levels of abstraction:
  1. High-level, e.g., `pageData.getHtml()`
  2. Mid-level, e.g., `String pagePathName = PathParser.render(pagePath);`
  3. Low-level, e.g., `buffer.append("\n")`

One Level of Abstraction per Function

- Mixing levels of abstractions – confusing
- What is an essential concept?
  - Rendering HTML
- What is a detail?
  - Appending a newline to a buffer

The Stepdown Rule

- Source code organization principle
- Program should read like a top-down narrative
- Descending one level of abstraction at a time



The Stepdown Rule

```
public void makeBreakfast() {
   addEggs();
   cook();
   serve();
}
private void addEggs() {
    Set<Egg> eggs = fridge.getEggs()
    for (Egg egg : eggs)
        fryingPan.add(egg.open());
}
private void cook() {
    fryingPan.mixContents();
    fryingPan.add(salt.getABit());
    fryingPan.mixContents();
}
private void serve() {
    self.take(fryingPan.getContents(0.6));
    friend.take(fryingPan.getContents(0.4));
}
```

Descriptive Names

- Remember:

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

- Ergo: functions should be predictable (based on their names)
- Do not be afraid of long names, but also don’t overdo it
- Long names can be difficult to parse when reading
- Especially if very long names differ in a minor suffix

Descriptive Names

Not useful:

```
private void loginUserInitializeSessionAndSetTimeout(
        User user) {

    user.login()
    session = new UserSession(user);
    session.setTimeout(user.getSessionTimeout())
}
```

How Many Arguments?

- Some argue: ideal No. of args $= 0$
  - more than 3 should not be used
- Just keep everything in instance variables
- Language-specific
- Questionable
- Testing considerations

Arguments

- Multiple arguments are often conceptually close
- Good idea: bundle them in a data structure

```
Circle makeCircle(double x, double y, double radius);
Person(String firstName, String middleName, String lastName,
       String street, String apt, String zipCode, String state);
```

vs.

```
Circle makeCircle(Point center, double radius);
Person(PersonName name, Address homeAddress);
```

No Hidden Side-effects

- Side-effects = lies:
  - function promises to do one thing
  - performs additional, hidden things
- E.g., unexpected modification of global state, closing a DB handle

No Hidden Side-effects

```java
public boolean checkPassword(String userName, String password) {
    User user = UserGateway.findByName(userName);

    if (user != User.NULL) {
        String codedPhrase = user.getPhraseEncodedByPassword();
        String phrase = cryptographer.decrypt(codedPhrase, password);

        if ("Valid Password".equals(phrase)) {
            Session.initialize();
            return true;
        }
    }
    return false;
}
```

Output Arguments

Functions most naturally thought of as transforming input (arguments) into output (return value).

Take:

```
        appendFooter(report);
```

vs.

```
        report.appendFooter();
```

- Language dependent!

Command – Query Separation

- Either *do* something or *return* something
- Doing both – often confusion
- E.g.,

```
    public boolean set(String attribute, String value);
```

```
    if (set("age", 17)) {
      ...
```

"If age is set to 17…"?

"If setting age to 17 is successful…"?

Command – Query Separation

A solution: separate command from the query

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

Prefer Exceptions over Error Codes

- Use of error codes – violation of Command-Query Separation

```
  if (deletePage(page) == E_OK)
```

```
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0) {
    SDL_Log("Unable to initialize SDL: %s", SDL_GetError());
    return 1;
}
```

Error Codes

- Lead to nested structures:

```
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;
}
```

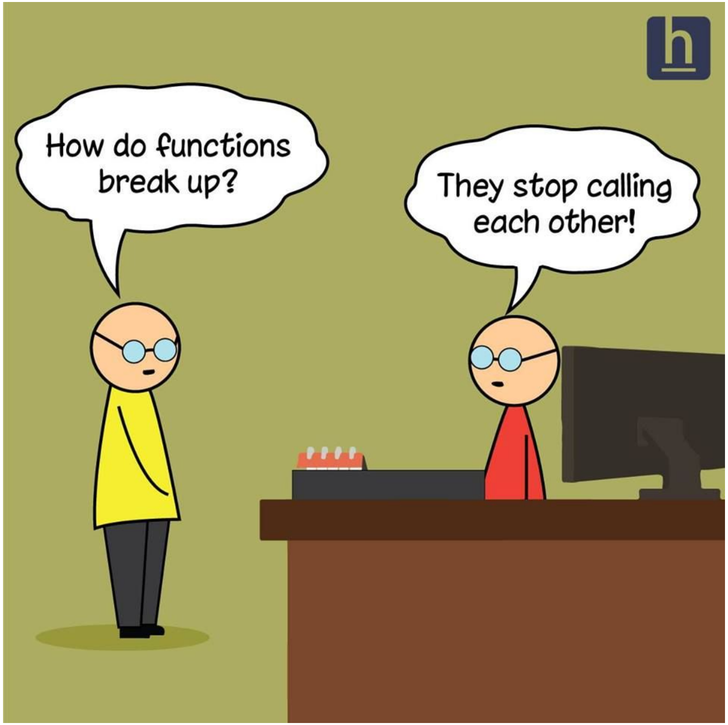
Exceptions

- Compare previous to:

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

Writing Functions

- Do not be afraid to write long, unwieldy functions initially
    - if that helps you writing an initial version
- But also: top-down / bottom-up
- Write tests, refine and restructure


How do functions break up?
They stop calling each other!