DESIGN: IMPLEMENTATION

Software Engineering
CS 130

Donald J. Patterson

Content adapted from Essentials of Software Engineering 3rd edition by Tsui, Karam, Bernal Jones and Bartlett Learning

IMPLEMENTATION

IMPLEMENTATION TOPICS

- Describe
  - Characteristics of good implementations
  - Best practices to achieve them
- Understand role of comments
- Learn debugging techniques
- Analyze refactoring

IMPLEMENTATION

INTRODUCTION

- Implementation:
  - transforming detailed design into valid program
- Detailed design may be done as part of implementation
  - Faster
  - Less cohesive and less organized
- Writing code, unit testing, debugging, configuration management

IMPLEMENTATION

GOOD IMPLEMENTATIONS

- Readability
- Maintainability
- Performance
- Traceability
- Correctness
- Completeness
- Other issues:
  - Relative importance ?
  - Tradeoffs ?

IMPLEMENTATION

CODING GUIDELINES

- Organization-specific
- Important for consistency
- Programmers can get used to them easily
- Usually mandate:
  - Indenting, formatting
  - Naming conventions (for files, variables etc)
  - Language features to use or avoid

INDENTATION

```
if (hours < 24 && minutes < 60 && seconds < 60) {
    return true;
} else {
    return false;
}
```

```
if (hours < 24 && minutes < 60 && seconds < 60)
{
    return true;
}
else
{
    return false;
}
```

```
if  ( hours   < 24
   && minutes < 60
   && seconds < 60
)
{return    true
;}         else
{return   false
;}
```

```
return (hours < 24 && minutes < 60 && seconds < 60);
```

More: Programming Style on Wikipedia

VERTICAL ALIGNMENT

```php
$search = array('a', 'b', 'c', 'd', 'e');
$replacement = array('foo', 'bar', 'baz', 'quux');

// Another example:

$value = 0;
$anothervalue = 1;
$yetanothervalue = 2;
```

```php
$search = array('a', 'b', 'c', 'd', 'e');
$r = array('foo', 'bar', 'baz', 'quux');

// Another example:

$value = 0;
$a = 1;
$yetanothervalue = 2;
```

```php
$search      = array('a',   'b',   'c',   'd',   'e');
$replacement = array('foo', 'bar', 'baz', 'quux');

// Another example:

$value           = 0;
$anothervalue    = 1;
$yetanothervalue = 2;
```

```php
$search      = array('a',   'b',   'c',   'd',   'e');
$r = array('foo', 'bar', 'baz', 'quux');

// Another example:

$value           = 0;
$a    = 1;
$yetanothervalue = 2;
```

More: Programming Style on Wikipedia

Overview
Downloading and Building
Developing
Contributing
Life of a Patch
Submitting Patches
View Patches
Life of a Bug
Reporting Bugs
Code Style Guidelines
Community

Code Style Guidelines for Contributors

IN THIS DOCUMENT

Java Language Rules
Don't Ignore Exceptions
Don't Catch Generic Exception
Don't Use Finalizers
Fully Qualify Imports
Java Library Rules
Java Style Rules
Use Javadoc Standard Comments
Write Short Methods
Define Fields in Standard Places
Limit Variable Scope
Order Import Statements
Use Spaces for Indentation
Follow Field Naming Conventions
Use Standard Brace Style
Limit Line Length
Use Standard Java Annotations
Treat Acronyms as Words
Use TODO Comments
Log Sparingly
Be Consistent
Javatests Style Rules
Follow Test Method Naming Conventions

The rules below are not guidelines or recommendations, but strict rules. Contributions to Android generally *will not be accepted* if they do not adhere to these rules.

Not all existing code follows these rules, but all new code is expected to.

Java Language Rules

We follow standard Java coding conventions. We add a few rules:

Don't Ignore Exceptions

Sometimes it is tempting to write code that completely ignores an exception like this:

```
void setServerPort(String value) {
    try {
        serverPort = Integer.parseInt(value);
    } catch (NumberFormatException e) { }
}
```

You must never do this. While you may think that your code will never encounter this error condition or that it is not important to handle it, ignoring exceptions like above creates mines in your code for someone else to trip over some day. You must handle every Exception in your code in some principled way. The specific handling varies depending on the case.

Anytime somebody has an empty catch clause they should have a creepy feeling. There are definitely times when it is actually the correct thing to do, but at least you have to think about it. In Java you can't escape the creepy feeling. -James Gosling

Acceptable alternatives (in order of preference) are:

- Throw the exception up to the caller of your method.

```
void setServerPort(String value) throws NumberFormatException {
    serverPort = Integer.parseInt(value);
}
```

IMPLEMENTATION

STYLE ISSUES - I

- Be consistent and highlight meaning
- Naming
  - Convey meaning
  - Be consistent
- Warning: If you can't think of a good name chances are you don't understand or the design can be improved
- Multicultural issues

```
a = b * c;
```

```
weekly_pay = hours_worked * pay_rate;
```

More: Naming Conventions on Wikipedia

IMPLEMENTATION

STYLE ISSUES - II

- Separating words, capitalization
  - c_uses_this_style
  - JavaUsesThisOne
- Indentation and Spacing
- Function/Method size
  - When is it too big ? When to break ?
- File naming
- Error prone constructs

IMPLEMENTATION

HUNGARIAN NOTATION

- `bBusy` : boolean
- `chInitial` : char
- `cApples` : count of items
- `dwLightYears` : double word (Systems)
- `fBusy` : float (or flag)
- `nSize` : integer (Systems) or count (Apps)
- `iSize` : integer (Systems) or index (Apps)
- `fpPrice` : floating-point
- `dbPi` : double (Systems)
- `pFoo` : pointer
- `rgStudents` : array, or range
- `szLastName` : zero-terminated string
- `u16Identifier` : unsigned 16-bit integer (Systems)
- `u32Identifier` : unsigned 32-bit integer (Systems)
- `stTime` : clock time structure
- `fnFunction` : function name

More: Hungarian Notation on Wikipedia

IMPLEMENTATION

COMMENTS

- Types:
  - Repeat of the code
  - Explanation of the code
  - Marker in the code
  - Summary of the code
  - Description of the code intent
  - External references
- Keep up to date !!

More: Funny source code comments

DEBUGGING

- Locating and fixing errors in code.
- Errors noticed by testing, inspection, use.
- Four phases
  - Stabilization (reproduction)
  - Localization
  - Correction
  - Verification

DEBUGGING II

- Heuristics:
  - Some routines will have many errors
  - Routines with an error tend to have more
  - New code tends to have more error
  - Particular ones: languages, parts, coders
- Tools
  - Code comparators
  - Extended checkers (lint)
  - Interactive debuggers
  - Special libraries
  - Others: Profilers, pre/post conditions, test coverage

IMPLEMENTATION

ASSERTIONS

- Pre-condition: condition your module requires in order to work
- Post-condition: condition that should be true if your module worked
- Assertion: Executable statement that checks a condition and produces an error if it is not met
- Assertions supported by many languages

IMPLEMENTATION

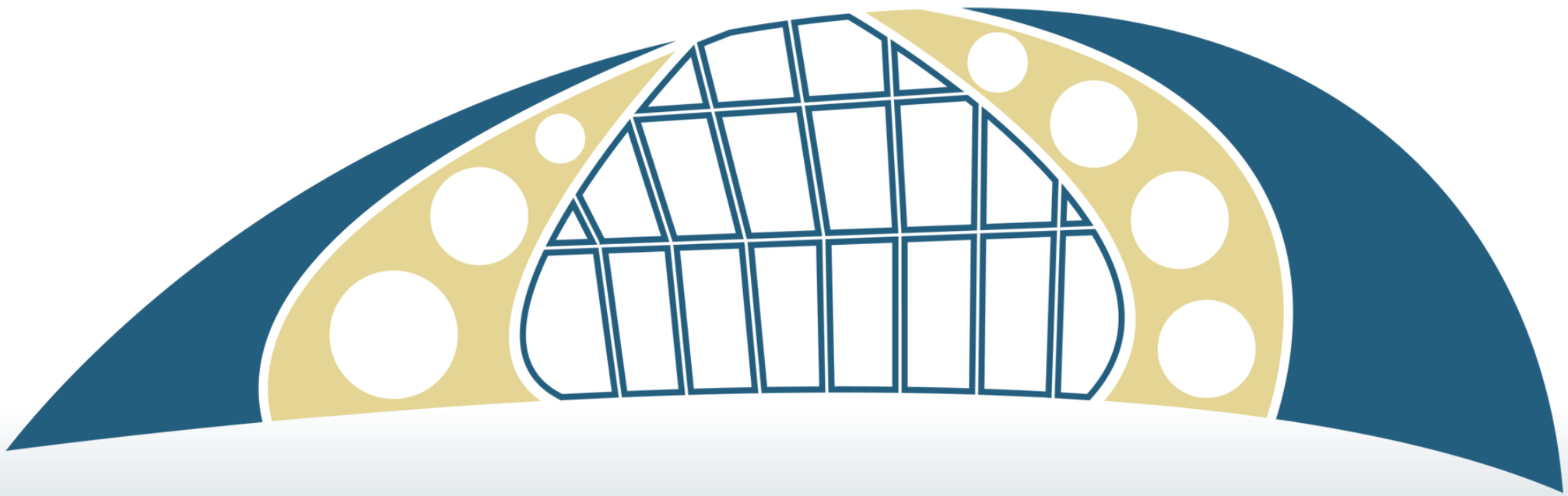
PERFORMANCE OPTIMIZATION

- Performance tradeoffs
  - Readability ?
  - Maintainability ?
- Correctness is usually more important
- Profiler: runs a program and calculates how much time it spends on each part
- Cost-benefit analysis
- Measure before ‘optimizing’

IMPLEMENTATION

REFACTORING

- Improving your code style without affecting its behavior
- Bad Smells
  - Duplicated code
  - Long method
  - Large class
  - Switch statement
  - Feature envy
  - Intimacy
- Refactoring
  - Extract method
  - Substitute algorithm
  - Move method
  - Extract class



WESTMONT INSPIRED

COMPUTING LAB