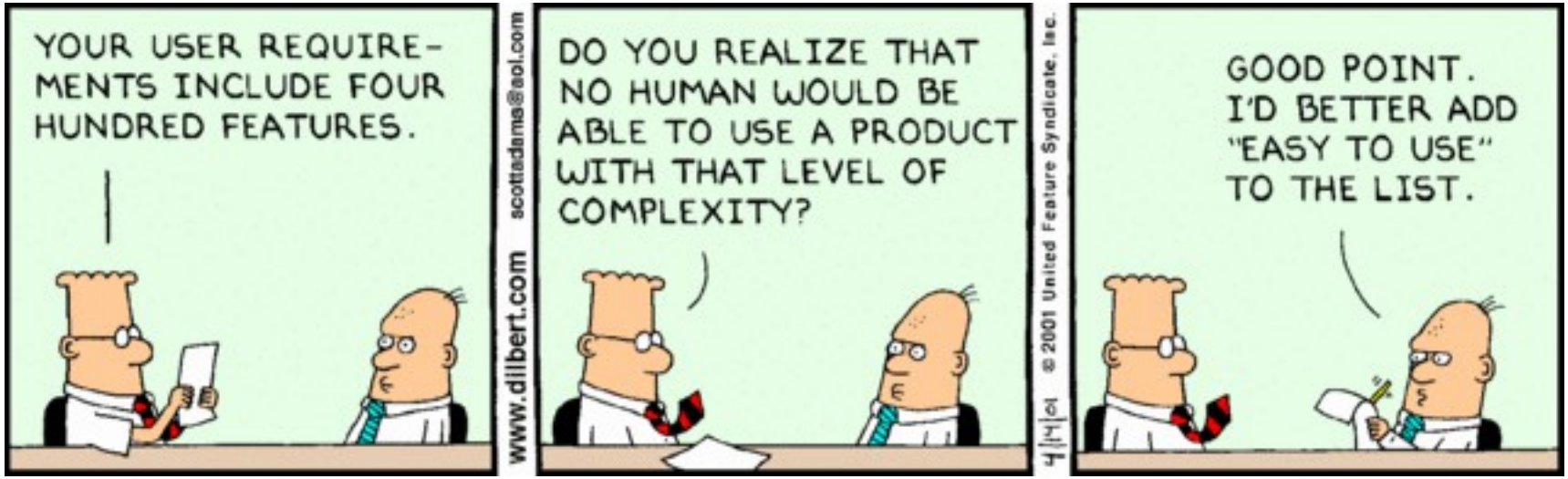
Software Design III

HW 1

- Functional/Non-Functional Requirements
- Professionalism & Presentation
  - Someone is giving you money (probably)
- Use cases vs. workflows
- Grammar, spelling, etc...



Where are we?

- Idea/Objective
- High-level requirements
- Users
- Use cases
- Workflows
- Components
- Functional Reqs
- Non-Functional Reqs
- UI mockups
- Interfaces
- Methods
- Data Structures & Algorithms
- 3rd party libraries/APIs/services

Today's Goals

- Understand importance of:
  - Design patterns
  - Language choices
  - Coding conventions
- Make implementation choices for your project
- Define a coding plan

Starting Exercise: Design a *Paragraph* class

Paragraph Requirements

- Sequence of lines – each line is a string
- Must provide:
  - List<String> alignedText();
  - String getLine(int i);
  - int getCountLines();
  - void addLine(String s);
- Formatting algorithm specified at runtime
- Able to add formatting algorithms without changing Paragraph class

Where are we going?

- Design Patterns
- Language Choices
- Code conventions
- Code structure
- Test plans
- Development planning

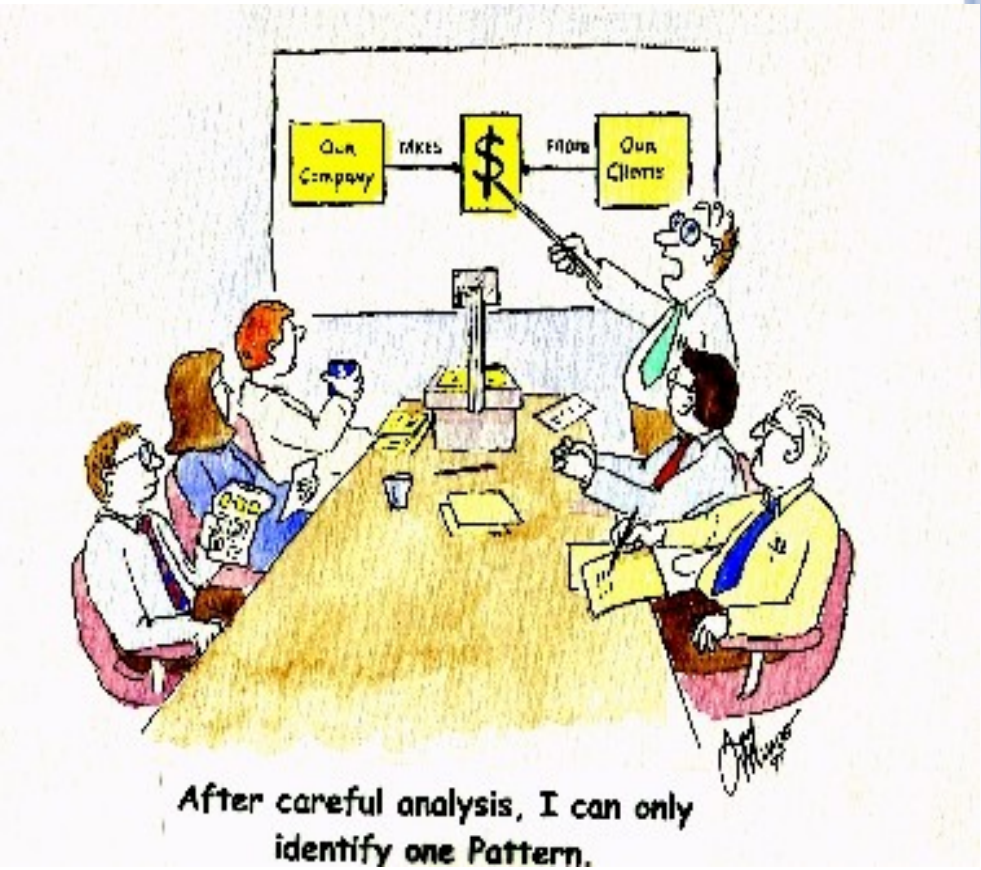
What is a Software Design Pattern?

Software Design Pattern

- “A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context.”
- *Design Patterns: Elements of Reusable Object-Oriented Software*

SW Design Pattern

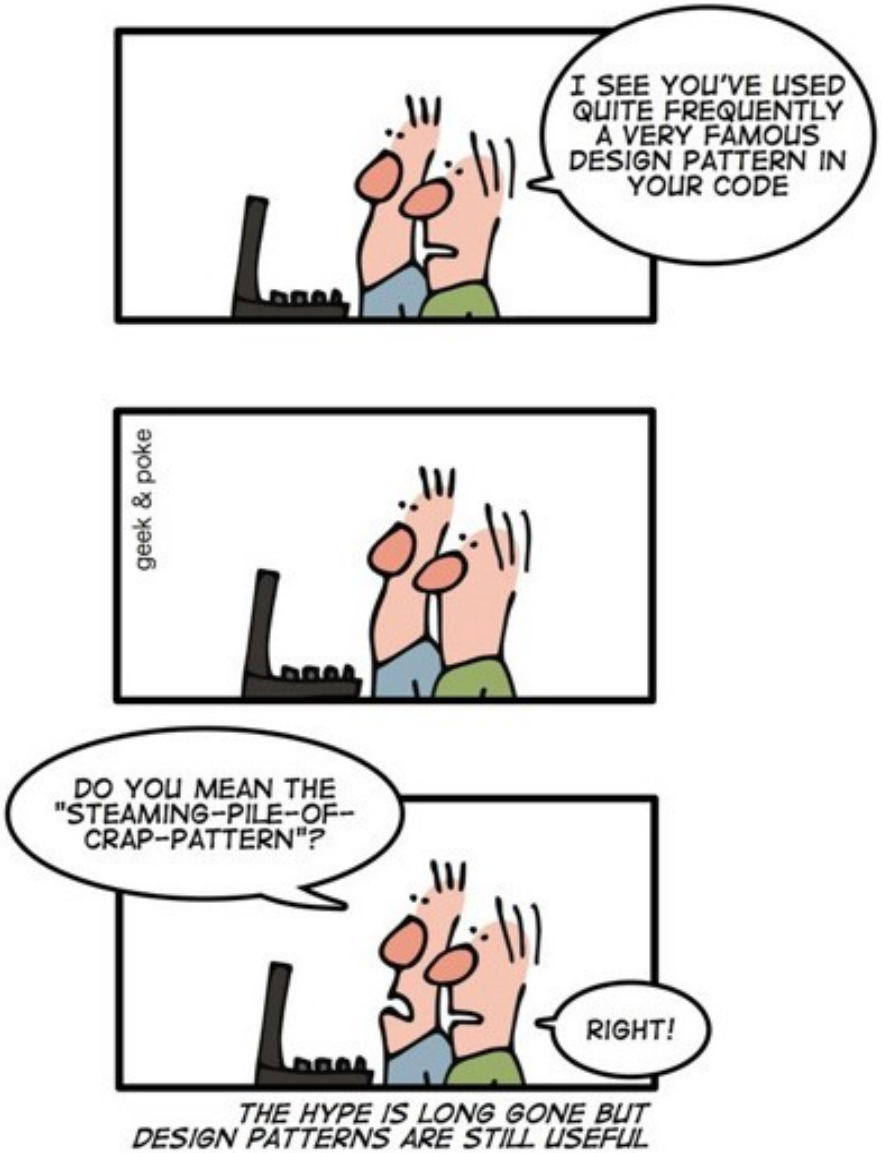
- “How can you distribute responsibility for design through all levels of a large hierarchy, while still maintaining consistency and harmony of overall design?”
- *A Pattern Language* by Christopher Alexander



Software Design Patterns: Types

Types of Software Design Patterns

- Creational
  - How will you create objects?
- Structural
  - How will you realize relationships between objects?
- Behavioral
  - How will your objects interact?
- Concurrency
  - How to manage multiple threads?



Examples

- See handout
- Implement:
  - Factory pattern to create ImageReader objects
    - GIF Reader
    - JPEG Reader
  - Adapter pattern to treat adapt Interns to Employees (IEmployee interface)

Criticisms?

Software Design Pattern Criticisms

- “'The 'Design Patterns' solution is to turn the programmer into a fancy macro processor”
- “The design patterns may just be a sign of some missing features of a given programming language (Java or C++ for instance). 16 out of the 23 patterns in the Design Patterns book (that is primarily focused on C++) are simplified or eliminated (via direct language support) in Lisp or Dylan.”

Software Design Patterns

- Which are critical to YOUR application?
- What languages provide support for the given design pattern?
  - And is easy to use?
  - And that you already know?
  - Language is NOT a replacement for understanding
  - Java is not the answer to everything!
- What languages make it easy or possible to implement the design pattern?

Coding conventions: Why use 'em?

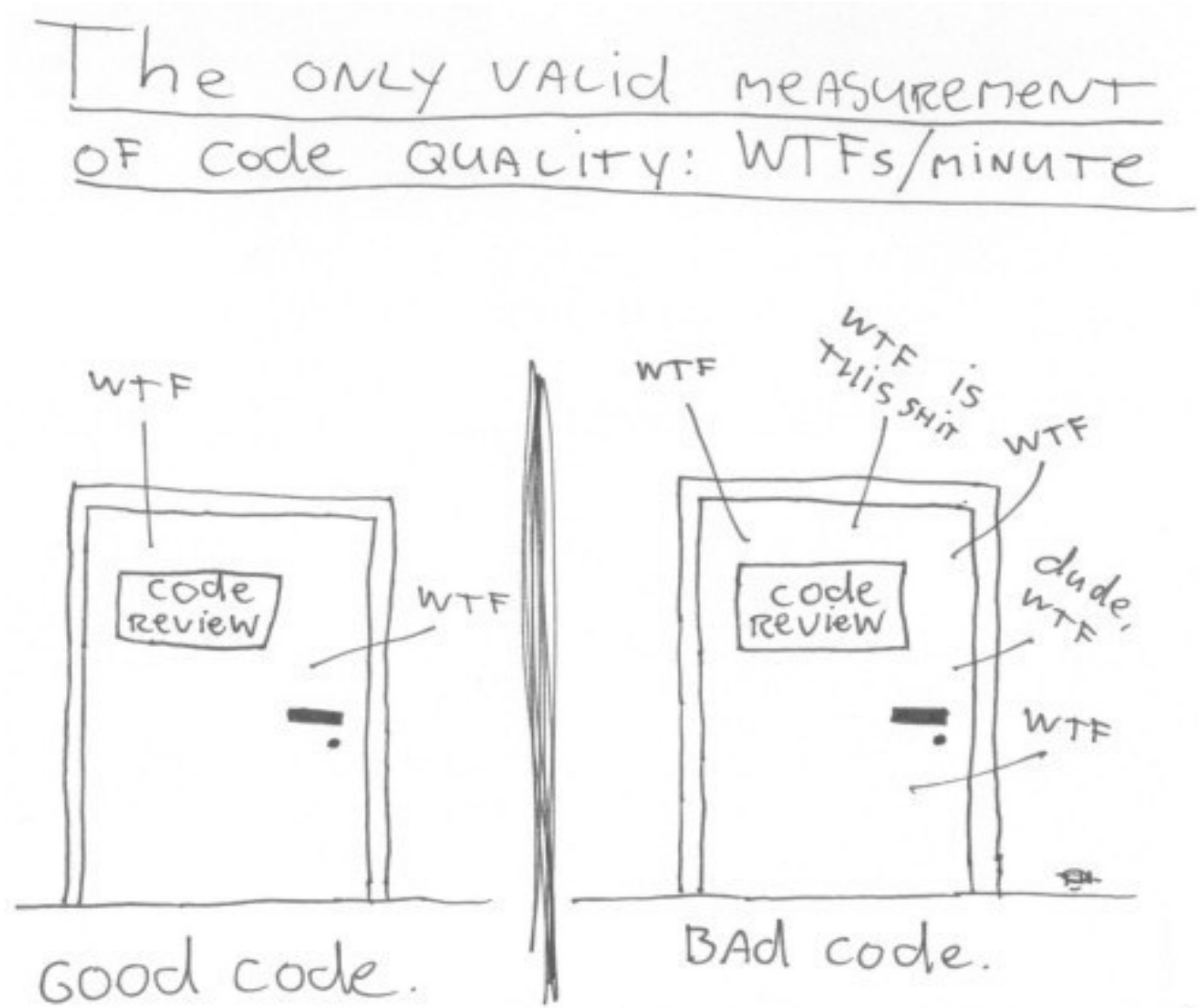
From Sun Microsystems

- Code conventions are important to programmers for a number of reasons:
  - 80% of the lifetime cost of a piece of software goes to maintenance.***
  - Hardly any software is maintained for its whole life by the original author.
  - Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  - If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create
- http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

Note***

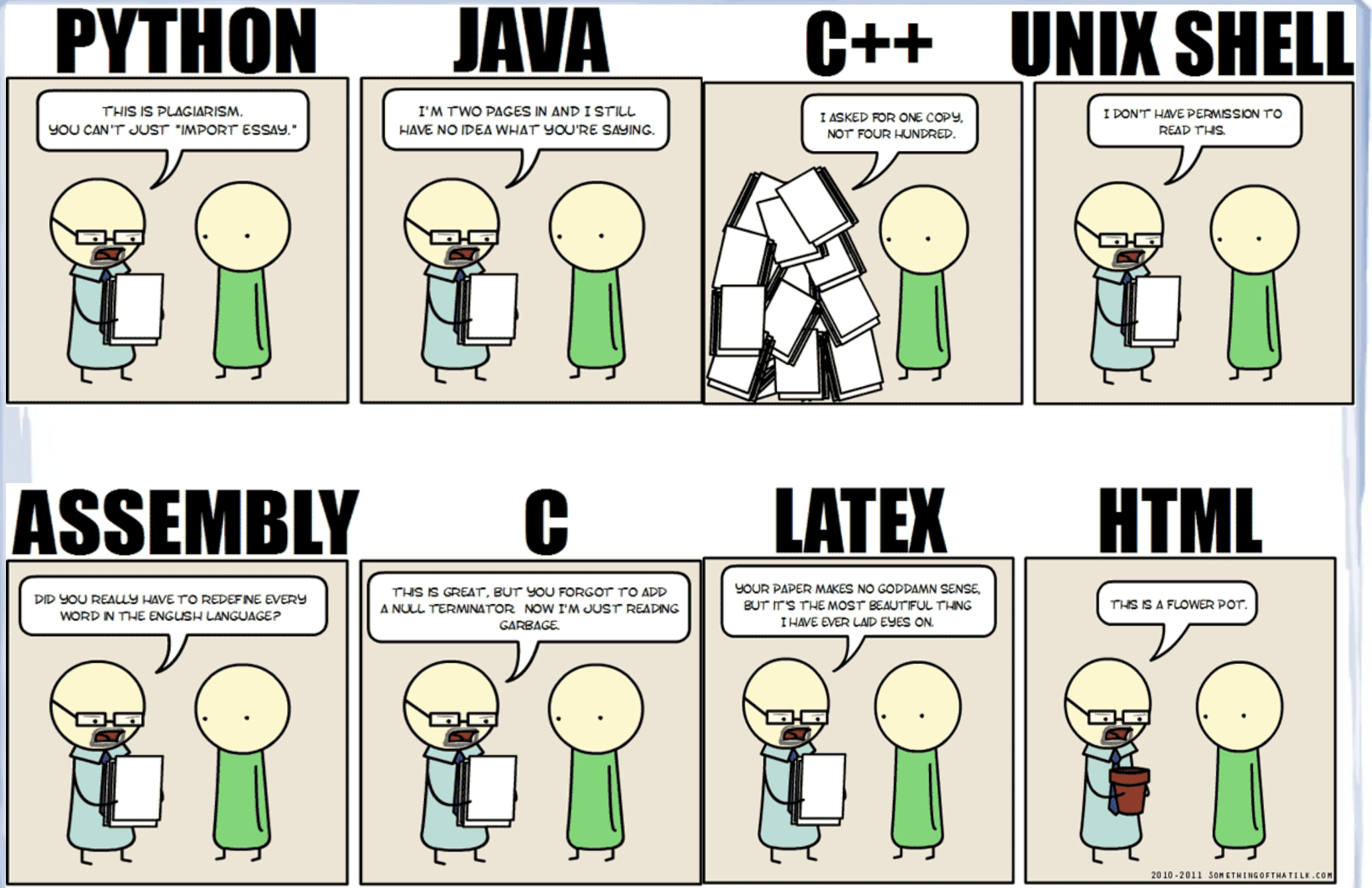
- “Maintenance typically consumes 40 to 80 percent of software costs. It is probably the most important software lifecycle phase.”
- From *Facts and Fallacies of Software Engineering*
- *http://www.codinghorror.com/blog/2008/03/revisiting-the-facts-and-fallacies-of-software-engineering.html*

Software Quality



Hmm, it seems that the language is important...

What makes languages different?
Everything is a Turing Machine so what does it matter?


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM

What language are you using? Why?

Language differences

- Available libraries
- “Speed”
- Size
- Pointers
- Garbage-collection*
- Standardization
- http://pesona.mmu.edu.my/~wruslan/SE1/Readings/detail/Reading-9.pdf
- Compiler tools
- Available workforce
- ***Conceptual simplicity & representation***
- ***Hardware optimization***

Recap

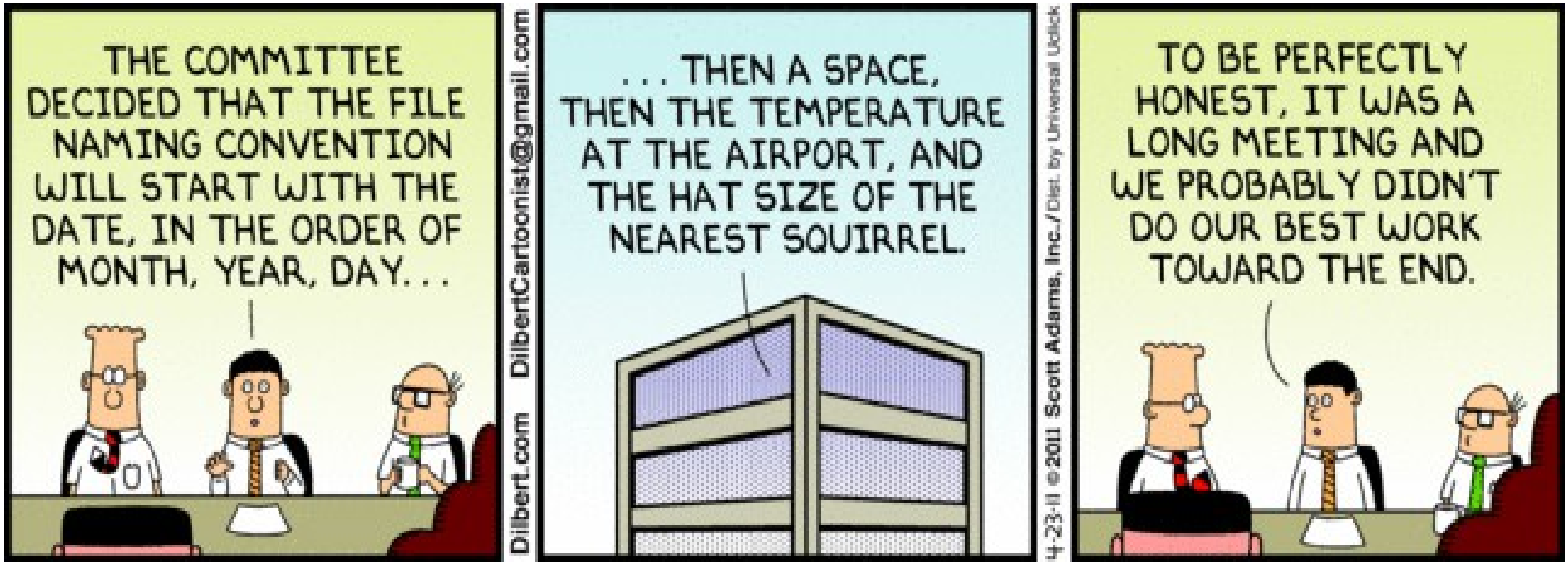
- Design pattern(s)
  - Which are critical to your application?
- Language
  - What is suited to your needs?
- Next...
  - Coding conventions

What are some types of conventions?

Types of conventions

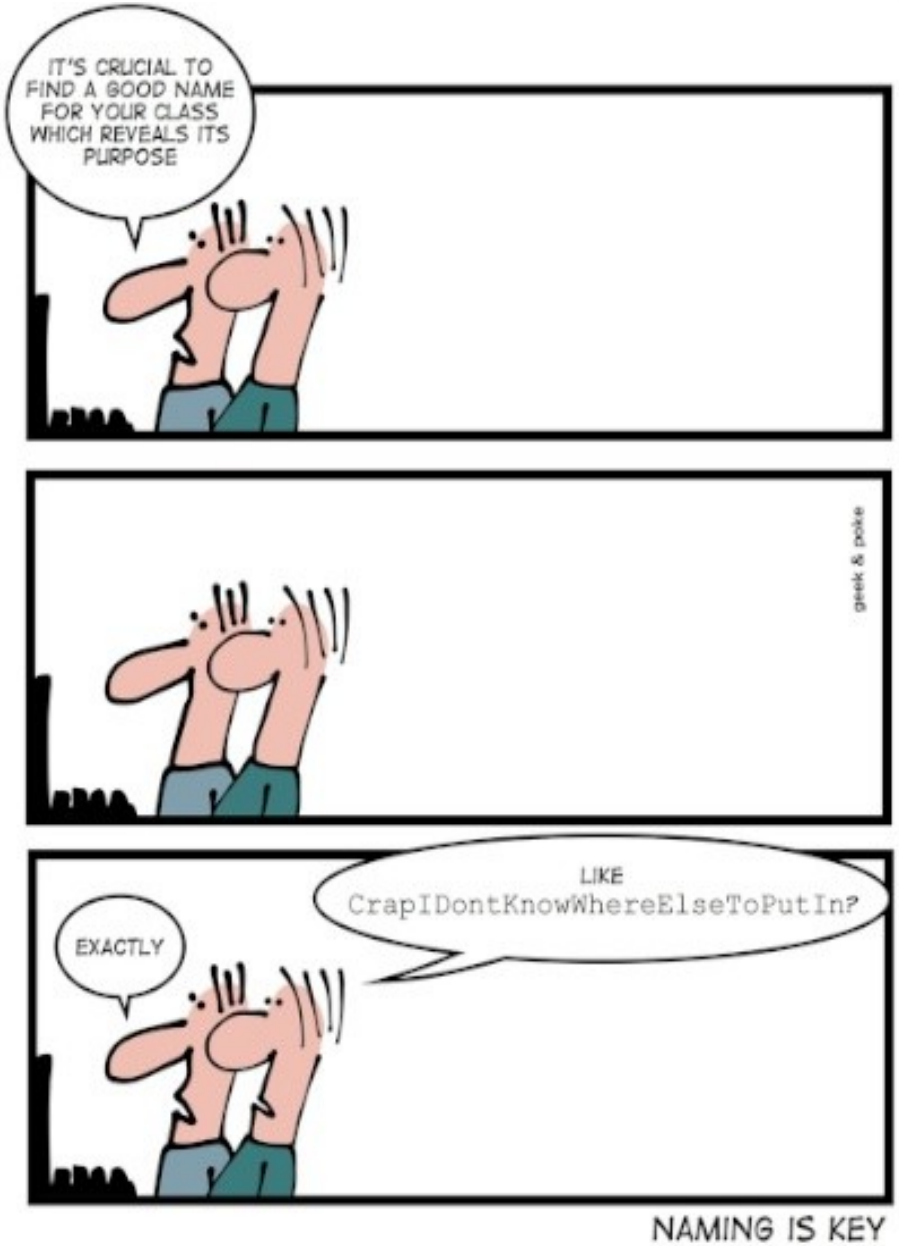
- Comments
- Indentation
- Symbol Naming (notation)
- File Naming
- Version system
- Coding
  - Statements-per-line
  - Single-return-per-function
  - Exception-handling

Why does notation matter?



Why notation matters

- Make the bad stuff stand-out
  - http://www.joelonsoftware.com/articles/Wrong.html
- Quicker understanding
- Semantic meaning
- Independent development (namespaces)
- Independent integration
- Independent testing
- Reduce cognitive overhead!



Example: Web application computing a function on user data and displaying data to the user

XSS anyone?

Some possibilities...

- Encode all strings right away
  - What about strings that go somewhere else?
    - Like a credit-card processor....
- Encode all strings when displayed to user
  - What if you need to write raw HTML?
  - That looks weird...
- Any ideas for the *right* solution?

Right solution

- Use notation to differentiate between encoded (i.e. “safe”) strings and non-encoded (i.e. “unsafe”) strings
  - sName vs. usName
- Examples:
  - usAddress = Request(“address”)
  - Write(usName)
  - sName = Encode(usName)
  - sName = usName

Notation: Items to consider

- Variable notation
- Class notation
- Function/Method notation
  - Including parameters & return types
- Namespaces
- Name length

What notation patterns have you noticed?

Some Notation Options

- CamelCase
- Mixed case
- Underscores
- Capitalization
  - All caps for constants?
- Prefixes
  - “m_” for member variables

Hungarian Notation?

Who knows what this is? What do you think?

Hungarian notation originally meant to convey semantic information about the variable NOT captured purely by its type.

Other Notation Choices (Examples)

- Verbosity:
  - SetScrollX vs. SetHorizontalScroll
- Capitalization
  - setScrollX vs. SetScrollX
- Underscores
  - SetScrollX vs. set_scroll_x
- Class Member indication
  - m_xPosition vs. xPosition vs. xPostion__

Recap

- Design pattern(s)
  - Which are critical to your application?
- Language
  - What is suited to your needs?
- Coding Conventions
  - What notation will you use?
- Next...
  - Other conventions!

File/Directory Structure: What to do?

Several “Best Practices”

- Maven Standard Directory Layout
  - http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
- Mozilla Source Code
  - https://developer.mozilla.org/en-US/docs/Source_code_directories_overview
- Microsoft
  - http://msdn.microsoft.com/en-us/library/bb668953.aspx

Directory Structure Themes

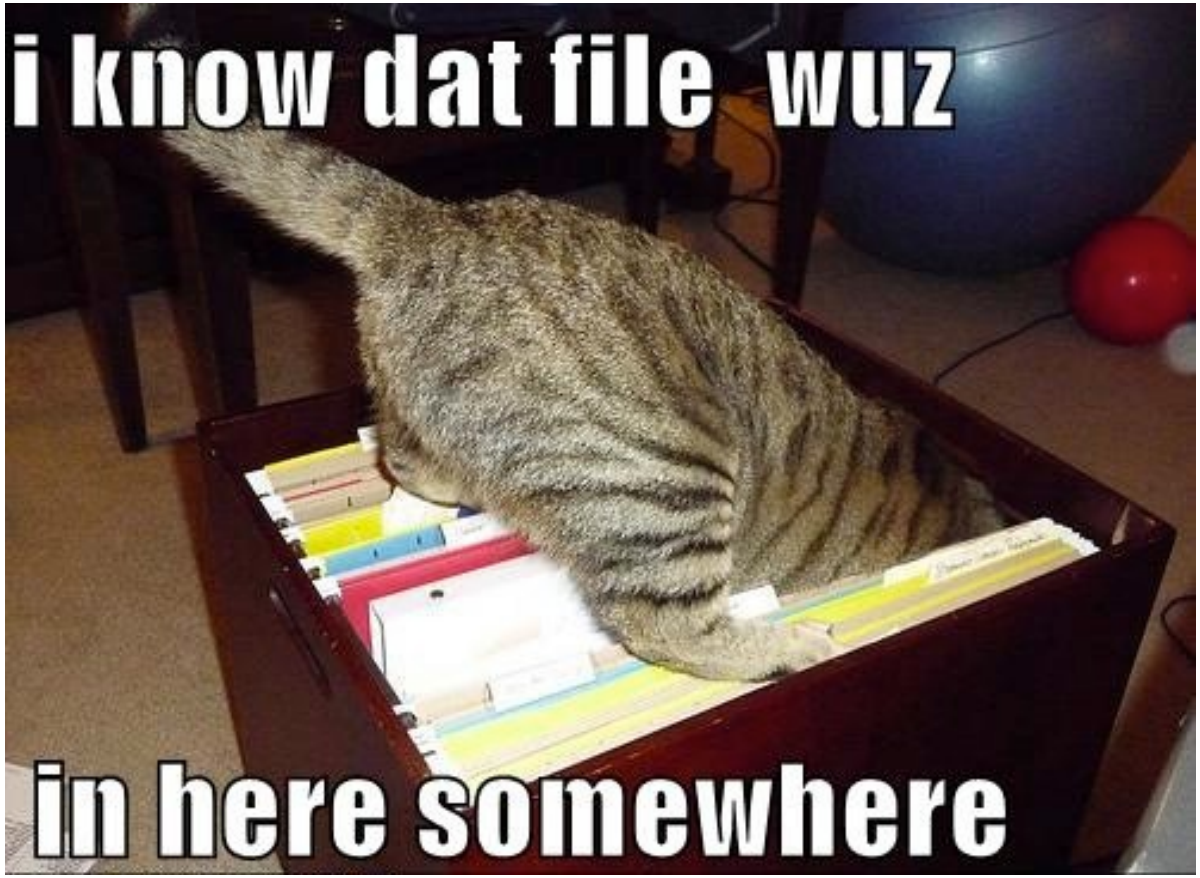
- Top-level: License, README, changelog, Makefile/build script
  - Then directories for source, include, libraries, documentation, external tools (e.g. build tools), resources (e.g. images)
- Separate source code, documentation, build scripts, configuration
- Group by file type (e.g. images, source code files, text files)
- Sub-divide on purpose/module – tests, components

Examples

- Apache HTTPD
  - http://svn.apache.org/viewvc/httpd/httpd/trunk/
- Android Privacy Guard
  - http://android-privacy-guard.googlecode.com/svn/trunk/
- Metasploit
  - https://github.com/rapid7/metasploit-framework
- Mozilla
  - http://mxr.mozilla.org/mozilla-central/source/

Directory Structure Goals

- Simple
- Well-defined
- Extensible
- Intuitive?



Where/How to store configuration information?

Configuration Storage

- **Good**
  - SQLite (via API)
  - YAML
  - Property List (.plist)
  - .properties
- **Bad**
  - XML
  - Serialized Java objects
  - Registry
  - JSON
  - INI file

YAML Example

```
public void testLoadFromStream() throws FileNotFoundException {
  InputStream input = new FileInputStream(
       new File("src/test/resources/reader/utf-8.txt"));
  Yaml yaml = new Yaml();
  Object data = yaml.load(input);
}
```

.properties example

```
Properties prop = new Properties();

ClassLoader loader =
   Thread.currentThread().getContextClassLoader();

InputStream stream =
   loader.getResourceAsStream("myProp.properties");

prop.load(stream);
```

Configuration Considerations

- Very application dependent
- What can't you save on the server side?
  - Server information
- What if don't have a server?
- Magic values in your code
- Aesthetic preferences
  - Layout, colors, widgets, favorites
- Behavioral preferences
  - Auto-open/auto-trust, security requirements, enabled extensions/plugins, number of threads, maximum memory usage, logging

Coding Conventions: What else to consider?

Code Conventions

- Single exit point
- Exception/Error handling
- Left-hand comparisons
- Control structures
- Use of braces
- Complex boolean expressions
- Code alignment
- “Magic numbers”
- Lists (trailing comma)
- Meaning vs. consistency

  'i' for array index
- Commenting & Documentation
- Logging

What is your strategy for handling errors?

Primary Requirements

- Application survives,if possible
  - Graceful termination if not
- Relevant parties notified
  - Global unhandled exception handler?
- Reproduce/Recreate

Problems with Exceptions

- Exceptions impede understanding control flow
  - Like goto
- But not always bad...
- http://blogs.msdn.com/b/oldnewthing/archive/2005/01/14/352949.aspx

Bad code

```
BOOL ComputeChecksum(LPCTSTR pszFile, DWORD* pdwResult)
{
  HANDLE h = CreateFile(pszFile, GENERIC_READ, FILE_SHARE_READ,
      NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE hfm = CreateFileMapping(h, NULL, PAGE_READ, 0, 0, NULL);
  void *pv = MapViewOfFile(hfm, FILE_MAP_READ, 0, 0, 0);
  DWORD dwHeaderSum;
  CheckSumMappedFile(pvBase, GetFileSize(h, NULL),
          &dwHeaderSum, pdwResult);
  UnmapViewOfFile(pv);
  CloseHandle(hfm);
  CloseHandle(h);
  return TRUE;
}
```

Not-bad code

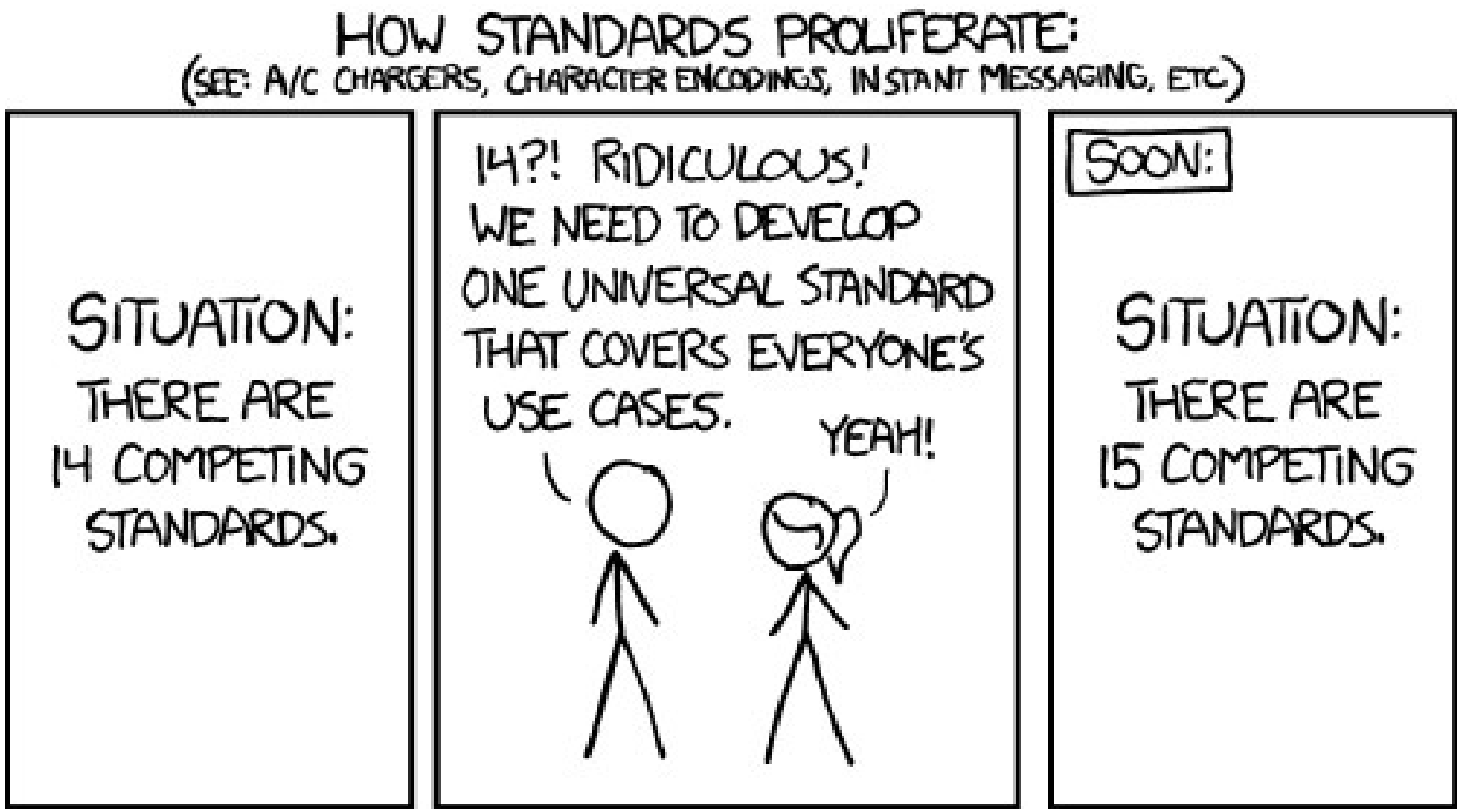
```
BOOL ComputeChecksum(LPCTSTR pszFile, DWORD* pdwResult)
{
  BOOL fRc = FALSE;
  HANDLE h = CreateFile(pszFile, GENERIC_READ, FILE_SHARE_READ,
      NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  if (h != INVALID_HANDLE_VALUE) {
    HANDLE hfm = CreateFileMapping(h, NULL, PAGE_READ, 0, 0, NULL);
    if (hfm) {
      void *pv = MapViewOfFile(hfm, FILE_MAP_READ, 0, 0, 0);
      if (pv) {
        DWORD dwHeaderSum;
        if (CheckSumMappedFile(pvBase, GetFileSize(h, NULL),
                               &dwHeaderSum, pdwResult)) {
          fRc = TRUE;
        }
        UnmapViewOfFile(pv);
      }
      CloseHandle(hfm);
    }
    CloseHandle(h);
  }
  return fRc;
}
```

Which is good?

```
NotifyIcon CreateNotifyIcon()
{
 NotifyIcon icon = new NotifyIcon();
 icon.Text = "Blah blah blah";
 icon.Visible = true;
 icon.Icon = new Icon(GetType(), "cool.ico");
 return icon;
}
```

```
NotifyIcon CreateNotifyIcon()
{
 NotifyIcon icon = new NotifyIcon();
 icon.Text = "Blah blah blah";
 icon.Icon = new Icon(GetType(), "cool.ico");
 icon.Visible = true;
 return icon;
}
```

Don't invent your own!



Coding Style Guides

- Windows (C/C++)
  - http://msdn.microsoft.com/en-us/library/windows/desktop/aa378932(v=vs.85).aspx
- Android (Java)
  - http://source.android.com/source/code-style.html
- Cocoa (Apple)
  - https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.html
- GNU Coding Standards
  - http://www.gnu.org/prep/standards/standards.html
- Google C++
  - http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Commenting

- Do it!!!!
- One practice:
  - Name of the module.
  - Purpose of the Module.
  - Description of the Module (In brief).
  - Original Author
  - Modifications
  - Authors who modified code with a description on why it was modified.

Classics

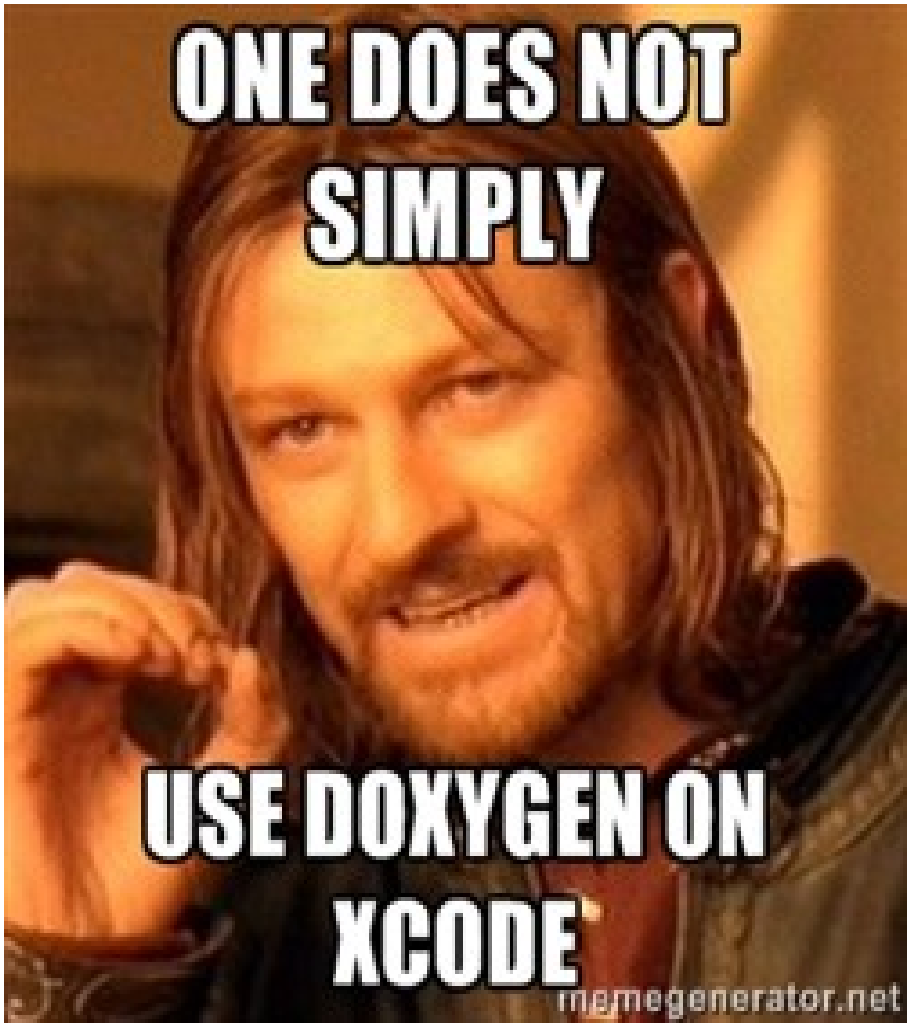
```
//
// Dear maintainer:
//
// Once you are done trying to 'optimize' this routine,
// and have realized what a terrible mistake that was,
// please increment the following counter as a warning
// to the next guy:
//
// total_hours_wasted_here = 42
//
```

Classics

```
/**
* For the brave souls who get this far: You are the chosen ones,
* the valiant knights of programming who toil away, without rest,
* fixing our most awful code. To you, true saviors, kings of men,
* I say this: never gonna give you up, never gonna let you down,
* never gonna run around and desert you. Never gonna make you cry,
* never gonna say goodbye. Never gonna tell a lie and hurt you.
*/
```

Auto-documentation

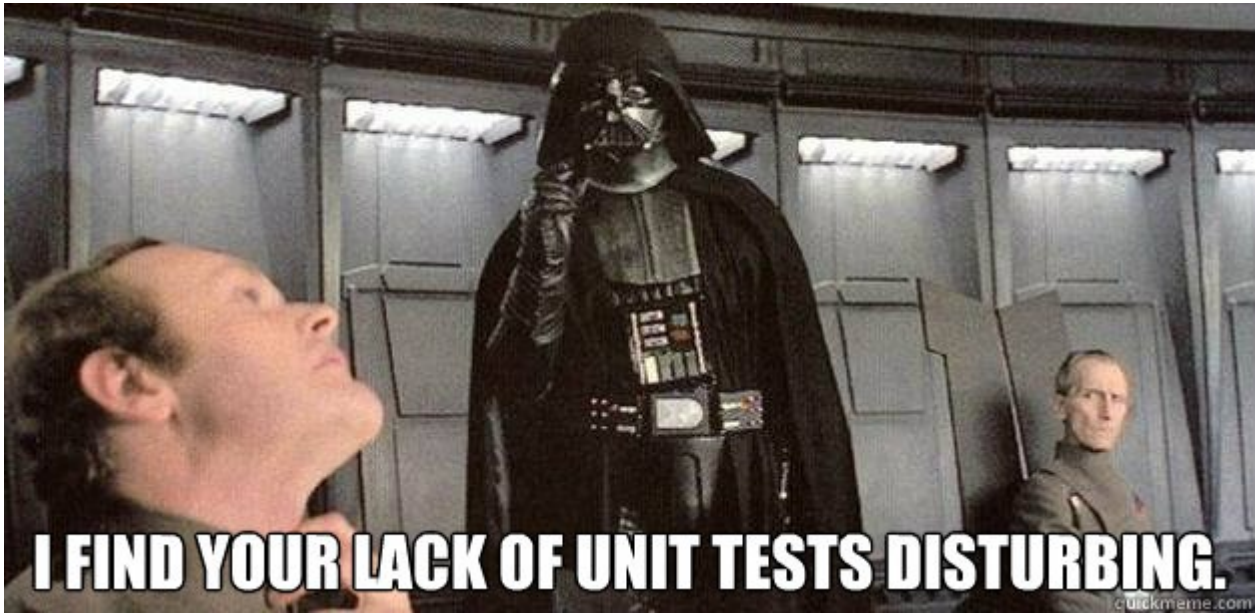
- Doxygen
- Javadocs
- Perldoc
- Epydoc



What's even better than comments?

Unit Tests!!!

- Shows how a module/component is used
  - CppUnit
  - PyUnit
  - JMock
  - JUnit



Unit Test Considerations

- Boundary conditions
  - Null strings
  - Empty strings
  - Null pointers
  - 0
  - 2^32
  - Timeout
  - “Leaky Abstractions”

JUnit Assertions...

- assertTrue
- assertFalse
- assertArrayEquals
- assertNull
- assertNotNull
- assertNotSame
- assertThat
- fail

Unit Tests

Calculator Example

Unit Test Exercise

- http://pastebin.com/HYjVVDFy
- Triangle Class
- Test for type
  - Scalene
  - Isosceles
  - Equilateral
  - Too big

What about dependencies?

- You object might rely on other objects
- How to test independently?
- Mock objects!
  - Mockito
  - EasyMock

EasyMock Example: Portfolio of stocks, value is from a StockMarket object. Don't want to bang against an actual service in testing, so make a mock StockMarket object

Unit Test Concerns

- Write Test first?
  - Test-Driven Development (TDD)
- Don't go crazy
  - A 50% solution in the hands of the customer is better than the 99% solution that's only on your development box
- Keep in mind what is critical
  - “Shipping your code is a feature. An important feature. Your code must have it.”

Recap

- Design pattern(s)
- Language
- Coding conventions
  - Documentation
  - Unit testing
- Next: Integration

Use Integration Tools

- You have a revision control system, right?
- Frequent integration/Continuous integration
- Tools:
  - Hudson/Jenkins
  - Apache Continuum
  - Buildbot

Benefits of Continuous Integration?

Benefits

- Quick bug detection
  - Quickly revert to working version
- Avoid release-day chaos
- Encourages frequent commits – more modular code
- Faster feature integration

Drawbacks of continuous integration?

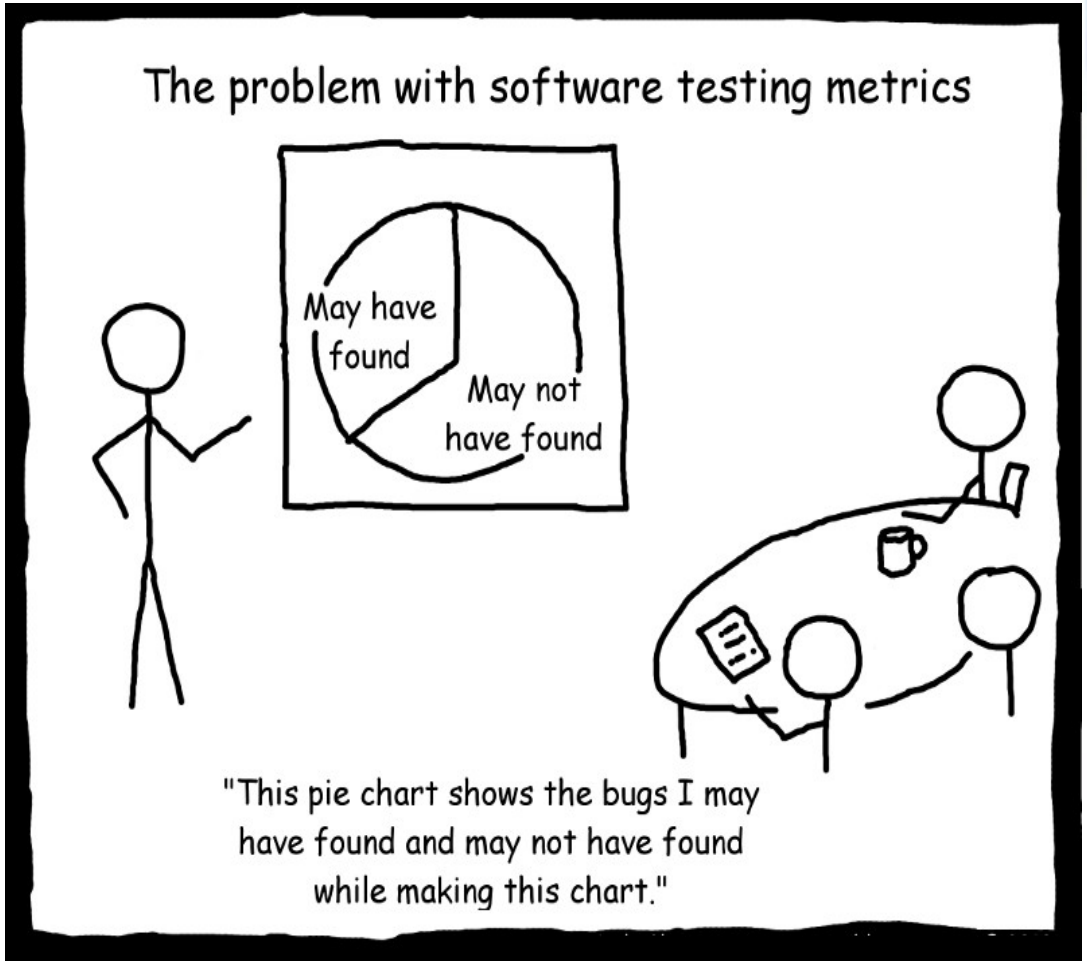
Recap

- Design pattern(s)
- Language
- Coding conventions
  - Documentation
  - Unit testing
- Integration testing
- Next: Dealing with bugs/issues/management...

How to manage bugs, feature requests, project management, etc...?

Task Tracking

- Integration with:
  - Version control system
  - Integration testing
  - Ticketing
- Examples:
  - JIRA (Not free)
  - Bugzilla
  - Trac
  - Redmine



Time Management: How long does it take you to code?

Time Management

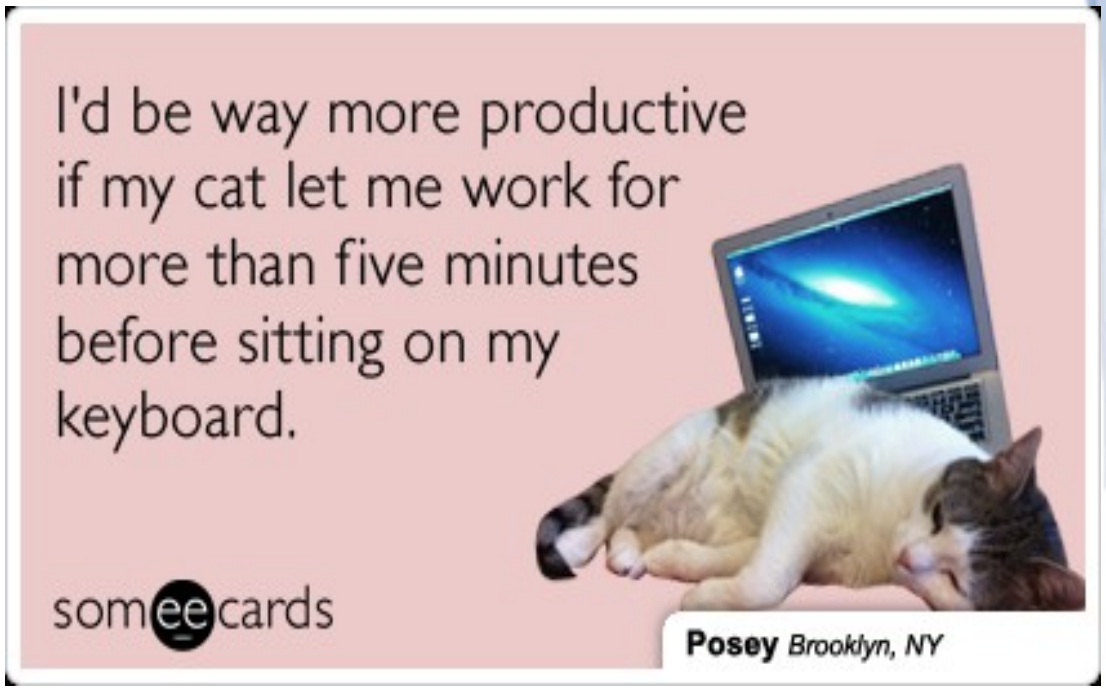
- Longer than you think!
  - Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
  - Planning Fallacy
- Not always a function of LOCs
- Rule of 3:
  - Multiply whatever you think it will take by 3
  - aka the P-factor (polish factor)

Divide & Conquer

- Create tasks from workflows
- Divide tasks into manageable subtasks
  - Aim for 4-6 hours (who can code for 8 hours straight??)
  - Include unit test development and execution
  - Time = P*(coding + testing + integration)
- Remember – you will get this wrong. :)
- Only YOU know your schedule
  - http://www.paulgraham.com/makersschedule.html

Environmental Factors

- Noise
- Interruptions
- Rabbit holes
  - Chasing minor problems
- Fighting your tools
- Meetings



Do we have any time left?

Start planning your project!!

Final Thoughts...

Top 10 Fallacies

- You can't manage what you can't measure.
- You can manage quality into a software product.
- Programming can and should be egoless.
- Tools and techniques: one size fits all.
- Software needs more methodologies.
- http://www.codinghorror.com/blog/2008/03/revisiting-the-facts-and-fallacies-of-software-engineering.html

Top 10 Fallacies

- To estimate cost and schedule, first estimate lines of code.
- Random test input is a good way to optimize testing.
- "Given enough eyeballs, all bugs are shallow".
- The way to preduct future maintenance costs and to make product replacement decisions is to look at past cost data.
- You teach people how to program by showing them how to write programs

Backup Slides

The most important thing?

“How do I gain that foundation of knowledge? I consume the available material and ask "why" a lot. If I look at a class hierarchy and it's design isn't immediately obvious to me, I ask why it was built that way and why was that way chosen over another. And to the answers to those questions, I keep asking why 'til I get to something I know already or until I get to a human reason. The reason for not stopping 'til I get to something I know is that I believe that **all learning is only as effective as well as it can be tied to what you already know**. How easily it is to learn something is directly related to how much you already know about related topics, so the more you know, the easier it is to learn more.”

Leaky Abstractions:
http://www.joelonsoftware.com/articles/LeakyAbstractions.html