Holistic Program Quality and Technical Debt

by Nathan Strutz
for CF.Objective() 2011

Are you bored yet?

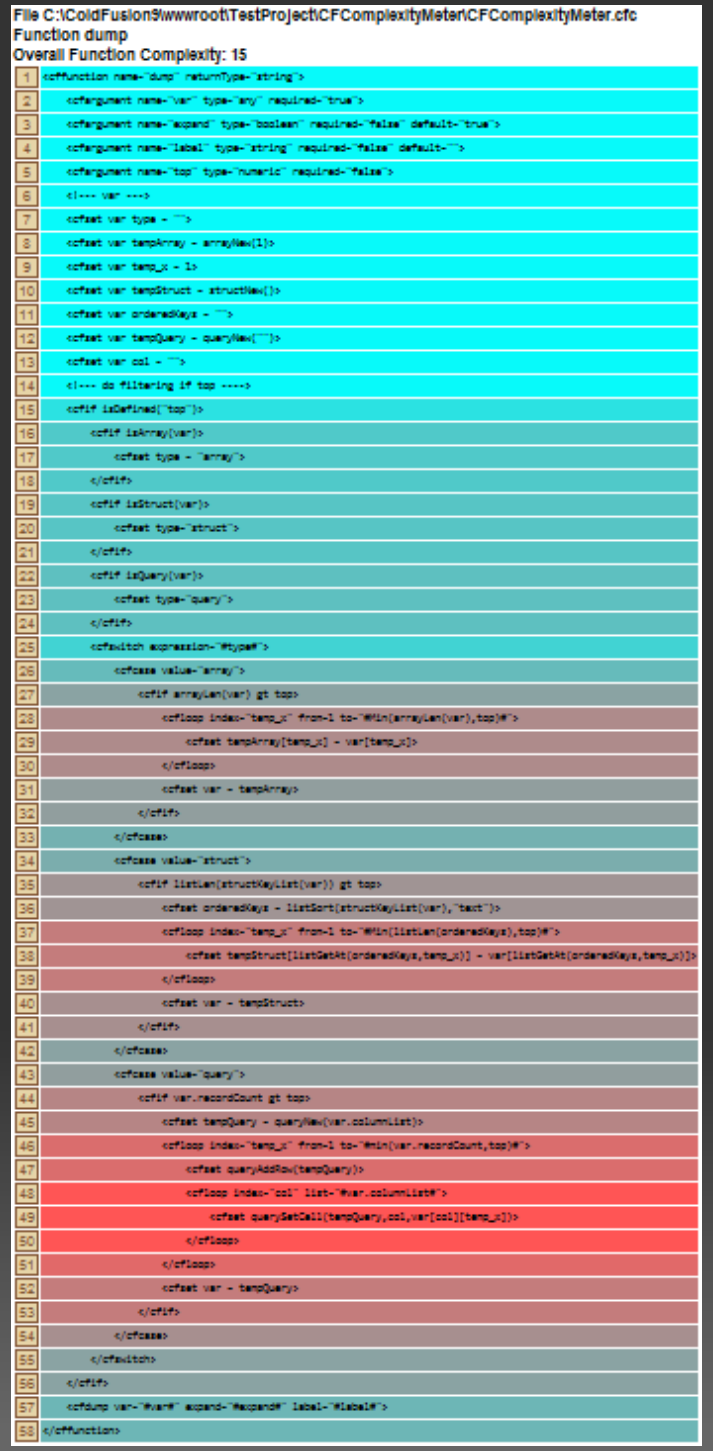
My day job...

Hackers, we have a problem

- The "Software Crisis" from the early days of computing, now 40+ years in
- Software projects are over budget, late, inefficient, don't meet requirements, code is unmaintainable and resulting software is just plain bad
- Sometimes, never delivered

Measuring software

- Speed & load times
- Lines of code
- Number of files, by type
- Size on disk
- Customer requirements
- Code Coverage
- Cyclomatic Complexity


File C:\ColdFusion9\wwwroot\TestProject\CFComplexityMeter\CFComplexityMeter.cfc
Function dump
Overall Function Complexity: 15

```
1  <cffunction name="dump" returnType="string">
2      <cfargument name="var" type="any" required="true">
3      <cfargument name="expand" type="boolean" required="false" default="true">
4      <cfargument name="label" type="string" required="false" default="">
5      <cfargument name="top" type="numeric" required="false">
6      <!--- var --->
7      <cfset var type = "">
8      <cfset var tempArray = arrayNew(1)>
9      <cfset var temp_x = 1>
10     <cfset var tempStruct = structNew()>
11     <cfset var orderedKeys = "">
12     <cfset var tempQuery = queryNew("")>
13     <cfset var col = "">
14     <!--- do filtering if top ---->
15     <cfif isDefined("top")>
16         <cfif isArray(var)>
17             <cfset type = "array">
18         </cfif>
19         <cfif isStruct(var)>
20             <cfset type="struct">
21         </cfif>
22         <cfif isQuery(var)>
23             <cfset type="query">
24         </cfif>
25         <cfswitch expression="#type#">
26             <cfcase value="array">
27                 <cfif arrayLen(var) gt top>
28                     <cfloop index="temp_x" from=1 to="#Min(arrayLen(var),top)#">
29                         <cfset tempArray[temp_x] = var[temp_x]>
30                     </cfloop>
31                     <cfset var = tempArray>
32                 </cfif>
33             </cfcase>
34             <cfcase value="struct">
35                 <cfif listLen(structKeyList(var)) gt top>
36                     <cfset orderedKeys = listSort(structKeyList(var),"text")>
37                     <cfloop index="temp_x" from=1 to="#Min(listLen(orderedKeys),top)#">
38                         <cfset tempStruct[listGetAt(orderedKeys,temp_x)] = var[listGetAt(orderedKeys,temp_x)]>
39                     </cfloop>
40                     <cfset var = tempStruct>
41                 </cfif>
42             </cfcase>
43             <cfcase value="query">
44                 <cfif var.recordCount gt top>
45                     <cfset tempQuery = queryNew(var.columnList)>
46                     <cfloop index="temp_x" from=1 to="#min(var.recordCount,top)#">
47                         <cfset queryAddRow(tempQuery)>
48                         <cfloop index="col" list="#var.columnList#">
49                             <cfset querySetCell(tempQuery,col,var[col][temp_x])>
50                         </cfloop>
51                     </cfloop>
52                     <cfset var = tempQuery>
53                 </cfif>
54             </cfcase>
55         </cfswitch>
56     </cfif>
57     <cfdump var="#var#" expand="#expand#" label="#label#">
58 </cffunction>
```

Programming is hard

- "Software is not governed by Moore's Law; more like Murphy's Law."
  - Douglas Crockford
- "Every time I think I am making progress, I come to realize that I only have more confusion over the 'right' way to implement something."
  - Ben Nadel
- "OO Almost Destroyed My Business!"
  - Marc Funaro

What is quality?

- There's no real, single, conclusive definition
- It's like chasing wind
- It's like herding cats
- It's like trying to slam a revolving door
- It's like being the best at make-believe
- Hmmmmm, the best at make-believe?

... the best at make believe

What is quality?

- How do you measure quality? How do you define it?
- Completeness, Conciseness, Consistency, Debugability, Efficiency, Extensibility, Maintainability, Portability, Reliability, Security, Structuredness, Testability, Understandability and Usability to name a few
- Quality has to cover the entire system, end to end

You need to travel all paths at once

This is your job, this is your life

- You aren't paid to take up space and write poor software
- Doing it right will pay your salary
- Doing it right will be good for your team
- Your code is the representation of yourself to your team
  - If you are virtual, your code is your physical self
- Doing it right will pay yourself back

So, What is Quality?

Completeness

Does it do everything it needs to do?

Conciseness

Does it do only what it needs to?

Consistency

Is it predictable in UI and in API?

Debugability

Is it easy to fix complicated things?

Efficiency

Does it feel like it's quick enough?

Extensibility

Is it architected well enough to grow?

Maintainability

Is it easy to find and fix things?

Portability

Is it easy to switch client software?
Is it easy to move servers?

Reliability

Is it available?

Security

Really good security programming
is really good programming

Security Detour

photo by Alex C. Balla, flickr.com/bluepoint951

Security detour

- Mediocre security is the same as insecurity!
- Security == Simplicity
- Bugs cause security problems.
  Complexity causes bugs.
  Simplicity removes bugs.
- But software is still complex

Security detour, owasp.org ten for 2010

http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

- Injection
- Cross-Site Scripting (XSS)
- Broken Authentication and Session Management
- Insecure Direct Object References
- Cross-Site Request Forgery (CSRF)
- Security Misconfiguration
- Insecure Cryptographic Storage
- Failure to Restrict URL Access
- Insufficient Transport Layer Protection
- Unvalidated Redirects and Forwards

Structuredness

Is it organized in an obvious way?

Testability

Is it easy to add tests to?

Understandability

Do you know what the system does?
Do you know how it does it?

Photo by Doc Searls, flickr.com/docsearls

I'd like to talk to you about your debt

Photo by Wally Gobetz, flickr.com/wallyg

Ward Cunningham said it best

- Because he said it first (1992)

> Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object- oriented or otherwise

Time == Money

$$V(t;T) = \int_t^T f(u)b(t;u)\,du.$$

your time is an investment
your knowledge is an investment
your work is an investment

What Classifies as Technical Debt?

Intentional	**Unintentional**
Not enough time for planning	Poor planning, or no planning
Make something static when it should be dynamic	Discover later how application should have worked
Leave features unfinished	Keep adding features, never refactor
Legacy code that no one wants to maintain	Legacy code that no one can maintain

Everybody borrows a little

No one knows the end product from the first few lines

> "*...while you're programming, you are learning. It's often the case that it can take a year of programming on a project before you understand what the best design approach should have been.*" Martin Fowler

If those wrong turns aren't cleaned up, you have created some technical debt

Interest Payments

- Interest is paid every time you work in a system that has technical debt
- It's the time it takes to work around the string and duct tape
- Interest is charged on the top
- The amount of debt determines how bad your interest payments are
- In a bad system, you may be paying over 50% to interest

When do you have a problem?

On the surface

- Growing dislike for the system admitted by the developers
- Small bugs that never seem to get fixed
- A bad UI is a clue to a rotten core
- Don't meet requirements like UI guidelines or security
- Running old versions of a framework or application server
- Lack of active developer knowledge
- Unused features

When do you have a problem?

In the code

- Lots of TODO and FIXME style comments
- Code that can't be refactored
- Code is too sloppy to make sense of
- Poor variable naming
- Files over 500 lines of code, functions over 200 lines
- CFCs / classes with over 50 functions
- Objects not well encapsulated
- No objects / CFCs whatsoever
- No intelligible structure
- No formal tests, no automated testing
- No automated build process
- No software releases for 6 months
- OOP done wrong


NO
SWIMMING
BEYOND THIS POINT.

When are you in over your head?

- Deep contempt for the system admitted by developers
- All knowledgeable developers have disavowed and moved to remote locations
- Frameworks or engines 10+ years old
- No software releases in 5 years
- Bug fixes take months
- Lots of "Sorry", "OMG", "WTF" and "$%#&*!!" comments
- Analysis revealing pure spaghetti code

Declaring bankruptcy

Declaring bankruptcy

- Rewrite. Throw it all away and start over
- Consider losing an hour's worth of work
- Consider a year or a decade's worth
- You may create new problems, new kinds of bugs, and features *will* be lost
- Scary!

Declaring bankruptcy

Declaring bankruptcy

- You can write better software in less time, now that you know what the system does and how it works
- Do it right this time
  - Best practices
  - Refactorable code
  - Go OO
  - Build for testability

A personal story

Avoiding technical debt

How do you avoid technical debt?

- Two perspectives
- Micro, Easiest win: Make your code readable!
  - Indentation
  - Commenting
  - Naming
- Macro
  - Architecture
  - Planning
  - Loose Coupling

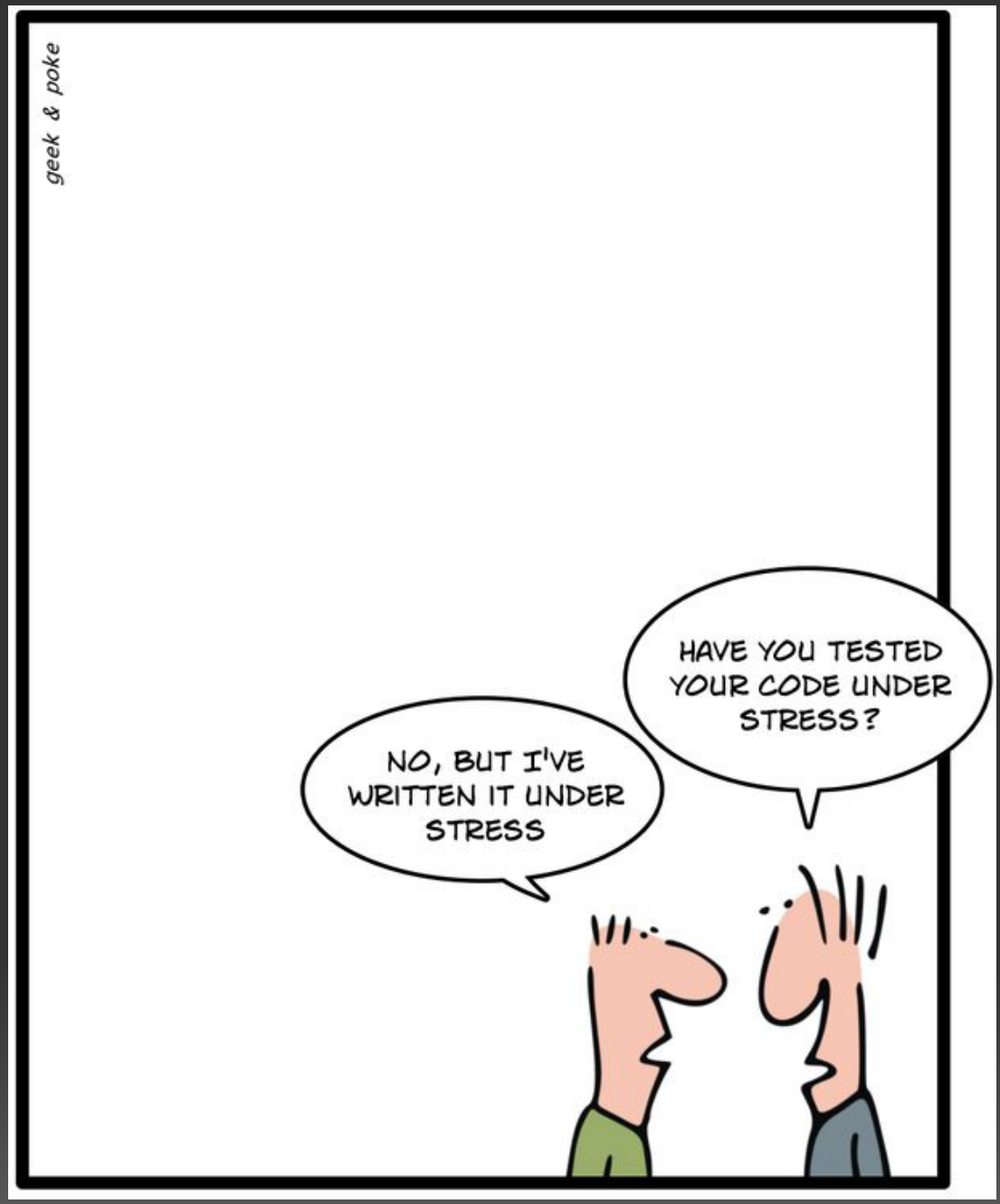
Separate layers, keep them that way

- Javascript in HTML
- HTML in Javascript
- CSS in Javascript
- CSS in HTML
- Javascript in CSS
- Model-View-Controller

Write less

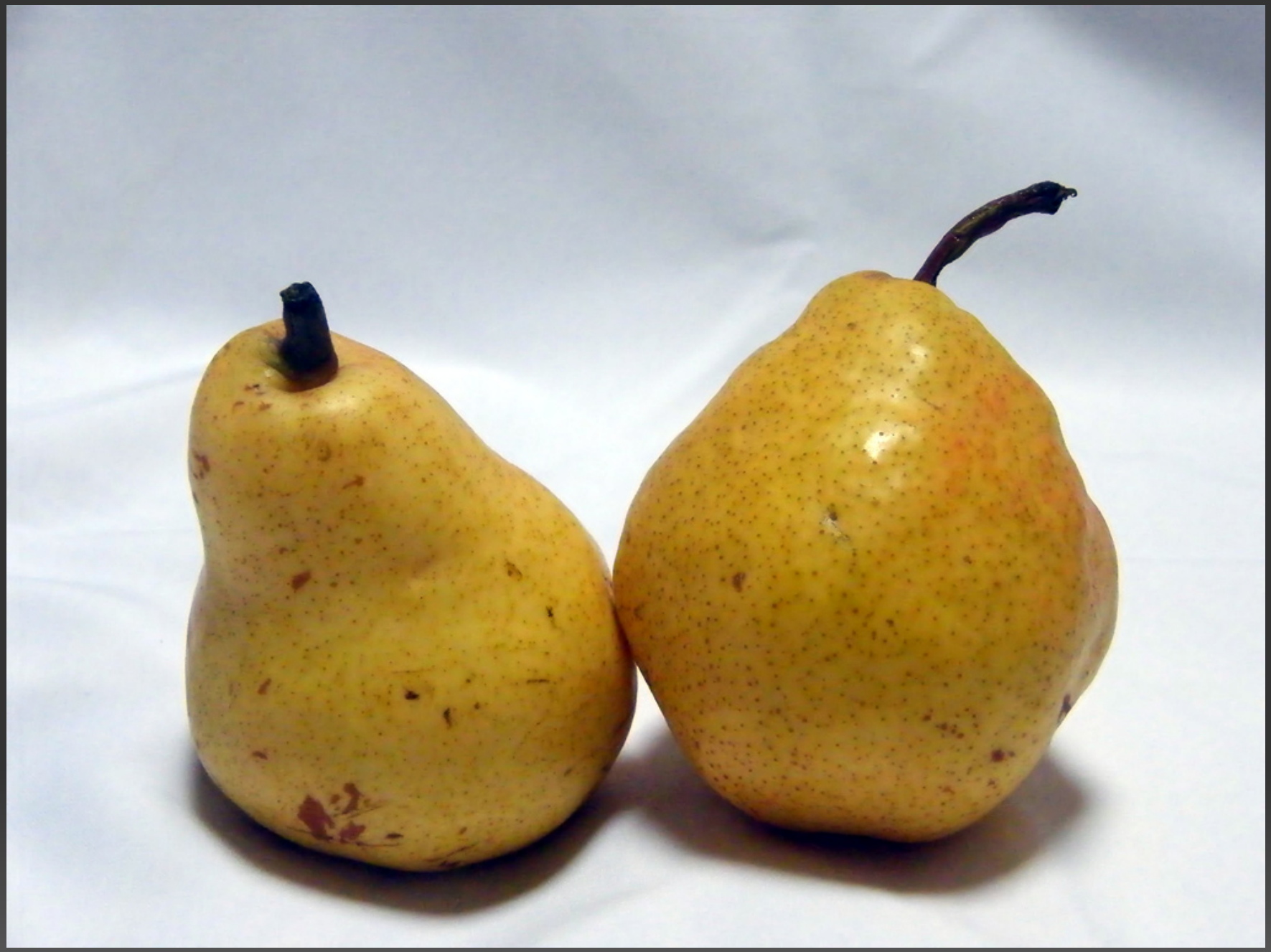
Features Have Cost

- Development time
  - obvious
- Deployment
  - always longer than you think
- Maintenance
  - more features means more to maintain
  - more bloat
- Load cost
  - downloading time, bandwidth, CPU & memory usage
- User confusion
- Training
- Potential to introduce bugs



Test around

Image thanks to geekandpoke.typepad.com



Increase knowledgeable staff

Refactoring as a way of life

Junk stuff

Aggressive code management


Document by automation

Make debt visible

Photo by James Jordan, flickr.com/jamesjordan

Conclusion

- The problems with software quality can be overcome, but it is a long, hard road
- It's your job to write good software
- Secure programming is good programming
- Take calculated technical debt risks
- It can take years to find the right approach
- Watch for the warning signs
- Make your code readable

Thanks

Bob, Marc & Emily.

You.

Nathan Strutz
strutz@gmail.com
@nathanstrutz
facebook, flickr, etc., etc.
www.dopefly.com