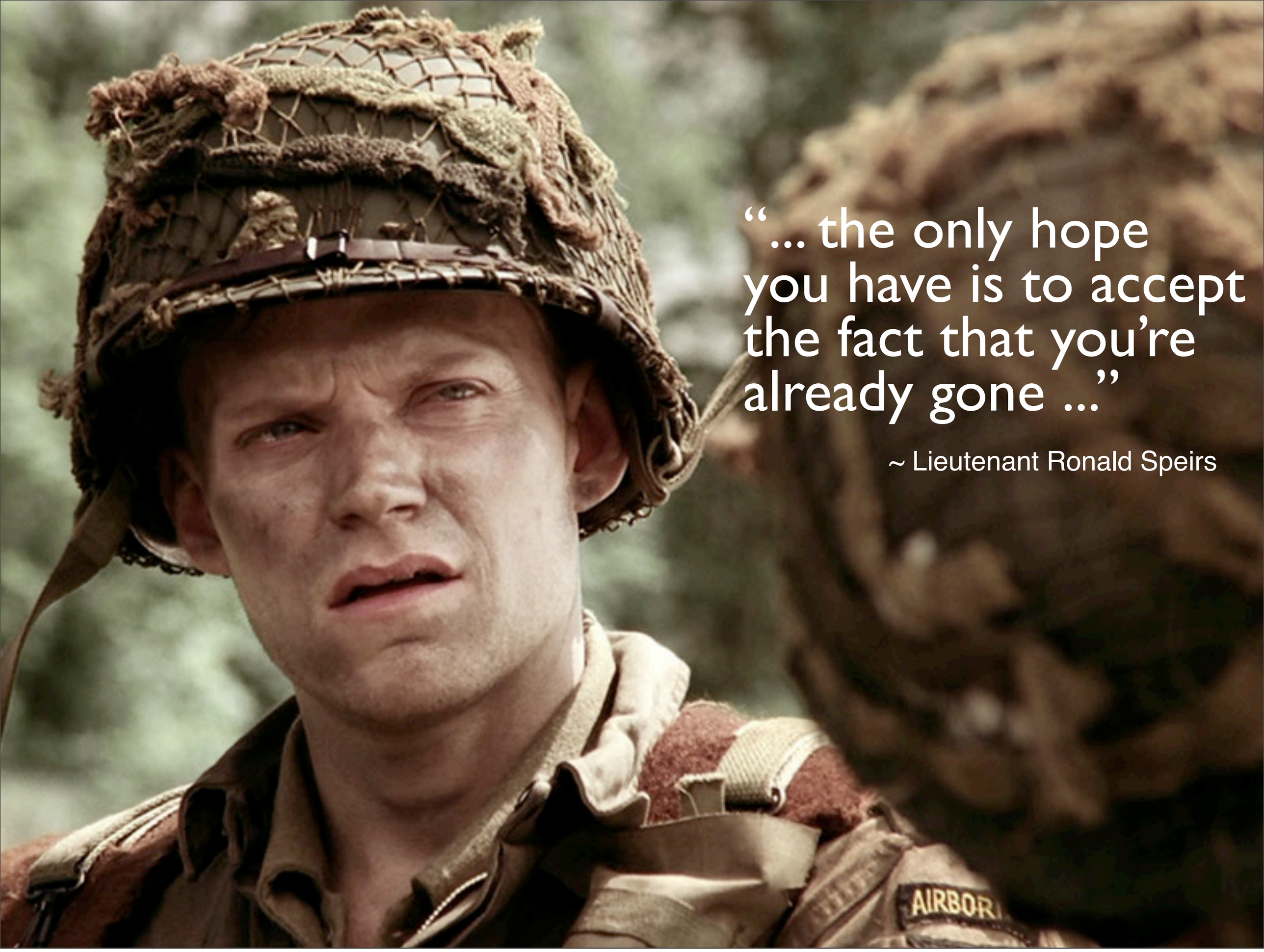
Lessons Learned in a Continuously Developing Service Oriented Architecture



Before we begin, I want to get serious for a moment.

[Band of Brothers “Accept that you’re already gone”]

Now, nobody’s dying in software development and I wish more folks realized this but...

The first lesson we learned is to accept the fact that our first attempt at new problem areas or domains will ALWAYS be naive to an extent and will never be the best we’re capable of. As soon as we accept that, we can go on to function as a developer should.

You make the best decisions you can with the information you have at the time and you leave yourself open to make improvements later.

Current Environment

Start to describe the current organizational environment at NC State.

Gives a frame of reference for where we were.

No Monolithic IT Organization

We have a central IT unit on campus that is responsible for core services such as email, ERP, human resources, student information systems, etc.

So you'll find pods of developers scattered throughout our University that may be in complete isolation or possibly lightly working with other groups.

The Problem with Small Teams

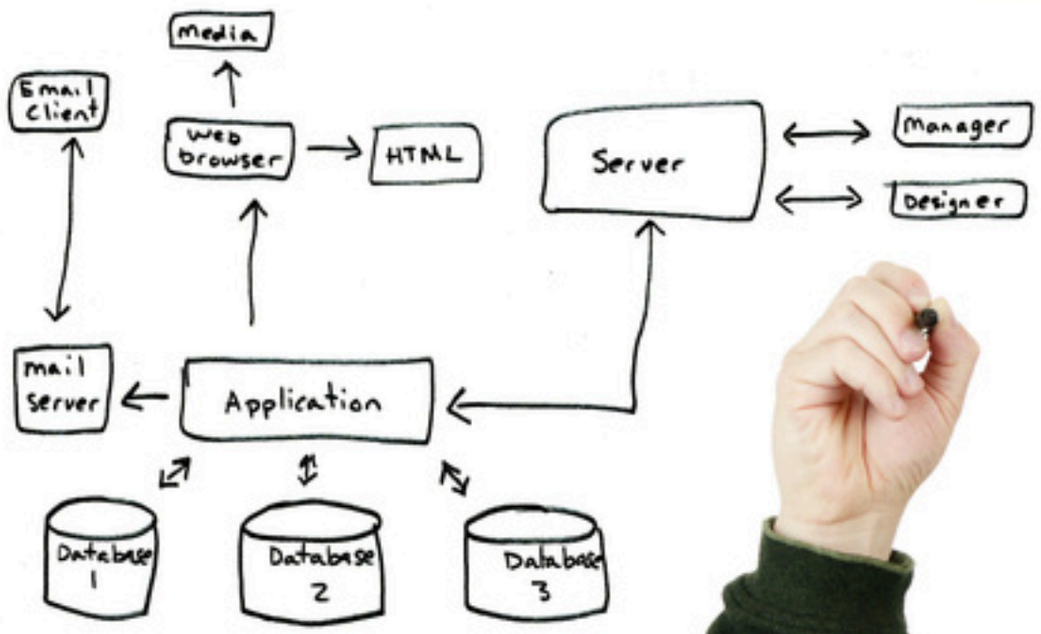
Small teams have a tendency towards a few negative behaviors and are sometimes in a bad spot when it comes to retaining talent and hiring new talent.

Group Skill-set

Teams that are completely decentralized need core competency in every employee and sometimes this doesn’t allow for the most efficient / optimal skillset spread.

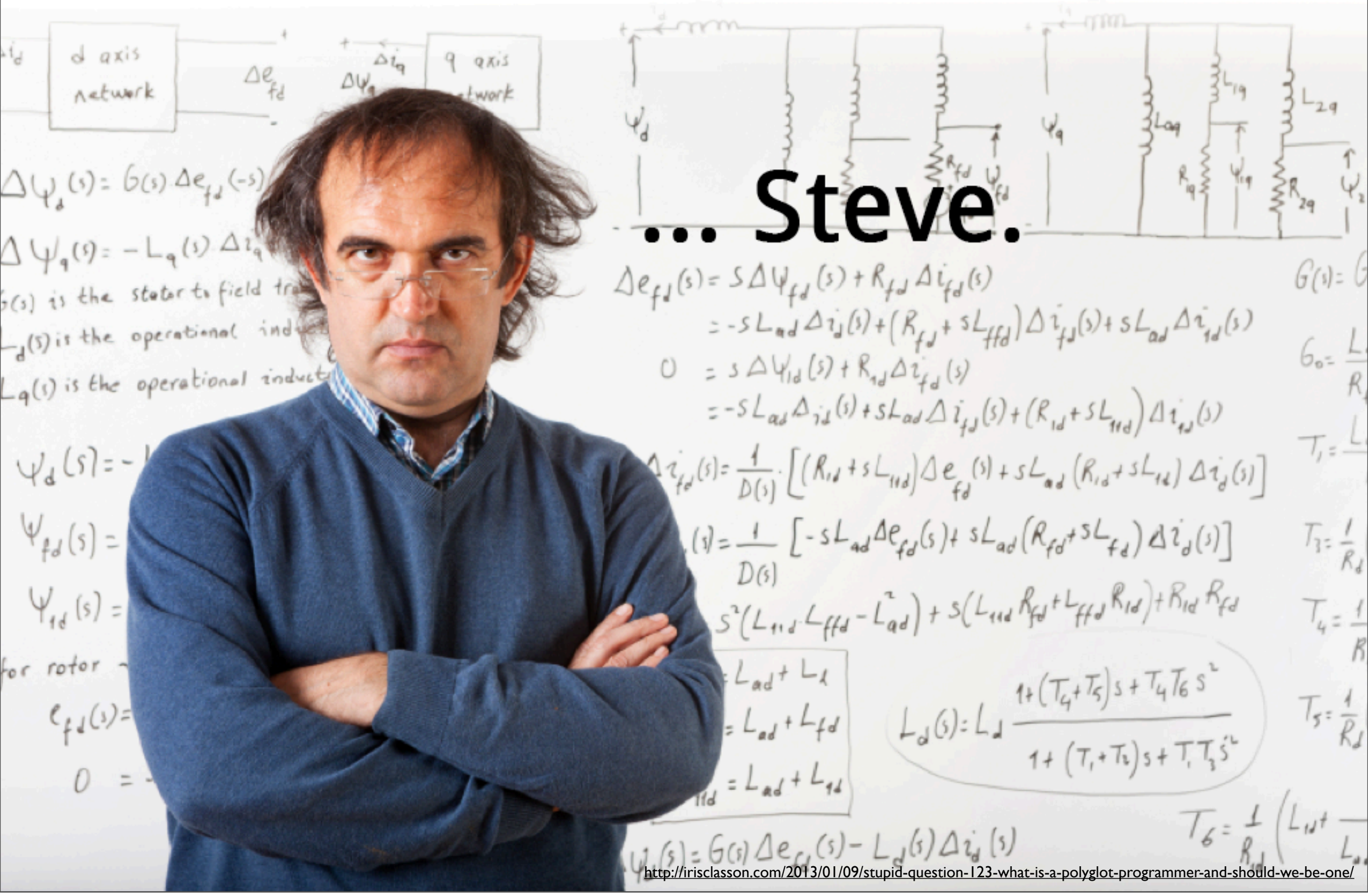
A lot of times, there is a real risk to end up with a group of “jacks/jills of all trades”

Alex



http://employeerightsguide.com/computer-programmers-job-description-salary-information-and-more/

Alex has just joined our mock group and he has ideas. Oh how he has ideas.



http://irisclasson.com/2013/01/09/stupid-question-123-what-is-a-polyglot-programmer-and-should-we-be-one/

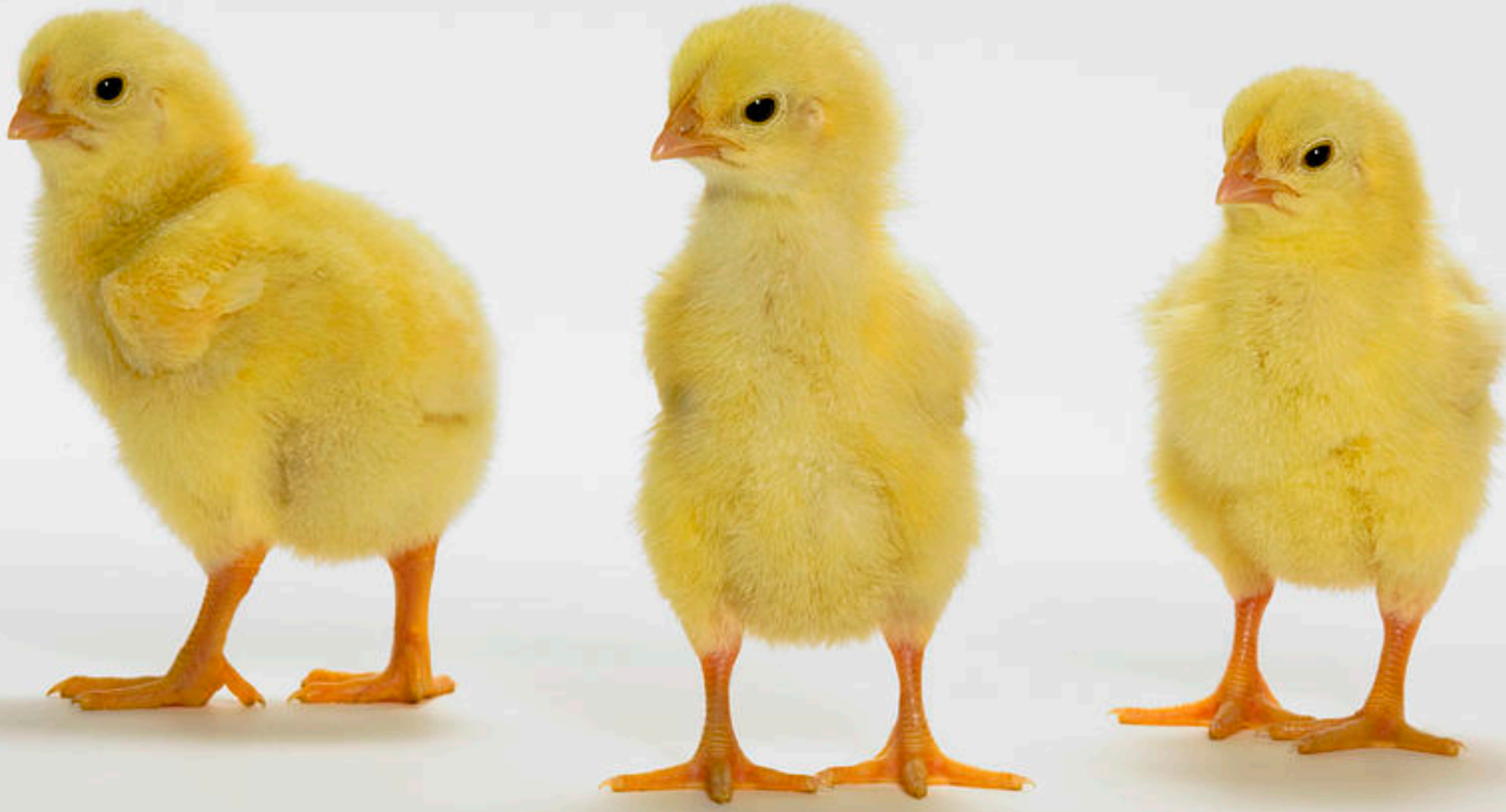
Steve... has been programming since before Alex was born. Steve laughs at Alex's youthful enthusiasm and remembers himself in Alex's shoes years ago.

http://faildesk.net/2012/08/24/it-horror-stories-scumbag-programmers/

Alex is replacing this guy. Bye bye guy.

The REAL danger with this guy is that he can talk the talk with management. This particular person KNOWS that he's writing bad code and that he's leaving nails for the next developer to step on.

The fact that he's fast and "gets the job done" makes him a stealth assassin for management and in the wake of his departure, Alex and Steve begin to smell the smells left behind.



Then, you have the baby chicks. They don't know what they're doing isn't the "best practice", but they have best intentions and you can't get upset with them because best intentions are exactly what you want in a group.

If you put baby chicks on a keyboard and expect Twitter to pop out, then it's not the chicks' problem when they fail.

`!isEnterprise()` **can lead to** `blinders.equip();`

Smaller pods of developers don't always have the long-term 10-yr plan to think about.

This may cause them to focus on the implementation at hand without a lot of forethought. This will scale to a point, relative to the amount of resources you have.

Focusing solely on day-to-day tasks can give a development team tunnel vision and causes them to miss the important breakthroughs; to miss the opportunities for greater insight.

http://postgradproblems.com/8-things-you-didnt-know-about-the-mighty-morphin-power-rangers/

Take for instance, the Power Rangers. You could not find a more motivated group. However, their downfall is that they only see 30 minutes into the future.

Every episode of the power rangers was the same...

Tuesday, November 12, 13

The show starts...

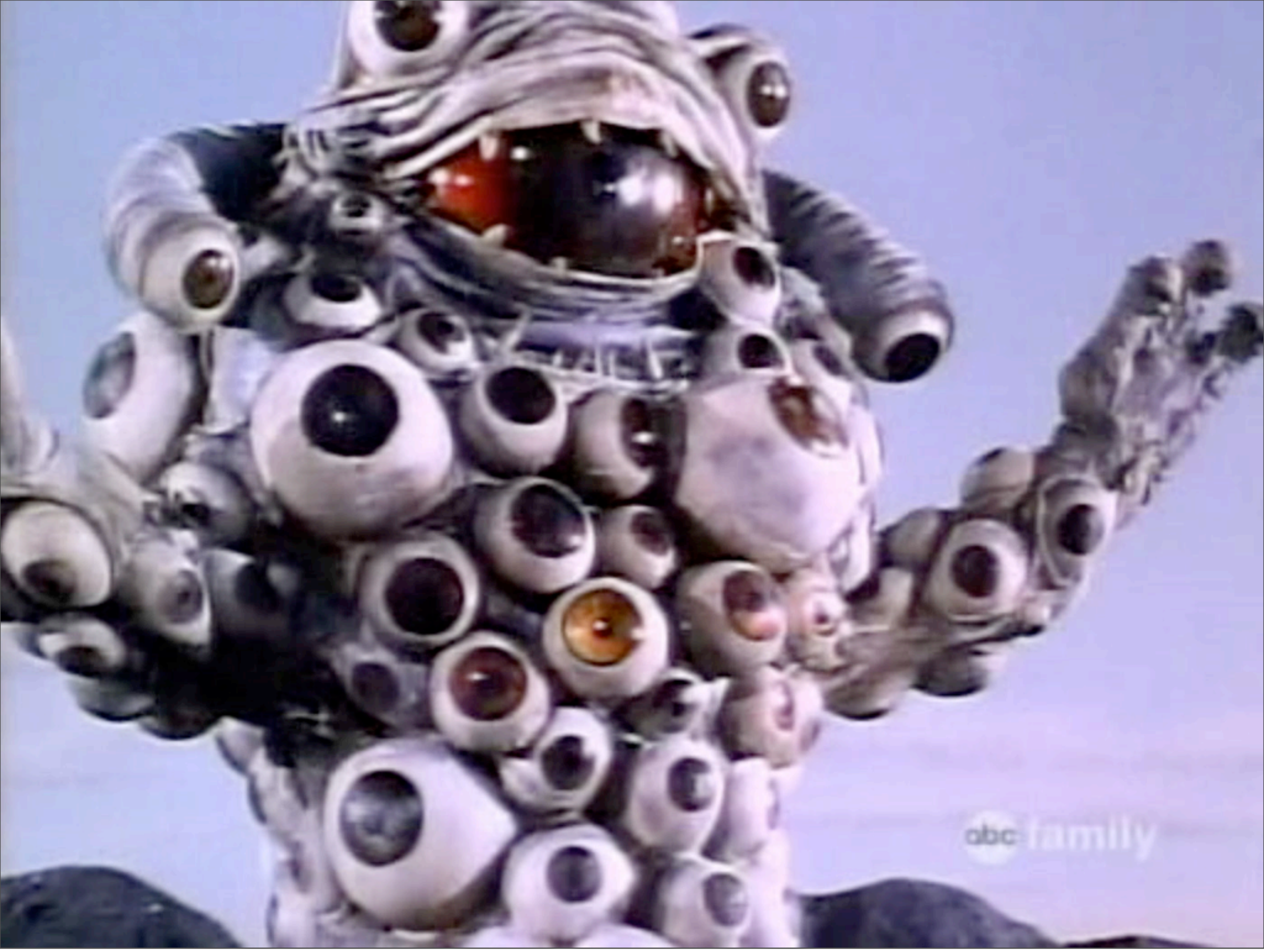
Tuesday, November 12, 13

The bad guys send the putty patrol and a monster...

Tuesday, November 12, 13

They fight the putty patrol and defeat the monster... or so they think...

Monster grows to some insurmountable size... and...



Tuesday, November 12, 13

Here comes the megazord...

The problem is that over time, the Megazords collect rust and you’re 20 seasons into Power Rangers: Jungle Jurassic Park and nobody knows what’s going on anymore.

Small Teams form Micro-Experts

While having experts is always a good thing, placing all responsibility for a product on one person is always going to have risks attached.

Small teams can begin to transform into a group of independent contractors that sometimes interact on shared tasks. Silo-ed developers don't grow as rapidly as collaborative developers. There just isn't an opportunity for cross-training.

Independent contractors have to bid on projects and always justify continued participation. This phenomena bleeds into the development group and creates what I like to call "THE right answer syndrome".

Experts like to do their own thing...

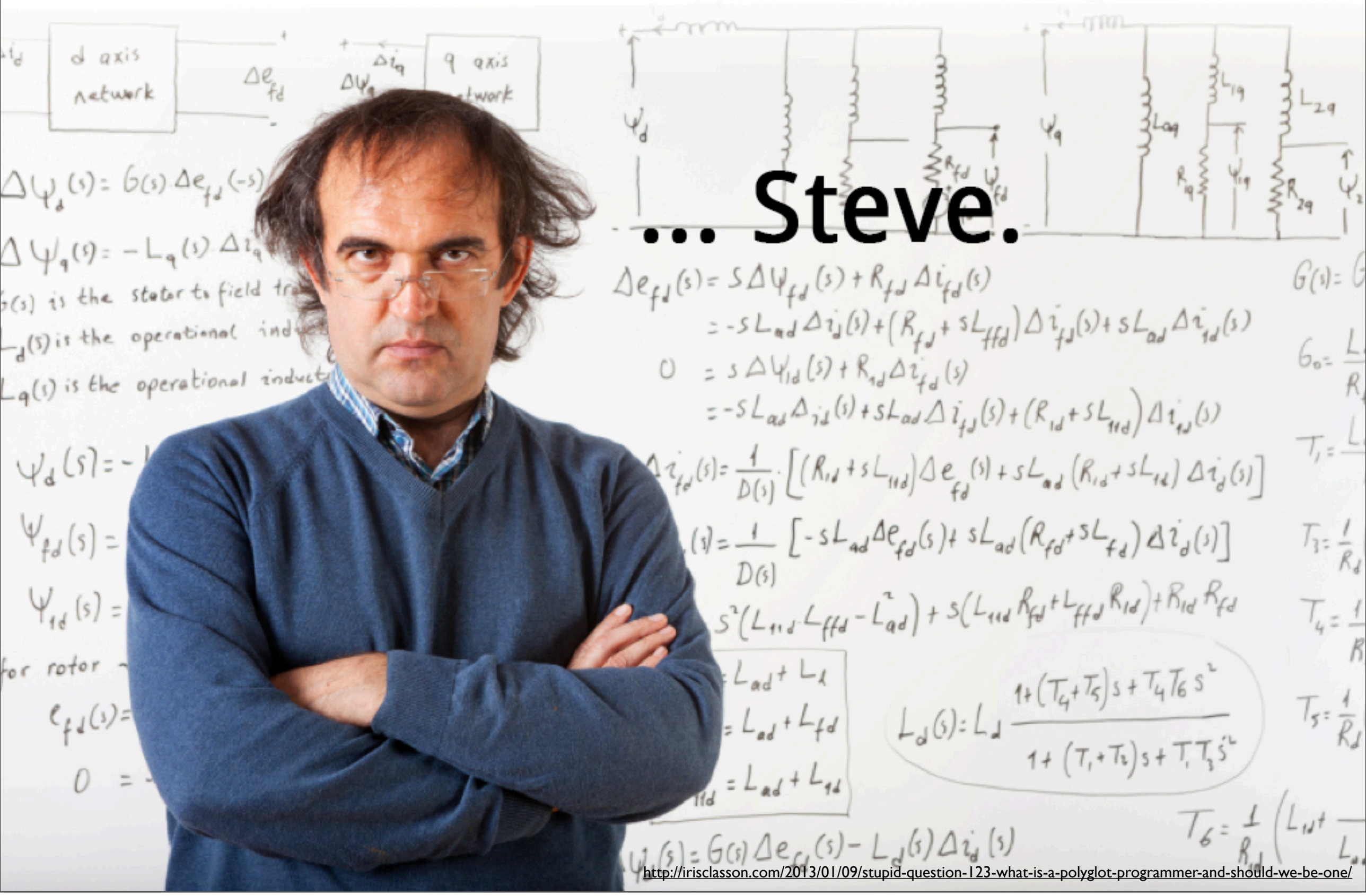
Project management on small teams can look like a post-it blizzard, email, issue trackers, stone tablets, and more. The key is to find something that truly works and buy into it completely.

Github was the first time recorded discussion was directly connected to individual lines of source in our organization.

System Architecture

Some experts may have more expertise in certain domains than others. This may lead certain developers to make problems more complex than needed or make technical decisions to peak their interests.

The fact of the matter is that “peaking interest” is not a basis for technical decision-making. Thought must be put into the decisions we make on how we implement our products because there is a wave of developers behind us that shouldn’t have to rebuild from scratch.



Steve is 60% of your small team's expertise. If Steve doesn't feel challenged at work, he **will** leave, given time.

With Steve's departure, he takes with him years and years of domain expertise that cannot be trained into less than 2 or 3 new developers.

The reality of the situation is that unaccounted development will create difficult-to-maintain applications that next-generation developers are going to want to rewrite instead of gain insight from.

Why are we here today?

My primary goal is to discuss how our team is working towards a Service Oriented Architecture and the lessons we're learning in the process that we believe are allowing us to function more efficiently in a resource-constrained environment.

tl:dr;

If there's only one thing you remember in this tornado of a talk, please remember this...

"There is NO right answer. There is NO silver bullet. However, there is ALWAYS a set of right answers to be selected from. You just have to get them out."

When we started this process, nobody really knew what the end result was going to be. However, stalling for THE right answer was an obvious blocker to productivity and would cost a lot of time and money in practice.

We had to recognize early-on that flexibility as a development group was going to be key to our success.

So, what was our problem?

We didn’t know there was a problem.

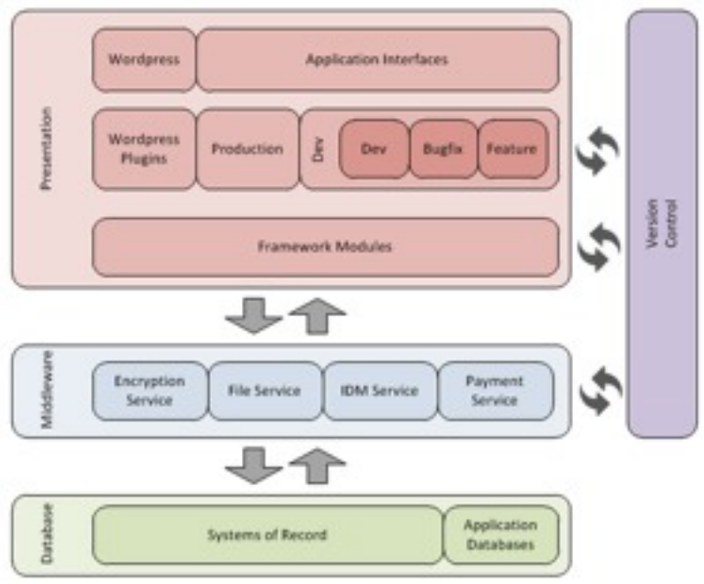
For a long time, we didn’t see any problem in the way we were managing day-to-day development. The job was getting done, so we thought no better.

As the team grew in responsibility (not necessarily resources), development velocity began to slow down and tasks that should by all rights take a few hours are taking days.

Projects are lingering for a long time and failing slowwwwly.

Take a step backwards.

Take a step backwards.



Duplication of Efforts

We identified several important service offerings where there were multiple implementations solving parts of the same problem.

Content Management

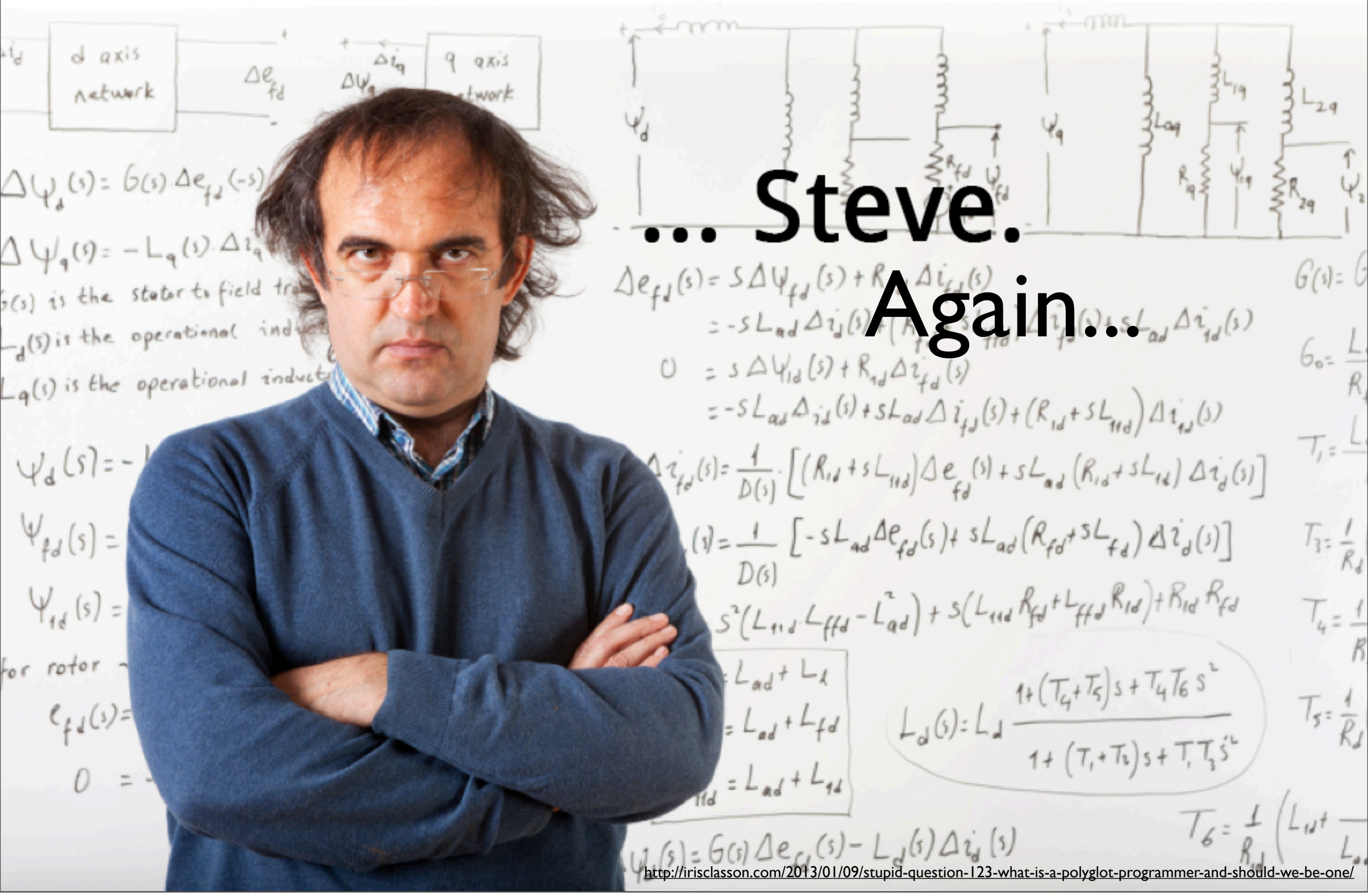
End-users are allowed to maintain HTML sites using whatever client they choose:

SSH + VIM... yes.
Notepad++ / SFTP
Dreamweaver
Contribute

All this means that we're also expected to support all of these technologies JUST for content management.

Application Development

* Access to directory information
* Large file storage and distribution
* Payment Card Industry Interfaces
* Multiple smaller applications that provide basic data collection and reporting



http://irisclasson.com/2013/01/09/stupid-question-123-what-is-a-polyglot-programmer-and-should-we-be-one/

Risk in Loss of Domain Expertise

We have a team of experts. Each developer on our team is given responsibility, end to end.

As developers leave the group, they take that expertise with them and the rest of the already overburdened group is left ot pick up the slack.

Stability and **Agility**

We want to find stability in the areas we know how to solve so that we can spend resources on addressing future goals in a more agile fashion. This will lead to breakthroughs that improve our entire application suite.

We want a greater sense of stability in the maintenance of “legacy” applications.

We want an approach to development like a feather: As we solve new problems, the feather floats left; then right, but eventually settles into a stable implementation after a display of agility. I’m not advocating flying off to outer space at every turn, but it’s certainly not good to find yourself in a self-mandated rut for comfort’s sake.

So we’re down the path!

We put a halt on feature development across many of our applications and committed to no new development for the duration of our infrastructure planning and migration.

This gave us the affordance to focus on the long-term problem; mitigating the distraction of day-to-day fires.

Content-based Service Offering consolidated in Wordpress

Most of our content owners / managers are not technical staff. Because of that, Wordpress has a lower barrier to entry than other CMSs we assessed.

For basic content needs, Wordpress has been great so far. However, there is the issue of tying dynamic data sources into our Wordpress-based sites. This influenced part of our decision toward an SOA.

Develop best practices and apply them regularly.

Develop a set of best practices to form an organic record of how you do software development.

Apply them in new development and be prepared to constantly iterate on the things that just aren't fitting.

Don't set your team up for failure by expecting to know everything up front. The lesson to be learned is that again, "This isn't 'Who Wants to be a Millionaire'.. so when Regis asks you, "Is this your final answer?" you better say, "It depends."

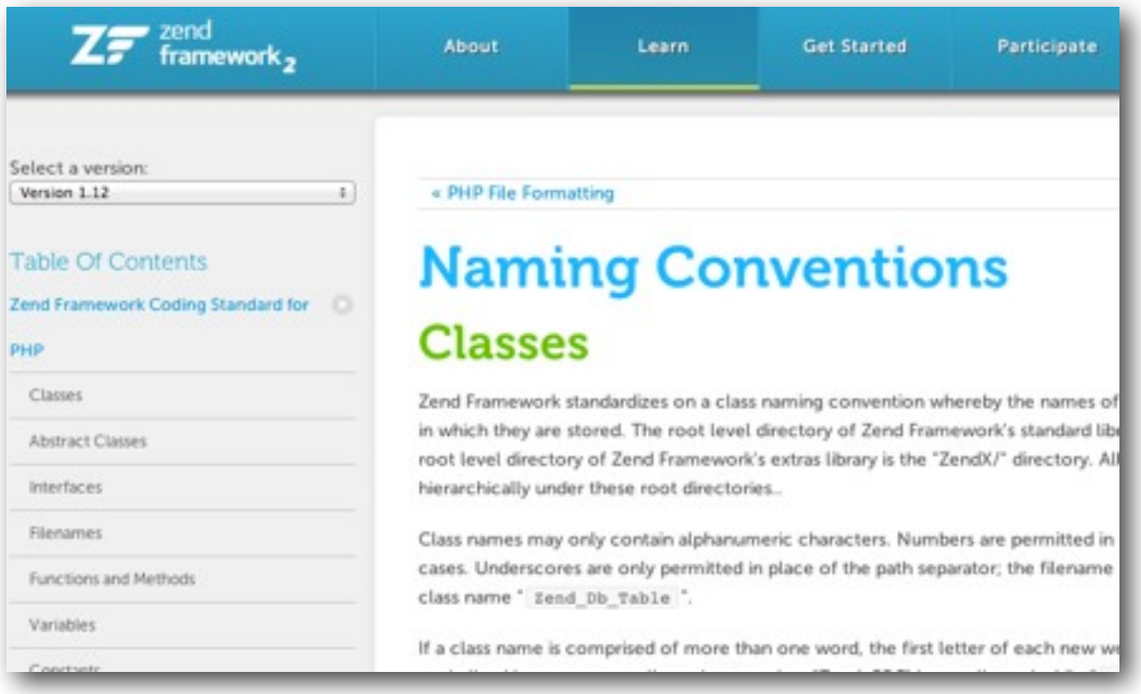
Make iterative advances.

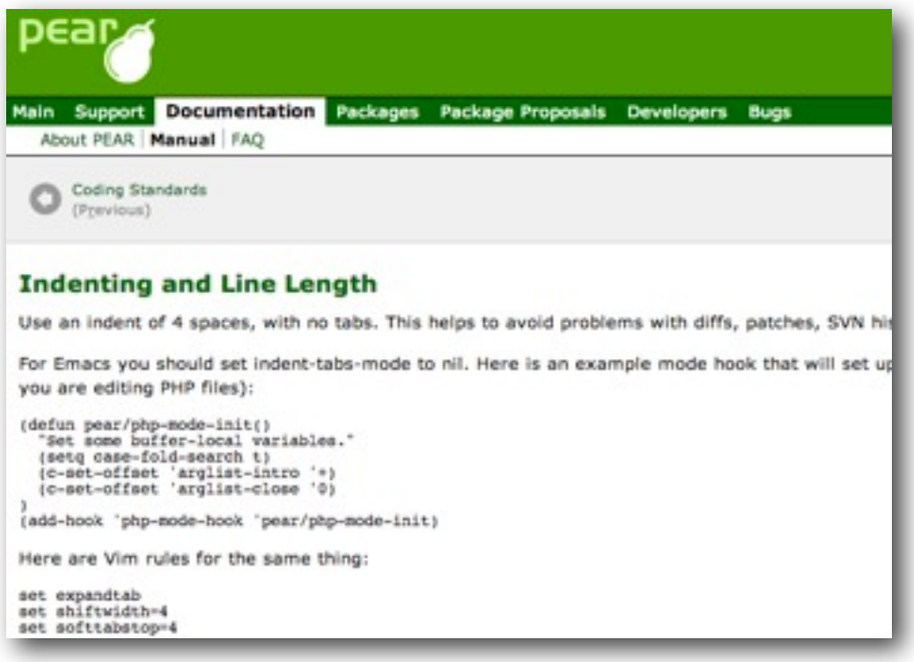
Make iterative advances in development projects, applying best practices as you go; always with an eye towards the next iteration.

Don’t create a revolution.

Do not try to change the world overnight. There are real habits built up that will take time to change.

Coding Standards





Pick a standard or write your own flavor of an existing. It doesn't matter.

We went with PSR (PHPFIG)

4.3. Methods

Visibility MUST be declared on all methods.

Method names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.

Method names MUST NOT be declared with a space after the method name. The opening brace MUST go on its own line, and the closing brace MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.

A method declaration looks like the following. Note the placement of parentheses, commas, spaces, and braces:

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

5.1. `if`, `elseif`, `else`

An `if` structure looks like the following. Note the placement of parentheses, spaces, and braces; and that `else` and `elseif` are on the same line as the closing brace from the earlier body.

```php
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}
```

Web Services: REST vs SOAP

We knew that we were going to be writing a bunch of web services that represented centralized service offerings we were providing.

We also had a goal of providing these services for consumption by other groups on campus. Soooo...

We didn't answer this question alone. We assembled a cross-organizational working group to talk about the problem and establish a standard for web services on campus.

First order of business... REST or SOAP.

Holy war ensues... not really.

NC State Interop Group is formed.

RESTful URLs and Actions

The constraints of REST describe the segregation of logical resources within a web service. These resources are acted upon using HTTP requests using the standard HTTP methods / verbs: GET, POST, PUT, PATCH, and DELETE.

Resource Naming

- Resource names MUST be represented as nouns.
- Resource names MUST use their plural form.
- Resource fields MUST use `snake_case` in all RESTful interactions. **Note: This excludes libraries that consume the service. Libraries are subject to the coding standards for the language they're written in.**
- There is no requirement for resources to map one-to-one with underlying models. The idea is to abstract away technical detail from the API consumer.

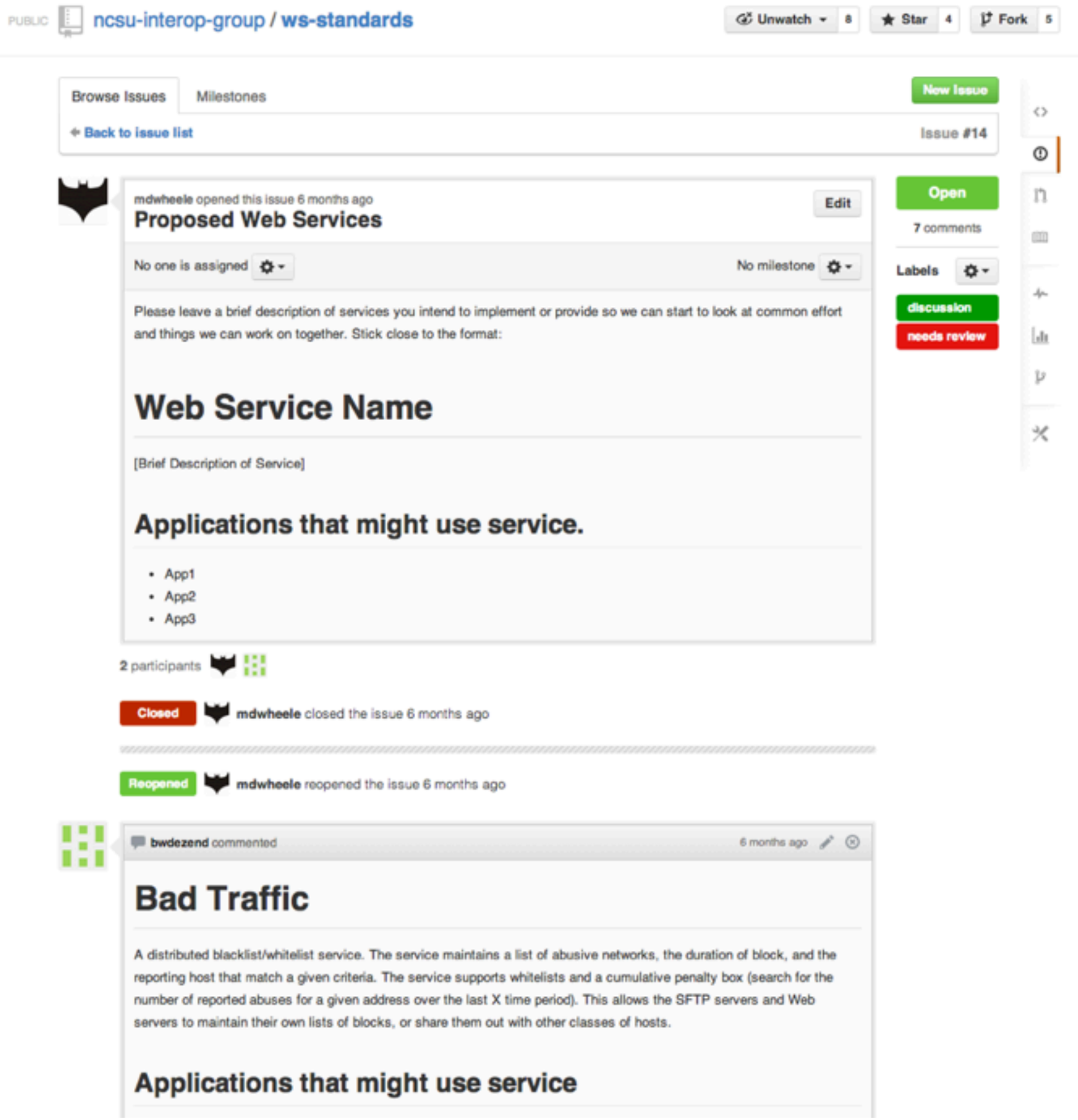
Filtering

- APIs MUST use a unique query parameter for every resource field that implements filtering.
- Filters MUST be mapped to the query string and MUST NOT be included in the resource mapping.
- Multiple filters on one resource field MUST be separated by comma.

Examples

- `GET students?major=mae` - Retrieves a list of MAE students.
- `GET students?major=mae,csc` - Retrieves a list of MAE and CSC students.

Immediate Benefits



We have a group of people to contribute towards the evolution of this standard and to start creating some really cool services that no single group would realistically be able to provide.

Iterate

It was maybe 48 hours after our group first approved WSSR-1 and we were writing a service that had a completely valid use-case to bend the standard.

That's okay. It means we didn't have a full understanding of the problem set up-front.

We expected this, remember?

Establish a “Definition of Done”

“A common understanding of when a development task is finished.”

This can transform a team from claiming best practices have prohibitive overhead costs. You simply accept these new practices as part of the “definition of done”.

For example, for a feature to be complete, it must: be specified, be coded, unit tested, reviewed by the team, and then merged into the repository. Only then can it be marked complete.

Notice there is no backlog item to write tests or meeting to do a code review. It’s simply accepted as part of the feature itself. More about this later in “Testing”.

* Should only require tasks under teams control
* Avoid having too many states; to do, doing, and done are enough
* Avoid tasks that are susceptible to getting “stuck”

Following these guidelines, you also pear down tasks into manageable, distributable chunks.

Keep in mind, our team’s DoD may not fit your team. In all of this, you have to find something that fits the team that you have.

Version Control

Version control is a common practice in development teams today. 10 years ago, only larger software companies consistently used version control systems. I was at a conference earlier this year and the question asked wasn't who IS doing version control, but who WASNT (3 hands in a room of 200+)

Somewhere along the line, we decided that the rules of software development did not apply to what we were producing.

Version control is, in my opinion, the single-most impactful service we've bought into.

Common Gripes

* I don't make mistakes...
* It's much easier to just save my files in live space and test from there
* My boss says it sounds like a bunch of overhead with minimal return

How do we sell it?

* Source control lays the foundation for best practice and is the beginning of a multiplicative return on investment
* Serves as an authoritative record to the state of an application and creates an open point of reference for a development team
* It's not about being able to rollback

http://git-scm.com/book

So how do I get started?

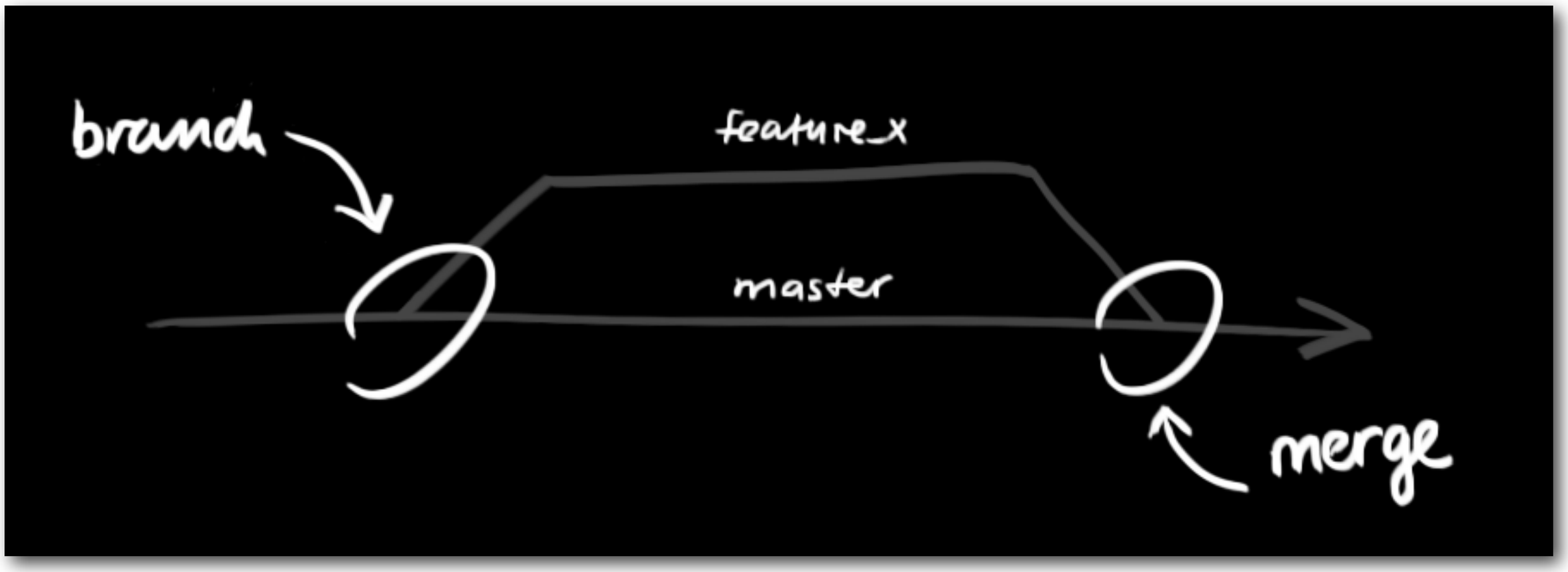
Git is free. It comes installed with OSX and most Linux distributions. Git is also available for Windows through a cyg-win'y terminal called Git-bash.

Command-line tools are scary! Get over it. While there ARE GUI applications for git, github, and others; you will get more out of learning the CLI.


github
SOCIAL CODING


Atlassian
Bitbucket

A better development workflow appears.



A starter workflow with git might look something like this:

1) A ‘master’ branch of development
2) When a feature is to be implemented, a new branch called ‘feature-awesome’ is created
3) When the feature is completed (DoD), the branch is merged back into ‘master’

Github-flow

* Everything in `master` is deployable
* To work on something new, create a descriptive branch off of `master`
* Commit to that branch locally and regularly push your work to the server
* When you need feedback, help, or think it's ready, submit a "pull request"
* After code review, the branch is merged into master and the code is deployed immediately

This is what Github does every day. They claim to each commit deployable code 23-25 times a day.

That means that Github.com is deployed 24 * ~80 a day; that's 1,920 times...

Source Commits Network **Pull Requests (23)** Fork Queue Issues (290) Wiki (98) Graphs Branch: master

Open **josh** wants someone to merge 11 commits into `master` from `charlock-linguist` **#1497**

Discussion Commits <> 11 Diff >= 9

josh opened this pull request about 16 hours ago

Charlock+Linguist

Use Charlock to detect if blobs are binary or plain text. The encoding is then passed along to pygments.rb when the blob is rendered.

/cc @brianmario

Open

+ 63 additions

- 40 deletions

All Pull Requests

josh, brianmario, and tmm1 are participating in this pull request.

josh added some commits about 16 hours ago

24825bd Use charlock for blob binary and encoding detection

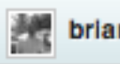
bb0b945 Merge branch 'master' into charlock-linguist

brianmario started a discussion in the diff about 16 hours ago

vendor/internal-gems/linguist/linguist.gemspec View full changes

```
...  ...  @@ -6,8 +6,9 @@ Gem::Specification.new do |s|
 6    6     s.files = Dir['lib/**/*']
 7    7     s.executables << 'linguist'
 8    8
 9         -  s.add_dependency 'escape_utils', '0.2.3'
10         -  s.add_dependency 'mime-types',   '1.16'
11         -  s.add_dependency 'pygments.rb',  '~> 0.2.0'
      9    +  s.add_dependency 'charlock_holmes', '~> 0.6.0'
```

1



brianmario repo collab about 16 hours ago

I'd change this to 0.6.2 - I yanked 0.6.1 since it had that bug

brianmario commented August 17, 2011

❤

bielkamp commented August 17, 2011

⚡

defunkt commented August 17, 2011

kneath added some commits August 17, 2011

d0a5311  Org n00b notice needs new words.

kneath referenced this pull request from a commit August 17, 2011

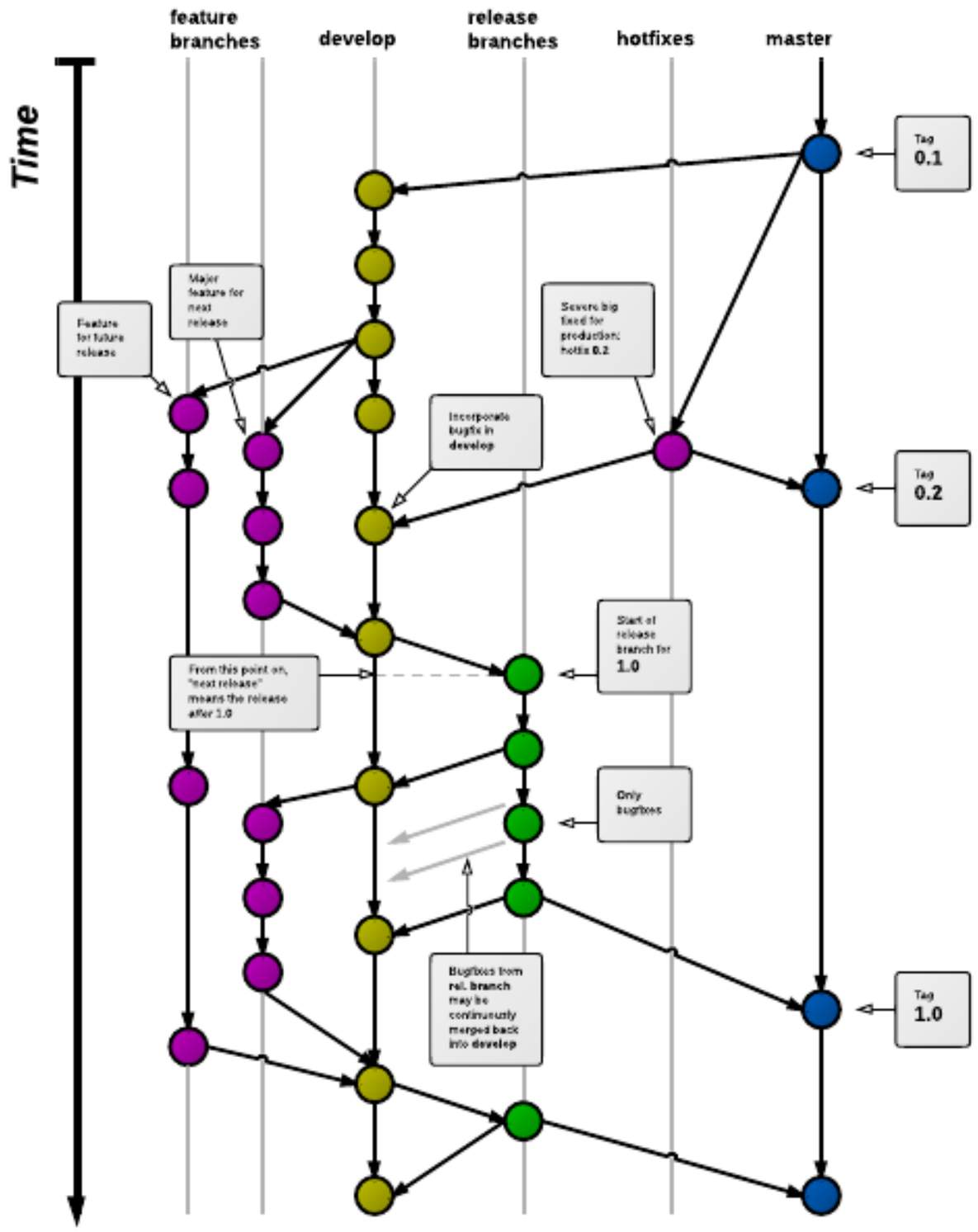
a9ab4eb  Merge pull request #1431 from github/orgselector

Merged  **kneath** merged commit `a9ab4eb` into `master` from `orgselector` August 17, 2011

Closed  **kneath** closed the pull request August 17, 2011

Git-flow

Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes
Bugfixes from rel. branch may be continuously merged back into develop
Tag 0.2
Tag 1.0

Pick something that fits.

Code Reviews

Common Gripes

* I don't make mistakes...
* I don't want someone else checking over my work. I know what I'm doing!
* So we're meeting to design, plan, architect, and then you're saying we meet again to review code... When do we get to work?

* Code review as part of a version control workflow creates a unique opportunity for pushing quality of code that goes into an application
* Reviews are a great time to enforce the current iteration of best practices your team has adopted
* It can foster discussion on how best to approach a new feature or bugfix

* Don’t make it personal
* Don’t make inquisitiveness an attack
* Always... always assume best intentions

Code review is a very sensitive subject that should be carefully approached, but definitely adopted. It’s highly dependent on the personality types you have on your team.

It’s important to have a team that can separate technical ability from self-worth as a member of the team.

Just because you don’t know something or didn’t quite hit the mark does not mean you don’t know; you just don’t know, yet. Without code review, you’d never get the input that what you’re doing wasn’t quite right. While it may work, it’s not the best you can do.

And if you’re on the other side of that coin and you’re frustrated and say, “What were they thinking?!”; take a step back and be happy that your co-worker is about to find mentorship in you. Be that mentor.

ncsu-multiauth/ncsu-multiauth.php

outdated diff ✖

```
...  ...  @@ -8,114 +8,252 @@ Author: Dustin Wheeler
234  +    {
235  +        $plugin_data = get_plugin_data(__FILE__);
236  +        $updater = new Auto_Update($plugin_data['Version'], "http://www.webtools.ncsu.edu/wp-updates
237  +        add_filter('pre_set_site_transient_update_plugins', array($updater, 'check_update'));   // D
238  +        add_filter('plugins_api', array($updater, 'check_info'), 10, 3);    // Define alternate resp
239  +    }
240  +
241  +
242  +    /**
243  +     * Disable the password fields for non-admins, to prevent password changes
244  +     *
245  +     * @param   mixed   $flag   Unused
246  +     *
247  +     * @return  bool
248  +     */
249  +    public function disable($flag)
```

2

mdwheele 5 months ago

Do we want to make a distinction between local and WRAP users in allowing the capability to change passwords themselves? It's not that we're dis-allowing them access to change passwords, it's that it makes no sense for WRAP. However, local users would probably want to be able to.

mafields 5 months ago

The password fields are actually accessible to local accounts, so they can change their passwords. The password recovery function is currently disabled though. We can change that.

Add a line note

Testing.

Griping

that's all.

The fact of the matter is... it's 2013. We've been through this with version control already. Do we really need to wait 10 more years before we buy into commonly accepted best practices in testing our applications?

I went to a conference earlier this year... Chris Hartjes.


Tests

“If there are no tests, it’s broken.”

This simplifies why to buy into testing our applications. It should simply be a part of everyone’s Definition of Done.

The next big feature might as well not even be released if we’re not going to test it. It’s like sending an airplane out of the factory with no wings.

Difference is, with software, things can seem to function and be horribly broken inside.

Unit, Functional, TDD, BDD... bunch of fads.

Unit Test

A method by which individual units of source code are tested to determine if they are fit for use.

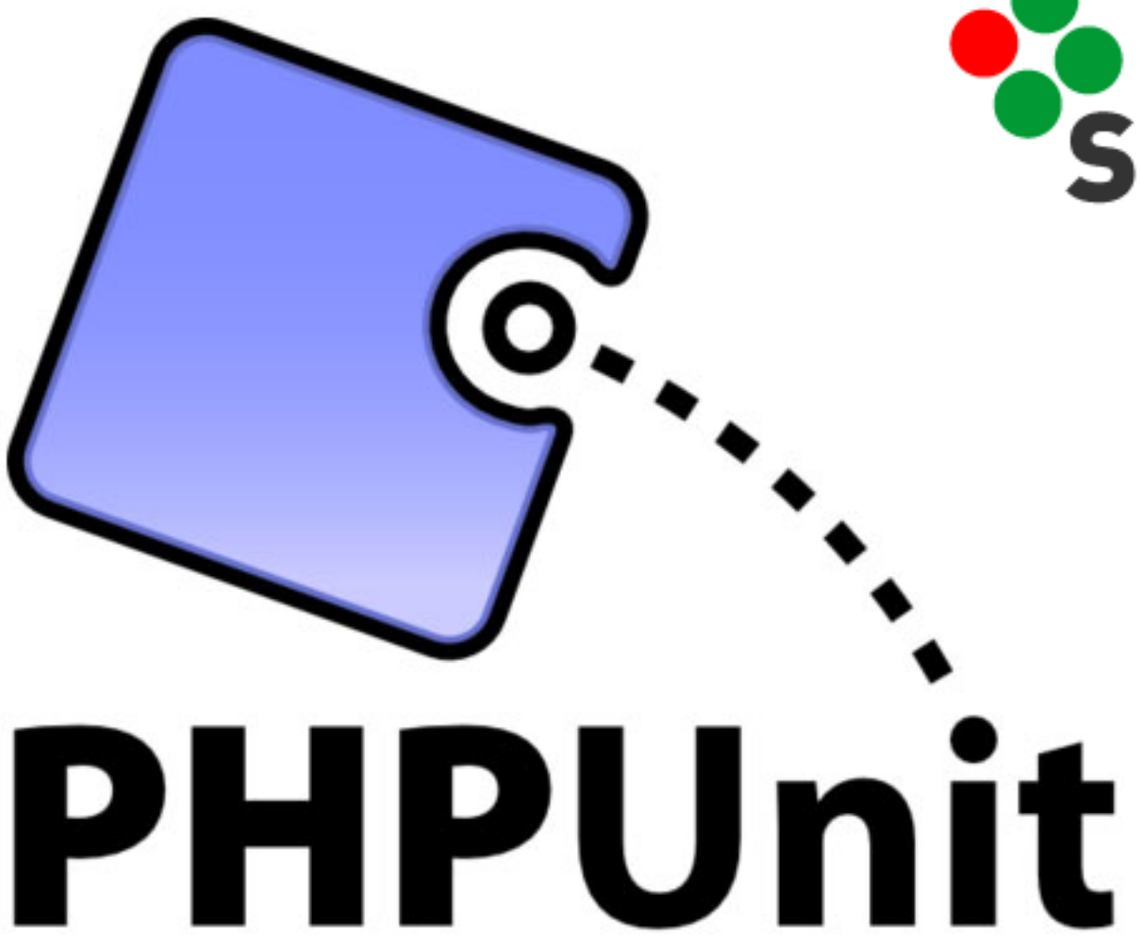
Is your code testable in isolation?

```
public class SuperMan implements SuperHero {

    private Cape cape = new Cape();
    private Spandex costume = new Spandex();

    public void fly() {
        // Do something exciting with cape and spandex.
    }

}
```

```
public class SuperMan implements SuperHero {

    public SuperMan(Cape cape, Spandex costume) {
        this.cape = cape;
        this.costume = costume;
    }

    public void fly() {
        // Do something exciting with cape and spandex.
    }

}
```

This becomes important when you start considering tests failing. If tests on YOUR code is passing and the app is breaking, that means you have an uncovered dependency that failed... which is GREAT!

Consider only using packages with tests for that very reason.



and more...

http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

file | 17 lines (14 sloc) | 0.278 kb

Open | Edit | Raw | Blame | History | Delete

```php
<?php

use Wheeler\Fortune\Fortune;
use Mockery as m;

class FortuneTest extends PHPUnit_Framework_TestCase
{
    public function tearDown()
    {
        m::close();
    }

    public function testCanCreateFortune()
    {
        $this->assertNotEmpty(Fortune::make());
    }
}
```

```php
public function testGetConstants()
{
    $enum = new MockEnum(MockEnum::One);

    $constants = $enum->getConstants();

    $this->assertCount(3, $constants);
    $this->assertEquals(MockEnum::One, $constants['One']);
    $this->assertEquals(MockEnum::Two, $constants['Two']);
    $this->assertEquals(MockEnum::Three, $constants['Three']);
}

public function testSetValue()
{
    $enum = new MockEnum(MockEnum::One);

    $this->assertEquals(MockEnum::One, $enum->getValue());

    $enum->setValue(MockEnum::Two);

    $this->assertEquals(MockEnum::Two, $enum->getValue());
}
```

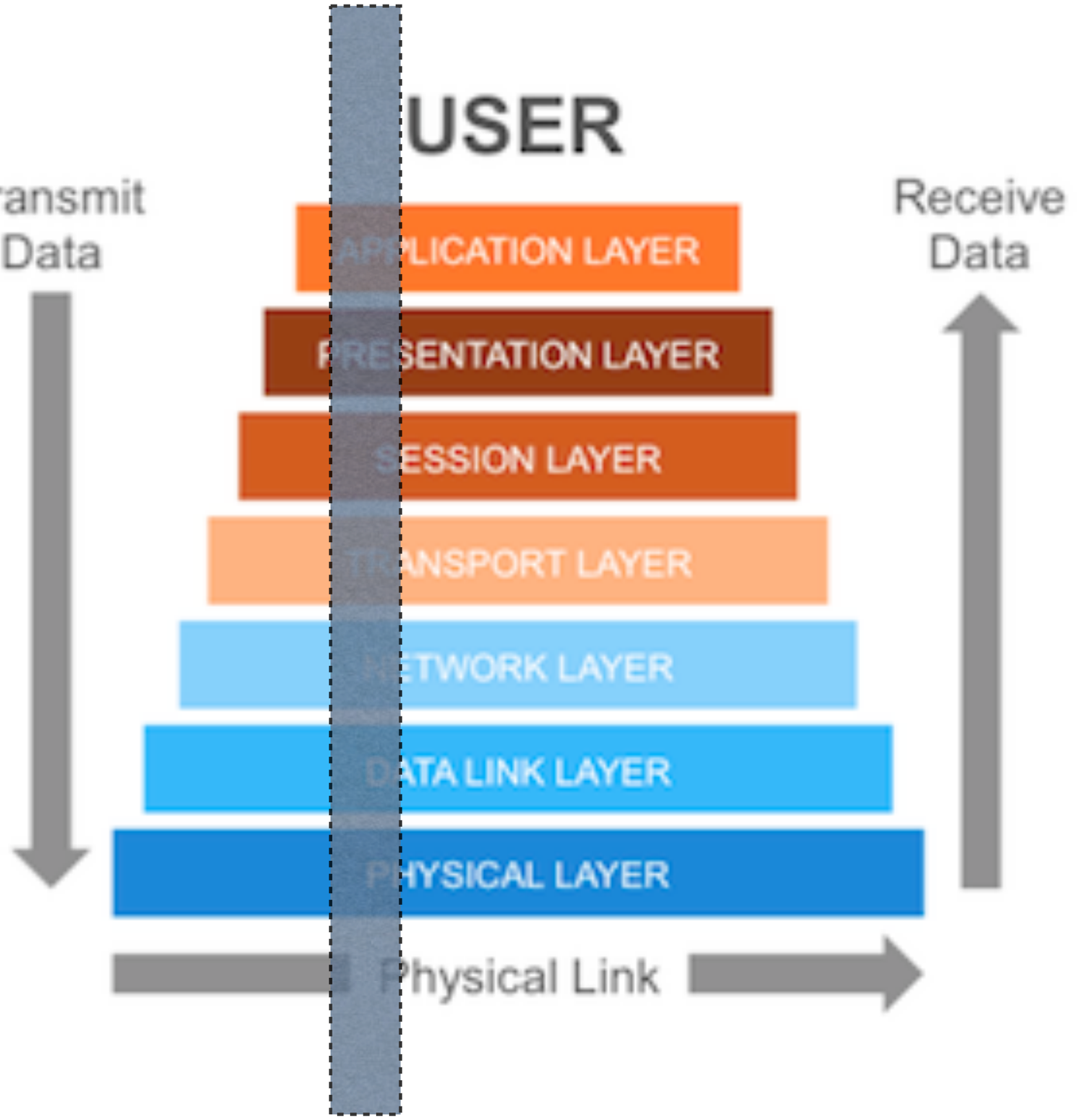
Functional Test

A quality assurance process that bases its test cases on the specifications of the component under test

Sometimes called User Acceptance Testing, perhaps Integration Testing.

This can include back-end service testing as well as browser emulation.

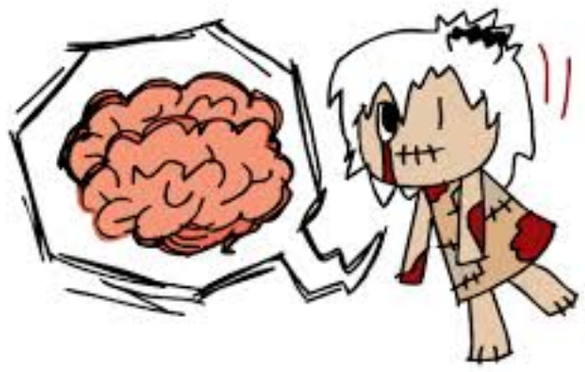
Think of it as testing a slice of your application stack.



Tuesday, November 12, 13

I HATE THIS IMAGE. ITS A PLACEHOLDER TO GET THE POINT ACROSS.

There's no "is your app testable" here.



```
var Browser = require("zombie");
var assert = require("assert");

// Load the page from localhost
browser = new Browser()
browser.visit("http://localhost:3000/", function () {

  // Fill email, password and submit form
  browser.
    fill("email", "zombie@underworld.dead").
    fill("password", "eat-the-living").
    pressButton("Sign Me Up!", function() {

      // Form submitted, new page loaded.
      assert.ok(browser.success);
      assert.equal(browser.text("title"), "Welcome To Brains Depot");

    })

});
```


This gets tedious but the idea at first glance is to test what your users are seeing.

There are tools to make this less of a chore and I'll get to those shortly.

Test Driven Development

* Write a test that fails
* Write code to make it pass
* Repeat

Does TDD make sense for everyone? Not at all.

But what it IS really good at is getting you “in the mode”.

Behavior Driven Development

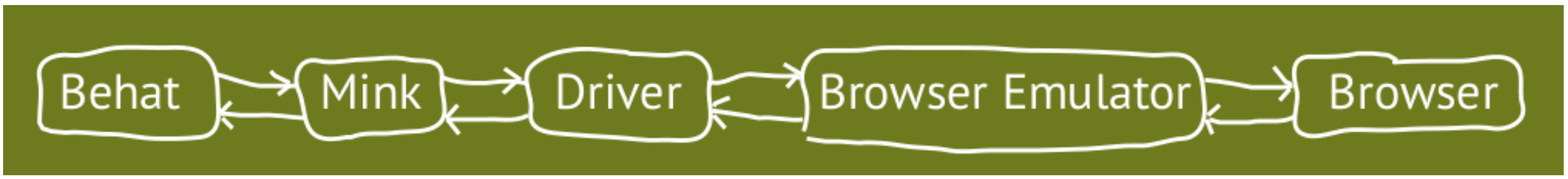
* Write a test that fails
* Write code to make it pass
* Repeat

I didn't mess up my slides... it's pretty much the same exact thing, just a different spin.

BDD is ... behavior driven. It's analogous to unit vs functional testing. BDD is testing the public-facing side of the application.

Behat & Mink

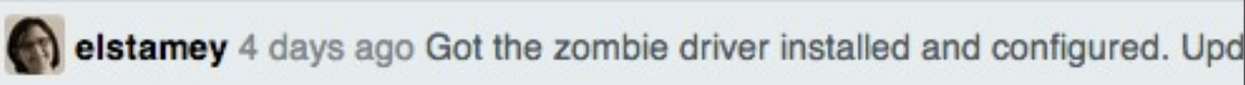
Now I know you're thinking, "Dustin, you're just making up stuff."



This is the most professional diagram I could find on the topic.

But you see that Behat is a testing framework that uses Mink to drive a headless browser emulator to test-drive your application according to rules you set.

```
Feature: ls
  In order to see the directory structure
  As a UNIX user
  I need to be able to list the current directory's contents

  Scenario: List 2 files in a directory
    Given I am in a directory "test"
    And I have a file named "foo"
    And I have a file named "bar"
    When I run "ls"
    Then I should get:
      """
      bar
      foo
      """
```



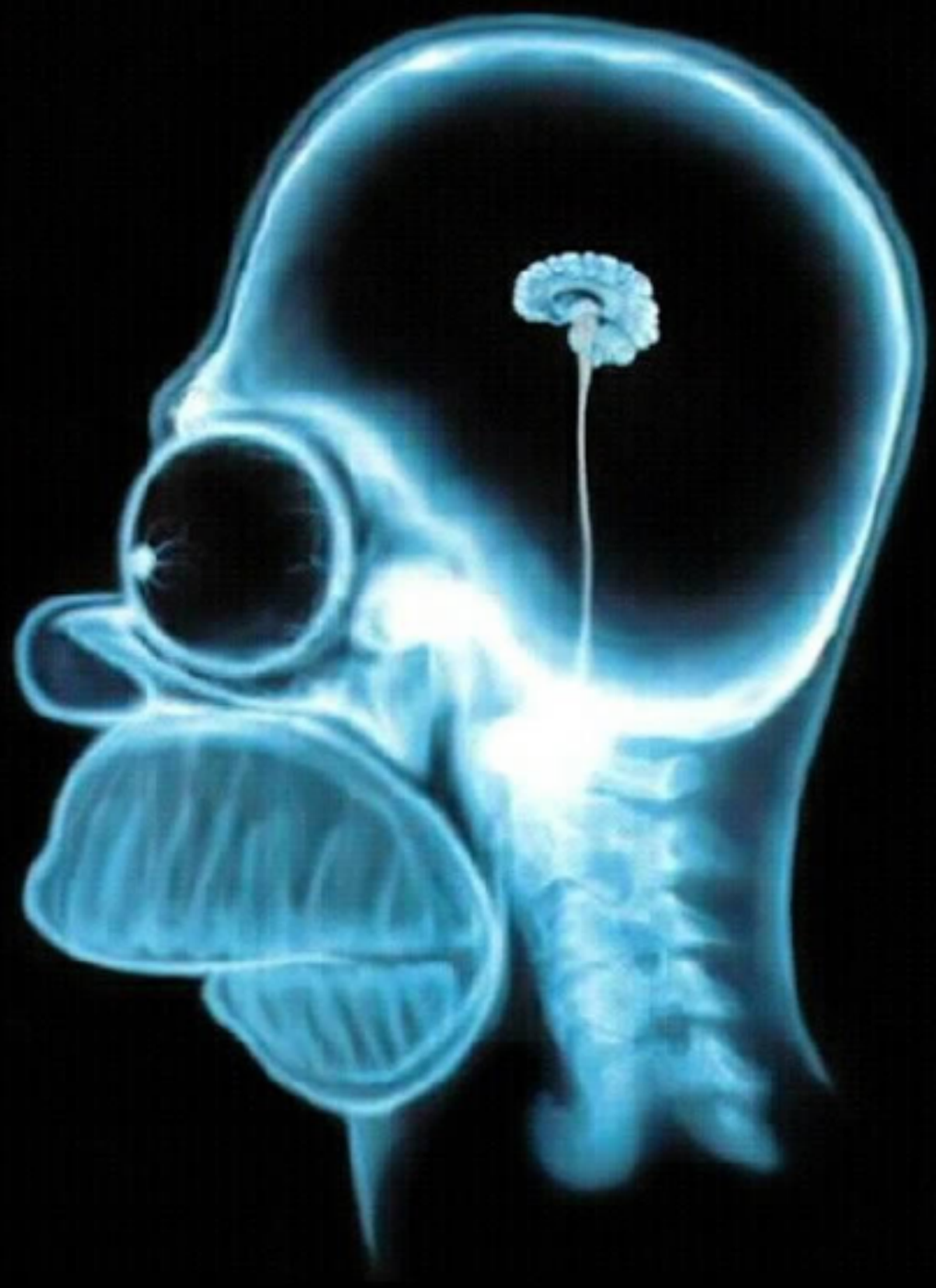
file | 11 lines (9 sloc) | 0.298 kb

```
1  Feature: Search
2    In order to find buildings on campus
3    As an end-user
4    I need to be able to search a buildings directory
5
6    Scenario: Find a building
7      Given I am on "/maps/client/"
8      When I wait for page to load
9      And I fill in "search" with "Page Hall"
10     Then I should see "Page Hall"
```

In Summary...

They’re important and form the pass/fail basis for whether or not an application is allowed to be deployed.

Talk a little about continuous integration and deployment... but not a lot of details.



Mitigating Brain Drain

Documentation

* External documentation works, but must be kept up to date
* **Meaningful** comments in code

External documentation will work just fine as long as it is made a “part of the process” like everything else. The first stop after a new feature is discussed and planned out should be an update to the documentation.

Meaningful comments are important.

```
public class SuperMan implements SuperHero {

    ...

    public void fly() {
        // If the costume is not equipped.
        if (!costume.isEquipped())
            return;
    }

    ...

}
```

```
    /**
     * This method takes a formatted string of year(s) and stores it as a multi-dimensional
     * array of operator+value expressions.
     *
     * Comparison operators can be encoded in URL.  Any PHP library wrapping this service should encode
     * the URLs.  For human usage, it may make more sense to allow unencoded operators.
     *
     * Operator Encoding
     * -------- --------
     * =        %3D
     * >        %3E
     * >=       %3E%3D
     * <        %3C
     * <=       %3C%3D
     *
     * Query parameter format:  ?year=2008,2009,1982-1989
     *
     * @param $year
     */
    public function setYear($year)
    {
        /** Split query parameter by comma for different years. */
        $years_raw = explode(',', $year);

        foreach($years_raw as $expression){
            /**
             * Every expression is either a single year or a range of years separated by a dash (-)
             */
            preg_match('/^([0]{1}|[0-9]{4})(?:-)?([0-9]{4})?$/', $expression, $matches);

            /**
             * From the docs on 'matches':
             *
             * If matches is provided, then it is filled with the results of search. $matches[0]
             * will contain the text that matched the full pattern, $matches[1] will have the text
             * that matched the first captured parenthesized subpattern, and so on.
             *
             * In this specific case:
             *
             * $matches[1] is the year (or beginning year, in a range)
             * $matches[2] is the ending year, if in a range; null if not range.
             */

```


Fix this code... lol.

SOLID Design Principles

* Write a test that fails
* Write code to make it pass
* Repeat

Tuesday, November 12, 13

Tuesday, November 12, 13

Tuesday, November 12, 13