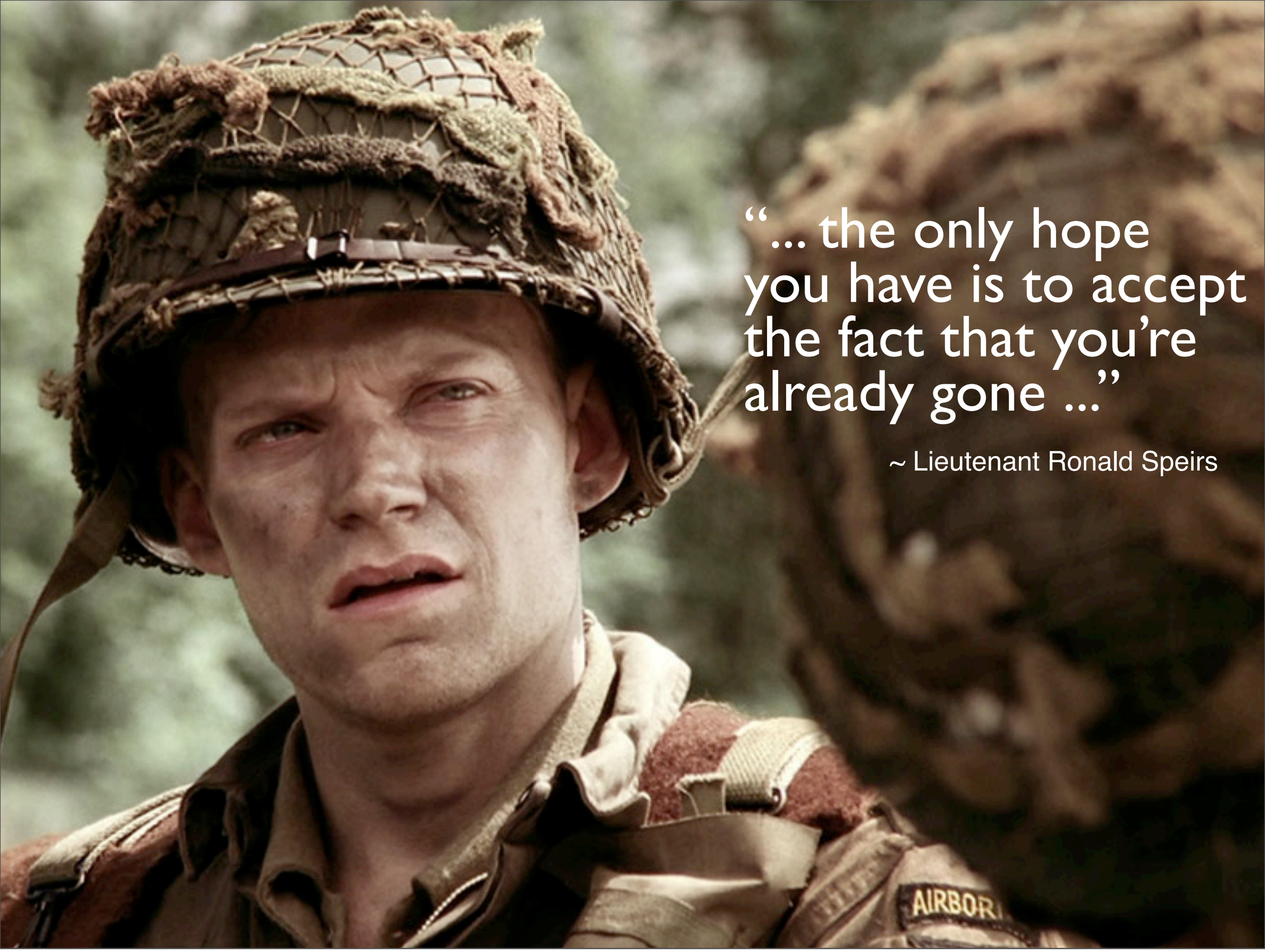
Lessons Learned in a Continuously Developing Service Oriented Architecture



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...

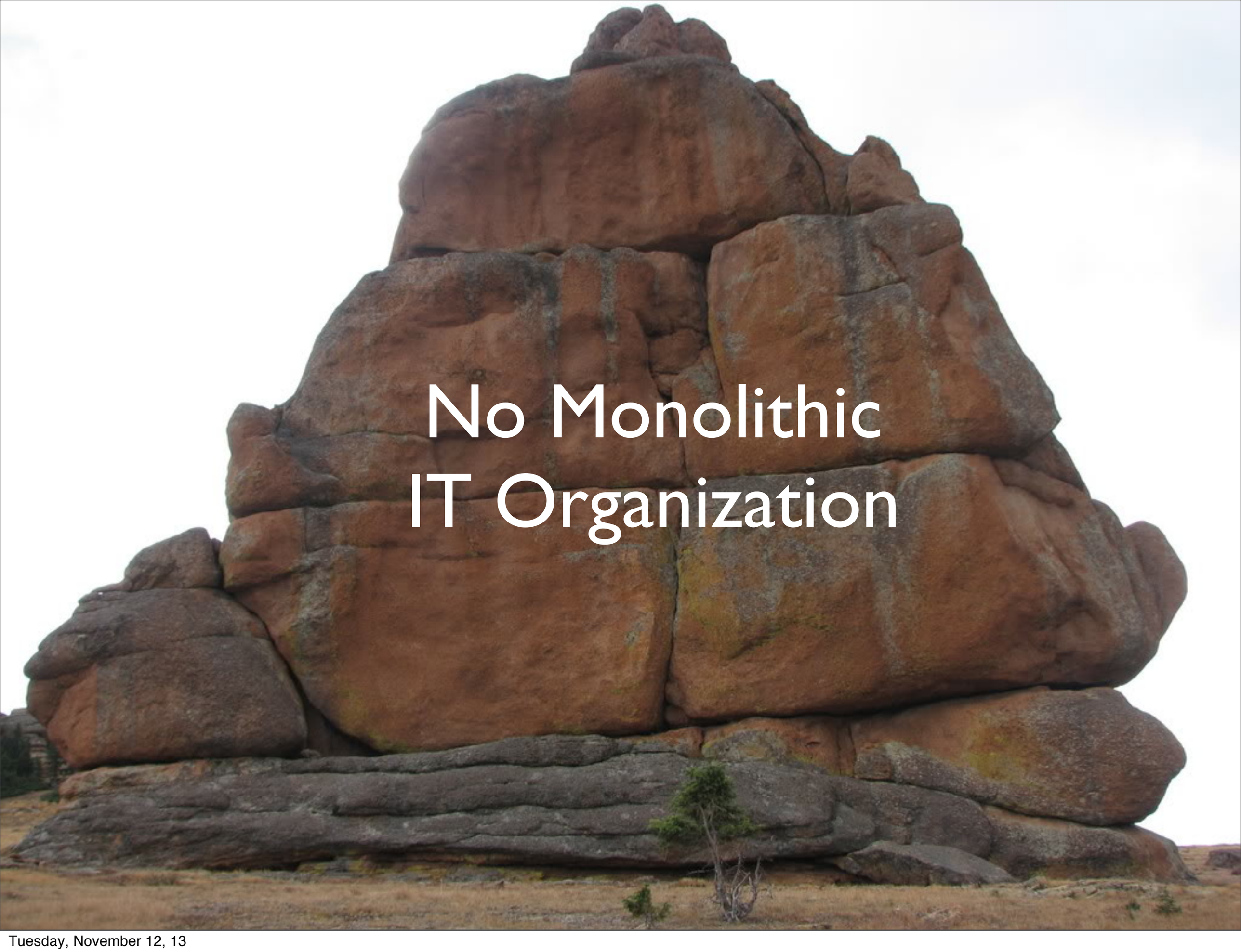
The first lesson we learned is to accept the fact that our first attempt at new problem areas or domains will ALWAYS be naive to an extent and will never be the best we’re capable of. As soon as we accept that, we can go on to function as a developer should.

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.

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.

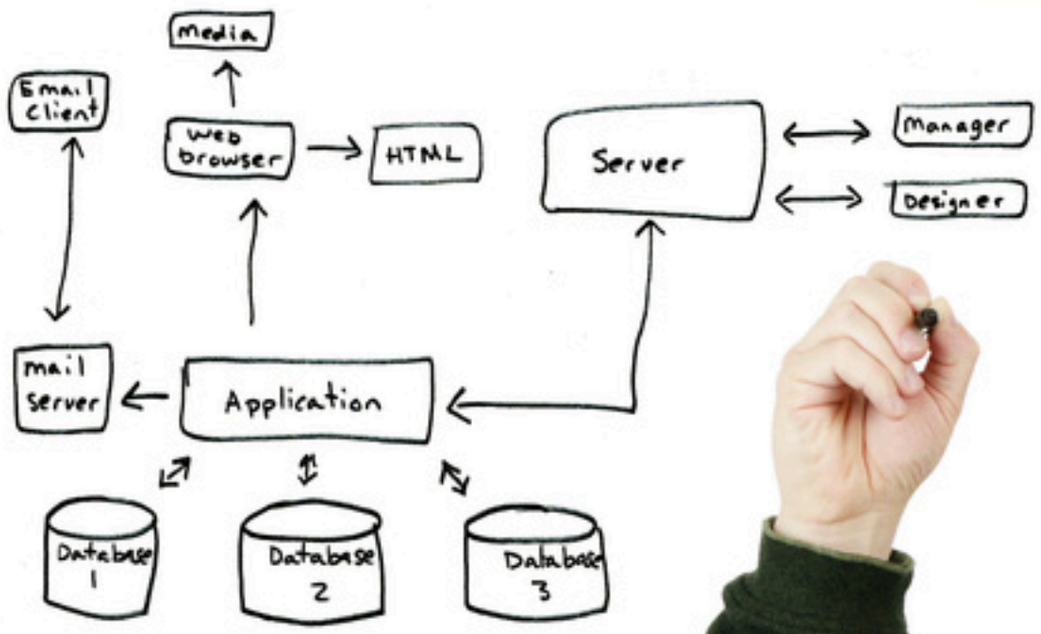
Group Skill-set

Tuesday, November 12, 13

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

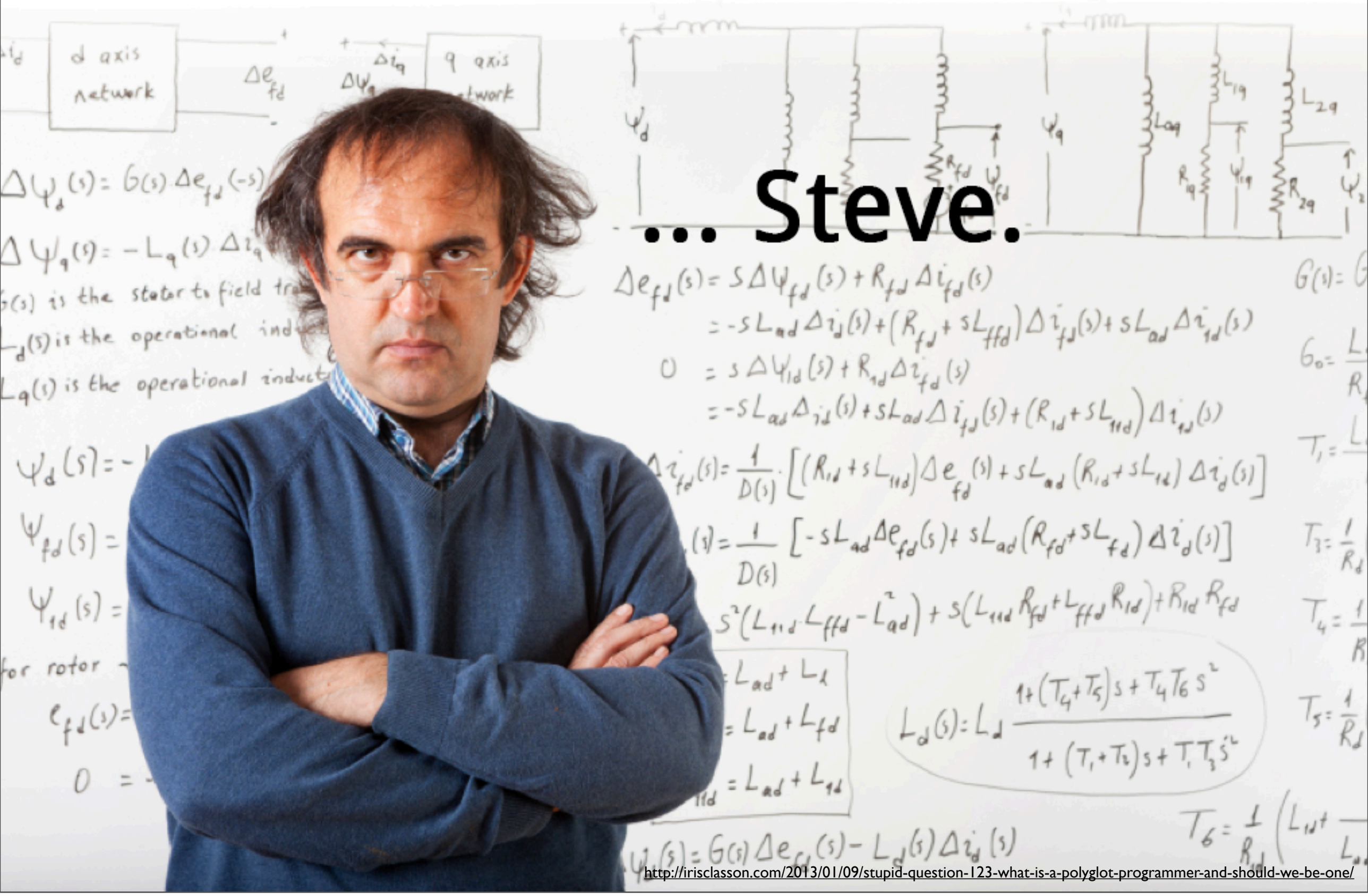
A lot of times, there is a real risk to end up with a group of “jacks/jills of all trades”

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/

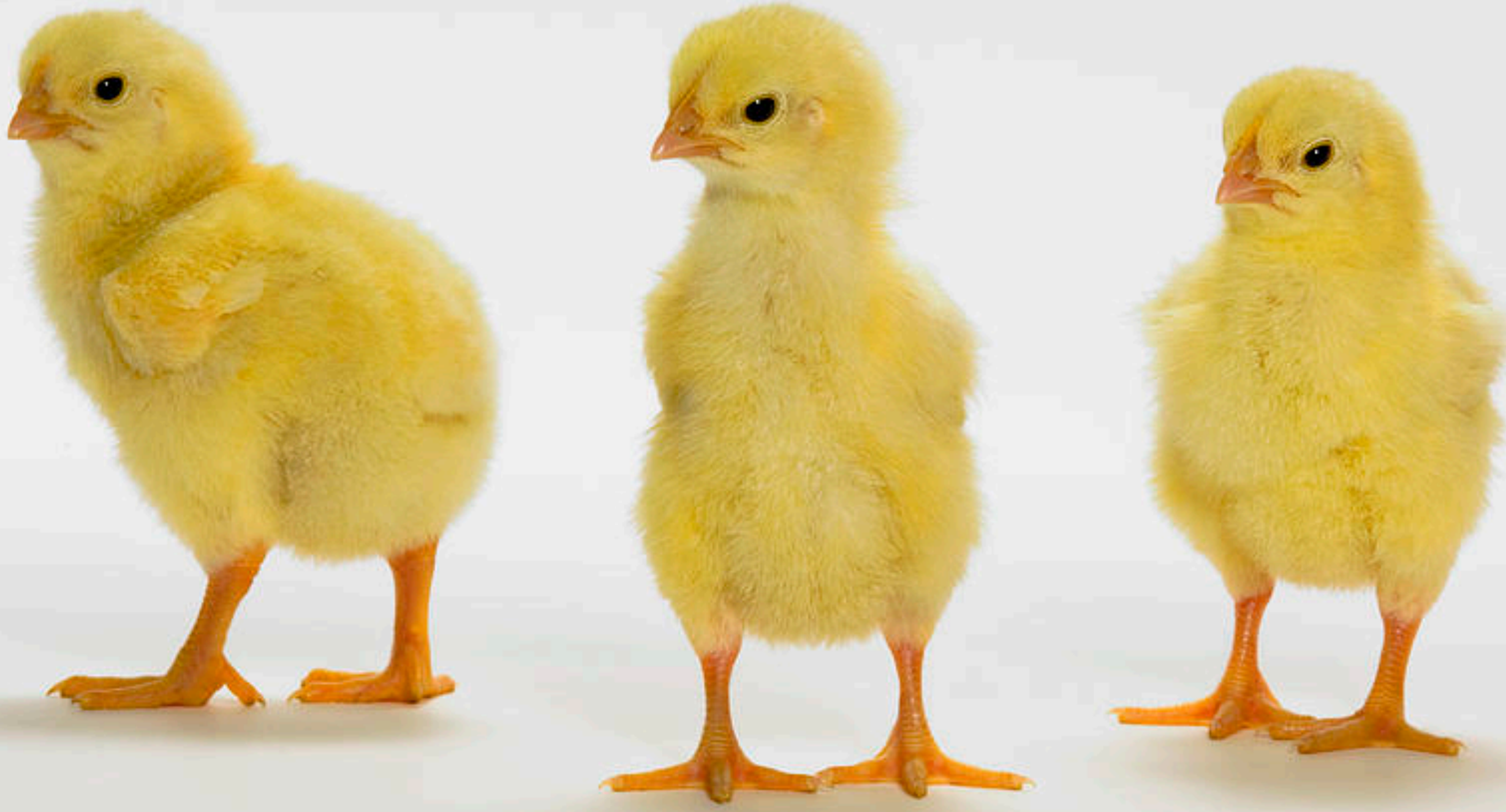
Steve... has been programming since before Alex was born. Steve laughs at Alex's youthful enthusiasm and remembers himself in Alex's shoes years ago.

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

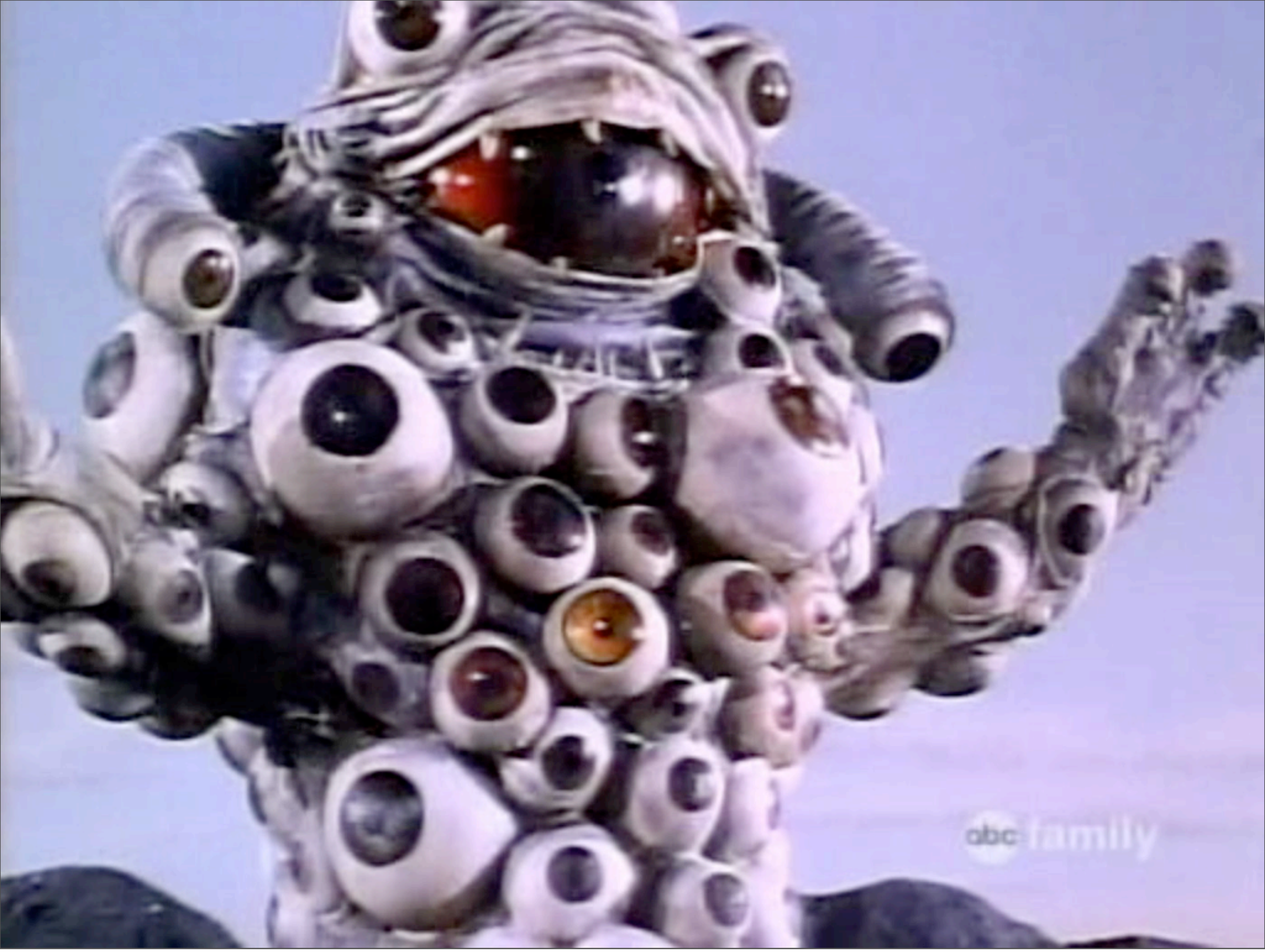
The show starts...

Tuesday, November 12, 13

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

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...

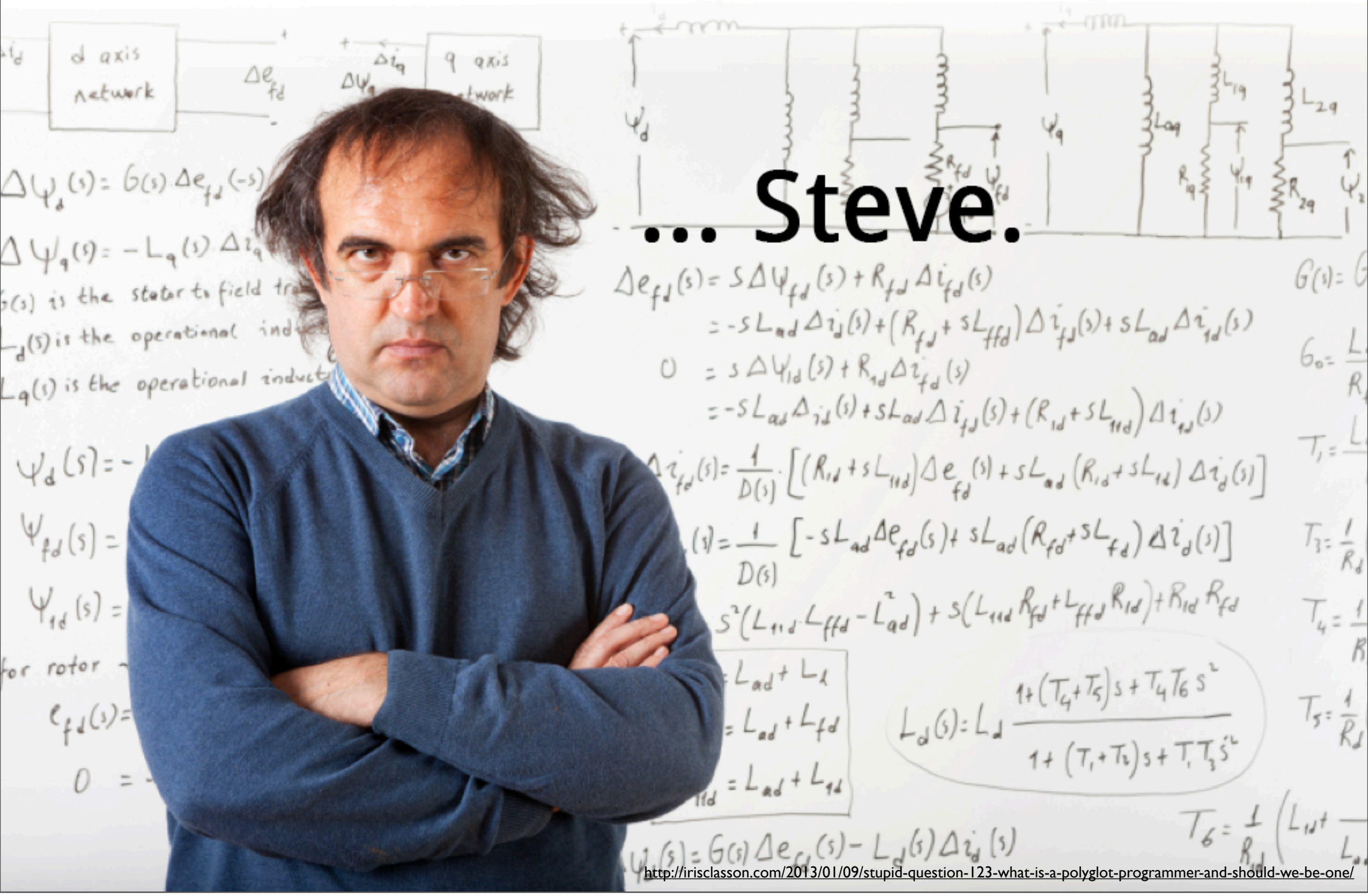
Project management on small teams can look like a post-it blizzard, email, issue trackers, stone tablets, and more. The key is to find something that truly works and buy into it completely.

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 them 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 our end product was going to manifest itself as. However, stalling for THE right answer was an obvious blocker to productivity and costs a lot of time and money in practice.

We recognized 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

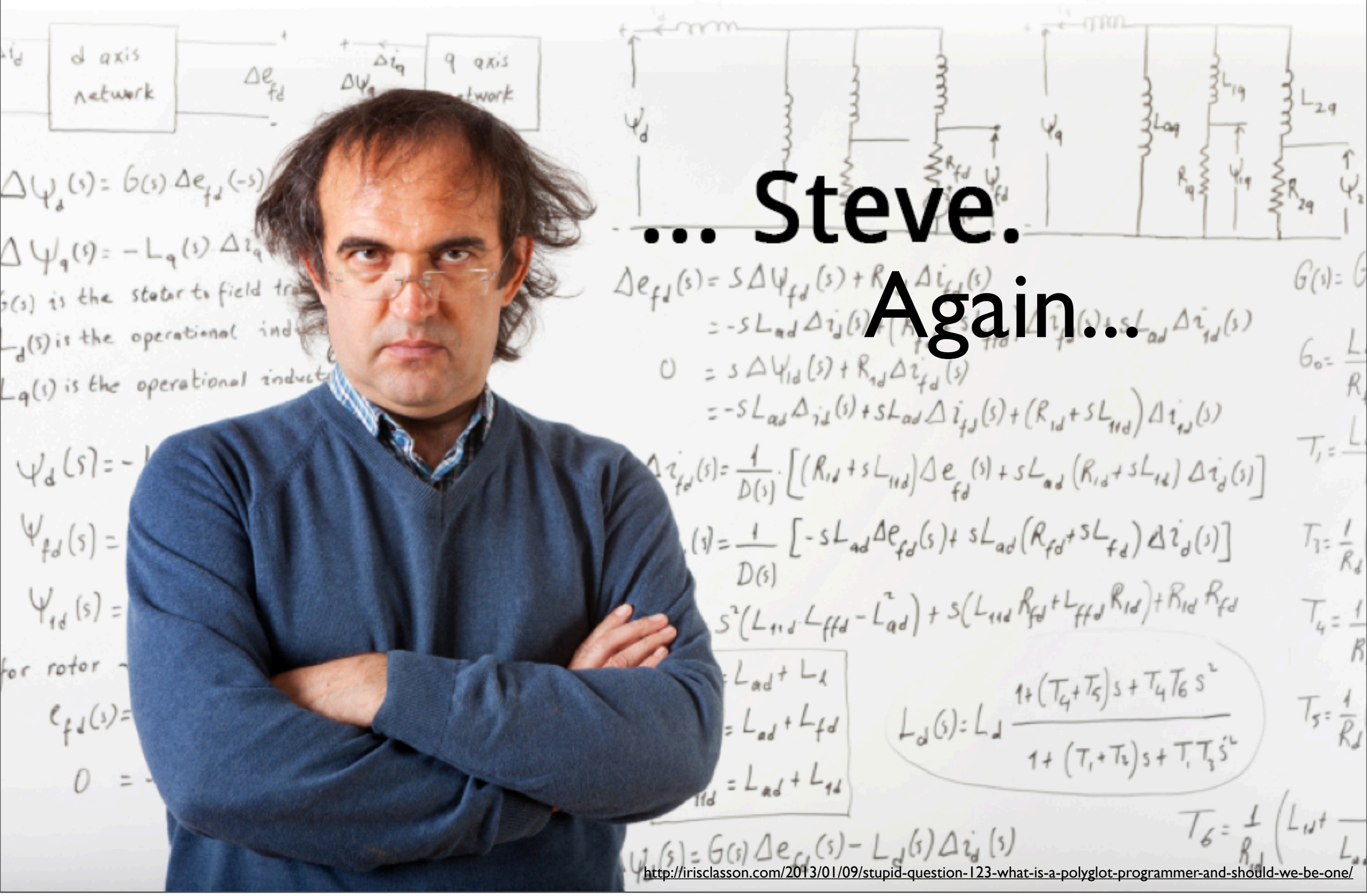
End-users are allowed to maintain HTML sites using whatever client they choose:

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



Risk in Loss of Domain Expertise

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

The development process has been waterfall-esque in that requirements are gathered, digested, and implemented in a black box with a sole developer taking responsibility for all tasks in the project.

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."

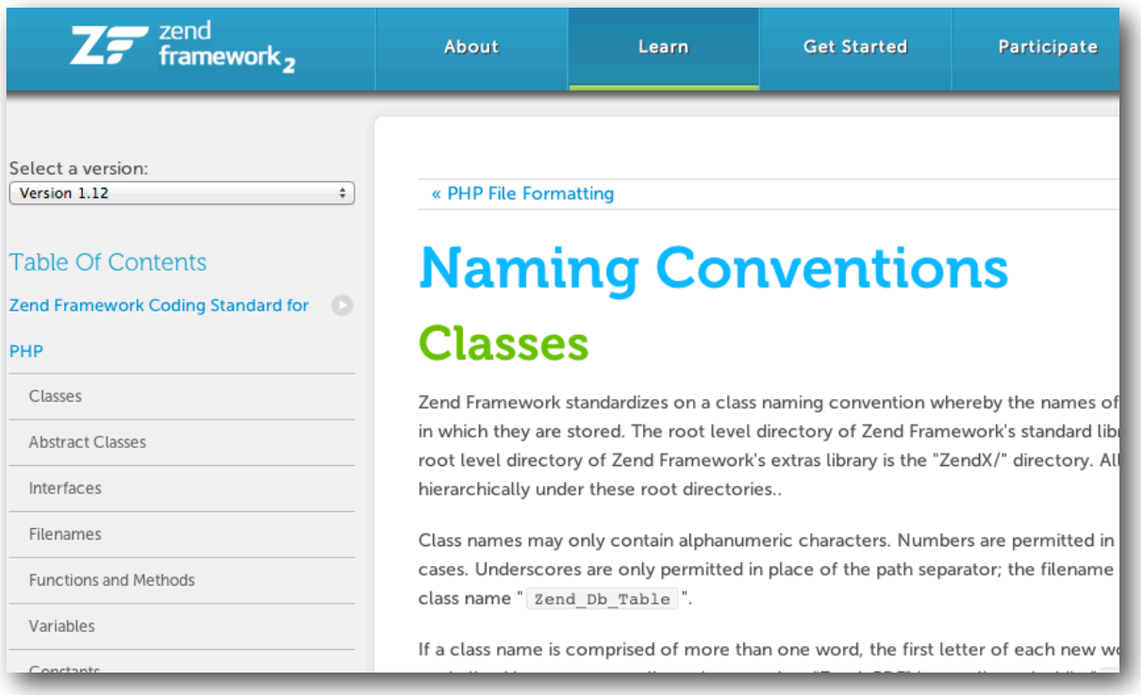
Make iterative advances.

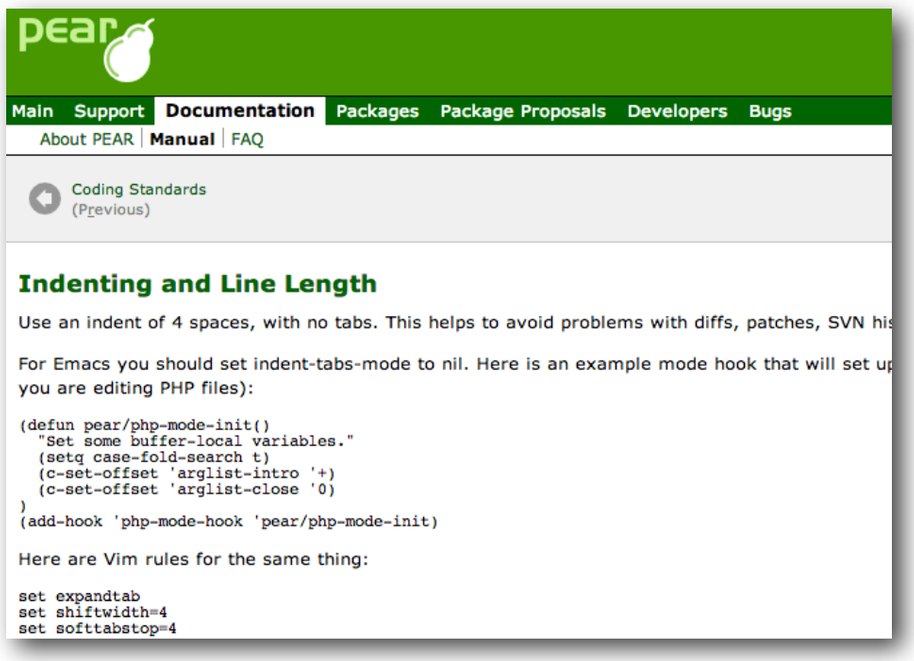
Make iterative advances in development projects, applying best practices as you go; always with an eye towards the next iteration.

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

PUBLIC ncsu-interop-group / **ws-standards**

Unwatch 8 | Star 4 | Fork 5

Browse Issues | Milestones | New Issue

← Back to issue list | Issue #14

mdwheele opened this issue 6 months ago

Proposed Web Services

Edit

No one is assigned | No milestone

Please leave a brief description of services you intend to implement or provide so we can start to look at common effort and things we can work on together. Stick close to the format:

Web Service Name

[Brief Description of Service]

Applications that might use service.

- App1
- App2
- App3

Open

7 comments

Labels

discussion

needs review

2 participants

Closed **mdwheele** closed the issue 6 months ago

Reopened **mdwheele** reopened the issue 6 months ago

bwdezend commented 6 months ago

Bad Traffic

A distributed blacklist/whitelist service. The service maintains a list of abusive networks, the duration of block, and the reporting host that match a given criteria. The service supports whitelists and a cumulative penalty box (search for the number of reported abuses for a given address over the last X time period). This allows the SFTP servers and Web servers to maintain their own lists of blocks, or share them out with other classes of hosts.

Applications that might use service

Tuesday, November 12, 13

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?

Tuesday, November 12, 13

Tuesday, November 12, 13