being good at software: a tragedy in three acts

matt stine (@mstine)
Global Field CTO / Chief Architect
http://www.mattstine.com

Pivotal
WARNING: this is a rant
probably no longer
three acts
Going Meta (Antipattern)

Never talk about the talk itself. The audience isn’t interested in mechanics; they came to hear about the interesting topic that you are uniquely equipped to expound upon. This would be akin to a theatrical (not a director’s cut) release of Star Wars Episode IV: A New Hope spending time discussing how George Lucas used a certain kind of lens, the temperature of lighting on the set, and the brand of makeup on the actors.
probably still a tragicomedy

because there will be tragedy...

...and there will be comedy
THIS IS FINE.
what is
“being good at software?”
principles > practices > tools
avoid the cycle:

(re)discovery

failure

flight
let's talk about principles
there are three constants in life...
change, choice and principles

— Stephen Covey
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.
paradox of choice
practices
if you’re still running the same process you ran 6 months ago, you’re not agile
the slider bar principle
is this service a microservice?

is this service HA?

is this service secure?
the answers to these questions are not binary...
...they are fuzzy
false dichotomies
monolith == evil
soa === evil
microservices == amazeballs
The Suck/Rock Dichotomy

Lots of people are passionate about software development (much to the confusion and chagrin of our significant others), and that unfortunately leads to what I call the “Suck/Rock Dichotomy”: everything in the software world either sucks or rocks, with nothing in between. While this may lead to interesting, endless debates (Emacs vs. vi, anyone?), ultimately it ill serves us as a community.

Having been in software communities for a while, I've seen several tribes form, thrive, then slowly die. It's a sad thing to watch a community die because many of the people in the community live in a state of denial: how could their wonderful thing (which rocks) disappear under this other hideous, inelegant, terrible thing (which sucks). I was part of the Clipper community (which I joined at its height) and watched it die rather rapidly when Windows ate DOS. I was intimately part of the Delphi community which, while not dead yet, is rapidly approaching death. When a community fades, the fanaticism of the remaining members increases proportionally for every member they lose, until you are left with one person whose veins stick out on their forehead when they try to proselytize people to join their tribe, which rocks, and leave that other tribe, which sucks.

Why is this dichotomy so stark in the software development world? I suspect a couple of root causes. First, because it takes a non-trivial time investment for proficiency in software tribes, people fear that they have chosen poorly and thus wasted their time. Perhaps the degree in which something rocks is proportional to the time investment in learning that technology. Second, technologists and particularly developers stereotypically tend to socialize via tribal ritual. How many software development teams have you seen that are not too far removed from fraternities? Because software is fundamentally a communication game, I think that the fraternal nature of most projects makes it easier to write good software. But tribal ritual implies that one of the defining characteristics of your tribe is the denigration of other tribes (we rock, they suck). In fact, some tribes within software seem to define themselves in how loudly they can say that everything sucks, except of course their beautiful thing, which rocks.

you are here

find your spot
the no free lunch principle
microservices will fix everything!
I'll keep saying this ... if people can't build monoliths properly, microservices won't help.

#qconlondon #DesignThinking #Modularity
I see you have a poorly structured monolith. Would you like me to convert it into a poorly structured set of microservices?

6:59 PM - 23 Feb 2015
What the hell have you built.

- Did you just pick things at random?
- Why is Redis talking to MongoDB?
- Why do you even *use* MongoDB?

[Red button: damnit]  [Green button: Nevermind FREE]
Microservices - Not A Free Lunch!

TUESDAY, APRIL 8, 2014 AT 8:54AM

This is a guest post by Benjamin Wootton, CTO of Contino, a London based consultancy specialising in applying DevOps and Continuous Delivery to software delivery projects.

Microservices are a style of software architecture that involves delivering systems as a set of very small, granular, independent collaborating services.

Though they aren't a particularly new idea, Microservices seem to have exploded in popularity this year, with articles, conference tracks, and Twitter streams waxing lyrical about the benefits of building software systems in this style.
YOU MUST BE AT LEAST 140 cm TALL TO RIDE NO 5G POWER
tradeoffs
the slider bars aren't independent
the
innovation
token
principle
Choose Boring Technology
March 30th, 2015

Probably the single best thing to happen to me in my career was having had Kellan placed in charge of me. I stuck around long enough to see Kellan's technical decisionmaking start to bear fruit. I learned a great deal from this, but I also learned a great deal as a result of this. I would not have been free to become the engineer that wrote Data Driven Products Now! if Kellan had not been there to so thoroughly stick the landing on technology choices.

Being inspirational as always.

http://mofunley.com/choose-boring-technology
choose amazon lambda
choose kafka
choose react
choose docker
choose boring technologies
known unknowns
unknown unknowns
optimize globally
how would you solve your current problem without changing your tech stack?
what about your current tech stack makes solving the problem impossible?
avoid the cycle:

(re)discovery

failure

flight
Welcome to Struts

Welcome to Struts! The goal of this project is to provide an open source framework for building Java web applications.

The core of the Struts framework is a flexible control layer based on standard technologies like Java Servlets, JavaBeans, ResourceBundles, and XML, as well as various Jakarta Commons packages. Struts encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm.

Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with standard data access technologies, like JDBC and EJB, as well as most any third-party packages, like Hibernate, iBATIS, or Object Relational Bridge. For the View, Struts works well with JavaServer Pages, including JSTL and JSF, as well as Velocity Templates, XSLT, and other presentation systems.

The Struts framework provides the invisible underpinnings every professional web application needs to survive. Struts helps you create an extensible development environment for your application, based on published standards and proven design patterns.

Struts is a project of the The Apache Software Foundation. The official Struts home page is at http://struts.apache.org/.

Struts is a volunteer project and all support for the framework is provided by unpaid volunteers. This documentation bundle and the mailing lists are the primary ways to learn how to use Struts. The next few pages are devoted to helping you understand what resources are available to you. Since Struts is a volunteer project, and our resources are limited, it is important that we first help you help yourself.

Struts in a Nutshell

A web application uses a deployment descriptor to initialize resources like servlets and taglibs. The deployment descriptor is formatted as a XML document and named “web.xml”. Likewise, Struts uses a configuration file to initialize its own resources. These resources include ActionForms to collect input from users, ActionMappings to direct input to server-side Actions, and ActionForwards to select output pages.
Why are you all here today?

Because Struts sucks?

Struts

webwork

Spring Framework

Tapestry

My Faces

a raible slide from 2005
Rod Johnson

"I wrote this book for architects and developers who have grown increasingly frustrated with traditional approaches to J2EE design, especially EJB. It shows what you can do right now to implement cleaner, more productive alternatives to EJB and move into the next era of web applications."

expert one-on-one™

J2EE™ Development without EJB™

Rod Johnson with Juergen Hoefler

Updates, source code, and Wrox technical support at www.wrox.com
spring!
RUN AWAY!!!
Web development that doesn’t hurt

Ruby on Rails is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.

Get Excited

Get Started

Get Better

Get Involved

“Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days.”

-Tim O'Reilly, Founder of O'Reilly Media

Read more quotes
Jim Weirich presents “Decoupling from Rails” at this month's Cincinnati Ruby Brigade user group.
Test-induced design damage

By David Heinemeier Hansson on April 29, 2014

"Code that's hard to test in isolation is poorly designed", goes a common TDD maxim. Isolation meaning free of dependent context and separated from collaborators, especially "slow" ones like database or file IO. The prevalent definition of "unit" in unit testing (though not everyone agrees with this).
Rails, You Have Turned into Java. Congratulations!

Posted on February 19, 2013 by Tim O'Brien

Frameworks atop Frameworks atop Opaque Hosting Platforms

Sure, Rails itself is straightforward, but the frameworks you slap on top of it can quickly become burdensome abstractions: RefineryCMS, Devise, Omniauth, Carrierwave, Unicorn, Rack Rewrite, Fog, New Relic, Foreman, AMQP, and Honeybadger, not to mention the extra magic that Heroku gems throw into the mix (backups and other fun).
RUN AWAY!!!
Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Current Version: v0.12.7

Download for OS X (x64)
left-pad
String left pad

left-pad
P.S: I've unpublished it from NPM.

Install

$ npm install azer/left-pad
What The Left-Pad Incident Teaches Us About JavaScript Dependencies

written by Dan Rowinski

The power and pitfalls of open source programming.
Welcome to left-pad.io!

## History

On March 22nd 2016, a terrible tragedy befell the Node.js community. A popular microframework for robust string expansion, `left-pad`, was removed from npmjs.com.

This resulted in broken deploys worldwide, and untold billions of dollars in economic ecosystem of String Manipulation as a service.

## A microservice saviour appears
never forget
___________ is too hard!
___________ doesn't work!
___________ doesn't scale!
___________ is dead!
"90% of tech is tribalism and fashion."
@littleidea #cfad
Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Manifesto for Software Craftsmanship

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also well-crafted software

Not only responding to change,
but also raising the bar.

The Reactive Manifesto

Published on September 16 2014. (v2.0)

Organisations working in disparate domains are independently discovering patterns for building software that look the same. These systems are more robust, more resilient, more flexible and better positioned to meet modern demands.

These changes are happening because application requirements have changed dramatically in recent years. Only a few years ago a large application had tens of servers, seconds of response time, hours of offline maintenance and gigabytes of data. Today applications are deployed on everything from mobile devices to cloud-based clusters running thousands of multi-core processors. Users expect millisecond response times and 100% uptime. Data is measured in petabytes - requirements simply not met by yesterday’s software.

An approach to systems architecture is needed, and we call this architecture Resilient, Elastic and Message Driven. We call
no tribe left behind
embracing constraints
doing hard things
from “hello world”
to
real world
No Silver Bullet

Essence and Accidents of Software Engineering

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill

Fashioning complex conceptual constructs is the essence; accidental tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically pass through them to strike them dead. A similar false hope is still enjoyed by the software manager who dreams of a silver bullet—something to make software costs drop as rapidly as computer hardware costs do.

But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity. In this article, I shall try to show why, by examining both the nature of the software problem and the properties of the bullets proposed.

But at least one thing is certain: although we see no as yet unmet challenge, there are no one-silver bullet solutions. We must learn to cope with the accidents, not to avoid them. We must fashion complex conceptual constructs in a way that makes accidental tasks arise in language, not in hardware.

Does it have to be hard?—Essential difficulties

Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any—inventions that will do for software productivity, reliability, and simplicity what electronics, transistors, and large-scale integration did for computer hardware.
break the cycle:

(re)discovery

failure

flight
a call to action
focus on principles
think in spectra
consider tradeoffs
choose boring technology
stop abandoning tribes
thank you

matt stine (@mstine)

http://www.mattstine.com

https://www.safaribooksonline.com/search/?query=matt%20stine
GREAT INDIAN DEVELOPER SUMMIT 2019
Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin