Thinking Architecturally

Nathaniel T. Schutta
@ntschutta
ntschutta.io
Thinking Architecturally
Lead Technical Change Within Your Engineering Team
Nathaniel Schutta

https://content.pivotal.io/ebooks/thinking-architecturally
Architecting is hard...
Many competing agendas.
Technology changes.
Constantly.
Feature not a bug.
Keeps things interesting...
We want to avoid legacy platforms.
But we can’t change things every few months.
“Our app has 4 different UI frameworks…”
Developers kept chasing the new hotness.
How do we avoid that?
How do we evaluate new technology?
I have no idea what language/framework/platform is “next”.
No one does.
But I can guarantee you this much:
It will be different than what we use today.
Five years from now we will be using something that isn’t invented yet.
Chasing the new thing.
Technology changes.
Constantly.
Tempting to always chase the “new hotness”.
Bleeding edge.
It’s fun!
Part of being in this industry.
The most important skill to have as a programmer is the ability to teach yourself new things effectively and efficiently.

You're going to be constantly growing and picking up new technologies. The ability to do that is more important than any individual tool or technology.

11:00 AM · Feb 28, 2019 · TweetDeck
422 Retweets 1.5K Likes
"Education isn't something you can finish."

-- Isaac Asimov (1920 - 1992)
Our understanding constantly evolves.
There are things you understand well enough to have strong opinions on, things you know just well enough to know what others' opinions are, and things where you can have no opinion. The complication in tech is that things move between these categories all the time.

1:42 PM · Dec 30, 2017

70 Retweets 218 Likes

In tech, there are:
- Things you're starting to learn about
- Things you know well
- Things you're no longer following
- Things you were an expert on but moved on from years ago and half-forgot

In tech, the point that you really understand something is the point it's becoming irrelevant

For example, I used to know all about radio networks, mobile networks and the Japanese mobile market (Mode! and PHS). Now, none of that matters. Most questions about smartphones are now moving rapidly down the list as well. On to new questions.
Let’s be honest...
Developers have opinions!
Often *very* strong opinions.
There's a big difference between knowing the name of something and knowing something.
Maybe we fear old things?
I've become old.

I've become wary of new technology and new frameworks.

I've become the grumpy fellow in the back who says, after someone demos GraphQL: “Isn’t that just like SQL queries from the front end... like PHP?”

Is this what is feels like to be a “senior developer”?
It's said that "senior" developers tend to be afraid of learning new things. This has always struck me as absurd, because who hasn't learned a ton of new things over the course of 10+ years of programming? Almost impossible not to.

It almost feels as though less experienced developers are projecting their own fears onto others with more experience.

I get the sense that many programmers are afraid to learn *old* things. As if the older things are impossibly arcane and complex, compared to the supposed simplicity and elegance of new tech.

They have a strong preference for the new. They'd rather use unfinished, buggy, unproven technology than older, refined, stable, tested tech. Even if it reinvents or indeed breaks the wheel.

So now there's a excitement about a "UnicornKit" that combined UIKit and AppKit into one. But do people realize that unless UnicornKit is simply UIKit, you're going to have to learn something else anyway?

Why is learning UnicornKit harder than learning AppKit? Many developers have successfully written both Mac and iOS apps. It's not as hard as you may think if you have no experience with AppKit.

Also, any new technology is going to be chock full of bugs. AppKit has its bugs, yes, but it's assuredly vastly more reliable and stable than whatever you're working with now.
Predictable hype cycle.
Year 01: this technology is radical!

Year 02: wow, it's hard to use, amiright?!

Year 03: you should never actually use this, esp. if you're just running PHP and a database.

Year 04: Machine learning and blockchain!

GOTO Year 02;
12:35 PM · Feb 13, 2018
13 Retweets 26 Likes
How do we know where not to use a technology?
Trial and error.
Developers tend to get bored quickly.
Learning keeps it fresh.
But we have to deliver business value.
Can’t do that if we’re always experimenting.
Have to commit at some point.
Develop some expertise.
Bleeding edge...means you will bleed!
Joe Armstrong
@joeerl

Dear <redacted>

Thank you for the latest version of your wonderful program with many new features.

Breaking my old code that worked in your previous version is a great idea and will provide me with many hours of fun searching for a work around.

Tearing out my hair
@joeerl

6:28 AM · Nov 15, 2017

73 Retweets 221 Likes

https://mobile.twitter.com/joeerl/status/930774515512201216
Pioneers...the ones with arrows in their backs.
What is your strategy?
How do we avoid dead platforms?
Without constantly changing direction?
Strategy.
Hope is not a strategy!
But it is what rebellions are built on.
We need to be deliberate.
There are a lot of bits out there...
New languages, techniques, approaches.
How do you keep up?
Develop a routine.
Block out Friday afternoon. Tuesday over lunch. Whatever fits.
Consider “morning coffee”.
Take 15-30 minutes in the morning to peruse the tech news.
Before the day gets away from you...
Attention is precious.
Attention is a resource. It doesn't scale.

10:42 AM - Mar 7, 2019 - TweetDeck

59 Retweets 166 Likes
“Attention is a bit like real estate, in that they're not making any more of it. Unlike real estate, though, it keeps going up in value.”

— Seth Godin

Don’t waste it.
Be selective.
Can’t read it all.
In fact, you’ll miss almost everything.
We cannot adopt every new thing.
How do we know where to invest our time?
Hacker’s Radar?

http://www.paulgraham.com/javacover.html
“I have a hunch that [Java] won't be a very successful language.”
Never written a line of Java, glanced at some books.
Need more than just a hunch.
“Judging Covers” can be a useful filter.
But beware bias.
Where is the community?
Are you skating to where the puck *was*?
Technology Radar.

https://www.thoughtworks.com/radar
TECHNOLOGY RADAR VOL.21
An opinionated guide to technology frontiers

Select an area to explore

- Techniques
- Tools
- Platforms
- Languages & Frameworks

Download Vol.21
Subscribe to Technology Radar

THEMES FOR THIS EDITION

Cloud: Is More Less?
Cloud providers rush new services to market in a competitive frenzy. Beware the rough edges and friction of early adoption.

Protecting the Software Supply Chain
Modern delivery pipelines protect more and more aspects of creating software artifacts as we evolve toward governance as

Interpreting the Black Box of ML
The power of machine learning (ML) matches its inscrutability. Explainability is growing in importance when choosing

Software Development as a Team Sport
Innovation thrives by pulling separate specializations into collaborative and cross-functional “10x teams.”
Remember Google’s 20% time?
Fallen out of favor in some circles...
Innovation Fridays.
Could you carve out Friday afternoons?
How about Tuesday Tech Talks?
Architectural Briefings.

https://github.com/stuarthalloway/presentations/wiki/Architectural-Briefings
One person does some research, presents to the team.
And no, you don’t need to be an architect to present!
Why should we use X?
What do you need to know to answer the “why”?
What do you need to know in order to use X?
Keep it short - 45 minutes.
Not a how to.
Beyond the initial documentation.
These are participatory events!
Attendees should be taking notes.
Asking questions.
Using their own experiences.
Do you agree? Why or why not?
By the way, you are up next week...
Pass the briefing filter?
Hands on time.
Workshop it.
Couple of hours.
A few exercises.
Focus on how to, simple setup.
Pass the hands on filter?
Time to trial it in the organization.
Real project work that is a good fit.
Probably not a “bet the company” project though!
The new hotness is not our only concern though.
Need to stay current on the things we are using day in day out.
The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

What leaks in practice?

We have tested some of our own services from attacker’s perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. Fixed OpenSSL has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.
Equifax Hackers Exploited Months-Old Flaw

by BEN POPKEN

Equifax announced late Wednesday that the source of the hole in its defenses that enabled hackers to plunder its databases was a massive server bug first revealed in March.

For the rest of the IT world, fixing that flaw was a "hair on fire moment," a security expert said, as companies raced to install patches and secure their servers. But at Equifax, criminals were able to pilfer data from mid-May to July, when the credit bureau says it finally stopped the intrusion.

"We know that criminals exploited a U.S. website application vulnerability," Equifax said in an update on its website Wednesday night. "The vulnerability was Apache Struts CVE-2017-5638." Equifax said it was working with a leading cybersecurity firm, reported to be Mandiant, to investigate the breach. Mandiant declined an NBC News request for comment.

Related: The One Move to Make After Equifax Breach

The Apache Software Foundation, which oversees the Apache Struts project, said in a press release Thursday that a software update to patch the flaw was released on May 20.
Most of the Fortune 100 still use flawed software that led to the Equifax breach

Zack Whittaker
@zackwhittaker / 1 week ago

Almost two years after Equifax's massive hack, the majority of Fortune 100 companies still aren't learning the lessons of using vulnerable software.

In the last six months of 2018, two-thirds of the Fortune 100 companies downloaded a vulnerable version of Apache Struts, the same vulnerable server software that was used by hackers to steal the personal data on close to 150 million consumers, according to data shared by Sonatype, an open-source automation firm.

That's despite almost two years' worth of patched Struts versions being released since the attack.

Sonatype wouldn't name the Fortune 100 firms that had downloaded the

Exactis said to have exposed 340 million records, more than Equifax breach

We hadn't heard of the firm either, but it had data on hundreds of millions of Americans and businesses and leaked it, according to Wired.

BY ABRAR AL-HEETI / JUNE 28, 2018 10:14 AM PDT
Marriott hack hits 500 million Starwood guests

© 30 November 2018

Technology

The records of 500 million customers of the hotel group Marriott International have been involved in a data breach.

The hotel chain said the guest reservation database of its Starwood division had been compromised by an unauthorised party.

It said an internal investigation found an attacker had been able to access the information between mid-2014 and September 2018.

The data breach, which is one of the largest in recent years, means that a vast amount of personal information about millions of people was potentially at risk.

The records included names, addresses, email addresses and phone numbers.

Marriott said it had stopped the hackers' access to the information as soon as it detected the breach.

It said that anyone affected by the data breach would be given a year's free identity theft protection services.
Oops.
Don’t think you’re a target?
The Three Rs of Enterprise Security: Rotate, Repave, and Repair
At high velocity, the three Rs starve attacks of the resources they need to grow. It’s a complete 180-degree change from the traditional careful aversion to change to mitigate risk. Go fast to stay safer—in other words, speed reduces risk.

—Justin Smith
What is your patching strategy?
What version of X are you on?
Some organizations have a policy of N or N-1.
Do they measure it? Do they enforce it?
What needs to change in your culture to stay at N?
What hurts more? Changing your patching strategy?
Or being on the receiving end of the latest “largest hack ever”?
Pros and Cons.
Every technical choice involves tradeoffs.
When we find ourselves presented with technology that promises to offer us drastic improvements, we need to look for the trade-offs.

— Susan J. Fowler
Production-Ready Microservices
Essence of design.
To paraphrase Harry Truman...
Give me a one handed technologist.
Should we use React or Angular?
Should we refactor to microservices?
Should we be on prem or public cloud?
any decent answer to an interesting question begins, "it depends..."

10:45 AM - 6 May 2015

540 Retweets 380 Likes
In many cases?
Balancing those opposing forces is the art of architecture.
No tech is perfect, don’t pretend it is.
Acknowledge the negatives.
What do you like about it?
What don’t you like about it?
What would you add?
What would you remove?
King of Java for a day...
You haven't mastered a tool until you understand when it should not be used.

Kelsey Hightower
@kelseyhightower

9:02 AM · Feb 13, 2018

1,989 Retweets 5,327 Likes
How does it stack up to alternatives?
The spreadsheet approach.
Options across the top.
Criteria down the left.
Criteria can be weighted.
Harvey balls.

http://en.wikipedia.org/wiki/Harvey_Balls
How closely does it map to the criteria?
Very effective...
<table>
<thead>
<tr>
<th>Feature</th>
<th>Angular</th>
<th>React</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>●</td>
<td>⌁</td>
</tr>
<tr>
<td>Community</td>
<td>●</td>
<td>⌁</td>
</tr>
<tr>
<td>Committer diversity</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Codebase</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Testability</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Update history</td>
<td>⌁</td>
<td>⌁</td>
</tr>
<tr>
<td>Maturity</td>
<td>●</td>
<td>⌁</td>
</tr>
<tr>
<td></td>
<td>Angular</td>
<td>React</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>Stability</td>
<td>★★★</td>
<td>★★</td>
</tr>
<tr>
<td>Extensibility</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Support</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Training</td>
<td>★★★</td>
<td>★★★</td>
</tr>
<tr>
<td>Hiring</td>
<td>★★★</td>
<td>★★</td>
</tr>
<tr>
<td>Corporate fit</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Usage</td>
<td>★★★</td>
<td>★★★</td>
</tr>
</tbody>
</table>
What criteria should you use?
How should they be weighted?
Up to you.
You can tip the scales...
Usually backfires.
Quality Attributes.
Sometimes called non-functional requirements.
Or quality goals, constraints, quality of service goals...
Cross-functional requirements.
Architecturally significant requirements.
The “ilities”!
Customers usually focus on functionality.
It’s what they “see” after all.
Obviously important we meet their needs!
But we have to look beyond that.
Vital that we focus on quality attributes.
Service level objectives if you will.
What are some quality attributes?
Maintainability.
Scalability.
Reliability.
Security.
Deployability.
Simplicity.
Usability.
Compatibility.
Fault tolerance.
Modularity.
The list goes on and on on!

What quality attributes do you focus on?
any decent answer to an interesting question begins, "it depends..."

10:45 AM - 6 May 2015

540 Retweets  380 Likes
Depends a lot on the type of software you build!
We don’t get to turn every knob to 11 do we?
Inverse relationships.
Maximizing one may minimize another.
Security and usability for instance.
It’s a balance.
Some are obvious to our customers.
If the systems won’t support the user load...
The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

What leaks in practice?

We have tested some of our own services from attacker’s perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. Fixed OpenSSL has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.
Equifax Hackers Exploited Months-Old Flaw

by BEN POPKEN

Equifax announced late Wednesday that the source of the hole in its defenses that enabled hackers to plunder its databases was a massive server bug first revealed in March.

For the rest of the IT world, fixing that flaw was a “hair on fire moment,” a security expert said, as companies raced to install patches and secure their servers. But at Equifax, criminals were able to pilfer data from mid-May to July, when the credit bureau says it finally stopped the intrusion.

"We know that criminals exploited a U.S. website application vulnerability," Equifax said in an update on its website Wednesday night. "The vulnerability was Apache Struts CVE-2017-5638." Equifax said it was working with a leading cybersecurity firm, reported to be Mandiant, to investigate the breach. Mandiant declined an NBC News request for comment.

Related: The One Move to Make After Equifax Breach

The Apache Software Foundation, which oversees the Apache Struts project, said in a press release Thursday that a software update to patch the flaw was released on March 22.
Most of the Fortune 100 still use flawed software that led to the Equifax breach

Zack Whittaker
@zackwhittaker  /  1 week ago

Almost two years after Equifax’s massive hack, the majority of Fortune 100 companies still aren’t learning the lessons of using vulnerable software.

In the last six months of 2018, two-thirds of the Fortune 100 companies downloaded a vulnerable version of Apache Struts, the same vulnerable server software that was used by hackers to steal the personal data on close to 150 million consumers, according to data shared by Sonatype, an open-source automation firm.

That’s despite almost two years’ worth of patched Struts versions being released since the attack.

Sonatype wouldn’t name the Fortune 100 firms that had downloaded the vulnerable version of Struts. It also couldn’t say how many companies had deployed the vulnerable software since the attack. However, it did say that the new version of Struts has been downloaded more than 3 million times as of last month.
Exactis said to have exposed 340 million records, more than Equifax breach

We hadn't heard of the firm either, but it had data on hundreds of millions of Americans and businesses and leaked it, according to Wired.

BY ABRAR AL-HEETI / JUNE 28, 2018 10:14 AM PDT
Technology

Marriott hack hits 500 million Starwood guests

© 30 November 2018

The records of 500 million customers of the hotel group Marriott International have been involved in a data breach.

The hotel chain said the guest reservation database of its Starwood division had been compromised by an unauthorised party.

It said an internal investigation found an attacker had been able to access the database and copy personal details, including passport numbers.
Fairly easy to convince people of the importance.
Others - are “invisible”.
Or at least harder to see.
Maintainability and simplicity for instance.
How do we get the decision makers to buy in?
What techniques can we use to influence them?
Outline the benefits.
Find common ground.
Avoid aggression.
Listen.
Have a conversation!
Can be hard to convince people!
Two approaches...
Find the influencers.
Influence the influencers.
Approach as equals.
Rely on the strength of your ideas and your reputation.
Your reputation speaks for you when you aren’t there.
Not sure what your rep is?
May not like the answer...
But you can work to change it.
Find common ground.
Reciprocity rules...
Be helpful.
Be respectfulful.
Research your ideas.
Use trusted sources.
Recruit credible allies.
Nothing wrong with bringing help!
Speak their language.
Avoid techno babble.
You may be impressed by the jargon...
Most customers aren’t.
What resonates in your organization?
Cost savings?
Developer productivity?
Speed to market?
Shape your approach accordingly.
The iDon’tDrive System.
Self driving cars are all the rage these days.
Let’s pretend our company has the answer!
VCs can’t give us money fast enough.
Customers are flocking to our stylish website.
Our team has been tasked to build a backend system.
The cars generate a lot of data.
Battery level, health of the engine, maintenance.
Basics

- Car “phones home” on a regular basis.
- Sends a standard data payload including VIN.
- Demand is extremely high.
- Expect millions of cars on the road in the next 3 years.
- Marketing is full of great ideas...
- System must be available 24x7.
Customer Facing

- Web interface as well as mobile apps.
- Allow an owner to check the stats of their car.
- When does it need maintenance? How’s the battery? Etc.
- Must be secure - only access ‘your’ car.
- Allow owner to summon the car to their present location.
- Push notifications for maintenance, low battery etc.
Company Facing

- Fleet generates a lot of information that can be mined.
- Data must be anonymized.
- Ensure only authorized users have access.
- Audit access to customer data.
- Push software updates to the car.
- Push recall/update information to customers.
What quality attributes matter most here?
What words/phrases stand out to you?
Basics

- Car “phones home” on a regular basis.
- Sends a standard data payload including VIN.
- Demand is extremely high
- Expect millions of cars on the road in the next 3 years.
- Marketing is full of great ideas...
- System must be available 24x7.
Customer Facing

- Web interface as well as mobile apps.
- Allow an owner to check the stats of their car.
- When does it need maintenance? How’s the battery? Etc.
- Must be secure - only access ‘your’ car.
- Allow owner to summon the car to their present location.
- Push notifications for maintenance, low battery etc.
Company Facing

- Fleet generates a lot of information that can be mined.
- Data must be anonymized.
- Ensure only authorized users have access.
- Audit access to customer data.
- Push software updates to the car.
- Push recall/update information to customers.
How would you rank them?
Depends on the perspective of the system!
From a driver/owner...
<table>
<thead>
<tr>
<th>Rank</th>
<th>Quality Attribute</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Usability</td>
<td>Customers will not read a manual on how to use the iDon’tDrive system.</td>
</tr>
<tr>
<td>2</td>
<td>Availability</td>
<td>Owner must be able to summon a car 24x7.</td>
</tr>
<tr>
<td>3</td>
<td>Security</td>
<td>Owner must be confident that only then can access their car.</td>
</tr>
<tr>
<td>4</td>
<td>Reliability</td>
<td>System must work as expected when called upon to maintain confidence in the system.</td>
</tr>
</tbody>
</table>
From a service center...
<table>
<thead>
<tr>
<th>Rank</th>
<th>Quality Attribute</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Security</td>
<td>Only authorized users should have access to the system.</td>
</tr>
<tr>
<td>2</td>
<td>Auditability</td>
<td>System should audit access to determine appropriate usage of the system.</td>
</tr>
<tr>
<td>3</td>
<td>Efficiency</td>
<td>System should be minimize the time service representatives need to find information.</td>
</tr>
<tr>
<td>4</td>
<td>Usability</td>
<td>System should require minimal training and allow new service reps to be productive quickly.</td>
</tr>
</tbody>
</table>
What architectural decisions result from that?
<table>
<thead>
<tr>
<th>Rank</th>
<th>Quality Attribute</th>
<th>Decision(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Usability</td>
<td>UX designers will be engaged and ensure the design requires no training to use.</td>
</tr>
<tr>
<td>2</td>
<td>Availability</td>
<td>System will be geographically dispersed across multiple data centers. Zero downtime deploys will be utilized.</td>
</tr>
<tr>
<td>3</td>
<td>Security</td>
<td>Standard three zone security will be employed. System will encrypt personally identified information and follow all security standards.</td>
</tr>
<tr>
<td>4</td>
<td>Reliability</td>
<td>System will be geographically dispersed across multiple data centers.</td>
</tr>
</tbody>
</table>
Questions?
Your turn!
What quality attributes matter most for your kata?
How would you rank them?
What architectural decisions might result?
Establish principles.
We can’t be everywhere...
We can’t be involved with every decision.
We can establish principles.
Guard rails.
Guide posts.
North stars.
Create the environment within which our projects can thrive.
But how do we know if projects are following our principles?
Fitness functions.
We’re all familiar with the second law of thermodynamics...
Otherwise known as a teenagers bedroom.
The universe really wants to be disordered.
Software is not immune from this!
We go through the thoughtful effort to establish an architecture...
How do we maintain it?
We can’t spend every minute of every day on every project.
How do we ensure teams continue to make good decisions?
We cannot predict the future.
I don't make many predictions...but I do study history....
That’s not entirely true.
One constant - change.
Architecture is often defined as the decisions that are hard to change.
Or the decisions we wish we got right.
But we *know* things will change!
Isn’t this approach anti agile?
Contributing factor to the “we’re agile, we don’t have architects” theory.
You definitely have people making architectural decisions!
Sure hope they are making good ones...
You’ll know in a year or two.
“Our app has 4 different UI frameworks...”
What do we do about that?
Maybe we should change our assumptions.
1. Build for now
2. Choose tech based on ability to evolve
3. Evolve one use case at a time

-- @randyshoup

Evolutionary Architecture – Randy Shoup – Medium
medium.com

10:55 AM · Jan 5, 2018
455 Retweets 813 Likes

https://mobile.twitter.com/martinfowler/status/949323421619548161
What if our architectures expected to change?
Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE

Neal Ford, Rebecca Parsons & Patrick Kua
An evolutionary architecture supports guided, incremental change across multiple dimensions.

— Building Evolutionary Architectures
Some architectures are more evolvable than others...
Components are deployed, features are enabled via toggles.
Allows us to change incrementally.
Also perform hypothesis driven development!
But how do we ensure the architecture still meets our needs?
How do we know if a solution violates part of the architecture?
Fitness functions!
Concept comes from evolutionary computing.
Is this mutation a success?
Are we closer to or further from our goal?
For architecture, it is all about protecting the ilities.
And balancing the tradeoffs.
We want to capture and preserve the key architectural characteristics.
First, we need to identify those key measures for project success.
Service Level Indicators if you will.
What can we measure?
Sometimes we let what we can measure dictate too much...
Just because we can measure it doesn’t mean it matters!
Lines of code anyone?
Once we have our metrics, we can set some goals.
Service Level Objectives.
SLO !== SLA!
Now we can create a fitness function!
Basically, a set of tests we execute to validate our architecture.
How close does this particular design get us to our objectives?
Ideally, all automated. But we may need some manual verifications.
For example...
All service calls must respond within 100 ms.
Cyclomatic complexity shall not exceed X.
There are no cyclic dependencies.

clarkware.com/software/JDepend.html
Directionality of imports.
Consumer Driven Contracts.

https://martinfowler.com/articles/consumerDrivenContracts.html
Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach. Currently Spring Cloud Contract consists of the Spring Cloud Contract Verifier project.

Spring Cloud Contract Verifier is a tool that enables Consumer Driven Contract (CDC) development of JVM-based applications. It is shipped with Contract Definition Language (DSL) written in Groovy or YAML. Contract definitions are used to produce following resources:

- by default JSON stub definitions to be used by WireMock (HTTP Server Stub) when doing integration testing on the client code (client tests). Test code must still be written by hand, test data is produced by Spring Cloud Contract Verifier.
- Messaging routes if you're using one. We're integrating with Spring Integration, Spring Cloud Stream and Apache Camel. You can however set your own integrations if you want to.
- Acceptance tests (by default In JUnit or Spock) used to verify if server-side implementation of the API is compliant with the contract (server tests). Full test is generated by Spring Cloud Contract Verifier.

Spring Cloud Contract Verifier moves TDD to the level of software architecture.

To see how Spring Cloud Contract supports other languages just check out this blog post.

**Features**

When trying to test an application that communicates with other services then we could do one of two things:

- deploy all microservices and perform end to end tests
- mock other microservices in unit / integration tests

Both have their advantages but also a lot of disadvantages. Let’s focus on the latter.

**Deploy all microservices and perform end to end tests**

Advantages:
Unit test your Java architecture

Start enforcing your architecture within 30 minutes using the test setup you already have.

Start Now

ArchUnit is a free, simple and extensible library for checking the architecture of your Java code using any plain Java unit test framework. That is, ArchUnit can check dependencies between packages and classes, layers and slices, check for cyclic dependencies and more. It does so by analyzing given Java bytecode, importing all classes into a Java code structure. You can find examples for the current release at ArchUnit Examples and the sources on GitHub.

News

Mar 31, 2019 – New release of ArchUnit (v0.10.2)

Mar 16, 2019 – New release of ArchUnit (v0.10.1)

Mar 16, 2019 – New release of ArchUnit (v0.10.0)
A fluent API for .Net that can enforce architectural rules in unit tests.
Today we’re releasing Scientist 1.0 to help you rewrite critical code with confidence.

As codebases mature and requirements change, it is inevitable that you will need to replace or rewrite a part of your system. At GitHub, we’ve been lucky to have many systems that have scaled far beyond their original design, but eventually there comes a point when performance or extensibility break down and we have to rewrite or replace a large component of our application.

Problem

A few years ago when we were faced with the task of rewriting one of the most critical systems in our application — the permissions code that controls access and membership to repositories, teams, and organizations — we began looking for a way to make such a large change and have confidence in its correctness.

There is a fairly common architectural pattern for making large-scale changes known as Branch by Abstraction. It works by inserting an abstraction layer around the code you plan to change. The abstraction simply delegates to the existing code to begin with. Once you have the new code in place, you can flip a switch in the abstraction to begin substituting the new code for the old.
Performance - average and maximum response times.
Scalability - average response times across number of users and requests.
Number of timeouts and application faults.
Nearing the next price tier with our cloud provider.
Hard failure of an application will spin up a new instance.
Alert when things start to go out of band!
Chaos Engineering.
Your turn!
What fitness functions would you recommend for your kata?
Don’t worry about implementing them!
Fitness functions remind us what is important in our architecture.
Informs our thinking about tradeoffs.
Different categories of fitness functions.
Atomic vs. Holistic.
Some characteristics must be tested in isolation...others cannot.
Holistic fitness functions test combined features.
We can’t test every possible combination!
Must be selective, driven by the value of the architectural characteristic.
Triggered vs. Continual.
Must consider frequency of execution.
Fitness functions can be triggered by something - checkin, QA pass...
Continual tests are just that.
Monitoring Driven Development!

http://benjiweber.co.uk/blog/2015/03/02/monitoring-check-smells/
Static vs. Dynamic.
Static tests have a fixed result - they either pass or they fail.
Nearly any test based on a metric.
Other fitness functions have a shifting definition of success.
Generally defined within a range of acceptable outcomes.
Automation is good!
Ideally most of our fitness functions will live in our deployment pipeline.
Not everything is amenable to automation though…
Legal.
Existing projects.
Temporal fitness functions.
Essentially a reminder.
Check for an upgrade of library X.
Break upon upgrade tests.
Clearly we want to identify fitness functions as early as we can.
The discussion about the tradeoffs is invaluable to our understanding.
Help us prioritize features.
May lead us to break a system up to isolate certain features.
We can’t know everything up front.
Fitness functions will emerge as the system changes.
But we should strive to identify as many as we can up front.
We can also classify fitness functions.
Key - critical decisions.
Relevant - considered but unlikely to influence the architecture.
Not Relevant - won’t impact our decisions.
Can still be very useful to identify the non relevant dimensions!
Keep fitness functions visible!
Need to review the fitness functions.
Are they still relevant?
Are there new dimensions we need to track?
Are there better ways of measuring/testing our current fitness functions?
Aim for at least an annual review.
Architectural Decisions.
They happen!
How do we document them?
Lightweight Architecture Decision Records.

https://www.thoughtworks.com/radar/techniques/lightweight-architecture-decision-records
Title/ID.
What is the problem?
List assumptions and constraints.
What are the options?
List the pros and cons of each alternative.
Which one did you chose?
Why?
Status - is this a proposal or has this been accepted? Is it now deprecated?
Consequences of the decision.
Consider a “time capsule”.
Screen cast or podcast of what you did and why.
Or just “why did we do this?”
Should be stored in version control.
Keep old decisions around - just make sure they are marked appropriately!
Your turn!
What architectural decision would you make on your kata?
What are the options?
Create an ADR!
Architecting is hard...
We have a lot to juggle!
Important that we thing strategically.
We can’t afford Resume Driven Design.
Good luck!
Questions?
Thanks!
Cloud-Native Java From the Source

The SpringOne Tour brings the best Cloud-Native Java content from our flagship conference directly to you. In 2 days, you’ll learn about both traditional monolithic and modern, Cloud-Native Java from the source. Experience valuable facetime with expert Pivotal speakers in both traditional presentation and informal Pivotal Conversations about modern Application Development, DevOps, CI/CD, Cloud and more.