CodeMash 2020

By Andrew Hinkle

This was my second year at [CodeMash](https://www.codemash.org/) and first year attending the Pre-Compilers. Weather held up well and was mostly just cold a couple days and wet the rest. I was impressed with the great content and can't wait to share.

Previous [CodeMash](https://www.codemash.org/) reviews go over what the conference offers, pros/cons, etc. Most of that stayed the same, so review them for more information on what to expect. I'm primarily a Microsoft full stack developer. The sessions I attend and how I review are from that perspective. I've taken on more team lead duties this past year, so I invested in more soft skills to help that initiative. With that said, let's review this year's CodeMash!

* Jonathan Danylko: [2020](https://www.danylkoweb.com/Blog/codemash-v2020-O6), [2019](https://www.danylkoweb.com/Blog/review-codemash-v2019-LY), [2018](https://www.danylkoweb.com/Blog/codemash-v2018-LR), [2017](https://www.danylkoweb.com/blog/codemash-2017-new-friends-old-friends-great-times-IS), [2016](https://www.danylkoweb.com/Blog/codemash-2016-my-exhaustive-week-DP), and [2015](https://www.danylkoweb.com/Blog/how-i-used-codemash-2015-to-determine-my-technology-plans-for-2015-97)
* Andrew Hinkle: [2019](https://www.danylkoweb.com/Blog/codemash-2019-part-2-FK)



# Virtual Machine to the Rescue

The Pre-Compilers are typically hands-on workshops, so if you choose a session on coding be prepared to have some setup. The recommendation is to perform any setup on a personal laptop, so you don't run into any issues with company resources and policies that may prevent you from performing some of the steps. I'm in between laptops at this time, so my employer provided a company laptop. Well that just wouldn't cut it, so I logged into Azure and created a virtual machine (VM) dedicated for CodeMash. I'm glad I did as I learned from others that they ran into troubles.

# Tuesday

## Functional F# Programming in .NET – A success story by Riccardo Terrell

[GitHub Repository](https://github.com/rikace/codemash-fsharpws)

I promote the concept of functional programming to my team. Keep your methods and classes honest meaning no hidden side effects. Given the same input you'll always get the same output. F# was designed from the Functional Programming perspective, so why not learn more by diving into a language dedicated to it.

[F# supports currying.](https://coders-corner.net/2014/03/30/currying-in-f/) This technique allows you to "pipe" the results of a method into next method. [Fluent interfaces and method chaining in C#](https://www.codeproject.com/Articles/640997/Fluent-interfaces-and-Method-Chaining-in-Csharp) can do something similar via the [Builder pattern](https://refactoring.guru/design-patterns/builder/csharp/example).

Immutability means there are no side effects/changes hidden within the function which makes them honest. If the function is honest and functional, then it does not need to be mocked. This is a core concept. Unfortunately, we didn't dig in deep enough to learn what to do when we have database dependencies. I've seen developers try to keep the core business logic as pure and honest as possible and pull dependencies on the database and services closer to the entry point of the application. The less you must mock the better.

let x = 5;; // assign 5 to x

x = x + 1;; // return false, because x does not equal x + 1;

x <- 7;; // error can't assign

|> // the expression on the left pipes forward.

"Drink your milk"

|> snake\_space

|> wrap("\_")

|> yell

Try to do everything you can using functional. mutable is used for interop and for performance in certain cases such as string concatenation.

;; is used to tell the compiler you are done.

// inline makes the input/output generic, but they also have to be the same type (int, string, etc.)

let inline add a b = a + b;;

add "Hello" " World" // Hello World

// Records are the same as Tuples with names

// Record types can be compared for equality.

let result = { Quotient = 3; Remainder = 1 } : DivisionResult

// with keeps all the same information and only changes what is after it.

let newResult = { restult with Remainder = 0 }

The last half of the session was a hands-on workshop. I followed the [F# Workshop](https://github.com/rikace/codemash-fsharpws/blob/master/doc/FSharpWorkshop.pdf) prerequisites and chose to use Visual Studio 2019 on the VM. Indentation matters!!! VS2019 tabbing fixed my issues. This set me back at the beginning of the workshop. For the rest of the coding sections I barely finished typing the code and verifying that the unit tests passed. Do not copy/paste the code as you will run into issues where the code doesn't quite line up to compile properly.

The workshop took us from basics through advanced topics. Thankfully the workshop is very detailed and once I got past the initial syntax setback the PDF flows well. Additional topics covered included discriminated unions, functional lists, domain-specific language (DSL), and more.

The advanced topics at the end were a bit overwhelming at the end including some very complex examples. Riccardo did this with the intent of illustrating F#'s capabilities which I appreciated. Riccardo did a fine job overall and the pacing was pretty good. I would have enjoyed more time coding the workshop examples allowing others to catch up and give me time to review the code to understand it better. Overall, I got what I wanted from the session, so thank you Riccardo!

## GIT: From beginner to Fearless by Brian Gorman

The workshop can be found as a paid [course](https://courses.majorguidancesolutions.com/p/git-from-beginner-to-fearless) offered online with videos so you may go at your own pace. Those who attended the workshop were given free access to the course and as of the writing of this article I still have access to the course, so bonus! Out of respect that this is a paid course, I've only included a few snippets to entice you to check it out for yourself.

GIT is a decentralized source control. No single point of failure.

Team members: clone specific repositories > make branch and changes > commit changes locally > push commits > central server.

Working directory is the local version on your machine. The .git folder tracks changes.

STAGE adds the changes to INDEX. Prepare the changes for tracking. COMMIT makes changes locally only.

PUSH and PULL. Is push same as a merge? No. Is pull same as a clone? No, clone brings down the entire repository at one time.

GitViz is nice application that visualizes your local GIT.

* <https://github.com/Readify/GitViz>
* <https://github.com/Readify/GitViz/releases>

The first half of the session went through the basics at a decent pace. While I struggled with the Merging due to a mistake I made in my setup, I was able to follow along, understand, and perform all the steps. My expectations were to get comfortable with GIT up through the merging, which I'm mostly there now due to this course, so thanks Brian!

* <https://git-scm.com/downloads>
* <https://git-scm.com/book/en/v2>

The second half of the course went through advanced topics including Amend, Reflog, Squash and Merge, Reset and Clean, Revert, Rebase, Cherry Picking, Stash, and Tag. Brian spent plenty of time explaining Rebase and provided links to the Explain Git with D3 tool to help. I added an example below.

* <https://github.com/blgorman/explain-git-with-d3>
* <http://onlywei.github.io/explain-git-with-d3/#zen>

// Rebase

git checkout -b dev1

git commit

git commit

git commit

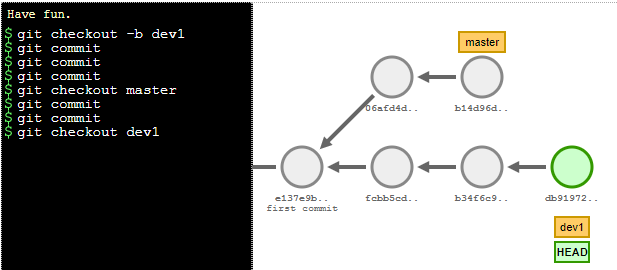
git checkout master

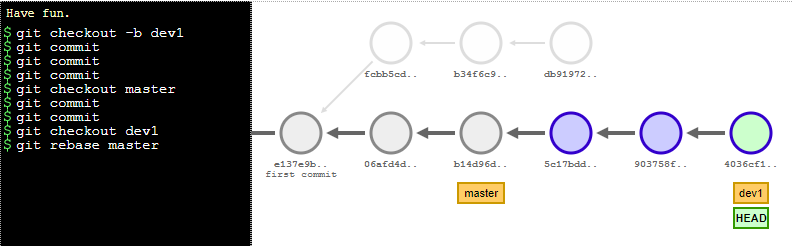
git commit

git commit

git checkout dev1

git rebase master





The tracking now shows the last 2 commits followed by the original 3 commits. If there are any merge conflicts in the process, then each commit of those changes would need resolved in the process of rebasing.

Brian did an excellent job going through the GIT commands. I learned a lot and I do feel more comfortable with it now. I do wish I had more experience going in to understand the advanced features he covered. The workshop felt rushed and you had to be diligent to keep up if you were trying to perform the commands at the same time. I was able to keep up through the basics covered in the first half, but after that I was mostly taking notes. There was plenty of interest in the advanced sections and due to the complexity, I can see why. Remember, this paid course is available online with videos and guidance that you may consume at your own pace.

# Wednesday

## Application Security, Basic, Intermediate, Advanced by Bill Sempf

This was another session based on a paid workshop, so out of respect I've written on the freely available information and a few small snippets to encourage you to attend [Bill Sempf's](https://sempf.net/) workshops in the future.

[OWASP](https://owasp.org/) is in the process of moving their WIKI content to <https://github.com/OWASP>.

Bill warned that some features won't work on company computers as your infrastructure team should have locked them down to prevent exactly what we were doing. That's why I created a short-lived virtual machine (VM) in Azure where I could experiment to my heart's content and in the end just destroy the VM when done.

What? Hacking's a bad thing? Well, yes, hacking is quite illegal, so you could lose your job, be sued, and probably go to jail. So yeah, don't do it.

You can get paid by companies who want you to find their security bugs by completing bug bounties at <https://www.bugcrowd.com/>. However, if you want to learn and understand the basics to help you protect your sites here's how you get started.

Setup the [OWASP Juice Shop](https://github.com/bkimminich/juice-shop). [Heroku](https://signup.heroku.com/) was recommended as a quick and free method of setting up the site for testing. Instructions are in the [OWASP Juice Shop](https://github.com/bkimminich/juice-shop). If you set it up properly it should look like the demo site <http://demo.owasp-juice.shop/#/>.

Download the [BurpSuite Community Edition](https://portswigger.net/burp/communitydownload). [Installing Burp's CA Certificate in Firefox](https://support.portswigger.net/customer/portal/articles/1783087-Installing_Installing%20CA%20Certificate%20-%20FF.html). When not using Burp, uninstall the certificate! Leaving the certificate on is a vulnerability that hackers can exploit on your machine, so only have it installed when in use. This is another reason to consider using a VM.

There are plenty of instructions at the sites above on how to get started and troubleshoot any setup issues.

Once my OWASP Juice Shop playground hosted on Heroku was setup and Burp installed I was able to use Burp to monitor all incoming and outgoing traffic. Burp includes options to handle basic Base64 and JWT decoding and more. I could see plain text information being POSTed to the site.

In one case I filled out the feedback page and saw how a POST was sent with my username and the number of stars. Well, well, well, how about I resubmit that same post and with Bill's username, 0 stars, and a horrible review! Oh Bill, that's scandalous! 2 achievements earned, score! That's one of the nice things about site, it came with a scorecard… if you can find it!

I've attended Bill's shorter session in the past at the [Central Ohio .NET Developers Group (CONDG)](http://condg.org/). I really wanted a deep dive into how to use Burp to hack the OWASP Juice Shop and I was not disappointed. He went over the common security vulnerabilities at a high level such as [SQL Injection](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md) which is always in the [Top 10 Web Application Security Risks](https://owasp.org/www-project-top-ten/). OWASP provides plenty of helpful advice in their [CheatSheetSeries](https://github.com/OWASP/CheatSheetSeries/tree/master/cheatsheets) for many of the vulnerabilities.

I would have liked more time hacking the site with the session focused on solving a few of the achievements to better understand the vulnerability. Describe a hack, the attendees perform the hack, light's dawn oh that's what he meant by that! Rinse and repeat. In his paid workshop that may be what he does. Overall, Bill's a great speaker, very personable, passionate about security, and it is a pleasure to attend his sessions.

## What to expect when you're concepting – Product Learning Lab by Saad Kamal

This workshop took you from the initial conception of a product to the creation of the Agile based tickets to fulfill them. As such this session targeted Product Owners and Managers but could be eye opening for Developers who aren't aware of the business side of the work they perform.

Hypothesis: What does the product stand for, why does your company exist, what's your elevator pitch? Start by asking "What if" statements to help.

Customer Focus: Put user's satisfaction first. Identify your market. Who are you targeting as a potential customer? Define your target customers (in this scenario we only included who we would initially target, though that may change over time).

What is their friction to change? Good product design makes the process easier to switch to the new process or product you're proposing. If the customer did switch to your product, what are the potential new frictions that you've introduced? Have they gained anything to switch?

Saad went through the list of interview questions. That led into development of persona where you defined their hopes and desires, pain points and frustrations, and friction to change.

Given all this information you build out your workload. An epic captures a large body of work that may span multiple releases. A feature captures what a user requires from the product that can be implemented in one release and then expanded/enhanced in future releases.

The rest of the session went over some additional features that you'll learn about by attending his workshop including prototyping, user journey flows, and minimum viable products (MVP) with assistance in understanding what that means for your product.

The Artemis room was on the other side of the session, so on occasion the presenter was hard to hear, and attendees were distracted. It was more of a minor annoyance that would not have been much of an issue if the microphone was louder. As it was, I strained to hear Saad most of the session but wasn't enough to seek further assistance than the few attempts made to increase the volume.

The session would have benefitted from keeping everyone focused on a specific Hypothesis. Everyone would have been able to follow along and grasp some of the concepts easier. As an example, once we reached the section on Epics, Features, and User Stories with everyone having arrived from such varying Hypothesis there was nothing to compare my efforts to understand if I truly got it. Were my Features too broad and should have been an Epic or too narrow and should have been a User Story? If the session was kept focused on a single path, then the session could have moved faster, and additional content could have been included in the workshop.

Given this session was labeled as Advanced I personally was looking for more information on how to build effective epics, features, and user stories. However, that was outside the scope of the workshop in its current format that Saad told me was targeted to beginners. With some adjustments this workshop could have accomplished that. I understand that this Pre-Compiler was based on an actual workshop targeting businesses. In that scenario they would have followed a more focused path, so I think it naturally would have occurred in those sessions and the points made here would be moot.

Saad is very capable and easy to work with. His quick wits allowed him to consume what we did and provide advice. If you are working for a company that needs assistance in this arena, schedule a workshop with Saad.

# Thursday

## Technical debt must die – Communicating code to business stakeholders by Matt Eland

Matt has written multiple articles on technical debt. I've read a few so far and they are great.

* [Communicating Tech Debt](https://killalldefects.com/2019/10/01/communicating-technical-debt/)
* [Strategies for Paying Off Technical Debt](https://killalldefects.com/2019/10/02/strategies-for-paying-off-technical-debt/)
* [Addressing Tech Debt without Killing Quality](https://killalldefects.com/2019/10/05/addressing-tech-debt-without-killing-quality/)
* [Functional Debt: The Price of "Yes"](https://killalldefects.com/2019/10/18/functional-debt-the-price-of-yes/)
* [The True Cost of Technical Debt](https://killalldefects.com/2019/11/09/the-true-cost-of-technical-debt/)

If you're interested in learning what it takes to deliver an excellent conference session, then read how Matt managed it for us!

* [Practicing (And Nailing) My First Conference Talk](https://killalldefects.com/2020/01/11/preparing-my-first-conference-talk/)

How to safely pay down debt? Manage Accumulation. Technical debt is code that's more expensive to maintain than it should be. Work with management to communicate and pay down technical debt.

[Toggl](https://toggl.com/) is a time tracking system that you may use to help understand where your time is spent.

Monitor lines of code added against removed and when they happened. If they happened right before a release maybe that's why you're having quality control issues.

Static analysis tools. Cyclomatic Complexity > Show in a visual graph as visuals help management

Analyzing Data

* Check Sheets - use data databars on your grids
* Fishbone diagram - high level issues with branching causes where each cause gets drilled down further to potential root causes
* Pareto charts - (80/20 rule)
* Histogram
* Run charts - data over time
* Control charts - incidents by month and compare to expected ranges
* Scatter plot
* Flow charts - highlight (Visio data annotations)

Presentation

* Can I keep our users happy?
* Will we still hit our strategic goals?
* Your goals

Communicate critical needs. Understand business needs. Start an honest conversation. Meet current and future needs.

Presentation structure

* Agenda > bottom-line up front > 1- 2 slides
* Orienting Information > 0 - 1 slides
* Supporting data 3 - 5 slides
* Recommendations or next steps > 0 - 3 slides
* What do you think?
* Don't show graphs that are way too busy

Personalities matter

* Anticipate objections / questions
* Handle questions gracefully
* Be prepared for a no
* Keep the focus on the topic
* No? > what are your concerns?
* Debt as risk > Example without paying down some of the debt certain features will be delayed

How do you perform tech debt remediation safely?

* Test Plans, manual tests, run unit tests: manual, CI/CD pipeline, code reviews
* Crop rotation
* Tech debt sprints
  + Normal > Normal > Normal > Tech Debt > repeat
* Dedicated Capacity
  + Tech Debt: 11%
  + Normal Work: 89%
* Dedicated Resource - alternate devs each sprint (untested)
  + Dev 1 - Normal Work
  + Dev 2 - Normal Work
  + Dev 3 - Normal Work
  + Dev 4 - Tech Debt
* Fix code that changes
* Only making changes to the code that is currently being worked on

Benefits of communication

* Building trust
* Gaining understanding
* Meeting the organization's needs paying of the right technical debt

This talk was very popular and a pleasure to attend. It was one of my favorite sessions at CodeMash. Matt was funny and witty. He hit all the points in a timely manner. You could easily tell that he rehearsed often given how well paced it was. I loved his hilarious example of a Drone company with issues ranging from inaccurate deliveries, incorrect billing, burning down houses, you know the usual. When his session is posted on [PluralSight](https://www.pluralsight.com/), I highly recommend that you watch it. Until then delve into his articles at <https://www.killalldefects.com>.

## A .NET Data Access Layer You're Proud of (Without Entity Framework) by Jonathan "J." Tower

[GitHub dal-without-ef](https://github.com/jonathantower/dal-without-ef), [Slides](https://github.com/jonathantower/dal-without-ef/blob/master/slides.pdf)

A pitfall of using Entity Framework is how easy it is for developers using LINQ to accidently use IQueryable when they think they are using IEnumerable or vice versus. IEnumerable filtering happens local after the data was fully loaded in memory. IQueryable filters in the database.

Entity Framework is a layer between you and the database. The query generated is automated and may not create the query you would have written.

Dapper is a lot faster than EF though EF Core has become more performant in the last year.

Change tracking is enabled by default for EF6 and EF Core. Tracking isn't useful in most web applications as the changes are brand new requests between pages. Manually turn off in your LINQ statement with [AsNoTracking](https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbextensions.asnotracking?view=entity-framework-5.0.0).

Lazy Loading is Dangerous as it can cause unnecessary calls to the database. See the example on page 10 of the [slideshow](https://github.com/jonathantower/dal-without-ef/blob/master/slides.pdf).

Object-Relationship Mapping (ORM) basically just maps the results of a query to an object and back again. There are options for more direct control of SQL syntax. Even when you use the "from...select" LINQ to SQL statements you won't necessarily get the query you expected. You could write the select statement in a string and then run the query. Parameterize to avoid SQL injection.

Micro ORMs map between objects and database queries. They are small and lightweight (EF does not fall in this).

Micro ORM is unlikely to have lazy loading, query caching, identity tracking, change tracking, OO query languages (LINQ), concept of a Unit of Work (UOW/transaction). Jonathan review a few Micro ORMs giving pros and cons and in the end recommended his favorite: Dapper.

Dapper supports extension methods built on the IDBContext. Uses existing built in ADO.NET objects like SqlConnection. It has [parameterized queries](https://stackoverflow.com/questions/7505808/why-do-we-always-prefer-using-parameters-in-sql-statements) to prevent [SQL injection](https://en.wikipedia.org/wiki/SQL_injection). Dapper has ASYNC versions. EF can simulate a Micro-ORM by using the FromSql command.

[Command Query Separation (CQS) Pattern](https://en.wikipedia.org/wiki/Command%E2%80%93query_separation)

* Command methods should
  + Not return anything (void)
  + Only mutate the object state (ex. id that was just created).
* Query methods should
  + Return results
  + Be immutable
  + Not change the object state

Check out the code samples on the last few pages of the [slides](https://github.com/jonathantower/dal-without-ef/blob/master/slides.pdf) to see the class structure come together. I would have liked to see the folder structure, but the concept was clear. Perhaps a simple app checked into Github that demonstrated the concepts would help. It could be a single page app with an in-memory database. That would just be gravy.

I like the separation of concerns of the Reads from the Writes whether it’s a database, file, or some other mechanism. I follow this pattern to a point already, but this gives me some more to think about especially in how it relates to the vertical slice architecture session below.

Jonathan did a fine job with the presentation. The pacing was good. I appreciated the code samples included, so it was more practical than theoretical. I also appreciated the fact that the [slides](https://github.com/jonathantower/dal-without-ef/blob/master/slides.pdf) were posted before the session started available for everyone immediately.

## Antifragile Teams by Charlie Sweet

[AntiFragile: Things That Gain from Disorder (Incerto)](https://www.amazon.com/Antifragile-Things-That-Disorder-Incerto/dp/0812979680/) by Nassim Nicholas Taleb

[Eustress](https://www.merriam-webster.com/dictionary/eustress) or AntiFragile means when stressed it gets stronger.

* A robust process is stressed and doesn't change much.
* A fragile process is stressed and gets worse and then stabilizes, but never recovers.
* An Antifragile process is stressed and gets stronger.
  + Like when you work out, your muscle is damaged, but then gets stronger.

The first quarter of the session discussed attempts in the past where stress was introduced and the results were great in some cases, worse in others.

The Wisdom of teams > Creating the high-performance organization

* Group: Strong, clearly focused leader, Individual accountability, Individual work products
* Teams: Shared leadership roles, Individual and Mutual accountability, Collective work products
* Both: Teamwork, Happy hours, Conflict
* Working group: When you need more than the sum of the parts.

[Maslow's Hierarchy of needs](https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs) (from the bottom of the pyramid up): Physiological > Safety > Love/belonging > Esteem > Self-actualization

Safety: Illustrated via [Project Aristotle](https://rework.withgoogle.com/guides/understanding-team-effectiveness/steps/introduction/) that evaluated communication styles, friends outside of work, similar interests, and management styles. None of these made a difference. It was primarily Psychological safety. You won't get in trouble or put down if you make a mistake or have a bad idea.

Love/belonging: Make connections between the teams. Make them feel like they belong.

Self- Actualization: [Flow by Mihaly Csikszentmihalyi](https://www.amazon.com/Flow-Psychology-Experience-Perennial-Classics/dp/0061339202). The Flow state is when you are totally focused on a task and are very productive.

What can we do? What can we do instead? What can we stop doing?

Premature or over-standardization. Story points should not be used as a metric to compare the effectiveness in a team or between teams. Burndown charts/velocity etc. can be helpful for a team but should not be used by managers.

Support teams. No one wants to be on the "B" team, "Maintenance" team, or "Support" team. The team that builds the product should fix the issues that come out of it. Cross train so teams can alternate between innovation and maintenance.

Long-lived teams should meet regularly to get a temperature reading, show appreciation, and share new information. Discuss puzzles - A scenario bothering someone, and they wanted to talk it out to see if a solution could be found. Complaints with recommendations (Ventilations and Vexations). Hopes and Wishes.

Take a Safety Assessment to understand where your team stands.

* When someone makes a mistake in this team, it is often held against him or her.
* In this team, it is easy to discuss difficult issues and problems.
* In this team, people are sometimes rejected for being different.
* It is completely safe to take a risk.
* Etc.

Change your mindset

* Embrace volatility
* Make it safe
* Power to the edges
* Don't add too much value
* Ensure appropriate amount of stressors

The volatility of moving people across teams if done properly can make the teams stronger

If someone comes to you with an idea and you add even just a small change to make it better their motivation drops in half because now, it's your idea. Push the power to make decisions to the edges to those who are closer to the work at hand.

This last point is important to me. I empower my team to make decisions and regroup when changes affect the work of others such as the architecture, patterns, and concepts. I like to build upon the shoulders of giants (team members). The sum of our parts is greater than the whole. I always felt that my suggestions to their ideas was mentoring, but this point of view shows it might be demoralizing. This is a tough one to reconcile and I'll have to think more on it.

Nassim was on point with the concepts. I would have liked the first quarter to move faster, so more could have been discussed on the later points. I learned much, reinforced some convictions, and found myself looking inward to decide if I'm helping or hurting my team. Mostly helping, I think. 😊

## Modular Monolith: The Best of Both Worlds by Seth Kraut

<http://tech.sethkraut.com>

We've all worked on a [Big ball of mud](https://en.wikipedia.org/wiki/Big_ball_of_mud). Deploying a change can be difficult, because it all needs deployed as one.

[Conway's Law](https://en.wikipedia.org/wiki/Conway%27s_law) promotes that organizations which design systems produce designs which are copies of the communication structures of these organizations.

[Domain Driven Design](https://en.wikipedia.org/wiki/Domain-driven_design) involves defining a [Bounded Context](https://martinfowler.com/bliki/BoundedContext.html). Sales may only need a subset of the information stored on a customer, and a support team may need different information with some overlap. Devs and business should talk the same language and the code should use the same terms.

Microservices vs Modular Monolith

* Microservices
  + Advantages
    - Decoupled and can be deployed individually.
    - You can build a really big thing like the death star with a bunch of small units like Legos.
  + Disadvantages
    - Complexity - imagine all the Legos for every box you've bought in the same box.
    - Slower - Every call to the service slows down the application that used to be run in memory.
* Modular Monolith
  + Single source code repository.
  + Deployed together.
  + All the benefits of a monolith with performance, Microservices with DDD bounded context etc., but cannot scale like Microservices.

Rules

* The best piece to split is the easiest.
  + Find a piece that is not dependent on anything else.
* Perfect is the enemy of the good.
  + Perfect is a blackhole that will never be dependent.

Example

1. Start with the Application (big ball of mud).
2. Identify the Mail utility has no dependencies and move that logic to a new Mail folder.
3. Identify the API endpoint has no dependencies and move that logic to a new API folder.
4. Identify Domain1 and split it out to a new Domain1 folder.
   * Domain1 could be your Customer, Orders, etc.
   * Rinse and repeat pulling out domains to their own folders.
5. Identify Common logic and dependencies between the domains.
   * In order to pull out domains and common logic you may need to refactor to remove dependencies or at least make them clearer.

project

API // Solution Folder

src

app

domain-ordering

service-mail

I was hoping for more details on the folder structure. The session targeted .NET and Java, so why folder structures? Why not create a project (.NET) or jar (Java) that represents each domain and the common logic? I can see the folder structure being a good first step and then project's or jars for the next.

This session introduced the basic concepts of DDD, and the pros and cons of Microservices vs Modular Monoliths. In summary Seth recommended focusing your refactoring to isolating logic with no dependencies from your domain logic and your common logic from your domain logic. The next steps from there dig into advanced topics outside the scope of this session.

If the session description included a little more information stating that it would introduce DDD concepts to separate your logic into folders I may have chosen a different session as I was already familiar with this technique. Updating the session description with those specifics would have helped. For those just learning the concepts this was an excellent introduction as mentioned by another attendee.

## API Gateways and Microservices: 2 peas in a pod by Santosh Hari

The first half of the session was a story of a project's progression from a monolith, to microservice, to containers. He discussed the trials and tribulations throughout the process and included some lessons learned explaining at a high level why decisions were made to transform from one state to another. While interesting it felt like it should have been a separate session.

API Gateway Vendors: [Azure API Management](https://azure.microsoft.com/en-us/services/api-management/), [APIGEE](https://cloud.google.com/apigee/api-management/), [AWS Gateway](https://aws.amazon.com/api-gateway/getting-started/), [Express gateway](https://www.express-gateway.io/getting-started/).

[An API Gateway sits between the client apps and the microservices](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern). It can be setup based on JSON, SOAP, REST, and Swagger.

[Add a new API](https://docs.microsoft.com/en-us/azure/api-management/import-and-publish): Blank API > OpenAPI > WADL > WSDL > Logic App > App Service > Function App.

Signatures, revisions and versions. Example: Catalog microservice changes signature. The API Gateway can handle the old and new signature requests?

Create a new API as a version of [Swagger Petstore](https://petstore.swagger.io/). Signature: Non-breaking changes. Revision: Breaking and non-breaking changes.

Mock responses. Set mocking policy to return a response based on the defined samples, rather than by calling the backend service. Your developers are your subscribers (customers). Add subscriptions and give the developers their keys. Keys should be stored in a Key Vault or equivalent. Key Vaults have different stores such as certificates, Secrets (Key/Value pairs), etc.

Provide developers with a playground. Keep rogue devs and infinite loops in check.

<rate-limit-by-key calls="10"

renewal-period="60"

counter-key="@(context.Request.IpAddress)" />

<quota-by-key calls="1000000"

bandwidth="10000"

renewal-period="2629800"

counter-key="@(context.Request.IpAddress)" />

<rate... />

DevOps all the things

* Use git to manage the API Gateway configuration.
* Monitor and measure
* Metrics such as setting up alerts when too many requests come in or such
* Cache is money; External cache can be created
* On-Premises APIs
* Site to site express routes

Security Options: Token, Oauth, OpenID, Certificates. If there is a bad actor, you can revoke the permissions.

Load Balancer manages the scale but not the versioning.

Santosh's presentation was fine and delivered well. As mentioned, the first half could be its own session. The second half of the session fared better in bringing me up to speed on the concepts and implementation of Azure API Gateways. I recommend spending more time on the when, how, and patterns to use API Gateways.

As an example, I had some questions that I wish were answered during the session. Here are the questions and answers (to the best of my understanding) I got from another attendee and from Santosh after the session. Thank you both for taking the time to answer these questions!

How many API Gateways should you have? What if you are concerned about cost? By environment such as DEV, QA, PROD? One for external and another for internal, so you don't accidently expose internal services?

From the attendee: Treat the version number like a router and point it to whatever environment you want. If it has a key or other security, you should be fine for internal. He recommended using load balancers for internal and environments other than PROD as they do the same without versioning.

From Santosh: 1 API Gateway for external, 1 for internal. Separate concerns. He said using an internal VNet or network will work for the internal sites if money is an issue. He also recommended for internal you could limit by IP addresses. He recommended using the developer version of API Gateway for DEV/QA work. Since it is configurable, he recommended storing some of the info in a Key Vault and use release definition token replacement to test the settings through the environments.

## Practical Data Modeling with MongoDB by Austin Zellner

Utility of data leveraged through query and access pattern.

* Data (BSON)
* Query (CRUD/Aggregation Pipeline)
* Access Pattern (Indexing)
* Each overlap and where they all overlap is Utility

How is the data stored?

* Document data model: Naturally maps to objects in code. Represent data of any structure. Strongly typed for ease of processing. Over 20 binary encoded JSON data types. Access by idiomatic drivers in all major developing languages.
* Type Data: Capture Field Value pairs, capture Arrays, etc... Versatile: Multiple data models. JSON Documents, Tabular, Key-Value, Text, Geospatial, Graph.
* JSON: Stored directly into BSON is the most common.
* Tabular (Relational): Capture column and value as Field Value pairs. Instead of left to right, top to bottom.
* Key Value: Key value captured directly as Field Value. If caching, can use TTL index.
* Text: Captured directly into Field Value
* Graph: Captured in Array
* Geospatial: Stored in GeoJSON
* Combination: Combine Multiple Models in One Document. You can have Geo, Key:value, Graph, Text, Tabular, etc. can be stored in the same document.

MongoDB is a [Polyglot](https://blog.cloudboost.io/what-is-polyglot-persistence-and-how-can-it-help-your-app-61d726292321?gi=b0615f742f57) system that allows querying against all of the different types supported by MongoDB in the **SAME QUERY**.

Simple Query.

db.collection.insert()

Versatile: Complex queries fast to create, optimize, and maintain.

db.customers.aggregate([

{

$unwind: "$address",

},

{

$match: {"addresss.location": "home"}

},

{

$group: {

\_id: "$address.city",

totalSpend: {$sum: "$annualSpend"},

averageSpend: {$avg: "$annualSpend"}

}

}

])

Aggregation to leverage utility. $lookup lets you LEFT JOIN between two domain level collections.

DO NOT USE as a crux to make MongoDB a Relational database, because your performance will get worse. Design queries to express meaningfulness of data. Since we are layering multiple dimensions, leverage queries to express the concepts represented in the data.

Fully Indexable. Fully featured secondary indexes. Index Types, Index Features.

Design Patterns

* General pattern is One Big Document.
* Application Specific Domain. Instead: MongoDB MultiDocument ACID Transactions.
* Just like relational transactions.
* ACID guarantees. This make transactions able to be rolled back within a transaction.
* Master and Working Collections.
* Master collections are stateless.
* Working collections are stateful.
* When initiate working document, read as needed from Master.
  + Duplicate only for performance.
* When working state changes, write back to Master.
  + Example: inventory consumed.
* If duplicated data changes in Master, write update to Working file.
  + Example: shipping address.
* Use Transactions to keep it all straight.

<https://university.mongodb.com/courses/M320/about>

JSON governance was added to keep the shape of data (int, decimal, string, etc.). However, to keep the data in sync between objects, it's up to the developer to keep the information in sync between documents.

Austin breezed through the session with ease. I was thoroughly impressed that the querying language supported querying different types within a document. The code sample was good and gave me a better idea of how to use it. I'm interested in learning more though as a long time TSQL user in the relational database realm it's difficult to contemplate a switch. The presentation felt targeted to developers like me and the message resonated with the audience. I'll need to prototype on my own at some point to get a better feel for it. I left the session with a better understanding of MongoDB, so mission accomplished.

## User Interviews: More than Just a Conversation by Ash Banaszek

Users are those who use your product to solve a problem. A user interview is meant to figure out what the user needs and provide it.

Individual Interviews are in a meeting room with a predetermined set of questions in a one-on-one setting. Group Interviews (Focus Group) are with multiple users posing questions to generate discussions.

Contextual Interviews are conducted in the context of the user's actual workplace during the process of work or immediately following. Collaboration for understanding, not a series of questions. Model data to find flows and patterns.

Set a research focus. Which process? What is important? What do we want from this? Who do we interview? If you aren't sure what is important, interview your stakeholders first.

Prepare Questions.

* Context / Details
  + Sequence (what do you do during the day when you first start?), Quantity, Specific examples, Exceptions, List, Relationships
  + How do you work with sales, underwriting, etc?
* Structure
  + Prepare Questions
  + Probing: Clarification, Code Words, Emotional Cues, Why, Delicate why, Indirect (leading), Outsider Explanation, Teach Another
* Mental Models
  + Compare Processes, Compare to Others, Compare across Time (Past)
* Ask Shortest Questions (Be succinct). Try not to ask your questions in different ways (I'm guilty of this). Treat your questions as a guide

Supplies: Paper/Pen, Recording device, Non-disclosure agreement (if applicable), Release form (if applicable), Know what you are bringing to the interview.

Identify your biases? What expectations do you have? Person, Application, Company, Role. Bias is easy and often unintentional. Assumptions: Users are resistant to change. They've probably used [application] before.

Fit in: Dress/Behave for the culture. Get your mind right. Make sure you've eaten. Take a bio break. Push away any other stressors. Review your goals and guide.

Basics of Interviewing

* Set the stage. Greet the user. Inform them of your goal. Do not tell them what you expect as that'll bias them. Set some ground rules. Give them a time estimate.
* Start recording. Must ask consent. Thank them for recording. Ask some comfort questions.
  + Anonymous: No one knows who you are.
  + Confidential: Only a select few know who you are.
* Beware of Guest/Host Model. We want to focus on partnership. Fine to take some water but lead them back to partnership. If taking something makes them more comfortable then do it within reason.
* 90/10 rule. 90% listening. 10% talking. Not a give and take.
* Talk about the present. Data is most accurate as it is happening. Do not ask the user to speculate on future behavior.
* Prompt, Don't Lead. Get the data you need to hear, not the data you want to hear.
* Avoiding Leading questions. Don't you think that if... Wouldn't it be better if... Do you prefer A or B. Some find it frustrating. How do you feel?
* Overly instructing. How do you create a New Account on the homepage? Maybe they don't use the homepage to create a new account. They know more than you do.
* User is the master. Interviewer is the apprentice. Do not demonstrate your expertise until the end.

Listening vs Perception. Ignoring: It's happening but you missed it. Perceiving: It's happening but you didn't respond appropriately like not recording. Listening: It's happening and you're trying to understand what is being done.

How to Listen. Listen to understand, not to reply. Focus on the message. Understand intent. Recognize tones and cues. Check for understanding.

Match the user's language. If they mispronounce something, go with it. If they use an insider slang, then if you're comfortable with it go with it, if it's odd then refer to it.

Do not fear silence. Silence allows for thought. Don't "fill it in". Leave silence after response.

How to take notes

* Don't type, clacking is distracting. Use time stamps for good quotes. Record when you started. Record the time when something happened. Write in context. Shorthand is key. Model on the fly with a flowchart or draw a picture.
* Better to have a second interviewer. One to moderate, one to take notes. Second interviewer does not talk past introductions.
* Keep track of threads. Potential for follow-up, then denote it. Triage. Come back at the end if needed. Keep an eye on the time.

Analyze with [Affinity Diagramming](https://www.nngroup.com/articles/affinity-diagram/). Read through notes (listen to recordings). Identify key points / ideas. Write each idea on a Post-It. Group and regroup ideas to find themes. Use themes to inform design decisions. Use different color for the theme name.

Reviewing Themes. Valuable insight on major points of feedback. How does this impact your design? How does the user see their role? What is their workflow?

Presenting. Mine audio snippets or "highlight reel". Review process. Review key insights. Here's how we can make it better.

Lessons Learned. Sell it to your stakeholders. Don't fall into the host/guest model. Know your research focus and key in on it. Make your work visible. Make it available for easy reuse. Plan > Do > Check > Adjust.

This session was packed full of useful information which is why this part of the review is long. [Ash](https://linkedin.com/in/ashmb) is an excellent speaker with great humorous quips and stories that flowed with the content for a well-timed enjoyable experience. If you have any interest in a topic she is presenting on, you are in for a treat. Thank you for a wonderful session and experience.

## Vertical Slice Architecture by Jimmy Bogard

[Slides](https://github.com/jbogard/presentations/tree/master/VerticalSliceArchitectures), <https://jimmybogard.com/>, [Example application using Vertical Slice Architecture](https://github.com/jbogard/contosoUniversityDotNetCore-Pages)

Jimmy started by reviewing the traditional architectures we use today. [Traditional n-tier](https://en.wikipedia.org/wiki/Multitier_architecture), [Domain Driven Design (DDD)](https://en.wikipedia.org/wiki/Domain-driven_design), [Onion](https://en.wikipedia.org/wiki/Onion_model), [Clean Architecture](https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html), etc. He went through the pitfalls of persisting a change through each layer that these architectures rely on.

A vertical slice has all the changes that need to be together stay together. Move code to a single place. As an example, you have an Invoice service with an endpoint for Approve(), Reject(), and Flag(). Move the logic inside each endpoint to a class: ApproveInvoice {}, RejectInvoice {}, FlagInvoice {}.

Next, model the requests: Input > Request Handler > Output. As the author of [AutoMapper](https://github.com/AutoMapper/AutoMapper) and [MediatR](https://github.com/jbogard/MediatR) (both available as NuGet packages) Jimmy utilizes them to handle messaging between them.

Commands and queries are isolated and follow this pattern. One model in, one model out. It provides complete encapsulation. A request comes in, all the work is done together, and then a response is returned. Merge conflicts should be rare as adding new features requires a new request, handler, and response.

* GET (Query) > Web App > Query > Handler > Response
* POST (Command) > Web App > Command > Handler > Response

He provided examples for Simple Queries, Parameterized Queries, and Paging/Sorting/Filtering Queries.

// Parameterized Queries

public class Index  
{  
 public class Query : IRequest<Model>  
 {  
 public int? Id { get; set; }  
 public int? CourseId { get; set; }  
 }  
…

The following is a simple command.

// Simple Commands  
public class Edit  
{  
 public class Command : IRequest  
 {  
 public int Id { get; set; }  
 public string LastName { get; set; }  
 public string FirstMidName { get; set; }  
 public DateTime? EnrollmentDate { get; set; }  
 }  
…

Each response is unique to the page. It's a [DTO](https://en.wikipedia.org/wiki/Data_transfer_object), not an [ORM](https://en.wikipedia.org/wiki/Object-relational_mapping) entity. Use AutoMapper to map between these objects. Scope the models only what is necessary for that page. He provided an example of a Simple Response and Complex Response.

// Simple Response  
public class Model  
{  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public decimal Budget { get; set; }  
 public DateTime StartDate { get; set; }  
 public string AdministratorFullName { get; set; }  
}

Command Response can be void, return a model, or success/fail (true/false). A model may be as complex as any other class, just remember not to return an entity.

When making a call to the database, Jimmy demonstrated how he uses [AutoMapper's ProjectTo](https://jimmybogard.com/automapper-linq-support-deep-dive/) feature to map the ORM entities to his response models. Regardless of using an ORM like Entity Framework, a MicroORM like Dapper, ADO.NET, or anything else this logic is encapsulated within a Handler class.

Common logic may be refactored out for reuse like a class, function, or extension method. Only couple logic if they change for the same reasons. [Refactoring 1st Edition](https://www.amazon.com/Refactoring-Improving-Design-Existing-Code/dp/0201485672), [Refactoring (2nd Edition)](https://www.amazon.com/Refactoring-Improving-Existing-Addison-Wesley-Signature/dp/0134757599), and [Refactoring.com](https://refactoring.com/) are great resources on how to refactor your code away from the Handler into Domain objects.

// Procedural Beginnings  
protected override async Task Handle(Command message, CancellationToken token)  
{  
 var course = await \_db.Courses.FindAsync(message.Id);  
 \_mapper.Map(message, course);  
}

In his [slides](https://github.com/jbogard/presentations/tree/master/VerticalSliceArchitectures) Jimmy shows you some basic refactoring techniques to manage complexity. Server-side validation is demonstrated. Controllers and Razor pages operate as routes, letting the Handlers do the heavy lifting. He goes on to show how Handlers can call other Handlers in Stackable Decorators that can handle transactions, unit of work, concurrency and retries, logging, and more.

The Rich Domain model should have unit test coverage. Perform integration tests of the handlers without mocking objects unless it's third party. Jimmy injects in-memory databases that is setup, so the entire vertical slice is tested with the request, handlers, and response.

Wow. That's a lot to take in. Jimmy has presented a new architecture pattern that feels clean and to the point. I for one will need to learn how to take advantage of the pipeline behaviors and handlers using .NET Core. Check out his [slides](https://github.com/jbogard/presentations/tree/master/VerticalSliceArchitectures) as I barely touched on the code samples he provided. He also included a link to an [example application that uses Vertical Slice Architecture](https://github.com/jbogard/contosoUniversityDotNetCore-Pages).

I'm not sure how I feel about not using Mocks. I like the idea of creating an integration test that tests the entire process from beginning to end as mocking will hide integration failures like forgetting to add a new rule to a factory or forgetting to call the new method you created. I think he makes a good case for using an in-memory database, but I'll probably stick to hiding the database calls in a class that I can mock, at least for now.

Jimmy's presentation was excellent. I liked the progression and tons of code samples to back up the architecture he proposed. I truly appreciate the [slides](https://github.com/jbogard/presentations/tree/master/VerticalSliceArchitectures) being posted before the session. As with any new proposal I'm sure some feathers were ruffled, but I've been having some interesting conversations with colleagues. As with anything new, create a prototype for something simple to get the concept down, vet it with the team before it's in production, and understand what maintenance on the application would be like. I'm excited to start a new project at home to learn this architecture. Thanks Jimmy!

# Friday

## Technical Leadership 101 by John Rouda

<https://www.ageekleader.com/>

* [Great leaders inspire greatness in others](https://www.quotes.net/show-quote/75889) - Clone wars anime
* Simon Sinek: "People don't buy what you do, they buy why you do it."
  + [Simon Sinek: Leaders eat last (book)](https://simonsinek.com/product/leaders-eat-last/)

People quit because you're a bad boss.

Building a Leader (FIRE): Foundation, Interior, Refinements, Exterior

Foundation

* Leader is the one who goes first
* IT Department Code: cultural norms
  + Retro to improve
  + Act like an owner (do what's best for the company)
  + We have each other's back
  + We choose people over process
    - Everyone is a person, people have problems
  + We question our assumptions
    - Don't do it because we've always done it
  + We think big and start small
    - Getting started is always the hardest part
  + We don't create waves... we ride them
  + We are intentional about our work
    - Meetings should have a purpose
  + We fail fast and recover faster
    - if it doesn't work, how do we get back to where we were (rollback)
  + We are not experts and will keep the posture of a student always learning
    - We make the complex seem simple
    - Stop talking in acronyms. Talk with business with their domain language
  + We make life easier for those around us
  + We are here to serve, not be served
    - Do what we can to make it easier for others

Interior

* Fix yourself first before you try to fix others
* Fact: money is a motivator. Pay people what they are worth
* Motivation: Autonomy, Mastery, Purpose
  + Autonomy: Freedom
    - Examples: Freedom of hours, remote work, what projects you work on, etc.
  + Mastery: Offer opportunities to train
    - Fun events can destress and promote productivity after the events.
  + Purpose:
    - The reason for which something is done or created or for which something exists.
    - Track down the real reason behind a request. Keep asking why.
    - To convey purpose, tell a story.
    - People want to make an impact, feel useful
    - You can do stuff as a team outside of work
      * Volunteer work such as at a soup kitchen at lunch

Refinements

* Field Trip: Go see how people use your products
  + [Transforming healthcare for children and their families: Doug Dietz](https://www.youtube.com/watch?v=jajduxPD6H4) - MRI creator. 75% of children had to be sedated to go into the machine. Made the MRI kid friendly with paint jobs, now only 10% need sedated. It's a very powerful Ted Talk.

Exterior

* Celebrating your team
* Give credit to individual and team
* Never blame an individual (it was your responsibility as a leader)
* Annual reports
* Certificates, anything to show improvement, achievement, etc.

What holds you back? FEAR. Who am I to do this? The imposter syndrome. Think about what you can do to mitigate your fears. Leadership takes time to learn. Something that takes five minutes to do, took a lifetime to get to this point to do.

John did a great job inspiring me to improve my leadership skills. These are all great points to remember and take to heart.

## Dealing with Disagreement by Tommy Graves

Two core contradictions

* We value diversity but diversity creates unproductive disagreements.
* We hire people because we value their ideas, yet we devalue those ideas in disagreements.

Why do people disagree?

* Usually have a clear path forward to resolve, then usually it's due to miscommunication and may be resolved by taking a step back and work on getting on the same page
* Different Evidence, then resolve by sharing the evidence
* Different levels of expertise, then resolve by sharing the knowledge
* Intractable disagreements
* Different Values
* Different Authorities
* Different past experiences
* Different fundamental frameworks such as different religious backgrounds
* Different intuition

Most things we believe are because it's obvious to us. Some fundamental thing that just makes us believe something. Looks like a chair. Must be a chair.

Why do we get things wrong? Conjunction Fallacy, Framing Effect, Bias Blind Spot, Dunning-Kruger Effect (Everyone thinks they are better than average drivers (lol)), Backfire Effect, IKEA Effect.

Our knowledge is very social. Someone disagreeing with you means that you are wrong. You should assume you may be wrong. Become humble, assume you may be wrong. This doesn't mean you give up, but you may need to not push as hard.

How likely do I think it is that you'll get the answer right? In past cases of disagreement, how often have I been right? We often are not in a good enough position to independently determine who is right.

You can't just give up your beliefs every time we get into a disagreement. So how do we resolve disagreements?

Use the phrase "disagree and commit." This phrase will save a lot of time. If you have conviction on a particular direction even though there's no consensus, it's helpful to say, "Look, I know we disagree on this but will you gamble with me on it? Disagree and commit?" By the time you're at this point, no one can know the answer for sure, and you'll probably get a quick yes." - [Jeff Bezos, 2016 Letter to Shareholders](https://blog.aboutamazon.com/company-news/2016-letter-to-shareholders).

... and reflect!

Three steps to disagreements. Disagree, Commit, Retrospect. Meet a week or month later and review how it turned out. What worked? What was valuable learned/done in the process? Right decision? Pivot? Many times, the disagreement was over something that was not critical. Better to do something than nothing.

Fostering a culture of disagreement and trust. This is critical for working with a junior engineer. Some of the ideas are great, some are bad. Many times, senior engineers will stomp on the junior's ideas driving them away to a place where they feel valued. If not harmful let them run with some ideas and let them own it. If you need to pivot after that, you pivot, but you gave them the opportunity.

if you disagree with someone you don't respect? Do they truly not have respect? Do others have the same opinion? Have they changed since your last encounter? Do they have your respect? In general, we should respect others.

Others are agreeing with you out of respect. How do you get people to disagree with you? It is extremely difficult to get people to disagree with you then ask them "do you REALLY agree with me?" Sometimes it's tricky to tell them something obvious wrong just to get them to disagree with you. Let them know that you are open to other opinions. Listen to them to foster an open forum for discussion. Mention some mistakes made in the past admitting that you're not always right.

Tommy has a passion for disagreements… in a good way. Through this session he methodically went through the most common forms of disagreement and offered solid advice on how to handle them. While there are nuances that may require additional help and guidance this is a great starting point. Such great advice, I included as much as I could. I liked this session a lot and look forward to hearing more from Tommy in the future.

## Hiring and Inspiring an Exceptional Team by Seth Petry-Johnson

[GitHub with slides](http://bit.ly/HiringAndInspiring)

Hiring

* Garbage in Garbage out. Hire people who share your values. Identify your core values.
* If you want to go fast, go alone. If you want to go far, go together. Build a group of team players.

[The Ideal Team Player](https://www.amazon.com/Ideal-Team-Player-Recognize-Cultivate-ebook/dp/B01B6AEJJ0) by Patrick Lencioni.

* Humble: Does not think less of self; Thinks of selfless.
* Hungry: Aggressively pursues goals.
* People smart: Emotionally smart in interactions with others.
* Combinations:
  + Only Humble: "The Bulldozer"
  + Humble/Hungry: "The Accidental Mess maker"
  + Humble/People smart: "The Loveable Slacker"
  + People smart/Hungry: "The Skillful Politician"
  + All three: "The Team Player"

Seth included the following gem of a quote. "So, tell us a little about yourself. I'd rather not, I really need this job." 😊

Screening for values and teamwork

* Get out from behind your desk!
* How they treat people outside of the office gives a better indicator about their personality.
* Take them to the local coffee shop.
* Doesn't matter where the interview happens.
* Screening for values and teamwork.
* [Hiring ideal team players: An interview guide to help you identify candidates who are humble, hungry and smart](http://bit.ly/TeamPlayerQs) is a list of questions to ask to evaluate the interviewee's values. Examples.
  + Humble: Can you tell me about someone who is better than you in an area that really matters to you?
  + Hungry: What is the hardest you've ever worked on something in your life? Can be your job, hobby, activity?
  + People Smart: What kind of people annoy you the most and how do you deal with them?

Eliminate siloed interviews. Panel of peer's review without leadership. Debrief. Who thinks they have value x? (humble, hungry, people smart)

Scare them with sincerity. Perfect fit or miserable? Fanatical about your core values and they'll be called out to them.

"[Design the Alliance](http://bit.ly/DesignTheAlliance)" with new hires. Example. How I like to lead: Be on time for meetings. How I like to be led: I don't watch the clock.

What's your preferred interaction style? How do you treat deadlines? Have you ever taken a personality assessment? How do you measure success? People who don't feel successful, don't stay. How can I tell if you're stressed/angry/upset?

Answer these questions first. Take digital notes (must be digital)! Review the notes once every quarter. One-Note page for each employee.

Effective 1:1s build relationships. Employees without 1:1 are 4x as likely to be actively disengaged. Employees with regular 1:1s are 3x more productive. "Bad management" is the #1 reason people give for changing jobs. Your ability to track 1:1s conversations is critical. Remember the Alliance, review your notes on how to interact with each other.

You cannot have an exceptional team without regular 1:1s. 1:1s are not status updates on projects. 1:1s are about building relationships. 30 minutes, once a week.

* 10 minutes their agenda. What's chafing them? No matter how small, by giving them that 10 minutes they know they always have time. What's on your mind?
* 10 minutes your agenda. Ask Questions. Give Feedback. Again, not project status updates! Build Rapport. How's life outside of work? How's ${childOrSpouseName} doing?
* 10 minutes flex in case either of the above take longer.

[One-on-Ones – Part 1 (Hall of Fame Guidance)](http://bit.ly/ManagerTools1-1). Employees whose managers are open and approachable are more engaged. Role power of hire/fire/raises is intimidating. You need to show real authenticate interest in their life.

Career Development. Training & Development, Flexible hours, cash bonuses. What work are you doing here that is most in line with your long-term goals? Giving them CSS work when they want to learn DB, isn't meaningful. Are there any roles in the company more interesting to you?

Improving the Team. How could we change our team meetings to be more effective? Who on the team do you have the most difficulty working with? Why? If you can identify problems early you can improve the situation before it escalates.

Give Feedback. Can't change the past, can change the future. Provide feedback at least quarterly or monthly, millennials need a quicker feedback loop.

Prepare. Ask the question to yourself, have data, prepare for responses and how you will respond. [How to Give Constructive Feedback to Motivate and Improve Your Team](https://bit.ly/1-1Feedback).

* Listen. Use What and How questions. Don't use Why as that can cause fight/flight response.
* Declare. Clearly articulate what needs to change. Change for the future, can't change the past. The more specific the better. Work with them when possible. Be redundant! Follow-up regularly. The more you show it matters to you the quicker the change.

Seth continued by reviewing the [DiSC](https://www.discprofile.com/) personality principles and how they help you understand how to speak/write emails to others not aligned with your personality type. What you feel is a very concise and detailed explanation on expectations may come off as condescending. Check out the [DiSC Overview](https://www.discprofile.com/what-is-disc/overview/) for details. Does everyone need to take a DiSC assessment? Nope, do it for your senior staff.

Seth somehow managed to pack this great advice in a single session without the audience feeling rushed. What a great speaker! Thank you SO MUCH for this great advice and having the slides posted on your [Github](http://bit.ly/HiringAndInspiring)!

## DDoS Attacks: Threat Landscape and Defensive Countermeasures by Chris Holland

[PDF of slides](http://bit.ly/ddos-talk)

Threats?

* Why take a site down? Conflicting Interests. Competition. Extortion. Axe to grind / Hacktivism. Distraction for a different attach.
* The IoT Threat. Internet of Things. Millions of Devices. Shared Factory-Default Passwords. Decade-Old Vulnerabilities with no upgrade cycle. Difficult to Secure.
* Other Threats. WordPress Sites via Ping-Back Protocol. Compromised Desktops, laptops, servers. Compromised Web Apps.
* TCP vs UDP. Every leg of the TCP handshake can be spoofed. UDP data exchange can be spoofed.
* HTTP Floods. If your router is not hardened, then hundreds of thousands of requests could be sent and processed consuming your CPU and memory. 1024 concurrent sockets. If your idle connections don't close fast enough there won't be any sockets available.
* HashDoS: Kill CPU with a Few Requests.
* Slowloris: Open Connection, and Keep it Opened: Feed 1 HTTP Header every few seconds. Most web sites can be taken down with just a laptop, not requiring a nation of bots.
* UDP Application Attacks.
  + Reflection is the act of pretending you are from one IP address and then asking another site to send you tons of info.
  + Amplification is when you make a request for big data, receive the data and return an even bigger response.
  + Reflection + Amplification is devastating.

What can you do?

* Fix IoT. IoT Devices: Look at MUD: Manufacturer Usage Description
* Reduce Reflection/Amplification. Close-Down open DNS resolvers. Use a Secure NTP Template
* Fix Spoofing. Stop Spoofing
* [RFC 2267](https://tools.ietf.org/html/rfc2267), [RFC 3704](https://tools.ietf.org/html/rfc3704)
* Fix WordPress
* Hardening. Reduce / Diversify Attack Surface
  + Reduce your exposure. [nmap](https://nmap.org/) yourself, nmap is a utility for network discovery and security auditing.
  + Use Dedicated Host Names. Even better different Domains
  + With different domains you can have different DNS servers
* DNS Sleeper Cells. Configure N DNS Providers. Assign Primary DNS Provider to Registrar. Primary goes down, switch to Secondary. Switch happens in seconds for popular TLDs: .com, .net, etc.
* DNS Redundancy. If you're primary goes down, you go to the next backup. DNS: Providers Selection. Anycast Infrastructure. DNS: Performance and Availability.
* Web Performance and Scaling. Performance > Scalability > Mitigation. Tune NGINX for High-Throughput Levels. Configure gzip/deflate compression carefully. Database indexes aligned with queries. Reduce number of DB queries per request. RDBMs backed by SSDs. Eradicate Table Locks (MySQL: ensure InnoDB). PHP: Ensure OpCache Properly Configured.
* Scalability. Application Code Needs Master / Replicated Awareness at DBAL. N Redis Data Caches. N Web Servers: NGINX. N Content Caching Servers: Varnish. CDN for Static Resources.
* Monitor: Performance and availability. Profile application. Fix bottlenecks. [newrelic.com](https://newrelic.com/) or [appdynamics.com](https://www.appdynamics.com/).
* Monitor Ongoing Performance and availability. [pingdom.com](https://www.pingdom.com/), [statuscake.com](https://www.statuscake.com/), [zabbix.org](https://zabbix.org/wiki/Main_Page) (Open source).
* Conduct Load-Tests: [loader.io](https://loader.io/).
* Monitor: Traffic and Crawlability Health. Ensure your traffic remains healthy. [analytics.google.com](https://analytics.google.com/), [webmaster.google.com](https://webmaster.google.com).
* Mitigation Platforms. Network Attacks: Absorb. Application Attacks: Blocking, Throttling, Caching. DIY / Vendor: [Reference the slides for examples](http://bit.ly/ddos-talk).
* Attack Playbook. Pause Marketing Campaigns. Engage Mitigation Platform. Notify Internal Stakeholders; prepare mailing lists ahead of time. Notify Clients/Customers; prepare messaging and delivery system. Gather evidence. Timeline of all events including extortion notices. Network server logs.

Chris did a fine job presenting and kept the pace very well. I now have a better idea of what DDoS attacks are and a multitude of other types of threats. Chris, thanks for providing plenty of links to resources in your [slides](http://bit.ly/ddos-talk)! If you're worried about the security of you network, then check out his slides to learn more.

## What's going on in there? Know what's happening in your software with Application

## Insights by Eric Potter

Roles we play when there is an issue: Mechanic, Crime Scene Investigator, Track Coach, Anthropologist.

If you already have an Azure account, Application Insights can easily be added to your application. Visual Studio 2019 > Right-click your .NET Core project > Add Application Insights Telemetry. It will register Application Insights and SDK. Application Insights uses CosmosDB as the backing storage.

Your application's Startup.cs is updated and configured to register Application Insights. The Application Insights DLL knows how to connect to Azure Application Insights, just need the InstrumentationKey. Do not add the InstrumentationKey to the app.config, use Key Vault or the like.

On each entry point or use dependency injection to inject Application Insights in a central place perhaps in a base controller. You could use Jimmy Bogard's Handler technique mentioned in the session above. Look at that session for more information.

var tel = new telementryClient new TelemtryConfiguration(instrumentationKey)

switch (i)

{

case 42:

tel.TrackEvent("SearchForMeaning"); // "SearchForMeaning" is a customEvent

break;

case 43:

tel.TrackException(new Exception("One too many"));

break;

case 44:

var rnd = new Random();

var val = rnd.Next(1000, 5000);

tel.TrackMetric("long wait time", val);

System.Threading.Thread.Sleep(val);

break;

case 45:

var rnd2 = new Random();

var val2 = rnd2.Next(1000, 5000);

var stopWatch = new Stopwatch();

stopWatch.Start();

System.Threading.Sleep(val2);

stopWatch.Stop();

tel.TrackMetric("long wait time", stopWatch.ElapsedMilliseconds);

break;

}

You can use Application Insights with desktop apps. Application Insights is not real time but it's close.

AppInsights - Logs (Analytics). Create a new query: exceptions, requests. Options are in the left panel. Can use where clauses based on any fields in the named columns, fields in your JSON (using extend).

customEvents

| where timestamp > ago(2d)

| where name == "SearchForMeaning"

customEvents

| extend UserName = tostring(customDimensions.Username)

| where UserName == 'somewhere@someplace.com"

pageViews

| summarize count() by bin(timestamp, 1h)

| order by timestamp

| render barchart

You can change by display time UTC to be local. If you really need physical location some near state boundaries.

customEvents

| extend UserName = tostring(customDimensions.Username)

| distinct

customEvents

| where timestamp > ago(7d)

| summarize count (timestamp) by name

| order by count\_timestamp desc

| render barchart

Different render options in the dropdown option

customEvents

|where timestamp > ago(7d)

| where name == "AddTime" or name == "UpdateTime" or name == "CopyTime"

| extend hourOfDay = datetime\_part("hour", timestamp - 5h)

| summarize count (hourOfDay) by hourOfDay

| order by hourOfDay asc

| render barchart

Every line is an AND, you can add OR inline. Query language is [Kusto](https://docs.microsoft.com/en-us/azure/kusto/query/).

let timeGrain=7d;

let dataset=exceptions

| where timestamp > ago(60d)

; // calculate exception count for all excptions

dataset

| summarize count\_=sum(itemCount by bin(timestamp, timeGrain)

| render timechart

I have not had the opportunity to use Application Insights much. The guidance provided here was an exceptional introduction with many detailed examples on implementation and queries. Just what I was looking for in this session. Please speed up the intro by speaking a little faster so there are fewer pauses. Once Eric got past the intro the session kicked into high gear and was well paced. Thank you again for all the great examples!

## Programmer burnout: how to recognize and avoid it by Santosh Hari

<https://santoshhari.wordpress.com/>

Santosh started by describing his burnout experience. It started off with his workload, a treadmill of sprints that just didn't seem to accomplish anything. He created his own personal paging monster where a notification pinged him every time something bad happened. He had no destressing activities outside of work.

What is burnout? Work related stress. Exhaustion, physical and emotional. Not a medical diagnosis. Can cause medical issues or result like depression.

Origins of burnout date back to the space scientists during the space race to the moon. When all fuel is spent, that is called burnout.

Software companies can become burnout shops. They view an employee that burns out is not considered a good one. Employees are expendable. Burnout is a personal failure.

Agile can contribute by becoming a death march. End of sprint deadlines where you never get close to being done. Agile activities typically stop doing retros as an example.

Burned out employee is a canary in a coal mine. It indicates a toxic culture. Trying to harden the employee is the wrong approach.

Cultural gaslighting. Program us to be constantly working with no downtime. Say no to the #Hustleporn. People flexing how hard they are working. How much better they are and you're lazy if you're not working extra hours. Working too much is the same as sending too many emails: SPAM, the results just are not good. The [Joy of Missing Out (JOMO)](https://www.psychologytoday.com/us/blog/happiness-is-state-mind/201807/jomo-the-joy-missing-out) is about being present and being content with where you are at in life. For Santosh this meant ignoring social media.

Lack of work life personal life boundaries. Don't monitor work notifications if you are not on the clock.

Superhero employee complex. Don't jump in to help others when not requested. If everyone is dependent upon you how will they do it on their own.

Factors at work. Workload, Control, Compensation, Community, Fairness, Values. You may not have control over what you work on, but maybe it's worth it if the compensation is good.

Burnout goes through phases based on your factors at work.

* Phase 1: Dialed in – all areas are good.
* Phase 2: Tolerating work – somethings are not working as well anymore. Example: workload/compensation.
* Phase 3: Stretched to the limit. Example: workload/compensation are now horrible, and control/fairness have dropped.
* Phase 4: Barely getting by. Example: workload/compensation/fairness are as bad as they can be and none of the other factors fair well either.
* Phase 5: Burned out. One developer who was active on a community GitHub project viewed the history of his contributions and could track when his burnout started up to when it was in full affect where he stopped contributing. [GitHub contribution graph to show burnout](https://flowingdata.com/2019/10/09/github-contribution-graph-to-show-burnout/).

What employees can do to reduce burnout. Understand your role. Reduce multitasking. Learn to say no. Reduce social media usage and notifications. Get a life outside of work. Get exercise and sleep. Take care of yourself. Self-care. A couple actions Santosh took were to stop the paging monster and spend more time with family.

What employers can do to reduce burnout. Sustainable workload. Autonomy. Recognition and reward. Work community. Fairness, respect, and social justice. Clear value and meaningful work. Be honest about which factors at work can't be adjusted and adjust the others for the better when possible.

[The No Asshole Rule: Building a Civilized Workplace and Surviving One That Isn't](https://www.amazon.com/Asshole-Rule-Civilized-Workplace-Surviving/dp/1600245854) by Robert I. Sutton PhD is a recommended read.

As a developer who has suffered through burnout a couple times, I have no interest in doing it again. Santosh's advice makes sense and will help us identify burnout and try to mitigate it. The presentation was well paced and well received by the audience. I personally thanked Santosh for presenting on the topic of burnout and support him giving this session to any and all who will listen to it. Thank you!

# Conclusion

[CodeMash](https://www.codemash.org/) 2020 was a wonderful experience. I met up with some colleagues and friends of old. I've learned so much that is applicable right now especially with the soft skills on leadership, hiring/inspiring your team, and more. Burnout is a real threat to everyone and I'm so glad to have learned more on how to prevent it. Vertical slice architecture is going to be a fun side project to learn. Thanks for another great year CodeMash!

***What was your favorite session at CodeMash 2020? Have you been attending their excellent soft skills sessions? What are your thoughts on the Vertical Slice Architecture?***