EPISODE 1006

[INTRODUCTION]

[00:00:00] JM: A data pipeline is a series of steps that takes large datasets and creates usable results from those datasets. At the beginning of a data pipeline, a dataset might be pulled from a database, a distributed file system, or a Kafka topic. Throughout a data pipeline, different datasets are joined, filtered, and statistically analyzed.

At the end of a data pipeline, data might be put into a data warehouse or Apache Spark for ad hoc analysis and data science. At this point, the end user of the dataset expects the data to be clean and accurate, but how often do we really have any guarantees about the correctness of that data?

Abe Gong is the creator of Great Expectations, a system for data pipeline testing. In Great Expectations, the developer creates tests called expectations which verify certain characteristics of the dataset at different phases in a data pipeline, and this helps ensure that the end result of a multistage data pipeline is correct. Abe joins the show to discuss the architecture of a data pipeline and the use cases of Great Expectations.

If you want to find all of our episodes about data engineering, check out softwaredaily.com or the Software Daily mobile apps. We've got hundreds of episodes and many of them relate to data engineering and data architecture.

[SPONSOR MESSAGE]

[00:01:34] JM: This episode is sponsored by Datadog, which unifies metrics, logs, traces and other advanced features in a single scalable platform. You can view SLO's in real-time. You can search group and filter SLO's by facets, like tags, and team, and services, and more, and you can see your error budget at a glance. Plus, you can easily share intuitive, rich dashboards to provide detail and extra context to internal teams and outside stakeholders.

Sign up for a free 14-day trial and Datadog will send you one of their famously cozy t-shirts. That's at softwareengineeringdaily.com/datadog. You can sign up for that trial and get a free t-shirt by going to softwareengineeringdaily.com/datadog.

[INTERVIEW]

[00:02:31] JM: Abe Gong, welcome to podcast.

[00:02:33] AG: Thanks, Jeff.

[00:02:33] JM: Explain what a data pipeline is.

[00:02:36] AG: Data pipeline is kind of generic term that data engineers use to cover a whole bunch of things. You hear people talk about ETL, which is basically moving data from one system to another. You hear people talk about transformations or aggregation kind of internal SQL logic in the database. There are pipelines that train and deploy machine learning models. There are data pipelines that serve dashboards.

In the world of data engineering, all of this is kind of just part of the great river of data flowing from upstream kind of raw data sources to downstream data products and use cases and anything in there you can think of as a pipeline.

[00:03:12] JM: The data pipeline might be broken up into these different steps, and different steps are handled by heterogeneous systems, like you got Kafka holding data in one place. You got the SQL database holding data in another place. You got a data lake, like HDFS or S3. With all of these different data sources you are writing and reading from a lot of different places, what are the problems that can result from having this heterogeneity of data sources?

[00:03:42] AG: I mean, it's big. There is a ton of surface area there and there a lot of things that can break. Some of the common problems you end up seeing are broken dashboards. When I talk at conferences, I'll often ask people to raise their hands and say, "Who remembers the day that the dashboard broke at your organization?" and almost everybody can raise their hand and talk about a time when some dashboard broke.

Machine learning models, you get the same thing squared because there is a lot more surface area where a machine learning model can break. Data products, when they're being deployed as part of a product experience, similar things. It's very easy for those to just go wonky.

I mean, I think this is kind of where you're going, but in general I think of the main issue there as, as those systems get deeper and more complicated, you have dependencies upon dependencies often maintained by multiple teams. Often, those teams have different skillsets and different levels of kind of engineering expertise. Among all those handoffs, it’s really easy for things to get lost in the mix.

[00:04:41] JM: I've heard you talk about this in the phrasing pipeline debt. Can you explain what pipeline debt is more broadly?

[00:04:50] AG: So simple analogy to technical debt anywhere else. I think, broadly speaking, most people today are describing technical debt as lack of testing and documentation for a given system. I think if you want to get really precise I would say that's like a core symptom and technical that is really unexpressed tacit knowledge or tribal knowledge that has to be on-hand if you're going to fix or maintain a system, but they come awfully close. Pipeline debt is the fact that an awful lot of data systems just don't have test or documentation today, and because of that we have to rely on all this tribal knowledge.

[00:05:24] JM: What's the cost of pipeline debt? How does it end up hurting me?

[00:05:28] AG: Oh! A bunch of different ways. I mean, I'd say the two main things are time and trust. On the times side, it's very common to talk to data teams that are, say, two years in to deploying and maintaining a system and to talk to them and say, “Yeah, we spend a big fraction of our time just chasing down bugs, just figuring out why the dashboard said that or why this column is not populating the way it used to.”

It's actually quite similar to what you see in other teams that have technical debt when something changes in the code and they don't know why and you have to trace it down. So that's one side. The other side aside from time is just trust. It doesn't take that many times of the dashboard breaking for everybody to start feeling like, “Well, can I trust the dashboard today? Do I have to talk to the data team before I check and rely on it?” A few issues like that can erode trust really, really fast.

[00:06:21] JM: A quote from you, “Data pipelines often become deep stacks of unverifi ed assumptions.” What do you mean by unverifi ed assumptions?

[00:06:30] AG: I'm thinking mostly of testing there, but testing with a little bit of a twist, and that the twist is in the software engineering world, we've gotten used to doing our tests at what I call deployed time or compile time. When the code changes, you want to test things.

In the data world, you want that, but it's not enough. The reason is, often, data pipelines are breaking not because you have changed any of the code in your system, but because the upstream data that you’re consuming, whether that's logs from an app, or data being bought from a data vendor, or something like that, those can change as well and, often, those will basically stress code paths that you had never anticipated and break things that way. I'm not sure if I'm exactly answering your question on the head there.

[00:07:14] JM: No. You’re answering it quite well. I think we’ll explore the contours of the unverifi ed assumptions and pipeline debt as we get into Great Expectations and in more detail.

[00:07:28] AG: Would an example or two be helpful there?

[00:07:30] JM: Sure.

[00:07:31] AG: One that I think is just really interesting to go to is when you say what's the average. There are a ton of assumptions that go into calculating something as simple as an average. Okay. Average is probably the main, maybe the median, maybe the mode. Okay. What do you mean? There're been some assumptions there.

If you’re talking about the main, you've got a numerator and a denominator. What goes in the denominator? If you say, “What was our average churn for customers last month?” Well, what counts as a customer? Was it only active customers the previous month? What defines active? If it's not the previous month, how long is your window for that?

Actually as you start to kind of reach into that you realize there are a whole bunch of assumptions that going to something as simple as what was our average church. If not everybody's on the same page about that, you can be drawn wildly different conclusions without meaning to.

[00:08:20] JM: Great Expectations is a tool for creating data pipeline tests. What is a data pipeline test?

[00:08:27] AG: A data pipeline test, let me just describe it in the language of expectations.

[00:08:31] JM: Sure.

[00:08:31] AG: Which is why the whole package is called Great Expectations, because we, on top of engineering, we like puns. An expectation is an assertion about data. Something like expect this column to exist, or expect the column to have such and such type in stream fl ow, whatever.

Beyond that, you can get into the contents of the data as well and you can say expect values in this column to fall between this minimum and this maximum at least 90% of the time. Things like that. We can do regular expressions. We can do daytime formatting. We can do missing values. We can do everything up to like correlations and like borderline anomaly detection. It's this nice clean assertion about what should the data look like at this point in the pipeline.

[00:09:15] JM: We've had integration tests for a long time. We've had unit tests. We've had smoke tests. Why haven't we had data pipeline tests?

[00:09:25] AG: Well, if you look, people do end up building this on the side themselves. But I think the critical difference is most of the tests that we are used to writing in software engineering really do test the code. The thing that you're defending against is if somebody changes the code in the future and it has an unanticipated consequence somewhere else in the codebase, let's make sure we pick that up.

The thing that's I think different about expectations and this notion of pipeline testing is we're testing the data. You're not verifying that a SQL query has such and such input and then will generate such and such output. That's not a bad thing to do. But in our case we're saying, "At this stage in the pipeline, here's what the data should look like." It actually starts to feel kind of like monitoring, although from a testing perspective and a trust in time perspective, it fills a similar role to tests in traditional code.

[00:10:16] JM: Most of the data applications that I think about are not fine-grained. They're not mission-critical. They're things like – I mean, they are mission-critical, but it's okay if there's some data that's slightly invalid. I think about a recommendation engine. It's not a big deal if Netflix accidentally thinks that I watched Les Miserables, right? That's not a problem, and Netflix gives me some recommendation based on Les Miserables. I just skip past it. That's not interesting. I'm not interested.

Similarly in a dashboard, you might have an average. If that average has been calculated from 30,000 data points and one of those data points happen to be invalid and it happened to be enough slightly bigger than the average, it's not a big deal. Why do we care so much about testing our data? If we're just getting these coarse-grained things like recommendation algorithms and averages and stuff, these are the aggregates of our data that we are getting. Why do we need to test our data pipeline?

[00:11:25] AG: I think I'm going to disagree with the premise on this one. I mean, this is not exactly what you said, but like, "Hey, data pipelines," they just produce stuff. There's noise. It doesn't really matter if we get it right. I'm not sure I believe that. A lot of my background is coming through healthcare. Some health and wellness, but also some working with really, really sick patients. Finding the right patients that like making sure they get to the right doctors, it really matters there.

Even stepping back to like the Netflix example, I grant you that a recommender system can absorb a certain amount of noise. But why would you allow more of that than you need to? If we’re treating this as engineering and we want the results to be reproducible and trustworthy, then presumably we want as few unintended errors in there as possible.

[00:12:12] JM: If you think about that example of the recommendation engine, you think about Les Miserables. The data somehow gets into the database that I watched Les Miserables. I'm not sure how that could happen or why that would happen, but more importantly, I have no idea how to write a test that would detect that I didn't watch Les Miserables. I mean, you've got all these different systems that are talking to one another. How would I know that it is impossible to get from some upstream system the fact that I did watch Les Miserables?

[00:12:47] AG: I think it is worth separating these types of errors in data into at least two categories. One would be what I'm going to call fat finger errors, which is the data as it was entered into the system was just wrong at the source. Truth be told, Great Expectations will give you some defense against that. But if we've got a gender field and you say you’re male and the data says it's female, if you just like to the system, there's only so much the system itself can do about that.

Now, if that were happening systematically and 50% of people's genders were being flipped, you might be able to pick that up, but I don't know if you could get one specific fat finger error by testing within the pipeline.

But there's another really broad set of issues that shows up as you kind of get into the thick of data pipelines, which is as the data is being processed or moved around, it's really easy to introduce errors there.

For example, you'll often see teams that are consuming logs from an app or something like that. They're using the data exhaust to build algorithms on top of. If the logging team changes the way the logs work, reality is not changing. You’re just changing sort of how you capture that and flow it through the system, and that could easily, easily mess up a machine learning algorithm or dashboard downstream. That's the sort of thing that I think we’re best suited to catch and deal with. This is good, kind of breaking out the categories of like what can and can't be tested.

[00:14:10] JM: Totally. Great Expectations is built around the abstraction of an expectation. What is an expectation?

[00:14:18] AG: Yeah. An expectation is an assertion about data. It's defining what the data should look like at some stage in your pipeline.

[00:14:27] JM: Can you walk through an example?

[00:14:27] AG: Yeah. In fact, let me give you a couple just so you get the flavor of them at different stages of the pipeline. For raw data coming in, so a semi-trusted source, something that's definitely outside of your control. You might be saying things like, "Hey, we want these columns to exist. We want them to have these types and we want to make sure that no more than 5% of values are missing in these critical columns." Just kind of verifying that things look okay as they enter the borders of your system.

If you get more into the middle of your logic, you might be doing things like I want to assert that the number of rows in table A is within 10% of the number of rows in a prime to make sure that our joins haven't gone wrong or we're not just dropping a lot of data when we do de-duplication or something like that.

If you get deeper into the system, say around a machine learning model, you might be looking at the distribution of input and output variables going to the ML model just to verify like, "Hey, is this being deployed on data that looks more or less like the date on which it was trained?" All of those are different species of expectation. The kinds of tests you want at different points in the pipeline are just going to be different depending on what your pipeline is doing there.

[00:15:36] JM: When are these tests going to be run?

[00:15:38] AG: How deep do you want to go?

[00:15:40] JM: Give me a number of places. I mean, if I think about unit testing, like I run unit tests as soon as I got a build ready. It's just on my local machine and those tests happen to satisfy my local build. Then so I push it to staging and then there's maybe some end-to-end tests that run in staging and I'm just talking about application tests. Then I'm going to merge my code into master. Then there're integration tests that are run. Then maybe the integration tests fails. You could think of all kinds of different places where Great Expectations style tests could be run. Tell me about where you see them most frequently.

[00:16:20] AG: Yeah. Let me give you three in the order that I usually see them, like see teams rolling them out. But I'll be very upfront. I think best practice here is still evolving. If you come back to me in a year, I may have a somewhat different answer. But here are the three. The first one is testing materialized data at rest in your pipeline. That would be something like you run nightly Cron jobs against your data warehouse to make sure that everything that was inserted into the warehouse makes sense, and then the downstream aggregations and all of that also makes sense according to your expectations.

We see people doing that first because it's pretty lightweight to deploy. You don't have to touch existing code. For most of the stuff you care about, it's going to be materialized somewhere. This is a lightweight way to just sort of bolt some level of testing under an existing system.

Second layer that we see is people deploying them in Airflow or other DAG orchestration tools and the value that you get there is you can actually control the flow of data through the system and that lets you do things like, "Hey, if the data is so broken that I'm afraid it's going to pollute downstream data processing, let's just halt the pipeline there. Not burden ourselves with the mass of cleaning that all up afterwards." But it's slightly heavier weight than Cron jobbing a data warehouse, because you actually have to go in and insert a few lines of code in the right parts of your Airflow Dag.

The last one, and this is the place where I think practice is still emerging the most, is testing new code as it's deployed in data pipelines. Actually, kind of PR's and traditional GitFlow to examine changes before they get merged to master or whatever the equivalent is in your system. For various reasons, I think that's the hardest one, and the main thing that I'd cite is getting good test fixtures for mock data to run through your code is actually quite a chore. It's often a lot harder than writing NewSQL queries or deploying a new Python notebook or something like that.

As I see people deploying expectations or data testing in general, usually they're starting with materialized data then they go to their orchestration and then eventually they work back to something like a GitFlow.

[00:18:30] JM: Okay. Let's go through each of those. The first one, I've got a data warehouse and there's a nightly job that refreshes the data in my data warehouse perhaps. Let's say every day at 2 AM, yesterday's exhaust data, click stream data from my advertising company that has been – It's bulb and written to HDFS and I run this big Hive job that queries HDFS and loads all my data into my data warehouse.

I now have got a materialized view in my data warehouse and it's also got some like rollups or aggregations that measure the averages. You're saying I could run at that point at let's say maybe 3 AM after the ETL job is finished. I run my Great Expectations tests against the materialized view in my data warehouse. Am I understanding that one correctly?

[00:19:24] AG: Yeah, exactly. It's exactly the kind of scenario I'm talking about.

[00:19:28] JM: So let's say it is some clickstream data. What kinds of data tests would I be running into that materialized view?

[00:19:34] AG: For clickstream, and I'll be honest. I haven't live super close to clickstream data in a while, so completeness, missing this. If you're already in a data warehouse, I'd assume that you're already probably structured, and so you don't have to test column existence and things like that. Although you might want to just be sure distributions over different types of clicks could be interesting.

We're starting to see people do stuff in aggregations around what I think of it state machines. Verify that logins are always followed by logout's or like session start always followed by session end. Yeah. You can do interesting things like that.

[SPONSOR MESSAGE]

[00:20:20] JM: DigitalOcean is a simple, developer friendly cloud platform. DigitalOcean is optimized to make managing and scaling applications easy with an intuitive API, multiple storage options, integrated firewalls, load balancers and more. With predictable pricing and flexible configurations and world-class customer support, you'll get access to all the infrastructure services you need to grow. DigitalOcean is simple.

If you don't need the complexity of the complex cloud providers, try out DigitalOcean with their simple interface and their great customer support, plus they've got 2,000+ tutorials to help you stay up-to-date with the latest open source software and languages and frameworks. You can get started on DigitalOcean for free at do.co/sedaily.

One thing that makes DigitalOcean special is they're really interested in long-term developer productivity, and I remember one particular example of this when I found a tutorial in DigitalOcean about how to get started on a different cloud provider. I thought that really stood for a sense of confidence, and an attention to just getting developers off the ground faster, and they've continued to do that with DigitalOcean today. All their services are easy to use and have simple interfaces.

Try it out at do.co/sedaily. That's the D-O.C-O/sedaily. You will get started for free with some free credits. Thanks to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:22:18] JM: This is useful because the data warehouse, at 9:30 AM, the data scientists are going to roll into the office and they're going to sit down in front of their machines. They're going to hook their BI tool up to the data warehouse and start making queries against it. If the data is invalid and they don't know, then they're going to make a bunch of business level decisions that are going to be incorrect.

[00:22:41] AG: I mean, at the very least, they're going to waste their day, right?

[00:22:44] JM: At the very least, they're going to waste their day. With that use case, you would need to take some time to actually set up data tests for your data warehouse. You would need to say, "Okay. Let's sit down and really think about what are the things that need to be true about our data warehouse every single day." We can look over the last three years and we've had the same data warehouse for the past three years. We've had similar user behavior. We've had similar clickstreams. We can say with some confidence the number of unique users that are going to be clicking on ads across all of our ad tech infrastructure is going to be 25 million people, or it's going to be between 15 million people and 35 million people, something like that. That way that's a very simple assertion that you could have about your data warehouse every single day.

[00:23:39] AG: Yeah, absolutely.

[00:23:41] JM: Then you can have more complex ones obviously. Okay, let's go even deeper on this particular example. What is the description language? If I want to test my data warehouse for the validity of data in that data warehouse, what does my test look like?

[00:23:59] AG: The way that we treat these in Great Expectations is they're all stored as JSON objects. You could store them other ways, but they're all very verbose descriptions of, "Hey, here's what should happen." I've been using a few of these names. I don't know if it was really clear, like expect column to exist. That is a type of expectation in our DSL. Expect column values to be between that with underscores in all the right places. That's the abstraction.

One of the choices that we've made in Great Expectations is the core set of expectations are all declarative this way. They're all verbose description of exactly what should happen, because that opens up some other cool possibilities that we can probably talk about in a few minutes.

Am I answering the question you're asking here?

[00:24:43] JM: Absolutely. Absolutely. I mean, you basically have a descriptive language that you say, "I expect this column to exist. I expect maybe in every row in this column to be between 5 and 15," something like that. How does the test hook up to a data warehouse? How does my testing infrastructure talk to a data warehouse in this instance?

[00:25:06] AG: In Great Expectations, everything right now is orchestrated through Python. Theoretically, you don't need to do that. You could rewrite in another language, but we do our orchestration just because so much data work is happening in Python, and Great Expectations currently knows how to essentially compile expectations to three different targets, and I use compile very loosely. It’s Python.

One environment is Pandas data frames. Anything you're doing in Pandas, you can apply expectations directly to the data frame. The second one is Spark. If you’re using Spark data frames, PySpark specifically, the same DSL can be executed against the Spark data frame. I'll put an asterisk there. There are a couple expectations that we have not yet implemented, but we’ll get there before too long.

The last one and the one that really expands the scope is SQL Alchemy. Anything the SQL Alchemy can establish a connection to, and that could Postgres, SQLite, Hive, MySQL, OracleDB, like lots and lots of use cases there, you can use exactly the same expectations as in other contexts and use SQL Alchemy as basically a compiler to execute them as SQL natively in your database.

The core philosophy there is, “Look, for data of any size, you want to be able to control where the execution happens. You don't want to have to create a copy of that data and pull it back and run it somewhere else.” Being able to run it in your environment is really important. Take the compute to the data.

[00:26:31] JM: Maybe I am not envisioning the infrastructure super well, but like if I have my data in Redshift, can I execute the necessary queries against Redshift?

[00:26:41] AG: Yes.

[00:26:43] JM: And that's because Redshift speak SQL?

[00:26:44] AG: That's right.

[00:26:45] JM: Okay. Got it. Basically, any data warehouse that speaks SQL, I can issue these queries against.

[00:26:51] AG: That's right. Again, asterisk. SQL Alchemy is not a perfect universal translator for all databases. So we're still picking up a fair number of edge cases where we're still kind of tweaking the internals of expectations to handle the peculiarities of each database.

[00:27:06] JM: Okay. The second example. I've got an Airflow DAG. An Airflow DAG is basically – I mean, it's pretty similar to the first example. It's just you've got maybe multiple steps. Maybe you've got like I want ETL or I want to load my veto clickstream data from HDFS into Spark and then I want to query Spark for some particular table or some particular data frames, some subset of that data. Then I want to put that data into a different place, or join it with something else, or load it into a data warehouse. Then I want to do something else with that data warehouse. Then I want to load into TensorFlow and I want to do something in TensorFlow and then I'll move it somewhere else.

[00:27:51] AG: I can see you're like building this DAG and you're having like nodes and edges.

[00:27:54] JM: Exactly. I mean, I definitely don't have the big data to do this, but I can imagine having the big data someday if I play my cards right. You could imagine tests along that pipeline. Different kinds of tests to validate that I'm not messing anything up along the way.

[00:28:12] AG: Yup. Totally. I mean, one of the really interesting things at sort of an applied math level is when people describe these DAGs, kind of nodes and edges of data flowing through. If you talk to software engineers, they usually think of each node as like a function, like a Lambda of data being processed through. Then the edges between them are data being materialized and sent from one to the other.

If you talk to a data engineer, they often do exactly the opposite and they'll say, "Well, each node in this DAG is a table, and then the edges connecting them are functions or processing."

[00:28:45] JM: Wow!

[00:28:46] AG: It's one of these like mind-blowing thing. But like the funny thing is they're both right, right? They're just the inverse of the other.

[00:28:52] JM: Amazing.

[00:28:53] AG: This first pattern we're talking about, you're saying, "Look, each node is data. We're just going to sort of insert taps and check that in the data flow. The second mode that we're looking at is the inverse of it, but it's equally true, "Hey, each node is like a data processing function in an Airflow DAG," and we keep saying Airflow. There are other alternatives out there, DBT, and Dagster, and so on. But at each node there, you can also test it and they both work.

[00:29:19] JM: The testing infrastructure in this case is basically the same. I mean, you said it yourself. It plugs into Spark or it plugs into one of these databases. Anything that speaks SQL, which is that's going to be the different materialized views.

[00:29:35] AG: Yeah. I mean, there are some specific mechanics to take care of there. For example, how exactly do you get your expectations into that data pipeline? They're stored in a JSON object somewhere. How does that JSON object get instantiated?

There are other issues like if you're going to hit a database with it, okay, we've got to get our credentials in a place where you can pipe that through to the database, unless you've already got the connection, but if you've already got the connection, how does his Great Expectations know about the database connection?

Without getting into all of the details, we've spent the last several months building out a set of abstractions that we call a data context that are basically sort of the simplest case of being able to connect all those things together. It's basically the common denominator between all of the things that we saw people building for themselves in Great Expectations last year. Our goal there is to get it to a place where you don't have to spend three sprints building out your own data validation system and getting all the connections. We've got a pretty basic version that'll work pretty well for everybody or almost everybody out of the box.

[00:30:38] JM: Sorry. Explain that in more detail. What is a data context?

[00:30:41] AG: It's the moving parts required to manage credentials and store expectations and put the results somewhere when you're done. All of those kinds of things.

[00:30:50] JM: Is it like the framework? Basically, the testing framework or the testing system?

[00:30:56] AG: Yeah, that's a good way to think of it. Another analogy that we're using a lot in our Slack channel is Great Expectations through last year gave you this like helpful concept of an expectation and some notebooks that give you some idea for how to start building them, but absolutely no idea how to deploy them.

Now we think of data context as sort of the Jango version of that in the sense that it's on Rails. It gives you sensible defaults that will work for most people, and then you can override anything you want to override.

[00:31:23] JM: When you see users interacting with Great Expectations, do you ever see the problem where your test just get out of date and then you're like, "Eh, I'm just going to give up on these. This is just too burdensome to keep updating," because you might have a test fail one day and then you'd go and you go and look at it and you're like, "Oh! I just tested that I have – The number of users I have falls outside the bounds of my normal number of users. Well, that's just because I had growth in users and like I don't really want to go and update my Great Expectations test every time I have a user growth."

[00:31:59] AG: Yeah. I mean, I think of this as notifi cation fatigue or in the false positives problem. The way we see most people dealing with that is by segmenting out what you consider a true error versus what you consider a warning. The idea is errors demand immediate attention. You may be blocking the pipeline and you have to go in and adjust something.

The bar for triggering an error ought to be pretty high. You ought to be pretty sure that like the pain of answering or picking up your pager and dealing with these is a lot lower than the pain that'd be caused by having that bad data propagated downstream or the dashboard blow and the CMO comes and shouts at you.

By giving people this kind of lever where they can control it, you can say some of these things are warnings, some of them are errors. That mostly alleviates that pressure. I think there's still a lot to do there, honestly. But, frankly, most the people using Great Expectations, like the idea of having zero tests, once you've lived in a world that has some tests, you'd much rather live in a world that has some tests than nothing. So disabling some tests and keep them going isn't too big of a deal.

[00:33:01] JM: Okay, let's go there then. I love worlds without zero tests, because I hate writing tests, and many of my jobs when I was a software engineer – I mean, maybe this is just because I was a low-level software engineer through the duration of my career. Many of my jobs involved writing tests and I just despised that process. The people who are actually using Great Expectations to write tests, are they basically the people who have gotten so burned by mistaken, like Datafl ows, like they've made some mistake where something got confi gured incorrectly and like an entire series of HDFS hourly jobs were written improperly and then they had to do something like a rollback Kafka or like replace Kafka and spend like eight days trying to replay Kafka properly and then they have to validate the Kafka data manually and it is just a nightmare? There are these people.

[00:33:56] AG: Yeah. Yeah, it's that the never again crowd. I still occasionally wake up in the middle of the night thinking about the one time that we had that client project that was due two days later and I screwed up the Spark job. Like most people who are pros in this industry –

[00:34:13] JM: Is that a true story? Can you tell me that story in the abstract?

[00:34:16] AG: I can tell it to you in the abstract. I was consulting with a company. They were relying on me to get – They had built out a bunch of data pipelines in SQL. They did not scale up to the scale needed for processing a lot of data for one of their clients. As an outsider, I was the critical guy to get the analysis done so that they could go and pitch this client, which for them was going to actually be like their big sale of that year.

They were really relying on me, and I was looking at taking an important job with that company. The night, because I was flying out to help present and get everything ready, and the night before, I realized there's a bug in the pipeline and we didn't catch it earlier and the results that I had shared to the CEO the day before were wrong in some pretty important ways.

I was professional enough to have signaled to him that this is still a draft, that take this with a grain of salt. It's possible there will be changes. But I had not anticipated how big those changes were going to need to be. The whole story was that risk of changing. There are 24 hours where I knew that I had 48 hours of work and I had no idea if coming out of the other side of that, any of it was going to work. It was terrifying.

[00:35:29] JM: Right. Yeah, the closest thing I've had to that is like I mess up something in Git and I get myself into a state in Git where I don't know how to rollback and I have to go and talk to like the senior engineer and say, “Look, can you enter the magical cherry picking and rebasing commands that get me back to where I need to be?” That's probably a cakewalk compared to reinventing like a dataset in HDFS. Maybe. Maybe not. It depends on the context.

[00:35:55] AG: Yeah, it depends on the context. It depends on – But the things that they have in common are like, “Shoot! I mean, over my head. Other people are counting on me and I can get us back in time.” That visceral fear, like a lot of people have that.

[00:36:07] JM: Yeah, so much anxiety. Since you're like building this platform and it’s open source community and you got a community of people. Any anecdotes you've heard from people that’s like nightmare stories? Can you drive it home further for the audience?

[00:36:21] AG: Oh man! Talk to any data engineer, like they've all got these stories. They’re out there. There a lot of the day the dashboard broke. But when you get out there and you say like, “Okay. What happened? Because the dashboard broke,” right?

[00:36:35] JM: Oh no! It’s not the dashboard. It’s something much deeper.

[00:36:36] AG: Yeah. It’s like all the salespeople came to me and said, “Wait, what happened to my bonus?” If sales are down 20%, who screwed up? Who's getting fired because of that? Shoot! That's a high-stakes conversation. There is the machine learning model that was deployed upside down. I've now heard of three of these. One was recruiting exactly the wrong candidates into a job pipeline. Just in the handoff between data science and data engineering, something slipped, a negative one looks –

[00:37:06] JM: Oh my God!

[00:37:09] AG: There's one of those that was like for patients in a medical setting. So recruiting exactly the wrong patients. Fortunately in that case, that was also medical triage. The doctors were very confused. But for a few months, they were looking at the wrong patients and wondering why we can't find anybody else is a better fit for this new clinical program we're spinning out.

This is why a minute ago when you talked about the Netflix example, how much is it really matter if I watched Le Mis or not. We're talking about like deploying the thing upside down and just not knowing for a long time, or like things where people's jobs are on the line. It's crazy. It's mostly untested today.

[00:37:49] JM: I do sometimes – The thing is it seems so hard to write these tests, because sometimes I'm on Amazon and I'm looking at the product recommendations for me and I'm like, "This is the dumbest product recommendation." I do not need like a sponge with like a selfie stick apparatus attached so that I can like sponge my own back.

[00:38:10] AG: I don't know what's in your click history.

[00:38:13] JM: Exactly. That's my point though, is whenever I see one of those things, I'm like, "Okay. What is the explanation –" Or when you get an ad, like when I look at my phone and I'm like, "What the heck is this ad?" Then I think back to what is the behavior I've taken in public settings on the Internet and you realize, "Oh! It's because I mentioned this thing on Twitter. Now I'm getting 50 recommendations for the things that are tangentially related or collaboratively filtered with something related." My point here is just like it can be very hard to anticipate in advance something that would be a rational outcome from a machine learning model. How do you actually create the proper test coverage?

[00:38:56] AG: I think this is another good place to separate out kind of what's a testing problem versus something else. Model explainability I think of as related to testing, like they're both giving you more insight into how complicated thing works. But it's not exactly the same thing as testing.

Going beyond that, like the fact that the world wants to sell you a bunch of weird crap, it's sort of an existential problem. It has nothing to do with the data pipelines that are processing it. I'm just hedging a little bit there, because I don't believe that testing alone is going to stop weird ads from popping up on Amazon, right? But what it might give you is at least the people running the system will be able to diagnose why that's happening. If they considered it a bad thing, then they can it slow it down.

Actually, I'm a little bit of an idealist on this. I hope that better data systems and more transparent data systems can actually make the world a more ethical place in general, or at least can be a lever that can be used to move the world in that direction. I'd hope that they could be part of the solution there.

[00:39:54] JM: We actually didn't run through the third example, the third use case. We talked about the data warehouse thing. We talked about the Airflow thing. The third one was what exactly? Like testing production systems or something?

[00:40:09] AG: No. Just the opposite, like testing in dev. As you're making changes to the pipeline code itself, testing before you roll it out.

[00:40:16] JM: Is this like somebody's working on a Jupyter Notebook and they're changing stuff?

[00:40:23] AG: Think about that Airflow DAG that we're talking about a minute ago. Most of the time, that DAG is just sitting there processing and you want to be checking the data as it goes through. But let's say that something has changed this quarter and there's a reason for you to go in and update it. Before you make those changes, yeah, you probably want to test that as well.

[00:40:41] JM: Something has changed this quarter.

[00:40:43] AG: Specifically, let's say – Oh, I'm making this up. Let's say you're processing logs that are coming off of an app and the folks who are generating those logs have introduced some new entries that your data pipeline was ignoring before, but you not want to do something with those. So I'm going to bolt a few new things on to my Airflow DAG to process the new log files.

[00:41:04] JM: Oh, okay.

[00:41:05] AG: Yeah. Like you've added new instrumentation in the apps so you can see when certain things are being clicked, and before that was just totally ignored.

[00:41:12] JM: I see, and I think you pointed out earlier that that can be kind of tricky because then you need to get some testing set of data from the people who are getting ready to push this change to production. You just say, "Hey, can you take your data from staging, or logging data from staging and send me a sample of it so that I can test it in my data pipeline?"

[00:41:31] AG: Yeah, exactly. They're not thinking of this as a sample of data. They're thinking of this as, "Oh! We added instrumentation to a few more places in the logs." There's this kind of an impedance mismatch in the way that the upstream team and the downstream team thinks about it there. Not always, but a lot of the times.

[00:41:46] JM: In that case, you're actually portraying a scenario where the logging team or the DevOps team or whatever, that they've updated the logs with a new field or they've changed some existing field to be represented with a double instead of an int or something. They're not thinking of logging data as production data. But in some cases, this kind of exhaust logging data can actually have a mission-critical production impacting use case.

[00:42:15] AG: Yeah, absolutely. You say mission-critical, that might be, "Oh, this is the thing that we base our dashboards on and that's how we steer the business," or it might be even more tightly tied in like that flows through into, say, a master user table that we used to profile users and decide which new features or something to surface. In some cases, actually flows right back into product.

I would say this is not a hypothetical situation. I'm making up numbers here. But I guess that 19/20 teams that instrument their apps and then that same data gets consumed downstream by data teams, I guess that 19 out of 20 of those do not ask permission and do not tell the data team that these changes are happening. It’s like a really common source of thrash for data teams.

[00:42:57] JM: In terms of actually building Great Expectations, describe the engineering that's gone into Great Expectations. What is going on under the hood? What have you built?

[00:43:09] AG: Yeah. This is the place where it’s probably good to talk about the history of the project just a little bit.

[00:43:12] JM: Please. Yeah.

[00:43:14] AG: So it started as a side project, just kind of nights and weekends collaboration a couple of years ago. James Campbell and I get, a good friend, who's working also in a data leadership role at that time, we both recognized that like, “Hey, having this abstraction would just be helpful in a lot of places.”

We started working on it on the side. At that point, it was basically a tool that you could use in Pandas in notebooks, and then if you wanted to, you could kind of write your own scripts to deploy it. That was helpful. It was interesting. We took that to Strata – What? Two years ago, and said like, “Hey, here's a thing we’re working on. It seems like the rest of the world might like this. Have at it.”

From that point, we started to get like really interesting direction from the community in general. One was getting pulled into SQL Alchemy and saying like, “Look, it's not enough for this just to work on data frames. It has to deploy against real infrastructure.” That totally made sense and was kind of compatible with what we've been thinking, but we got pushed there way faster than I would've guessed. Spark came later.

The other big changes that have happened over the – What I would say is at that point in the kind of software craftsmanship process, the main things we were thinking about are what are the control planes that you need to really describe what data looks like and how data works in a meaningful way?

For example, one of the innovations that came out of that period is this parameter called Mostly. Most of the expectations you can say mostly equals .95, and that means you don't need to insist that this expectation is true on every single row. It just needs to be true at least 95% of the time, and it turns out that that one little feature makes it way more useful to a lot of data teams because it's often not the case that you can insist that your data is 100% correct. But if you can say, "Look, it's 80% correct," and then I'm going to kind of ratchet that up later. That turns out to be a really, really useful affordance. Things like that.

The kind of quality of our decision-making at that time was a lot of really fine-grained back-and-forth on exactly how APIs should work and exactly how people should be able to express themselves. It's kind of early chapter.

[00:45:18] JM: Take me forward in the timeline.

[00:45:19] AG: Yup. Moving through the next year was kind of building out to address other backends. Taking all those affordances and making them work in SQL or making them work in Spark. There's a fair amount of kind of real engineering we have to do to start to make that work. Then I'd say for the last six months, the thing we've been focused on much more heavily is how do you get this into actual deployment, which is where that data context concept we are talking about before is. The way we think of that is the goal is to be able to deploy in a day, right? Most teams that have deployed Great Expectations prior to the middle of last year, they were spending at least a month to build their own data validation system and they were mostly building the same thing. So let's just give kind of standard boilerplate. So instead of a month, it can be a day. That's one big direction.

The other direction is extending the usefulness of expectations beyond just testing. I'd say there are two big directions here. One is this notion of expectation is useful as a test, but it also turns out to be really useful as documentation. In the last few months, we've deployed what we call compile to docs, which is the ability to take expectations and translate them from JSON objects in machine-readable format into human readable format. That's something that you could put, it uses data documentation or a data dictionary, or something that you could put into a master data management tool.

For example, we've seen people using – What is it called? GCP's. They have a data management tool, and I'm blanking on the name. But you can kind of pipe your own documentation into that and being able to say not in JSON, but like bullet points and graphs, like, "Here's what the data should look like at this stage in the pipeline," is really useful for creating visibility and helping teams just kind of share what they're doing with other stakeholders.

[00:47:08] JM: This project came out of Superconductive Health, right? The company that you were working on before?

[00:47:16] AG: Sort of. I was starting up this data consulting company in the healthcare data and definitely saw the need. But at that point, James – I mean, James was in a role in government, and he and I were the main collaborators on this. It didn't become a company thing until quite a bit later.

[00:47:33] JM: Fascinating. Can you tell me more about that progress? You're working on a data consultancy, like a data engineering consultancy specifically for healthcare companies. So like healthcare companies that have data engineering problems. You say, "This is a very specific domain. I want to go into this domain and specialize in it," and then out of that work came Great Expectations.

[00:47:55] AG: Yeah. I mean, put it this way, if you're working as a data consultant, what you really don't want to have happen is for you to finish up a contract. Wash your hands and say, "Okay. Great. Everything is running. Go for it." Then the two months later for them to call you back and say, "Hey, the dashboard is broken or our machine learning model is doing something weird."

As a consultant, you usually want to be pretty buttoned up about what's the scope, what are the requirements? How do we define those in a way that we can contract based on it? There's some notion of kind of self-defense here, right? You want to be in a place where you can kind of demonstrate the quality of your work? Then if the dashboard breaks, be able to say like, “Well, look. We built it right for the Dataview ahead at that time. The problem isn’t our software. The problem is that your upstream data has changed. Go talk to your data vendor.” There's a little bit of selfishness there and just like wanting to make sure that we could show that our quality of work was very good and then defend that.

On the flipside, there’s also just an altruistic side, like I've dealt with that waking up sweating in the middle of the night because I think about the bad data project for a long time. I just don't want to live in a world where that's true anymore. I think James and I both recognize that and just wanted to solve that problem once and for all. Sort of rambling answer, but this isn’t like a super crisp, like I knew what I wanted when I went into it kind of story.

[00:49:16] JM: It's interesting, and it's not – Also, in entrepreneurship terms, it’s not really atypical, right? This is how a lot of companies get started. You start off in one direction and it exposes you to problems in another direction.

What was the moment where you said like, “Oh, this is a big enough problem to focus my entire work on. I shouldn't be doing consultancy. I should just go all-in on data pipeline testing.”

[00:49:42] AG: In the middle of last year, we started to get a bunch of inbound requests from companies and they'd say, “Hey, we’re not a healthcare company,” but I've seen they have something to do with this Great Expectations project, and we really need that. Like, “Her are a bunch of dollars. Can you solve that problem for us?”

As an entrepreneur, you get 5 or 6 of those and you start to think like, “Okay. There’s a pattern here. People are really feeling this pain and this isn’t just some niche tool.” After several of those we said, “Okay. Let’s take a look at what it would take to like really make a go of productizing this thing and pushing it forward and really making it a success.

[00:50:20] JM: What’s been the hardest part of building Great Expectations so far?

[00:50:24] AG: I think the hardest part is that like a tool like this has to be built in direct contact with data because it's all about picking up edge cases and being able to expressively identify like, "Here's what my data really looks like."

I think more than most software engineering, there's been kind of a three steps forward, one step back process of trying things out in new context and just like really getting it to fit into the cracks. Early on, I talked about how getting the parameters and the API for each expectation, right? That required quite a bit of craftsmanship of where we found the patterns that really worked.

Now I would say we're going through a similar thing, but it's more about deployment and making sure it works with all of the infrastructure. For example, the thing that I'm going to be working on probably this weekend is we're building out a QA grid that will take all of the expectations and a whole bunch of sample datasets and just kind of rapid fire them against lots and lots of different databases to make sure that like everything works all the way through on all of them.

The crazy thing is we don't even believe that that's going to catch all the edge cases but it's going to catch some of the edge cases that otherwise people would run into in their own environments. It's just like that's the level of kind of depths of testing that you need to make a tool like this really robust.

[00:51:42] JM: How do you test Great Expectations itself?

[00:51:47] AG: How do you test the tests? Yeah. Two layers there. Each expectations, we have really pretty deep parameterized tests to make sure that it works correctly, throws errors in the right places and does everything that it needs to do. As I said before, a big part of that is we've had to handcraft a whole bunch of toy examples of data to make sure that you stress all the code paths.

That part of the codebase is in really good health, but as of today, that is being run through our CI against Pandas, Spark, SQLite and Postgres. So if you're using Redshift, it's possible that Redshift behaves a little bit differently when you hit it with certain SQL commit query. This test grid that I'm talking about is intended to cover exactly that case of just making sure that all the idiosyncrasies of all these different data processing systems, we have some way to have a net to catch those.

[00:52:39] JM: The users of Great Expectations, what kinds of applications are they using? Are they using it for – Can you tell me about like – Are they using it for anything that you didn't expect?

[00:52:52] AG: It's all over the place. People are using Great Expectations, some are at tech firms. Some are at startups. Some are at fortune 100 companies. Some are in government. A lot of data consultants actually I think maybe picking up on some of the same use cases we saw. There's no one domain.

I'd say the main – Most teams that are using Great Expectations are getting to the place where that task acknowledge problem we talked about earlier is starting to feel really heavy. It's often at the point where you're going from 3 to 5 data scientists and data engineers at the team, getting to a place where you have multiple stakeholders around the organization who are really relying on you. It's not so much by domain or by problem shape and more by when you get to the point where just maintaining something out of your own head just doesn't scale anymore. What we see is that's usually around 5 people on the team.

[SPONSOR MESSAGE]

[00:53:55] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have Datastax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

Datastax provides Datastax enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. Datastax enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run Datastax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce.

To learn more about Apache Cassandra and Datastax's enterprise, go to datastax.com/sedaily. That's Datastax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to Datastax for being a sponsor of Software Engineering Daily. It's a great honor to have Datastax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[INTERVIEW CONTINUED]

[00:55:35] JM: Let's zoom out a little bit, because you have worked as a data consultant. Now you're working on a particular problem in data engineering, but the world of data engineering has a lot of different outstanding problems. What do you see is the biggest unsolved problems in data engineering? The acute problems that you're not working on today.

[00:55:55] AG: I see – I don't have a really crisp way of stating the problems. Others do, and I'm sort of waiting to see which are those formulations really wins. But just like we talked about this kind of dual nature of data pipelines, I see really interesting stuff going on in what I call the opinionated DAG runner world. I think Airflow was the first example of, "Hey, here's a way to orchestrate DAGs and visualize them and think about them." I think there are a whole bunch of successor technologies. I say successor. It's not clear to me how they'll integrate with Airflow or where that's going to go. But DBT is getting a lot of really interesting pick up. There's the elemental crew over at Dagster Prefect. I don't know them. I would like to know them, but I can tell they're thinking about the problem in a similar way.

Kedro – I don't know if this is on your radar, but McKinsey Consulting acquired a company called Quantum Black, and they have an open source framework called Kedro that manages DAGs. I haven't dug into it super carefully, but it looks really interesting. Anyway, four really promising companies in addition to all the stuff going on around Airflow says to me that there's a really interesting set of things going on in that space.

[00:57:04] JM: What about there's a project that came out of Lyft called Flight. There's a project that came out of Uber. I can't remember the name of that one, but I think there's another –

[00:57:12] AG: Meta something. Metafl ow?

[00:57:15] JM: I don't know.

[00:57:15] AG: No. That's Netfl ix, I want to say.

[00:57:17] JM: But the thing that came out, Flight, was what came out of Lyft. I don't know if you saw that one.

[00:57:21] AG: I haven't dug into it carefully, but also in the space. Yeah.

[00:57:26] JM: What problems does Airfl ow not solved in the DAG space?

[00:57:31] AG: That's a really good question. My perception there is that – I mean, Airfl ow started off as a scheduling engine, right? Then it just got really, really ambitious. The thing that felt to me like very new and very useful is it let you express yourself in DAGs. It let you draw the nodes and draw that edges and visualize them, and that was really powerful.

But the way Airflow does it, it's thinking about those as blocks of code to be executed and it turns out that there's like a small but really important mismatch in executing the code and writing the code and thinking about kind of what are the units of analysis for composing real DAGs.

Airflow still lives quite close to the metal, and all of these other projects that I've described live much closer to the author, much closer to the developer. I think the interesting evolution to the space is going to be figuring out how those two tie together.

[00:58:23] JM: Wait. You said the author versus the developer?

[00:58:26] AG: No. Those are the same person. What I would say is Airflow is good at figuring out what should be executed. What is going to run? But if you're writing code and debugging it and kind of thinking about that as decomposable units, there are a lot of small ways in which the affordances of Airflow just don't quite match with the way you'd want to think about it.

For example, lots and lots of things happen in Airflow is side effects. It's not a functional language in the sense of inputs going to outputs, which is how you're going to be thinking about it if you are really working in like a mental DAG framework, or at least how I'm thinking about it. Maybe my mental model doesn't translate to everybody there.

[00:59:04] JM: Oh! So you're saying that with Airflow, data is actually getting overwritten as supposed to just completely new data being created?

[00:59:13] AG: Not just overwritten. If you get into specific functions in Airflow, a lot of the time the first line in the function is fetch data from somewhere and the last line is write data out to somewhere. The weird thing is Airflow doesn't give you really any tools for inspecting what data is coming in or what data is going out. If you're building data tools, you probably want to know that.

[00:59:36] JM: Interesting.

[00:59:37] AG: I'm overstating it a little bit there. There are some tools for doing it. It's just you can tell when you really play with it that that's not what was originally in the authors' minds when they built the thing.

[00:59:47] JM: As far as the business for Great Expectations, do you have a vision for what the company looks like?

[00:59:56] AG: For the time being, we are running partially in consulting mode and then kind of plowing all of our effort back into the open source project. It puts us in this fun place where we just build a cool thing and see how far it goes. Down the road, I do have a lot of ideas for productizing it. I mean, I think this – I say productizing. The commitment we're making to the community is everything that's in Great Expectations now as open source will always be open source. There's never going to be a rollback where we try and commercialize the core thing. But this notion of being able to test data and document it and collaborate on those things, there's a ton of scope there for, for example, inviting non-developers into the workfl ow or tools that bridge the gap between data science and data engineering.

I'm not too stressed about exactly what a future product will look like. For now, it's clear something is happening here. We want to build that out. See how big the community can get and just make it as useful as we can.

[01:00:53] JM: Totally. That's smart. Before you're in data engineering, you got a PhD. If I have this correct, public policy, political science, and complex systems.

[01:01:06] AG: Yep. Yep.

[01:01:07] JM: What does that mean? What were you trying – I mean, the fi rst two kind of makes sense. What is a complex system?

[01:01:12] AG: If you're going to go into data engineering, PhD is a pretty long detour. I'm not sure I'd recommend it. But if you're going to, then I would highly recommend a place like the Complex Systems Lab. At Michigan, it's this interdisciplinary lab where the lingua franca is applied math. It's physicists and bioinformaticists, and the odd social scientist like me all getting together and using mathematical models. Just talk about how the world works.

My dissertation while I was there, I was in public policy and poli sci, but my dissertation was this like very kind of applied math computation heavy project where I was downloading blogs. I was scraping the contents and then running natural language classifiers on them or natural language processing rather, so text classifiers, and horrible timing.

In the middle of the Obama administration, I was classifying like the blogosphere for Civility, except for some like kind of angry stuff on the right. There wasn't much going on there. Every once in a while I wonder like, "Oh! Should I just go and dust that off now?" because there'd actually be an interesting conversation about civility in today's world.

But back when I was doing the work, it was really interesting technical exercise, but I don't know. Maybe I should've seen the writing on the wall for where we're going. Long answer to your question.

[01:02:28] JM: No. That's totally fine. Was your transition from academia to data engineering, was that just by virtue of the fact that you realized, in order to study these problems, you would need to write big SQL queries or use distributed systems and that naturally led into working on engineering problems?

[01:02:50] AG: Yeah. I mean I'm fascinated by how people interact together. For the dissertation, I realized there was starting to be a lot of data on the web about that. In the process of writing, I wrote 99 different web crawlers to make the data collection work and I realized like, "If I worked at Google, this would be easy." A blogger was really big at that time. If you want to be studying social systems through the lens of the Internet, like, man, tech is the place to be.

[01:03:21] JM: What about biological systems? If you want to study biological systems, should you be in tech or should you be in academia?

[01:03:27] AG: How many PhDs do you want me to have? I don't know for biological systems. It's an interesting question.

[01:03:34] JM: But you're a medical consultant. Data engineering medical consultant, right?

[01:03:38] AG: Oh, yeah. But I mean most of the work I was doing there was things like figuring out how billing and claims data flows through the system. From that, trying – Dirty secret in American medical practice is very little structure data is captured on patient health. Almost all the structure data that's captured is about billing. Most of the best in class algorithms today are being built on building data.

[01:04:03] JM: Really?

[01:04:04] AG: Yeah. Our doctors called the stained glass window. They said like, “Okay. We can't really see the patient's chart. We can't really see what's going on.” But based on the fact that they were billed for this catheter insertion and these drugs and they had a hospitalization at this time, I'm pretty sure that patient is suffering from X. Those are the kinds of systems I was building.

[01:04:26] JM: So reasoning about a patient's actual condition from billing data?

[01:04:30] AG: Basically. Yeah.

[01:04:31] JM: How effective is that?

[01:04:33] AG: Way better than having no data at all, but still pretty fuzzy.

[01:04:40] JM: What's the future of academia?

[01:04:41] AG: I’m not sure I want to talk about this one on the air.

[01:04:44] JM: It’s all right. The thing is the people who are listening this who are in academia are the ones who are considering to –

[01:04:52] AG: Yeah. Get out. Get out now.

[01:04:55] JM: They’re actually looking for more reason to transition out. Give them some solace.

[01:04:59] AG: Yeah. Seriously, what I would say, is if you are thinking about the reason to stay in academia is because you really love research as it goes through the peer review process, like the academic peer review process. I found that I just couldn't stomach that. It was so slow.

They talk about progress in academia happens one funeral at a time. I definitely felt that. My advisors were great. The program was super, super supportive, but I could tell that like I was applying these new methods and new methods open up new questions, and I’m like really interested. I couldn't tell if change in the academic discipline was going to happen within the horizon of my career, like 7-year, 10-year clock. Were things going to change fast enough for that to happen?

If you're looking for reasons to get out of the academy, what I would say is in industry, you can attack a lot of the same questions. Now you've got to find the right companies. You got to find the right commercial reasons to pursue them, and that definitely is a filter. But there's a lot of really interesting stuff and you can just get at it faster. Yeah, if you're the impatient type, you might think about not staying in academic research forever.

[01:06:08] JM: Abe, thanks for coming on the show. Great talking to you.

[01:06:09] AG: Thanks, Jeff. This has been a lot of fun.

[END OF INTERVIEW]

[01:06:21] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I’m a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There’s very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[END]