EPISODE 307

[INTRODUCTION]

[0:00:00.3] JM: Data scientists need flexible interfaces for displaying and manipulating datasets. Data engineers need to be able to visualize how their data pipelines wire together databases and data processing frameworks. DevOps engineers need dashboards to understand their monitoring data at a high level. All of these programmers are building data applications. Data applications let us visualize and manipulate datasets effectively.

In today's episode, Dave King joins the show to describe the growing importance of data applications and how data application development is changing. Many knowledge workers use flexible tools like spreadsheets and business intelligence applications, but when you build a domain specific data application for a knowledge workers, you can unlock a higher degree of leverage for that knowledge worker.

We discuss data applications and the future of knowledge work in this episode with Dave King. Full disclosure; Exaptive is a company that was founded by Dave King, and Exaptive is a sponsor of Software Engineering Daily.

[SPONSOR MESSAGE]

[0:01:12.3] JM: Couchbase is a NoSQL database that powers digital businesses. Developers around the world choose Couchbase for its advantages in data model flexibility, elastic scalability, performance, and 24 by 365 availability to build enterprise web, mobile, and IoT applications. The Couchbase platform includes Couchbase, Couchbase Lite, which is the first mobile NoSQL database, and Couchbase Sync Gateway.

Couchbase is designed for global deployments with configurable cross data center replication to increase data locality and availability. Running Couchbase in containers on Docker, Kubernetes, Mesos, or Red Hat OpenShift is easy, and at developer.couchbase.com, you could find tutorials on how to build-out your Couchbase deployment. All Couchbase products are open source projects.

Couchbase customers include industry leaders, like AOL, Amadeus, AT&T, Cisco, Comcast, Concur, Disney, Dixons, eBay, General Electric, Marriott, Neiman Marcus, PayPal, Ryanair, Rakuten/Viber, Tesco, Verizon, Wells Fargo, as well as hundreds of other household names.

Thanks to Couchbase for being a new sponsor of Software Engineering Daily. We really appreciate it.

[INTERVIEw]

[0:02:40.3] JM: Dave King is the CEO of Exaptive. Dave, welcome to Software Engineering Daily.

[0:02:45.8] DK: Hi, thanks for having me.

[0:02:47.4] JM: Today we're going to talk about data applications and we will get to how you build data applications and how that ecosystem is developing. Let's just start with the question of what is a data application?

[0:03:01.0] DK: I think the best way to think about a data application is to just think about some of the software that we use today and we don't necessarily think of it as a data application. For example, everybody I know who buys plane tickets pretty much buys them through Orbitz or KAYAK. My mom will buy plane tickets through Orbitz.

When she does so, she doesn't feel like she's consuming some miracle of data visualization or data science, even though what's happening behind the scenes for Orbitz to work is actually pretty sophisticated, right? There's a whole bunch of data munging that it's doing from different sources. There's actually some pretty clever visualization in terms of that matrix that says if you left a day later or came back a day earlier, here's what the price would do. There's a whole bunch of sliders that let you adjust things and see the changes, or sort.

It is dealing with a lot of data, it's a pretty sophisticated application, but my mom doesn't think of it that way, because, ultimately, she can just click on a flight and book a ticket. That's what I think makes it feel like an application, and the reason I bring that up is that in lots of other areas where we're dealing with large amounts of data, it doesn't feel that way at all. There's no sort of proverbial booking of the ticket. You have some complex application, you do some analysis, you have these visualizations, but it's just reporting.

I like to sort of ask people, "Imagine if Orbitz could do everything that Orbitz does today, except book a ticket. What would that be like?" That would just be like travel agents, right? That didn't sort of change the way our travel works.

When I talk about data application, that's what I'm talking about, sort of moving from an environment where we're used to like looking at data and reporting on it, to more of an environment where we're just doing our work with data as part of the backbone.

[0:05:04.7] JM: If we think about this from an engineering perspective, from, let's take like an ops person that is looking at this dashboard that is an aggregation of a bunch of metrics that have accumulated and that's data from logs and stuff and it's being abstracted into higher level of thing that they're looking in our dashboard. If they want to actually interact with that, it's more process of going to the command line. It's probably a little more complicated than merely clicking within their dashboard. You're kind of talking about data applications as something where it's more interactive than that mere consumptive experience.

[0:05:41.7] DK: Yeah, I think I'd add on that in a couple of ways. I think you're exactly right, that the reason DevOps people look at dashboards is, ultimately, because they want to take some action if there's something wrong. I think it's a good example that if you see in your dashboard that a server is dangerously close to running out of memory, you can't just reallocate or add memory to the server within that dashboard. Maybe there's some specific tools that let you do that, but those tools are data applications.

Likewise, I used to work in high volume manufacturing and we would create all sorts of dashboards for getting a sense of what was going on with a factory and if we had any defects coming off from the manufacturing line. If you see a defect coming off the manufacturing line that you think is systemic, you want to be able to immediately put a stop ship on those units and not ship them out the door.

Those sorts of decision-making processes, those actions, a lot of software sort of sticks us in a bit of a dichotomized world, like on one side you pull your data, you do your reporting, you come to some conclusions, and then you sort of slide your chair over to a different system where you do stuff. Then, after you do stuff, you slide back to the first system and you see if it had effect. What I'm interested in is how to make that more seamless.

[0:07:04.6] JM: Makes a lot of sense, because it's kind of like a false dichotomy between this read only world of dashboards and the right-heavy world that's perhaps bloated with too much options and complexity for anybody who is not highly technical, like an engineer, a DevOps person, or a data scientist to interact with.

It reminds me an episode we did recently with the Ionic framework, which is framework for building cross-platform mobile apps, and the guest I had on was discussing how there are these big enterprises that will have 800 internal application — It's a company with 100,000 employees. It's no surprise that they have at least 800 different domain-specific applications. In interesting that they're able to use Ionic to build cross-platform apps. Basically, just the idea that you have all these different roles within the company and they have very specific things that they need done, and having an application that fulfills that specific need can give an employee on different levels of technical knowhow, a lot of leverage. Just having these domain-specific application can give you a lot of leverage.

[0:08:24.1] DK: Yeah, I think you're hitting on a couple interesting points in that last set of statements you've made. One is you're talking about sort of the role of domain-specific expertise, and I think how domain-specific expertise is really different than implementation-specific expertise, right? The person that understands the domain of a particular problem, the person that understands all of the challenges around running a factory, or specific problems around sales, or marketing, or drug discovery, that domain expertise is very different than the expertise to build a software application that makes their job easier.

You have this separation between the people that know the problems they're trying to solve and the people that have to implement those problems. I think that any tool that helps to democratize development a bit and helps to sort of shrink the links in the chain between the domain-matter expert and the implementation I think can have some real advantages, and that's like the mobile app platform you're talking about. That's one of the things it does, is it sort of makes it easier for people to build things, and as a result, you can end up with these much more customized applications.

I think, ultimately, that's something that we all want in lots of different areas, right? In medicine, there's a focus on precision medicine. We don't want a drug where the dosage of the drug was sort of the average of some clinical trial. We want a drug where the dosage is based specifically on our body mass, or our genetics, or our symptoms.

In all of these areas, anytime technology is good enough to allow much more customized solutions, I think that's a win. Traditionally, in software, I think we've been stuck with two options, and that's sort of like the make versus buy option. In the first case, if you want something very specific, you got to make it, it's expensive, it's time-consuming. The people that tend to make it are not always software professionals, so sometimes they're inefficient in the iteration. That's the make side.

If you buy it, you're buying a product that's trying to be sold to more people than just you, so it has a number of generalities that may move it away from your specific use case. I think now we're hitting a point with technology where we can offer people the customized feeling of making an application themselves and the customized outcome, but more in a timeframe and a price point of a general off the shelf platform.

[0:11:11.7] JM: When you're talking about somebody building an application for them self, what are the frictions preventing the average employee from being able to build applications on top of their data? For example, if I'm a marketing person, or if I'm a salesperson and I want to build a rich application, or if I'm somebody who's working at a research laboratory and I want to build like a rich application, but I'm not a software developer, what are the frictions that are preventing me from being able to build things on top of my data?

[0:11:47.1] DK: Yeah. Okay, good question. I think there's a multitude of them. One friction point I think is just awareness that such a thing as possible. There's sort of a cultural friction. Right now, we have a culture where software doesn't necessarily get built by non-software people.

That is changing. I think one of the things that we're seeing with data scientists as a role is that data scientists — A lot of data scientists were biologists or other scientists that started writing a lot of statistic scripts and they sort of created this hybrid between working with data and writing some programming.

I think that as there are more languages that are easier to use, it used to be that all languages were compiled and is very complicated to use. Now, we have much more sort of interpreted languages where you get quicker feedback, the whole process of building things is easier. I think that's the first thing, is the sort of the cultural friction, is just realizing that if you're a domain-matter expert and you want to build a rich application, maybe, actually, you can. That's the first friction point to get over is that mindset.

The second piece to get over is the toolkit for doing so. It can still — Even if you're willing to dive in and build an application yourself, it can still be technically daunting to do so. That's where, I think, the sort of future of application development that I'm particularly interested in is this idea of component-based development that makes it very easy for a less technical person to take a technical component that somebody else might have built and put it connected to their data.

The third friction — This sort of cultural friction, and then there's this technical friction. Then, I'd say that the third friction point that comes in is around specialization. This is something that we've seen over, basically, for a very long period of time, all now is sort of moves towards more and more specialized areas.

Where this makes it difficult for building applications is that the person who knows something about their data doesn't, which is the example you're giving, someone has some data and they want to do something with it. They don't necessarily know anything about the algorithmic layer. They don't necessarily know anything about the visualization layer, or the web layer to allow for interactivity.

It used to be that if you were going to make software, you made a program, and that program covered all three layers of the technology stack. Now, if you're going to make a program, you might just focus on data, or algorithm, or visualization, and so I think for someone that's coming at it from a particular direction, like in your example, someone who has some data, they don't necessarily know how to use machine learning algorithms, or even clustering algorithms, or natural language processing, or any of these stuff. That becomes a barrier for their experimentation.

[0:14:47.2] JM: The most prominent bell of the data application idea, or the data application platform that you're describing that I think of is Excel, because we've had Excel for a pretty long time at this point, or spreadsheets, whatever you want to call it. I know people who have never had formal training in programming, they probably have never had formal training in Excel, they've just kind of put their noes to the grindstone and they get to the point where they are very sophisticated at building whatever they need to do in Excel, who build very rich applications in Excel.

I'm convinced that data applications are a thing that people are building, they want to build richer data applications. You are working on Exaptive, which is a platform that allows people to build these rich data applications. Explain what Exaptive does.

[0:15:40.0] DK: Sure. Build on the Excel analogy, because I think you're right, that's a great example of a technology that sort of sat in this middle ground between programming and applications. Excel, I think, was successful for many reasons, but there's four particular ones that I think are apropos to this conversation, which is, one; the technical barrier was low enough that people were willing to try, sort of gets to that cultural bit. It didn't feel like you were sitting down in a text-based code editor. People were willing to try it.

The second thing is that it was generalizable. People use Excel for financial modeling, but they also use it for planning their wedding, or making to-do lists. It's like the core idea of Excel is incredibly horizontal as supposed to vertical. I think that's the second piece of its success.

Then, the third piece is that third parties can build on Excel's functionality, that you can have plugins, you have new formulas. The fourth piece is that there was a coding environment baked into Excel so you could — If you knew Visual Basic for applications, you could sort of do anything you wanted in Excel. Those four things created — There's a reason why Excel was the killer app. I think it wasn't any one reason. It was the sort of confluence of those four things.

What we're trying to do with Exaptive is we're trying to take those principles and apply them for today's world where the data that a person wants to work with is not just sitting in an Excel file anymore, and it's not even something that necessarily lends itself to being put in an Excel file either — And I don't mean just because it's too big. I mean, that's one thing, right? All these buzz of big data, it's just hard to put it in Excel file.

Totally separate from that, it's that the data are not necessarily just tabular data. It's not something you would just model as rows and columns. It's not necessarily data that are easy to get from one place. It might be a combination of Twitter data, and census, and medical data. There's all of these new challenges, and the question that we started with when we were building Exaptive was, "How do we give people all of that sort of expressive power of Excel, but upgraded to modern data science?" Make it easier to pull data from different sources. Make it easier to work with link data, instead of tabular data, and make it easier to build applications with all of the sort of fancy interactive visualizations that people were getting attached with with d3 and WebGL, and then be able to use all of the things coming out of the machine leaning revolution around deep learning and all these algorithms.

What we're building with Exaptive in a lot of ways has a vision very much like Excel, where instead of formulas that go into the cells, there's components, and those components can be written in any language. They can be written in Python, they can be written in R, they can be written in JavaScript on the client's side. They can use any third party library or open source library, and every one of those components sort of becomes like a formula in Excel that can be connected together to work on these data structures in a way that's interactive.

[SPONSOR BREAK]

[0:19:14.2] JM: Simplify continuous delivery with GoCD, the on-premise, open source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines, and you can visualize them end to end with its value stream map. You get complete visibility into and control of your company's deployments.

At gocd.io/sedaily, you can find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent, predictable deliveries. Visit gocd.io/

sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery, are available. Thank you to GoCD and thank you to ThoughtWorks. I'm a huge fan of ThoughtWorks and their products including GoCD, and we're fans of continuous delivery. Check out gocd.io.sedaily.

[INTERVIEW CONTINUED]

[0:20:26.2] JM: Now, if I'm a data engineer, or a data scientist, I spend 90% of my time cleaning data, doing stuff like how do I connect to the SQL database and how do I get that data imported or queried correctly into my application? How do I make sure that there's integrity to this data, there's no missing entries that are going to mess up my averages. What's the shortcut to making that easier? If you've got somebody who's got a database, or SQL database, or census information and then they want to pull from a Twitter API and they want to be able to do this with components and they are non-technical or they have not a programmer's level of technical proficiency, give me the top-down understanding of how that person can build a data application.

[0:21:25.8] DK: Sure. One of the fundamental differences that I alluded to when we're talking about Excel, between Exaptive and Excel, is this idea that, with Exaptive, we're not trying to constrain things to tabular data, to row and column data.

The way this manifests very concretely is that we've designed Exaptive to be perfectly comfortable consuming relational databases, or Excel files, or tabular data. The way that gets modeled in the application is, as I said, of entities with attributes. I think this is important, because if you sort of think about getting used to Excel, it's not the most obvious way. It's not the intuitive way that people think about data. Maybe it is now, we're all so used to Excel.

The way we tend to think about data is that data are observations about things. Those things are the entities. If I have patients, I know their ages, I know what medicines they're on, I know their heights and their weights, but the patients are entities. If those patients are taking drugs, those drugs are entities and the drugs have a certain formulation and they have a price and they have an efficacy and all that sort of thing.

One thing I've been very interested in for a long time is how we make it easier for people to work with mental models as supposed to data models, if that makes sense. Part of what you described in terms of the challenge of working with data is really the challenge of sort of schema synthesization, which is that every one of these sources you described; my own dataset, and API, to Twitter, or census data, every one of those things has its own schema.

That schema is just the way the tables and the fields are laid out. That is a data model, but that's not really the mental model. The mental model goes back to this idea of entities, like people that have things they say on Twitter, and they have incomes reported on the census.

What we've done in Exaptive is we've built Exaptive on top of a linked data model that's based on technology in semantic data. It might be worth — For the listeners that aren't familiar with semantic data, I feel like this is sort of like a mysterious thing when people talk about semantic data. I actually — I don't think it needs to be that complicated.

The idea of semantic data is that any data you want to represent can represented as a set of three word sentences, where you can think about those sentences like a subject, a predicate, or an action, and then an object. If you were think about — Let's think about an Excel file where I've got — Let's say I'm a teacher, I've got kids in my class, I'm making an Excel file with each kids' name, and their mother's name, and their father's name. You can totally imagine how that Excel file would get build out, right? Kid's name in a column, and then mother's name in a column, and father's name in a column.

The problem with this is let's say that I have a kid in my class that has a divorced mom, and I don't know the father. If I put a null in the father's name, if I don't fill that in, what does it mean? Does it mean that I don't know the father's name? Does it mean there isn't a father? If there isn't a father, is it because the father is divorced, or is it because the child was adopted and actually doesn't know the father? There's all these ambiguity.

Then, the other problem that comes in is let's say we have a homosexual couple with a kid and there's two fathers, and now this messes up my data model, because I have this heteronormative data model, and so now I have to put two father's name in one column, or I have to add another column.

The data model totally falls apart, but the conceptual model is really easy. Anyone listening, it's just easy to be like, "Well, some people have one dad, some people have two dads, some people have dad and a mom." The idea with semantic data is you just take the same idea of the data and you break it into a three word sentences. You would say, "Mom, Stephanie; Dave, dad; Rodney," and you would just sort of build on it from there.

The beauty of that is that it handles all of these other cases really, really nicely, because you can just leave off sentences. There's nothing that prevents me from having Dave; mom; Stephanie, and adding another row that says Dave, mom, and — You can just add rows. What this means in terms of linking data is it becomes very easy to concatenate datasets.

If I go out to a database of customers and I get a whole bunch of information and I turn it into these three word sentences, these entities, and these attributes, and these values, and then I go out to Twitter and I find out some other things about what people are saying and I just add those. Then, I go out to some natural language processing and do the sentiment on those statements on Twitter and I add those as three word sentences, "Sentence one sentiment; good. Sentence on sentiment; bad," this starts to build a graph instead of a table. It starts to build a graph, meaning nodes and edges all interconnected.

[0:26:41.5] JM: You're talking about the data representation. There's also this notion of components that you've mentioned before, and the components idea is important, because in Exaptive, you have these components that are encapsulated blocks of code that you can wire together. There is this lower level notion of code where you can actually just work as program if you want to work as a programmer, but there's a higher level of notion of pulling together these blocks of code and this visual representation.

Also, the idea that these components are things that you can reuse overtime, and so maybe you could have a programmer that could write — I can imagine a company where a programmer writes several components and then people who are not programmer, or they're technical people who are not coders, can pull those components together and they can understand what they're doing, but they don't necessarily have to program.

Then there's also just like the network effect of people in the community potentially being able to share different components. Can you describe what a component is and perhaps how the system of different components works together to make it easier to build applications sort of like you use libraries as a programmer?

[0:27:56.9] DK: Sure. Libraries is exactly the right place to sort of start thinking about this, which is that all of software engineering is about abstraction. I'm a software architect, and a programmer myself, and this is what I love about building software, is I love the act of abstraction. I love the fact that you can build a piece of code and you can wrap it inside an interface, and then it can be used either by yourself later, or by other people that don't know anything about what's inside that black box abstraction. That's exactly what makes programming so powerful. That's what has allowed open source to really take off. People produce all sorts of open source libraries that other people use in other things.

I don't want to claim that the idea of components, sort of the Exaptive idea of components, is in any way that we thought of componentization. This is just the tried and true abstraction in programing. Where we are trying to go with it is really pushing this reuse aspect, which is that just because you can make something a component doesn't mean that it's easy to reuse.

What I mean by this is that there's a big difference between something being modular and something being combinatorial. What I mean by that is that if you think about your cellphone and the plug for the charger, those two pieces are modular. The plug for the charger fits in to the plug for the phone — Fits into the phone. It's not combinatorial, because it's not like your charger can connect to anybody else's phone. It can only connect to the phone of that particular brand.

To contrast this, think about Tetris, for example. The reason Tetris is such a fun game is not because there's a specific set of blocks that go together. It's because every block can connect with every other block in a number of different ways, and that's what makes it so fascinating to play.

To bring this back to coding, while the idea of abstraction is not necessarily new, and the idea modularity is not necessarily new. The modularity in abstraction that goes into traditional software development, it puts a big emphasis on modularity as supposed to being combinatorial. We have lots of libraries that can be connected together, but they're not necessarily easy to connect to every other library you might want to use.

One way this manifests is like language specific things. It's easy to use a bunch of different R libraries, but it's not necessarily easy to connect R functionality to some SIDEKIT learn functionality in Python, where it's easy to connect some components written in C++, but not necessarily easy to connect those to something that was done a web frontend.

The idea behind Exaptive when it comes to the component abstraction is to move from simple modularity to more flexible combinatorics. In order to make that happen, we absolutely we want to make it easy for people to write code. I'm a programmer myself and I hate any system that I can't actually write code in. I don't want to be overly constrained.

The first step in Exaptive is that if somebody has technical programming expertise and they want to make a new component, they can do that and they can do that in languages they already know. Then, that component gets wrapped in this Exaptive component layer which defines inputs and outputs. That allows it to then go into a data, a visual dataflow programming environment in which it can plug and play with any other component that has inputs and outputs, all has inputs and outputs in this dataflow even if the insides of components are very different. That's the basic idea.

[0:31:55.8] JM: Okay. You've got a component that can basically be anything, and then in the input format and the output format, you've basically got an interface into the Exaptive — I guess, an interface that defines the world within the component how that gets translated to the world outside of the component, which is the Exaptive world.

What are the constraints that that interface enforces and how do you make component comply with that interface so that you can have guarantees about how these components are going to communicate with each other.

[0:32:34.7] DK: Yeah, that's a great question. Designing that interface is a challenge in finding the sort of right degree of abstraction. We wanted to create something that gives people a lot of flexibility and a lot of expressiveness, but adds a little bit more structure than just sort of pure programming.

The basic idea is that the inputs and the outputs of every component are data payloads, and those data payloads can either be simple types or complex types. An example of a simple type would just be like a string, or an integer value. If you have a component that, let's say, is going to go do a Google search. You can take an input, let's say, it's just an integer, or how many — Or let's say it's a string. What am I going to search for? That becomes just a simple string value.

That string value comes in to the component and then the component authored gets to decide how that input gets processed. That's sort of the simplest idea, is that we're just defining inputs and outputs from the point of view of sort of a messaging system. This is a lot like sort of a micro-service architecture, there's just messages that come in and then there's messages that go out.

Where this becomes a little bit more specific to a data-centric application is that we have a concept around entities and groups of entities which are our data structures. You can think about those just like JSON objects, JavaScript object notation. This has become — There's been lots of different standards for representing data in the web, there was XML, and now there's JSON, but what both XML and JSON have in common is that they both support sort of schema-free designs.

In JSON, you can have an objective that has an object inside it that has different numbers of attributes in the sub-objects. What we've done is we've elevated that JSON concept into a specific entity-based data concept and we called it a duffle. The duffle is like a duffle bag, you put data in the duffle, and then you send it on to — You just sort of send it out your ports as if you're a component. You sent it out your ports.

Then, it's in the Exaptive world of the dataflow, and those data then flow to any other components that are wired together. When the data hits the input ports of the next set of components, then they get received and the next set of components do whatever their component authors coded them to do with those inputs.

One of the pieces to add here that I think traditional programming doesn't really have modeled the same way is that in traditional programming, if you have two objects and they both have two interfaces, it's like those interfaces absolutely just have to line up.

In Exaptive, there's actually three roles which is that there's — In this example, if you had two components, there was the author of component one, the author of component two, but then there's the person that's building the applications, building the app, and we call our app Xap. There's the Xap builder who is wiring component one to component two.

What we've done in our visual Xap builder is we've given — In the visual environment, we've given the person who's wiring those components together a fair amount of control over how data in the Duffle gets mapped into the inputs of the second component. By doing that, we get around a number of the interface issues that tend to come up when you're trying to connect things together up close.

[SPONSOR BREAK]

[0:36:26.1] JM: When you are continuously deploying software, you need to know how your code changes affect user traffic around the world. Apica System helps companies with their end-user experience, focusing on availability and performance. Test, monitor, and optimize your applications with Apica System. With Apica Zebra Tester, Apica Load Test, and Apica Synthetic, you can ensure that your apps an APIs work for all your users at any time around the world.

Apica Zebra Tester provides local load testing for individuals, small teams, and enterprise DevOps teams to get started quickly and scale load testing as your needs evolve. Apica Load Test ensures that your app can serve traffic even under high load. Apica Synthetic sends traffic to your website and your API endpoints from more than 80 different countries, ensuring wide coverage.

Right now, you can go to softwareengineeringdaily.com/apica for a webinar about the real ROI of API testing. You can also find past webinars, such as how to optimize websites for fast load time. Go to softwareengineeringdaily.com/apica to find the latest webinars on load testing and lots of other topics, and check out Apica System for testing, monitoring, and optimization.

Thanks again to Apica for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:37:56.1] JM: It sounds like you take a lot of inspiration and consideration from these different data interchange formats that have been successful overtime for various reasons, like JSON, XML. I think you could also maybe say SQL is kind of like a touch point. It's interesting, because you're moving to this world where you have a more visual interface, so you want to have this idea of a component.

Let's look from a higher level. What are some examples of data applications that you've seen people build, and how does the individual, or the team that is constructing that application, how do they work together on it, and how does it evolve overtime?

[0:38:45.1] DK: We've done a lot of work in the health sciences, and in the biotech space, and that's just been an area when we founded the company a lot of us had interest in trying to use technology to help data analysis in life sciences.

We've seen a lot of Xaps get build with our platform in the areas of trying to work with fairy complex medical data, genetic data, and things like that. What's interesting about these sorts of use cases is that people talk a lot about data visualization, but it's rarely effective to just visualize the data. You usually need to visualize some algorithmic operation on the data.

I like to make a distinction between data visualization and information virtualization. One of the things that we try and facilitate with Exaptive is to make it easier to actually convert data into some more user information. Now, to your point, that frequently requires a team of people. That's not always something that just one person does.

One of the things that's really powerful about a visual programming environment and, as originally a C programmer, I was very skeptical of visual programming environments sort of earlier in my career. One of the things I realized when I became a software architect is that you really need system level diagrams in order to help collaborative complex — Building collaborative and complex things, that that system level diagram becomes the holy grail of development efforts, because it's the right level of abstraction. It's not down in the code, and it's not all the way up at the final product. It sort of sits right in between.

The idea of a visual dataflow environment is that it's like instead of having to make a system level diagram and try and keep it in sync with the code that you're writing, making the system level diagram is the code that you're writing. That becomes a really cohesive environment when lots of people are working on things, and it also changes the type of reviews that happen. I'm used to doing sort of very high level reviews and then these very low level code reviews. One of the things we see at Exaptive is that by pulling up the dataflow, which is — I've used that term a number of times. The dataflow is the visual representation of how the components are connected together and how data flows through them.

That provides an entry point into the application that really generates sort of a different level of review and discussion. The other thing it lets you do is lets you put in a placeholder component for, let's say, an algorithm that's going — I don't know. It's going to do some sort of curve fitting or something like that, you might put in a placeholder algorithm and wire the whole thing up and then annotate that component and say, "This has to be developed into something better by a different person." It allows for distributed work by allowing certain people to just work on certain components and still knowing that they're all going to connect together at the end.

[0:41:56.6] JM: Yeah. Of course. It sounds like, in some ways, the things that people are working on, I think this maybe late 90s, or early 2000s, where people have this idea of UML, and these big Java applications, and they were like, "In the future, you're just going to be able to create UML and that you'll derive your Java application from your UML diagram," and this was kind of the future for a while. It kind of didn't work out, I guess. I'm not exactly sure. I'm not enough of a student of history to understand why that didn't work out. Do you have any perspective on that?

[0:42:33.4] DK: Yeah. It's funny that you said, because I've totally forgotten about these experiences you provided me of a few disastrous projects I had that I entirely started that way with these UML diagram.

[0:42:45.1] JM: This has been your dream for a longtime.

[0:42:46.9] DK: Yeah — No. Then, I hit a button and it generated all of my stub classes and all of these. Yeah, I never really found it to be that effective. I think it is a valid dream.

[0:43:01.1] JM: It is a valid dream, exactly. That's the thing, it's too early. Too early is as good as not at all.

[0:43:09.6] DK: I think I'll shed some light on why I think the time is right now and sort of what I think the difference is, which is I think when I was working those tools, it was probably like 15 years ago, specialization across the technology stack was not was it today. I think that it was much easier to build applications then, where your small team could build every aspect of the application. Whereas, today, that is pretty much impossible. If you want consume certain types of data sources, you're going to have to go out to APIs and use them. If you want to use certain types of machine learning, you may end up using third party providers, things like Watson, or Lambda on AWS. There are all of these sort of specialized micro-surfaces.

I think what's happened is that the act of developing software has just inherently become a lot more fragmented, and that the act of building good software is now the challenge of connecting together those fragmented pieces.

The idea of a UML diagram, going back to the original idea of UML diagram, as a way to show how different fragments are connected. That, I think, is a necessity in today's technological landscape in a way that it wasn't 15 years ago. I think that that really — One; it's a necessity. Two, the other thing that's changed is because of that fragmentation, there are a lot more software providers that are providing just very specific fragments in terms of solving a problem, like natural language processing is a great example. There's a lot of effort — Or image processing, image analysis.

There's a lot of providers making these very specific algorithms for categorizing images, or extracting sentiment from text, but they don't have anything to do with the data. They don't have anything to do with the visualization of the results, it's just this very small piece. The combination of those two things, I think, is why Exaptive is sort of the right paradigm for this — I won't just say Exaptive, just the idea of going back to the idea of UML diagram being your application. I think it's the right time for that now, because that's what modern applications are, they are diagrams of lots of disconnected pieces.

[0:45:37.0] JM: I completely agree. Thinking about where we're going, there is, obviously, so much development in machine learning going on right now. Machine learning is interesting, because there's a large population of machine learning projects where you need a human in the loop doing something, and part of the work gets outsourced to Mechanical Turk or Scale API or something else and you just have -- or translation for example. If you need something translated, we did a show about translation recently with a company that does human-in-the-loop translation where they will do a first pass of translation that goes through a machine learning system, through a trained model and they get the translation as good as they can and then they have humans basically review.

But where I see this going is the things that we have the human-in-the-loop doing are going to get increasingly complex and we'll have increasingly technical human-in-the-loop jobs and this is kind of where, when I try to project myself into the future and think about where do the new jobs come from? Because obviously truck drivers are going away and all these other automation things that we could talk about. A lot of the new sort of blue collar work is going to be things like data standardization or I even think like this kind of work; connecting together these systems, these data models, these UML type of diagrams that are things that you can understand even if you're not highly technical.

So I guess my question is, where does the information system construction move towards, as an industry, where it involves and increasing number of less technical people and what is the changing dynamic between the technical engineering roles and the people who have a more moderate amount of technicality?

[00:47:48] DK: Yeah, your question gets at something that actually I've spent a lot of time thinking about and it's a big philosophical because I think what you're getting at with your question is, as computer technology and machine learning technology evolves, what room is left for the human? This is something I'm particularly interested in because I watch this technology around artificial intelligence and around machine learning in a lot of cases trying to push humans out of the equation. What I'm particularly interested in is how do we design technological systems in which there is a collaboration between computers and humans?

So where I see the answer to that is where it comes to ingenuity and creative problem solving. When you were asking your question, you used the term "technical" a lot, talking about technical people and less technical people. One of the things I want to do is I just want to unpack that a little bit to say that there are lots of technical people in the world that are technical in different areas. A subject matter expert can be an extremely technical person. A geneticist can be extremely technical. But they are not technical in terms of writing computer software, which is how you were using the term.

Where I see the movement with technology is where the people that are technical in asking questions in domain matter expertise, in creative problem solving, those people will be put more and more in a position of being able to creatively explore the spaces that they're interested in. They'll be able to do that because they'll have tools that make it easier for them to do that sort of symbiotic exploration, which is that computers are extremely good at all of these brute force problem solving approaches, but they're not very good at being creative.

Humans on the other hand are very creative, but the problem is we're sort of like correlation-finding machines. We don't want to believe there isn't correlation in things. So we'll see patterns even where they don't exist. So I think there is a fantastic opportunity to marry these things together where -- this is what I'm interested in with data applications is how do you create data applications that allow humans to do what they're best at, which is creative investigation, hypothesis formation, and allow computers to do what they're best at, which is reigning in the humans and saying, "Oh, well statistically there's not a correlation here, but look at the data over here."

To answer your question about where I think, sort of what's left as we start moving in these directions of increased automation, is what I get excited about is I think what's left is a set of jobs that are really about creativity, and there are two types of creativity; the technical software developer should be able to be creative with the thing that they are building and all of the value add of it, not the plumbing that they need to build to make that thing useful.

So if somebody is a technical programmer and they're interested in algorithms, they should be able to just work on the algorithm. If someone's a technical programmer and they're interested in visualization, they should be able to just make cool interactive visualizations and they shouldn't have to worry about all the plumbing code, or the glue code. That's creativity in that area, and someone else who is trying to solve problems with technology, they should be able to be creative in the problem solving arena without having to worry about where they're going to get the components from to turn their attention to.

At the end of the day, what we're really interested in with Exaptive is, how can technology facilitate innovation? We live in a world today where we are building technological platforms for facilitating all kinds of things. We facilitate finding new music, through things like Pandora. We facilitate finding a place to stay through Airbnb. We facilitate getting dates through things like OkCupid. We facilitate catching cabs through Uber. But we don't have any software platforms today that are actively facilitating idea generation. The whole reason we names the company "Exaptive" is after this biological term called "exaptation".

Exaptation is this thing in evolutionary biology where trait that evolve because of one set of survival advantages, it's serendipitously co-opted for something completely different, and thrive there. So it turns out that birds feathers evolved originally because they kept animals warm by trapping a lot of air, and that got serendipitously co-opted many generations later for flight. So it's a completely serendipitous thing. The reason I named the company after this is I sort of feel like this is the dream of all software developers, is we want to build something cool and we want it to get used in ways that we maybe never even imagined.

We want our programs to come to life. We want our libraries to get used. We want our GitHub projects to get forked. Programming is very much a creative endeavor and the benefit of that is we want these things to go out in the world and we want them to be exacted by other people who have other ideas for things they can do. The whole vision behind these data applications in the Exaptive platform is to lower the barrier for exactly that kind of creativity, and that's what I get excited about.

[00:53:43] JM: It is this recursive desire to give people the creative access because all of us remember that feeling that we had the first time we messed around with programming --

whether it was Hello World, or making some calculator app, or whatever -- and sensing that feeling of empowerment and as you progress down your career, you start to realize that the value and the money, frankly, and the satisfaction is all in giving people tools and giving people primitives that they can use in ways that will surprise you. The surprise that you can get out of that product creation is as valuable as any amount of money you could get out of it.

I share your optimism about these tools giving people more creativity in the end and even as a stop-gap between where we are now and this world where people who are domain experts can also be technical creators, they can explore their domain more creatively in the future. I think about this company I interviewed recently on Babble, that translation company I referred to, and they were talking about how when they have something that needs to be translated, like something about fishing for example.

If you need something about fishing to be translated from Spanish to English, you'll send it to somebody who's a fan of fishing. They speak Spanish and English and they're a fan of fishing so that they get some amount of pleasure out of that translation act. Because they're reading the actual content. It's very interesting how these systems are developing where we can route tasks to people who will enjoy doing that task more.

[00:55:29] DK: Absolutely. Just to chime in on that, I think that one of the things that I get excited about in terms of component based development is the idea that people should be able to love what they do, and they should be able to make a living doing it. One of the things that we're putting in place with Exaptive is what we call the Zap Store, which is a whole marketplace for these sorts of components so that someone who, you know, you gave an example of someone who loves fishing. But I think about someone who loves analyzing images from NASA image feeds or something like that. They might not necessarily know what that's good for, except that it interests them.

I'm an entrepreneur, I started my company, I've done a lot of entrepreneurial pitches and in the entrepreneurial world it's all about "what problem are you trying to solve? Etc." But when I put on my programmer hat, what I get excited about is not specific problems be solved, but tools that can be used in all different ways and I envision a world where people can work on something because they enjoy it. They can work on a component that analyzes NASA images, or they can work on a component that accesses census data, and they don't need to know ahead of time all of the perfect uses for it.

They can just build it because they think it's useful, they think it's exciting. They can put it into the component store and it can get used into a myriad of Zaps and they can get paid for it. I envision the ultimate Zap Store is where programmers get to focus on what they're good at, and other people get to focus on what it's good for.

[00:57:15] JM: Yeah. Okay, so just to wrap up, we didn't talk much about the engineering of Exaptive, but I'd love to get a picture for how it's build. So to make this more concrete, you have this product, the Exaptive Studio, which is sort of like an IDE for building these data applications that we've been talking about; for building these rich interfaces for dealing with data and building applications out of components. Can you talk about the tech stack and how you are building this component environment? Because we're describing an ecosystem, but we're also describing this IDE. So maybe, just give me a picture of the tech stack?

[00:57:59] DK: Yeah. So the idea behind Exaptive is that you can run these different components in different languages. So want to make that available. We've made that available to people in a cloud-based instance, though organizations that are running Exaptive have also installed Exaptive behind their firewall. But what it means is that there's a client and a server side piece, obviously. The server side piece uses Node.js on the backend to have a high performance server architecture there. It uses Docker for allowing for different components to be run in secure sandboxes so that a certain component can't interact with another component. So Docker is in the stack for that.

There's something we call "domain hosts", which allow components to be run outside of a Docker environment if you don't want, sort of on your own server. So that's another layer to the stack, so there's this sort of multi-tenancy option with Docker and then the ability to pull components onto your own server where they don't incur the Docker overhead. Then there's a component on the server side to being able to dynamically load scripts and to call sort of like a proxy server to be able to call out to other domains. Basically what you've got on the server side is a Node.js server and a Docker computation server that can also be scaled in a number of different ways.

Then on the client side, we're running an HTML5 application that's Javascript on the client. What's interesting about the Exaptive JS dependency that's running on the client side is the way that it communicates with the server to dynamically pull client side components into the browser in a way that is object oriented. If you look at a lot of D3 examples, which have become very popular, you'll see that a lot of those D3 blocks rely on hard-coded div names. So you have a D3 scatterplot, it's looking for a div called "scatterplot" and the problem with that is that if you then want to create a second, if you want to have multiple instances of these scatter plots, unless you go through and change all the div names, you end up with them stepping on each other's toes.

What Exaptive does on the client side is pulls these client-side components into sort of safe divs that are isolated from each other on the name space side of things but can still communicate with each other through the data flow. So that's a little bit of the overall architecture. The goal in the architecture was to keep the Exaptive core, that client server system I just described, to be very compact. Because we want the majority of the functionality to come from the extensibility of the components themselves. So in fact, when we built Exaptive, we built the core for being able to run these data applications, and then we built the visual IDE as an Exaptive application.

So I think you mentioned recursivity at some point. We're sort of living that and the Exaptive application is itself and Exaptive data application.

[01:01:09] JM: Very interesting stack. The Docker abstraction is obviously perfect for having these different programming languages in different environments and you just have a standardized way of interacting with each other. I was looking at Exaptive and I was like, "Oh, this would make a lot of sense if it works with Docker," and it does.

So I guess to close off, when you look out in the space of platforms and open source projects that are growing in prominence, things like Kubernetes and you mentioned Serverless earlier, there are also these things like the Google cloud APIs that are coming out that really seem like a sign of things to come where Google basically exposes an API for computer vision and you just give it an API request and it identifies the thing in the image, which is pretty incredible.

What are the things that excite you the most and that you feel could have the most potential in Exaptive?

[01:02:10] DK: Well, I think the thing that excites me the most is that what you describe is happening, that there is this proliferation of very specific targeted technologies all being made available as a set of microservices and that is what excites me. Because I think that's the future of programming. It's not, you know, bigger and bigger monolithic programs that do more. It's smaller and smaller specialized services that do a certain thing very well. That's what I get excited about because what Exaptive is trying to do is not replace any of those or compete with any of those, it's trying to offer a glue to allow those things to come together.

Maybe instead of saying "a glue", the problem with glue is once you glue stuff together it's stuck. Exaptive is a little bit more like Velcro. So you can put stuff together, you can use it, and then you can very quickly reconfigure and I think that that's the key. When technology is changing very quickly, the application you build today is not going to be the right application in three months, or six months, or maybe not even in one month. So I think, just as important as being able to quickly build things with all of those sort of new offerings you described, is the ability to also very quickly retool the things you built.

That's where this sort of component based architecture also has a lot of benefit. It's not just in how something can be built, it's how it can be evolved with new requirements.

[01:03:46] JM: All right, well Dave, this has been a fascinating conversation. That's for coming on Software Engineering Daily.

[01:03:51] DK: Thanks so much. My pleasure.

[END OF INTERVIEW]

[01:03:57] JM: Thanks to Symphono for sponsoring Software Engineering Daily. Symphono is a custom engineering shop where senior engineers tackle big tech challenges while learning from each other. Check it out at Symphono.com/sedaily. Thanks again Symphono.

[END]