EPISODE 1341

[INTRODUCTION]

[00:00:00] KP: As developers hone their craft, becoming more productive often means learning utilities and tools that run on the command line. The right combination of various parsing commands chained together through pipes can enable engineers to quickly and efficiently automate many ad hoc data processing tasks. In this episode, I speak with Adam Gordon Bell about some of his favorite command line tools. We also discuss his role as a developer advocate for Earthly, a powerful tool for building software in a repeatable and understandable way. Adam is also the host of the CoRecursive Podcast.

[INTERVIEW]

[00:00:41] KP: Adam, welcome to Software Engineering Daily.

[00:00:45] AGB: Great to be here, Kyle.

[00:00:47] KP: So the way we initially connected was I wanted to talk to a few people about different development tools. One of the ones that I guess initially was the keyword search that got us connected was for JQ. So I want to get into that and some others. But I thought a great place to start would just be to unpack your development story. Tell me a little bit about how you first got interested in coding.

[00:01:07] AGB: Yeah. So what happened was I took a class, I think, in grade 11, in high school, where they taught us Turbo Pascal. And I don't exactly know why I took this class. But I did have a computer at home. And we got to like split up into groups of two and make like a Yahtzee game. And it was super fun. It was just like building things. Like it was like a DOS program. And I don't know if you're familiar with Yahtzee, but you like roll dice. And then you have to like score things like kind of using poker type hands. So there's a lot of just writing like if statements and trying to draw things on the screen. But it was like I just really got hooked to that iteration where you like change something and then run it and see what happens. And then it

probably didn't hurt that it was a game. This was a super fun experience for me. And yeah, I've just been chasing that high ever since, I guess.

[00:01:57] KP: Yeah, I had a similar path. I did some Turbo Pascal at one point. And I think in those days, it was – I don't want to sound like an old man complaining about how easy it is that kids have it. But it was a lot more arduous to get anything even compile and run. Have you noticed great changes in the developer experience over the years?

[00:02:14] AGB: I mean, definitely. I don't have a great sense for what the beginner experiences, because I'm not a beginner anymore.

[00:02:22] KP: Oh sure, yeah.

[00:02:23] AGB: Yeah. One thing I would say is like it was harder, like Turbo Pascal, there wasn't like extensive tutorials or whatever. Like we had like some textbook. But on the other hand, it was a very small world, right? Like now it seems like you could just like try to learn about react and never get finished. Like it was an easier time, and the things you had to learn were much more contained and small, I think.

[00:02:47] KP: So if the foundations began in grade 11, were you on sort of a typical computer science track? Or how'd you get to where you are?

[00:02:54] AGB: Yeah, so I went to university for computer science. And I enjoyed that, especially when we got to actually build programs, and it was less theoretical. And then I left school, got a job as a software developer. And yeah, I was just so excited that I was getting paid to like build things. Sometimes I get frustrated day to day. And I try to think like – I try to remind myself, like there was a time I was just so excited that somebody would pay me to do this. But, yeah. And then, I don't know, my career has continued from there. And now I work in developer relations.

[00:03:30] KP: Tell me a little bit about what that role is like. What is someone in developer relations do on a typical day?

[00:03:36] AGB: Yeah, it's a great question. So I've been doing it for like almost exactly a year. So I'm not the most knowledgeable about the whole spectrum. But I could tell you what my day looks like. So our company makes developer tools. And so it's important to kind of be able to verify that they work to talk to developers. So I've done things like making video tutorials for Earthly, doing blog writing. Speaking at conferences, or applying to speak at conferences. Writing documentation around how you use various software. In October coming up, we have Hacktoberfest. Our project is on GitHub. So we try to help people who want to help contribute from the outside. Yeah, things like that.

[00:04:20] KP: Very cool. I want to ask you about Earthly, but let's do some prerequisites. I would hope any listener to Software Engineering Daily already knows about Docker. Otherwise, we've got archives to go through. But I'm not convinced every listener is going to know about Make. Could you tell us a little bit about the Make tool?

[00:04:37] AGB: Yeah. So Make is like a traditional Unix build tool that it has basically targets that you can specify for doing various tasks. And it's traditionally used with C a lot of times, but it can be used with anything. So you could describe with a Make file how to turn a C file into an object file, and then those object files into an executable, how to include headers. So it's a somewhat declarative way to specify steps in a build process and dependencies between them. And the idea is that if you change a file, then it knows what to do to produce a new executable that has those updated results and kind of doing the minimum amount.

[00:05:20] KP: So what is Earthly solution do?

[00:05:22] AGB: Earthly is a free source available command line tool for building software. I think it's helpful to maybe understand the backstory of it. So Vlad, who originally created the project, he worked at Google for some time, where they have a build tool called Blaze, and they open source as Bazel. If you work at Google, or if you kind of follow how Google does software development with like a large mono repo, it's a really great development experience. It's very easy to build software, have it happened quickly, distribute the builds, and not have flaky builds. And it does this a number of ways. But one of the ways is by having a sandbox. So when you are building things, it puts things in a sandbox so that you're sure that there's not missing dependencies that you haven't included.

Vlad left Google, and he sort of wanted a tool like this, but he wanted one that could work with the way that most people develop software, which doesn't necessarily tend to look like the Google giant mono repo and thousands of people working process. So what he noticed was that, well, sandboxes kind of exists in the form of containers and Docker containers. And so could I take something like a MIG file, as we were describing, and then use this kind of Docker ability to have sandboxes. Put those two together and get something much like Blaze or Bazel that gives you a way to make builds that are fast, that can be parallel, but that are also always reproducible. You're always getting a repeatable result, and avoid the problems of like flaky builds.

[00:06:55] KP: Could you give some examples of the pains people experienced when they have one of those flaky builds? What is earthly saving me from?

[00:07:02] AGB: Yeah. So imagine that you have some sort of – Like Jenkins is pretty popular , continuous integration product, and you're working on your code locally, Kyle, and you run some unit tests, but maybe not all of them or something. And then you commit your code, and then Jenkins starts to build it. And then it runs into a problem. Like maybe there was something on your local that was making it work which you don't have installed on the Jenkins server. Or maybe there's just a test that fails sometimes, and it requires some dependency that you don't have. So you don't tend to run it locally. So then you have to try to figure out what this problem is and kind of track it down, which can be a big pain, especially if you're going to just try to like make a change, and then push it again and see if Jenkin runs it again.

But by having a build process that you can run totally locally on your machine, because it is sandbox, because it's in a Docker container, that means that you can do the build locally just as easily as you can in CI, like they're identical processes. So that kind of saves you from having to deal with that flakiness. Or at least the flakiness should be consistently happening across your local environments in the build server so it'd be easier to track down.

[00:08:16] KP: And what does a quick start look like? If I want to introduce my software application to this process, how do I get going?

[00:08:24] AGB: Yeah. So there is a Earth file, which is similar to a MIG file. It looks a little bit like a number of targets. So let me give you an example. So I'm going to build my Scala project, and I have a number of steps for it. I need to compile my code. I need to run some unit tests. I need to run some integration tests. So I could make each of those as a target in a text file, one called compile**:** and then I would put whatever the steps are that I would run. To do a compile, I would put a test step and I would have the test step extend from the compile step, and then do the various test commands, similar for integration. And I might build up a whole bunch of these over time, linting, etc. And then I would just run this by calling a certain target using Earthly. So if I wanted to run the integration tests, I would say, "Earthly.integration test." And then if it depended on the compile step, like it kind of worked backwards and figures out what needs to be executed to get that result. So that's kind of what using it would look like.

And then in your continuous integration process, like if you were using Jenkins, or GitHub Actions, or whatever, you would just call the same action. So rather than building up complicated logic within your build server within a YAML file for GitHub Actions or something, you can just put all the logic in this kind of Dockerized format, and then it's easy to work with.

[00:09:49] KP: Can you give me a sense of how that helps with collaboration and team exchange and things like that? What do I share with my coworkers?

[00:09:57] AGB: What do you share with your coworkers?

[00:10:00] KP: In other words, are we all building to the same image? Should I be checking that Earthly file into GitHub? What are some of the common workflows people follow?

[00:10:08] AGB: Yeah. So you put this file in to your source control the same way you might put a MIG file or some sort of bash script that creates your program. And then that should mean that anybody can do this same step. If somebody needs to change the process, then the file is there and available. So one thing that's great about this kind of simple format that I'm describing is that it's simple. So oftentimes, at least in the places that I've worked, at some point, these building processes can get very complex. And there're a lot of steps. And there's just like one guy who kind of knows how it works. And if something needs to be changed, you need to go to

that guy. And he's like the build guru. Whether officially or not, like that's his job. He's the only guy who understands this.

So one thing we try to do with Earthly is make this whole format very simple and something that you can understand at first pass, because we want it to be something where anybody on the team, if they need to add something new to the build process, they can just step in and change it. So yeah, it's in source control. It's fairly easy to understand compared to other options out there, and everybody can work on it together.

[00:11:15] KP: As languages has evolved – I guess I started with C way back when, and I used MIG to do all my compilations, because that was the easiest way to do it. There was a tool outside of C built for that. But we see a lot of projects as time has gone on like NPM for node that helped with a lot of these steps. Do you see that affecting adoption? It seems obvious that C and probably Go programmers would take right to this. Is there a good use case for node developers as well?

[00:11:42] AGB: Yeah, it's a great question. Like where we see people using Earthly is often they have more than one tool. So yeah, NPM is a great example. I mentioned Scala earlier. In both those cases, you have a tool, NPM or SVT, that you probably do a lot of these things with. But what happens when you start having more than one language you're dealing with, right? And now you want something where you can say compile the backend in Java, and then run some integration tests against it. They were written in JavaScript using NPM. Like you can start writing, say, bash scripts, that kind of wrap around these like common actions, the cross-languages. You could use a MIG file, or like you can use Earthly. So that's really where we see a lot of uptick, is like when you're using more than one language –So there's more than one tool, and you need ways to do things that cross them.

[00:12:35] KP: As projects mature and get bigger, I often see the build steps and sometimes the unit tests kind of exploding. And there being a few maybe Jerry rigged ways, we skip things with flags and stuff like that. How does Earthly help with these larger scale, more mature solutions?

[00:12:52] AGB: Yeah, I mean, I wish I could say that there's like – We have this solution that just makes everything easier. I think that we do in a sense, but in a sense, like building software and testing it is just hard. So what we try to do is just offer a way to do so very declaratively. And we try to – We do. We're able to infer kind of where things can be run in parallel. But I think if you want to have a really solid and reliable build, you need to invest in it, right? Like you may have software and it fails in a certain case. So you start adding tests for that. And then these tests build up over time. You need to consider whether they're valuable or not. If a test fails 30% of the time, like is it really validating? What's happening?

So Earthly tries to make builds quick. It tries to make them reproducible. Even more so, like we don't want a build where sometimes it works and sometimes it doesn't. But I think it's just an area that requires a lot of investment if you want reliable software. I talked to Richard Hipp who works on SQLite. He spends three days just doing his build process for each release of SQLite. And that's because SQLite, it's used everywhere. It's a very important product. And like in order to test it correctly, he just needs to run, like, he said billions of tests. I don't think there's a tool that can make that happen in 15 minutes. It's just a complicated build process. So it requires investment.

[00:14:19] KP: Good advice, for sure. But I was actually leaning towards what I think are some clever optimizations Earthly does. In the same way Docker will cache the layers as I push them. So maybe my first build is hard in the future ones become pretty easy. Do I get any of those sorts of niceties when I work with Earthly?

[00:14:35] AGB: Yeah, yeah, exactly. So Earthly uses build kits, which is open source projects that's also used by Docker in the newest versions, and it does layer caching. So yeah, as you were saying, it is able to detect changes in the file system. And then it uses a union file system to cache previous results so that if you – Say you have the process we laid out earlier with compile and test steps, and now you make could change to the test to a specific test, it will be able to know that it doesn't have to recompile things if the tests were outside of it. It's using the file structure, and caching to determine where steps can be saved.

[00:15:14] KP: Do you have a common persona? Who is the first person introducing Earthly into an organization?

[00:15:21] AGB: Yeah, so I think that, often times, what we see is somebody hears about Earthly maybe from listening to this, and they try it out as maybe on a side project or something like that. And they like it. And they are probably a developer who's been doing this for a little while and is aware of challenges around builds. And then they kind of bring it into their company. So that person is probably a senior developer. Maybe he's the build guru person or she that we described earlier, the person that everybody goes to with the build. So they're frustrated. Or maybe it's just somebody who knows that this is an area that could use an improvement. And yeah, that's usually how we see the product get introduced.

[00:16:04] KP: Make sense. What's the future for Earthly? Or maybe it exists? Or it could exist? Is there room for an Earthly as a service type offering that a lot of tools that are free and available can offer mature or more enterprise-oriented scale or options? Anything like that in the planning?

[00:16:21] AGB: Yeah, there is, but I can't tell you what it is.

[00:16:24] KP: Maybe come back when it's ready, I guess.

[00:16:27] AGB: Yeah, we built this tool, and it seems to be quite popular. So what we're doing is trying to, yeah, build an offering around it. And we're very focused around the software development lifecycle, like improving it. And yeah, wait and see. We have some ideas around where this is going.

[00:16:46] KP: Very cool. Yeah, I got interested in Earthly in the same way I first got interested in Docker. It filled all these obvious pain points and seemed like it could be helpful for a lot of development processes. I also like that it has that developer appeal. I like working at the command line, having control via straightforward text, or things I can look at in a text editor for configuration, that kind of thing. I'd love to expand our conversation, talk a little bit more about the secret tools like that that are in your tool belt. I would hope most developers are familiar with Grep. And maybe we've each got our own little recipes that we'd like to run. What are some of those things you've leaned into heavily over the years.

[00:17:23] AGB: So JQ is a command line tool for querying JSON documents. So I have used it for years for pretty printing a JSON document, which is the least of its uses. But say you make a curl request and you get back some sort of big Jason thing and it's hard to read because it's all on one line or whatever. Pipe it to JQ, and JQ will just print it out nicely. But also, JQ stands for JavaScript Query, I believe, and it can do all kinds of things. It can transform an extract elements from JQ. So I spent some time recently just trying to understand the tool, because it's one of those things. It's a super powerful tool. Whenever I have a problem, it was there, but I never really mastered it. So yeah, I've spent some time trying to figure out how it works. And I've really become impressed with it.

[00:18:11] KP: Could you expand a little bit on some of those querying features? What might someone be able to get quickly out of a JSON document?

[00:18:19] AGB: Yeah, so the easiest thing you can do is just extract items from a JSON document. So it lets you treat a JSON document that you pipe into it as if it's a JavaScript object. So if the document is like an object and then it has a key called Kyle, and then a key on that called like last name, you can just tell JQ dot Kyle dot last name, and it will print out that value. So that's one of the easiest ways that you can use it. It's just extracting single or multiple values. You can also use it to extract arrays. So you could have a JavaScript document that contains an array of items. You could extract certain elements from that array. You could use it to take those array items, and take them out of the JavaScript context and put them each on a new line, which is very useful if you want to pipe it into some other Unix command. And it just keeps going from there. So it's actually a fully featured. The query language for JQ is actually a fully featured like Turing complete programming language, which I didn't really know going in, but you can do almost anything in it. But what I found is usually you just want these couple basic things, how to extract things, how to transform things.

[00:19:31] KP: Yeah, maybe it's my lack of imagination. But all the use cases I'm coming up with are – When I'm doing some sort of integration development consuming or interacting with a REST API, I'm sending and getting back JSON, and maybe I'm just interested in one element. So wouldn't it be great if I could grab that? So definitely appealing to me is sort of a work a day CLI tool. Does this belong in integration tests or in some production pipeline?

[00:19:58] AGB: I mean, that probably depends on your use case, right? I think if you're using it inside of an integration test, then, I mean, maybe if your integration test is a shell script, maybe you make a request, and then you want to pipe it through JQ and check a value and return true or false. You could do that. I have not done that. So I'm not sure.

[00:20:19] KP: When you think about the landscape of tools you can work with, we've got on one end sort of the Vim and command line, hardcore, bare metal sort of approach. And of course, every day, there's new clever IDE tools and plugins to be functioning as higher levels of the stack. Maybe we even say no code is sort of an icing on the cake. Where do you spend most of your development time?

[00:20:42] AGB: It's definitely more in the command line type space. So I'm using VSCode. But I do have Vim bindings in it. So that's kind of a middle place, I guess. And I definitely spend a lot of time in the command line. And I've been learning more and more recently. I've been using the command line more and more in recent history. And I've just been amazed at the type of workflows that you can build just by gluing these simple tools together.

[00:21:08] KP: Are there any tools you'd like to highlight? Things that maybe aren't as popular as they should be, or things developers should be picking up that they don't already use?

[00:21:16] AGB: Yeah, there's a number of tools that I think are pretty cool. One is Fuzzy Finder. Fuzzy Finder is a tool that just lets you easily do a fuzzy find in your command line. So Fuzzy Find is sort of just like a text search. So the way that I use it is like if I'm looking for a certain directory in the command line, I can type like LS and then star-star, which I have like bound to Fuzzy Finder. And it will bring up a kind of text UI, where as I type, it will filter down through all my paths until I find what I need. So it’s super handy just to be able to quickly find out what I have on my file system in my computer.

Another one that I use is called **[inaudible 00:21:57]**. I'm not really sure if the name makes a lot of sense to me. But basically, when you're changing directories using CD, **[inaudible 00:22:05],** you can just replace your CD with it. And the cool thing that it does is, as much like the Fuzzy Finder, it lets you change directories just using a substring of the directory you're working in. So oftentimes, like I have all these folders, and they're like nested three or four levels deep, and

say I want to change from one to another. I don't want to have to like CD../../ and like kind of build up the path that way.

[00:22:30] KP: Yeah, Java is particularly bad with that.

[00:22:32] AGB: Yeah, yeah, exactly. The Java namespaces, yeah. So **[inaudible 00:22:36]**, you can just alias your CD to **[inaudible 00:22:39]** and then it will let you CD just based on a part of that path. So you could just put in the name of the class, or the namespace you're looking for, and it'll figure out where to go. It's a super neat tool.

There's this tool called Funky, which has an interesting name. But Funky is a command line tool for building up bash aliases. So you type like Funky/a and then get image is what I have. And then it gives you like a little prompt where you can write in a command. And it's a little hard to describe, but it just allows you to write little shortcuts within your terminal. And then they are stored in a file that's called .funky in that directory. So it lets you have like little specific commands within your directory. So I have a blog for Earthly, and there's a bunch of commands I have to run in order to like start up the site, or shut down the site, or generate images. And so this allows me just a very smooth way to write a bunch of little aliases. And it kind of gives me that experience like I'm working in like a REPL, but I'm at the command line, which I really like.

[00:23:42] KP: Very cool. I think I want to try that one out.

[00:23:45] AGB: Yeah. I could keep going.

[00:23:47] KP: Yeah, let's do another one. I think these are all chained together very well. I like them all so far.

[00:23:51] AGB: Yeah. So there's this tool called McFly. I don't know where people come up with these names, to be honest, I guess there're only so many names out there. So if you're in your terminal, and you hit Ctrl+r, you can get a history of the commands you've run and use that to easily narrow down to what you're going to – To the command you're going to execute. Let's say you're looking for the previous like complicated Docker command you ran with like volume bindings and all this stuff, like you can use your Ctrl+r to easily find these. So McFly takes this

concept and extends it. So the creator made this observation that when you're trying to find recent commands using your history, like there's actually a bunch of information that's being thrown out by just looking blindly at all the commands in your history file. One is like whether the command successfully ran or not. So if you ran something and it failed, like you don't really want that to autocomplete as like a possible thing you want to run in the future, because it's probably just wrong.

Another one is what directory you're in. So as I mentioned with Funky, sometimes there's specific actions that you run in specific places, and they're not really relevant in other places. And another one is just one when you go to pick from a list of options, like which ones do you select, right? If I usually type Earthly and then autocomplete it to X, then maybe that should be the first option when I look again. So McFly actually uses a very small like neural net to keep track of all these details. So rather than this giant history file, it actually keeps a dictionary of the various commands you run in various directories and tries to learn what the best options are to present to you. I guess it's like a very small reinforcement learning algorithm. So it's a very neat optimization to try to suggest options.

[00:25:34] KP: Fascinating. I want to try that one for sure. Well, that's a great list. I'm going to ask you a sort of impossible question. But do you have a sense of the time I'm going to save on a weekly basis if I really invest and train myself on these tools? How do I know I'm going to get the return on investment I'm looking for in productivity?

[00:25:52] AGB: Yeah, that is an impossible question. I would say that maybe you should just try to note while you're working like what do you spend a lot of repetitive time doing it and what could be improved? And just spend a little bit of time trying to improve your workflow in areas where you see that a lot of time is spent, because I think the thing that can happen is that not just with command line, but with IDEs, or whatever, like you just get really into like customizing everything to be perfect. And you can burn a lot of time just adopting things that are – I think that if you spend the time to try to fix things that you know are problems in your current workflow, it's just a lot more valuable than trying to polish everything on your local environment or try out every new tool. It is fun to try out new tools. But practically, you have to be careful about the time you spend on adoption.

[00:26:43] KP: Good advice. I totally agree, and I see this as something that novice programmers often under appreciate a little bit. If I ask you to be the guy selling the argument for getting down into the command line and being more productive, what are some of the things that would entice me to explore these tools if I'm not yet too familiar with them?

[00:27:02] AGB: So let me sell you on it. So Unix is an operating system built around this philosophy of files, right? And Linux, and MacOS, and I guess to a certain extent, Windows, inherit from it this idea of like files being very preeminent. And at the command line, there are a lot of tools that are based around files and pipes. And it's a very powerful little language once you learn it. So your command line is not just a way to fire off commands, but it is a way to compose together little commands that can do things. So yeah, you could get something from a file. You could grab results from it. You could pipe that into, I don't know, a Python one liner that does some lookup on some service, and then write the results back to a file. You can definitely have, with these small amount of tools that you've combined in various ways, a lot of functionality available to you.

[00:28:02] KP: I'm wondering if you've seen any – It seems amenable to me that someone who's really good at command line and piping things together would not only fit naturally via Earthly user, but would probably mix those two spices that they'd come up with the Earthly files that have a couple clever Unix-y things to do a step in the build process. Have you seen any especially clever use cases in the people who have deployed it?

[00:28:26] AGB: That is a good question. I think you're definitely right, that Earthly is very much looking at the world through this command line perspective. And so we do tend to have users that use the command line a lot, as well as the developers on the team. Like one interesting thing that slowly emerged with the open source Earthly product is that a lot of our unit tests are actually shell scripts. And I don't think anybody planned that, but it fits very well with how we're doing things.

[00:28:54] KP: Can we explore a little bit about being open source and the growth of that? Are you building a community? Or is it simply just to have the code be public and shared and open?

[00:29:03] AGB: Yeah, that's a great question. Actually, I should clarify, we are using the BSL license, the business source license, which I guess is not open source in the true sense of it. I'm not a lawyer, but the way I understand it is the BSL license turns the code into open source after some time has passed. And it behaves mainly like an open source license, except with the caveat that you're not able to compete with us as a company on our commercial product. And this license is something that was, I think, pioneered by MariaDB. And the idea that MariaDB had to my understanding is basically we would like to be open source, but we would like to prevent Amazon from competing with us in AWS.

[00:29:46] KP: Yeah, makes total sense to me. Does that mean then do you find that people react to that? Is it going to be more or less likely you'll see contributors and PRs? Or are you early in that process?

[00:29:58] AGB: Yeah, we do see contributors. So we do our development out in the open, in GitHub, we do have contributors. We're participating in Hacktoberfest, which we did last year, and was a lot of fun. A lot of time we get contributors that help with documentation or with examples, but we do get occasional like really substantial contributions from the outside. We also – Actually not myself so much, but the other members of the team are often contributing to build kit, which is what we use to run the builds, the Earthly builds. And so we often find edge cases or improvements that we would need for our specific use case. And so that is also an open source project. And so we're off doing contributing up to the projects that we use.

[00:30:42] KP: Very cool. Do you have Hacktoberfest plans in place for this year?

[00:30:47] AGB: Yeah. So we have a bunch of tickets that we've labeled. I think we labeled them with hacktoberfest and good first time contributor. And yeah, if you want to participate in hacktoberfest, this is run by DigitalOcean, then they will give you, I think, a T-shirt for contributing. I forget the exact ramifications. But we will also give you some stickers if you come and help. And in the past, we've gotten a lot of great features. We have now syntax highlighting in a whole bunch of editors, because we didn't have anybody in the team who used Emacs. But somebody came in hacktoberfest and contributed syntax styling for Emacs, and for textmate, and added examples for various languages. So you can now use Earthly to build COBOL. I'm

not sure how much that happens. But somebody contributed did an example. So there's lots of fun if you want to help contribute to the project, then we'll send you some stickers.

[00:31:38] KP: Well, tell me a little bit about how people can get on board with that. I know there's a lot of zero time people haven't made their first open source commit yet and are looking to do so. How can someone like that especially get involved?

[00:31:50] AGB: Yeah. So you can go to github.com/earthly/earthly and look under issues for the issues labeled hacktoberfest. You can also jump on our Slack. If you go to earthly.dev, you can jump in our Slack channel and ask us questions. And yeah, we try to help people out and try to do whatever we can to encourage people to contribute. Yeah, and so Earthly is written in Go. Some familiarity with that would probably be helpful. But also, we have docs, and we have examples where we're trying to show how you can build things with Earthly. So if you're not familiar with GO or the technologies we're using, sometimes examples or documentation can be a great way to contribute. So yeah, just jump on our Slack channel and ask for help.

[00:32:35] KP: When I think about Docker, I mean, not that you have to use it in every project, but it's just so ubiquitous. It's sort of the go to solution. Do you think Earthly will eventually grow into that position that every project adopts it? Or does it just suit the needs of certain projects?

[00:32:50] AGB: That's a good question. I mean, I definitely think everybody should adopt it.

[00:32:53] KP: Well, that's a softball one, but why?

[00:32:57] AGB: I mean, the strengths of Earthly involve – Yeah, as I said, when you are dealing with more than one language, where you're probably working on a team, where you care a lot about the speed of your build and the reliability of your build process. I mean, I think that that covers a large percentage of software development, but there're probably cases where it's not exactly a fit.

[00:33:21] KP: Totally. Let's talk a little bit about CoRecursive, your podcast, CoRecursive. What can listeners find if they tune in?

[00:33:28] AGB: Yeah, CoRecursive is my podcast about software development. And the way it works is I like to have somebody on and have them share the story behind software being built. So it kind of has a story type structure. So I mentioned earlier, Richard Hipp, who created SQLite. I had him come on and explained the process of building SQLite, which was actually like super interesting. He started building SQLite for a battleship. It was supposed to be the database that ran on a battleship, where when they take damage, they don't want basically to get a message that cannot connect to database. They need something that's rigorous and responds well to the conditions of war. And that's where he started with that.

Also interviewed Brian Kernighan about the early days of Unix and the C language and times at Bell Labs. So that is kind of the way that I structured the show, is somebody building some piece of software and they come on and tell me and the listeners the trials and tribulations of building software.

[00:34:28] KP: Are there any trends you're especially excited about in the software world?

[00:34:32] AGB: Trends that I'm excited about? Yeah, so I wrote this article for the Earthly blog about command line tools. I mean, I guess we kind of covered this trend, but there're a lot of people building new command line tools. So a lot of times in Rust or Go that kind of fit this model that we were speaking of about being able to compose things together. So they kind of have the spirit of these original Unix tools, but try to extend them and improve them in some way. And so I'm pretty excited about that. A lot of people embrace seeing this kind of like Unix as an IDE concept.

[00:35:03] KP: I guess the antithesis of that is to say all software is going to move to no code solutions. What do you think the evolution of the software engineer's role is going to look like as we go into the future with both these possibilities in mind?

[00:35:17] AGB: Yeah, I really don't know what. I've seen some of the like – What is it? GPT3 and GitHub Copilot type demos, and they're pretty impressive. So what does the future hold? I mean, I guess, the number of people working in software development has just continued to increase. And I think that when new abilities appear, like low code, that doesn't necessarily take away from the number of things that are being built. I've heard in the past that when they're able

to produce more electricity, they think that that will cause the supply of electricity to go up, right? And the price of electricity to crash. But what they actually find is that demand increases to match supply. So like as things get easier to build, there's still lots more software to be built. So I don't think it will put a dent in the growth of our field.

[00:36:08] KP: Makes sense. Yeah, totally agree. Where can people find the CoRecursive podcast and you online?

[00:36:15] AGB: So I can be found online on Twitter or anywhere under Adam Gordon Bell. CoRecursive is corecursive.com, or just search for CoRecursive or Adam Gordon Bell in your favorite podcast player.

[00:36:28] KP: Adam, thank you so much for taking the time to come on Software Engineering Daily.

[00:36:32] AGB: Thank you so much for having me. It's been fun.

[END]