Presenters:
Karl Fogel, Open Tech Strategies
James Vasile, Open Tech Strategies

Moderator:
David Myers, Getty Conservation Institute

Note: Q&A transcript begins on page 8

DM: Thanks for waiting everyone and thanks everyone for joining us for this Arches webinar, which is going to provide an introduction to open-source software projects. My name is David Myers and I am on the Arches team at the Getty Conservation Institute in Los Angeles. And I'm going to be moderating the session today. And we decided to hold this webinar because we heard some interest from members of the community in learning not just about Arches, but how to participate in an open-source software project. And you know, how to help such projects typically work. So we're very pleased today to have presenting Karl Fogel and James Vasile, who are both partners at Open Tech Strategies, and both have had very long and broad experience working with open-source software, and through Open Tech Strategies they both provided open-source guidance to the Arches Project, since its very inception, which has been extremely helpful. And much appreciated. Before I turn it over to Karl and James, I just would like to go over a few housekeeping items in terms of how this will work, especially for the audience here. If you have questions today on the topic of the webinar, feel free to enter them into the Zoom Q&A interface. The chat function is disabled because this is a webinar. So please do use a Q&A interface for those questions. If someone has asked a question that you also want to know the answer to, you could upvote that question and we'll definitely get to those. If you need to see captions or if that would be helpful, this webinar offers live captioning, which you can enable by pushing the button that should be at the bottom of your screen that says CC or closed captions. And I believe you can also select which language for captioning. So this webinar is currently being recorded and we will make the recording available within approximately one week; you'll be notified if you registered when the recording is available. And also we'll be notifying the community through our social media channels and also on the Community Forum. So with that, I would like to turn it over to Karl and James to get in to talking about open-source software projects.

KF: I'm going to do a share screen and that should take over. Here we go. OK, can everyone see my screen? Great. Thank you. Hey, everybody, thank you for being here. It has been a pleasure to work with the Arches Project, to see it be so successful. It's one of our one of the open-source products that we point to as an example of kind of getting open-source right from the start. In this presentation we're not going to assume any technical expertise. Some people here may be programmers, some may not be. But

there's no need for you to have that kind of expertise to understand what we're talking about. These are all kind of concepts that make sense, even if you've never written a line of code. To start with (and let's see if I can get the slides moving here) let's just define what open source is, because there are often especially in sort of non-specialist media, there are various kind of divergent definitions that sometimes get used. But there is actually within the industry a very clear definition. There's a formal wording that I'm not going to go into, but essentially open source is code that is released under a very nonrestrictive, a very liberal copyright. There are no restrictions of how anyone uses the code, how anyone changes the code, how they share it or with whom they share it. That's it. That's the definition. You will hear sometimes other names for open-source software, so sometimes depreciated is OSS. - sometimes is called free software. People sometimes try to straddle all sides and just call it free and open-source software, which is then sometimes abbreviated as FOSS. Then to avoid that confusion, people have the word libre, which you will see sometimes, and that can sometimes be written as FLOSS, doesn't matter. They all refer to the same thing.

Open-source licenses: this is the first point of confusion that often comes up because there are a number of open-source licenses; in practice, though, there are only a few that matter. And those few pretty much all say the same thing. They require that you attribute the code accurately. You can't take credit for you can't put your name on someone else's code and claim you wrote it or something like that. They do allow all the provisions that I listed earlier. Among open-source licenses, here's only one real distinction that matters: whether the license is reciprocal or not. Reciprocal is sometimes called copyleft and by the way, this is the license that Arches is under and what it means is that if you have received a copy of the code, like you downloaded it and maybe you made some changes and you're distributing your changed version, you must distribute your changes under the same license. So it's contagious in a sense. The freedoms that you got with the code, you must share out in any derivative works. And then there are the non reciprocal licenses, which are sometimes called permissive or non-copyleft. But those are really the only two distinctions that matter. In the context of Arches, we're talking about copyleft and that will become relevant later in the presentation. Just a few cautions: you will occasionally see confusion about something like non-commercial open source, which is not a thing. Because you're allowed to use open-source or anything. That's part of the definition. That means you can use it for commercial uses. There's also no such thing as academic-only open-source, no such thing as government-only or a license that says you can only use this for ethical purposes, you can have that license. But it's not an open-source license. Also merely showing the code, but not granting all those freedoms in the license, just merely publishing the code so that it's visible. That's also not open source. I'm only saying is so that you if you run into someone saying we have a government open-source license, that governments only use it for free. Well, it's not open source. Arches is truly open source, by the way.

OK, now we'll talk about why you would want to do that. I'll hand it over to James.

JV: So people do open-source for all kinds of reasons. We surveyed a lot of our clients and a lot of folks in industry and asked them what they were getting out of open-source or even what they intended to get out of open-source. Why would you take your code and free it in this manner? Because going open source involves adopting a set of processes that take time. They take energy, they take focus, they take intention and making that investment of both the software that you write as well as the time and attention it takes to do it well is a measure that should be done with some sense of what you're trying to accomplish and what you hope to get out of it. So we asked people: what do you hope to get out of it? And we received a pretty short list. We were kind of amazed at how coherent it was of reasons people do open source. And this list is the set of things. And I'll just go through it pretty briefly. The first reason that we often talk about, which not is not the most prominent reason people do open source but is one that we want to put in people's minds, is for insight into what it is people are doing with your product. One of the great things about open source is that people participate in the production of your software. They participate very directly with your team who are helping to make the software and they do that participation through bug reports, through feature requests, through documentation changes, as well as through actually developing the software alongside you.

And that tells you an awful lot about what it is that people want from your software. And that, of course, helps you to make better software as well as to understand what is happening in the market for that software. That helps drive a lot of strategic planning in what to develop next, but also how to position yourself in whatever market you've entered or whatever policy space you've entered. The market insight is one of the big advantages, and it's one of the advantages that a lot people doing open-source aren't quite aware they could be getting. So we're hoping people do more intentional delving into market insight over time. But the big reason why people tend to do open source is for kind of an easy way to set up a framework for partner collaboration. We used to do inter company or inter agency collaboration through trade associations, through consortia, through heavyweight mechanisms that required a lot of time to set up a lot of time to design, a lot of legal logistics, a lot of financial investment and what I like to say these days is that a open-source project is the easiest, fastest, lightest weight way to do what we used to do with things like trade associations. If you have a bunch of parties that want to work together to achieve a goal, make something together, there is an entire off the shelf framework in open source that can help you do that quickly and well, using practices that have been socialized and iterated on and tested over time. So as a framework for partner collaboration, open source is quite powerful. Another big reason we hear people doing open source is to ease the fear of vendor lock in. When you have software that is proprietary and licensed from a vendor or sold by a vendor, you are entirely dependent on that vendor for everything that has to do with that software. If the vendor sunsets the software, pivots in a new direction, decides to do something that is not on your roadmap or decides not to do something that is on your roadmap, you don't have a lot of options. You can ask them to change their ways, but you might not be able to get them to say yes. And the nice thing about open source is that once you have received your copy of the software, you're up and running: you don't need the vendor anymore. In fact, in a really well-

run open-source ecosystem, you should have multiple choices for vendors to help you get it up and running and to exploit that software for whatever direction you want it. If you want to take in an entirely new direction, you can. But you don't have to worry about getting the vendors permission or agreement to make the software work really well for you. The real upside of that is stability, that you can continue to use it for years after a vendor has moved off in another direction. And your well laid plans will not be suddenly destabilized by a vendor shifting direction. So open source gives you that power in a way that proprietary software simply cannot.

The other thing that we see a lot of is people trying to lead standardization effort, which is to say, sometimes you want everybody to speak the same language, to use the same technology. So that everyone can be working harmoniously alongside each other, so that different pieces of technology and different work streams can flow together well across entity boundaries, across an entire industry, across the world. And open source is particularly good at that as well. We find that open source tends to improve product quality at a very technical level and that's simply because getting lots and lots of folks able to fix problems means that small problems tend to get fixed by the parties that care most about them. So if there's anything that is really bugging a user of software, they can apply their own resources to fixing it and they can fix the things that matter to them. And so product quality from the perspective of each individual user will tend to be a little bit higher than it does in proprietary software where changes only get fixed if they affect everybody. One thing that we see a lot of is software that is made by a very small group of people in the proprietary world can expand the number of developers who are looking at the code, who are developing the code, once it goes open-source, giving more people access to the software gives you a larger pool of people who are technically knowledgeable about the software and working on it. And one of the big upsides of that is that when it comes time to hire somebody to help work on the software, there are people outside your organization who have technical knowledge of the software and have left a public record of having that technical knowledge through their participation in the public process. And so when it comes time to hire somebody to help you work on the project, the pool of people who have worked on it in the past is visible to you and easy to gain access to. And that's one of the big advantages and a big reason why companies will take a piece of software and decide to make it public and open-source in this manner.

Another thing we really see a lot of is people working on internal collaboration which is to say that there are so many processes that have been iterate on in the open-source world to make software development efficient and especially efficient across boundaries, entity boundaries, internal team boundaries, time zone boundaries, district distributed geography boundaries are all problems of the open-source world has tackled and in today's world, where we're all much more distributed than we used to be, the methods that open-source has developed over time to work on projects together despite not having a lot of face to face meetings, despite not necessarily all being in the same org chart turn out to be really especially useful. So we've seen companies do something they call inner sourcing, which is they apply

open-source methodology and process to their internal structure and increase internal collaboration across departments within an organization. So those are kind of the main open-source goals that people are tackling from a process perspective. We also see a lot of innovation in the open-source space. It turns out that when you let anyone take your software and do whatever they want with it, people are going to take it and surprising things. And some of those innovations will be useful to you. Some of them won't but all of them will tend to make the software in aggregate more valuable to the world as a whole. And when it comes time for you to go in new directions, you will often find that there are folks in your ecosystem who have already investigated those new directions and can be brought in to partner with and collaborate with to help you get there faster. We also find that open-source really plays a large role in morale and retention of teams. People like working on open source. The open-source method of collaboration tends to be something developers like. And perhaps most importantly, developers really like to see their work put out in the public, put out in the world, because it gives them a calling card when they go to get their next job, it gives them a real currency in the workplace. And so for most people, they like working on open-source projects. Most technologists we talk to would prefer to be working in an open-source mode, and they prefer to work for companies that will let them do that. In fact, some developers will only work for companies that will let them do that.

More strategically. We find that open source is a marvelous way to disrupt a marketplace. Right. If you have an ecosystem full of software that is very proprietary in nature, and is not really responding well to needs, but is sort of too fixed in its market position to compete with, moving in an open-source direction with a competing product is often a way to disrupt a landscape and to change its configuration and to create new opportunities. And we've seen this, especially in the mapping space, for example, where once upon a time, the mapping space was extremely proprietary, with very high license fees. And then, you know, maybe about fifteen years ago, folks started doing a lot more open-source mapping software and shifted the industry and created lots of new software that was much more responsive to people's needs. The flip side of being able to disrupt an incumbent in a space with open-source software is that if you are the incumbent, you might consider moving in open-source direction to prevent yourself from being disrupted, to prevent somebody else from debuting in open-source product that disrupts what you are doing. So we often recommend to incumbents that are in a proprietary position to figure out where they are going to start stretching an open-source direction simply to hold off the insurgents who want something open-source and would disrupt them otherwise. Open source is also a pretty good way to engage with users. Open-source projects tend to have lots of different entry points for getting help, for talking to the team that's making the technology for asking for support for reporting bugs. All of that is user engagement. That, again, teaches you a lot about what your users need, but gives the users the engagement they want to feel like they are getting responsive support from a project.

Open source often plays a role in branding and credibility. The ability for your users to know what software they are running allows them to have some more trust in it. It allows them to see that other folks

are looking at it. So even if they are not looking under the hood, they know that somebody could and they know that that keeps projects honest and perhaps more secure. And so from a branding and credibility perspective, we consistently see that open-source products have more trust among their user base than proprietary software does. And the reason why is because you have this transparency you have the transparency that allows anyone to just look at the code, see exactly what it's doing, see exactly what it's not doing, and to test the assertions made by the producers of any sort of technology product. And that really helps folks understand what is going on. With that, with the software and gives them an easier way to engage, but also a way to engage with that higher degree of trust. All of that leads to your product having a reputation for being open, for being trustworthy, for being a place where folks can engage in ways that suit their needs, not yours. As soon as people start to realize that they can not only get what they need from your project but they can trust your project over the long haul, that leads to them wanting to adopt your project more. And that's where you get product reputation benefits. And there's one more goal that is not on the slide but we have seen a lot somewhat recently is one reason to do open-source is to get better at open-source. And we see a lot of a lot of organizations doing small open-source pilot projects as a way to help them understand more viscerally what the benefits are of doing open-source and for them to understand how we would integrate into their existing strategy and their existing process. So doing open-source for the sake of getting better at open-source seems like an odd thing to do. But it turns out that those kinds of pilot projects are what help be successful in doing open source when it really matters down the line.

And now Karl is going to talk a little bit about how one gets involved into open-source project and what the collaboration actually looks like.

KF: Thank you, James. Yeah, there are two questions that newcomers to open-source or people have heard about it, but not really done it in a hands-on way, ask: 1) how are these projects governed? And we'll talk about that later. And the other is OK that's interesting- here's an open-source project. I'm interested in a project. How would I meaning myself or my organization, my team, get involved in that? There's this open-source project is this hive of activity. But from the outside, it's not clear what the routes in are. The answer is generally that you start out with very lightweight investments of attention and effort. That's really as simple as like asking questions in the mailing lists or how many projects have like a live chat where the developers hang out during business hours and whatever times zone they're in. The next level of involvement might be like you're filing an actual bug report and takes a little more effort than just asking a question, because you have to give a good recipe for how to reproduce the bug. So that a developer can pay attention. There may be follow up correspondence with the people who are reading a bug report. So they can clarify and diagnose and eventually fix it, things like that that, you know, even just posting about your experiences using the project, the software product, or adding some stuff to an existing bug report. These are very, very easy things to do. Like the most you have to do is register an account somewhere and sometimes not even that in order to do that lightweight participation. But then

the next level is you see other people asking questions about the project. And by now you've got a little bit of knowledge about how it works and you recognize some of the stumbling blocks, these newcomers who are coming slightly after you are running into and you start responding to their questions. And importantly, you do it in the same forums where you were asking your questions that leads to perhaps you start contributing improvements the documentation so these questions get answered in advance. And there's always a document or some set of documents that tell you, OK, if you want to actually make a contribution? Here's the mechanism by which you do that. And eventually that that same mechanism is used, if you're a programmer you actually start contributing code, like a bug fix or even an enhancement or a new feature. All of this stuff is visible to the world. And it's saved. It's archived so that the questions that you answer now, your answer stays there, and it will be available for people to see in the future.

And actually, a mature open-source project is really two things. It's a code and it's a gigantic warehouse of already answered questions that has been indexed by Google and all the search engines. And if you go online and you search for that project and some question, you usually get an answer from some of these forums. So when you participate in any way, you're already helping users in the future. Mature projects tend to have multiple onboarding pathways, and the best ones document those pathways And all the people who are experienced in the project will sense after talking to you for a few moments, what the right pathway for you is if you are trying to become more involved in the project. So that's just something you advise projects to. You don't just look for someone coming in and immediately making a code contribution, have lots of ways to get involved from the beginning. And I will hand it over to James.

JV: So when we talk about code in an open-source project, we often use the word upstream. And because you're going to hear that a lot, I just want to define it really quickly. Upstream is the original open-source project. So if you are downloading Arches from GitHub or from Getty, you will be receiving it. And you are downstream from that project. So everything that you get that you receive from somebody else, from an established project is upstream from you. And when you take that upstream work, you can just take, then you can run it as is if you want to. You could also modify that product and run the modified version. So you would be downstream from the upstream project running a modified version, for example, if upstream displays text using white letters but for some reason, white letters don't work for you, you prefer your text to be blue. You can just change your version of the software to display text in blue and upstream will continue producing software that works for them with white letters and you as downstream get software that works better for you using blue letters. Now that sort of divergence turns out to be over time costly to maintain. If you make a lot of changes every time you make a change to your software, you have to then maintain your specific changes to the software. Upstream will maintain their version and they'll do a pretty good job of that. But anything you change, you have to keep running over time. When upstream upgrades, you will download a new version, a changed version of upstream, and then you have to take your changes and apply them to the new version. You have to reimplement your changes.

Now, usually that's pretty easy. You just there's software that will help it, help it happen automatically. But then you also have to test those changes in the new version. And make sure they're still working. So every time upstream makes a new version, you have to integrate your changes into that new version. Usually this works great, but sometimes when upstream makes a change, their change doesn't work with your changes. For example, upstream has white letters - they might decide to put those white letters on a blue background. So they change the background to blue and then suddenly your text is showing up as blue letters on a blue background. And it's very difficult to read. So you have a conflicting change. And then you have to go and fix that. You have to decide what to do and maybe you'll decide to go back to white letters, or maybe you'll decide to go to red letters. But you will have to make that change yourself. There is a solution to this. And the solution is to take your improvements, your blue letter innovation and say to upstream: hey, would you like blue letters? Blue letters are better. Here's why. You convince upstream that your blue letters are better and they take your changes And now when they decide what to do with a new version they release a new version. It already has the blue letters and you can just run it. You don't have to reapply your blue letter changes and that saves you the cost of maintenance. However, that's a whole process when you present your changes upstream and you say, hey, would you like blue letters? Upstream might not want blue letters. They might need to be shown why it is better. There might be a reason why blue letters works less well with their changes and they might need additional changes upstream to make that work for them. So there's a negotiation process, a conversation process that happens between upstream and down to decide what it is the upstream version going to look like and is upstream going to want to take over this change and maintain it for you.

But if they do take it, you get you get this really big benefit, which is that you do not have to do that maintenance yourself over time. And you can just sort of receive the project in a form that is more suited to your use. When you do software implementation as a downstream, you will very often hire vendors to do that for you to make your changes and we cannot emphasize this enough. Your vendors should be required as a matter of contractual requirement to submit those changes upstream and to work with upstream to land those changes in the upstream project as much as possible, because that will reduce your maintenance costs over time. There is prior art in procurement, language, and procurement discussions to help to help vendors help require vendors to do that. but also to help support vendors in doing that. And if anyone wants more information about that, please just let us know afterwards. And we're happy to share with you whatever we have. But it's very important that your vendors are doing that upstreaming, because if they are not doing that upstreaming, then over time, they are going to cost you as their customer, the benefits of open source, which is this reduced maintenance cost over time. And eventually, if you make enough changes to your downstream version, the cost of reintegrating those changes with a new upstream version will be too high and you'll be unable to upgrade.

And we do see this occasionally where folks who are operating downstream don't get to make their changes, don't get to upgrade the newest version of upstream, because it would mean that they would

have to forgo some of their changes would be too expensive to re-implement some of them. So we highly recommend that you manage upwards, push all your changes upstream and really require your vendors to do that. And then make sure they're actually getting it done. In order to work with upstream to get all this done. You know, there is that negotiation and that negotiation takes many different forms and that is when we start talking about what is governance of an open-source project look like? Who gets to decide what changes are going to land in upstream and who gets to decide what changes will be maintained over time up there? And Karl is going to talk a little bit about that in the next slide.

KF: Thank you, James. Yeah, so governance in order to understand governance in open-source software, and this is not going to be a detailed treatise on it, we have a whole other webinar about that. But the key thing you need to understand is remember that anybody can copy the entire project, essentially all the code, they can also copy the bug reports and whatever. You can basically duplicate the entire project and go off and run it yourself. In fact, that sometimes happens. It's rare, but it does. It's called forking the project like a fork in the road. Once you think about the implications of that, you realize that a thing becomes possible in open-source governance that is not possible in normal governance. Like you, if you ever you hear some industry testifying before Congress about how well we prefer the lobbyist say we prefer to have a voluntary compliance or a self-governing system and everyone knows that that's code for we want to pollute all the streams without oversight or something. But in open-source software, this actually works out OK. Not that it doesn't pollute all the streams, but the reason self governance works is: you're governing a particular copy of the code, if you download Arches - what you're doing is going to archesproject.org, you're clicking on the download button and you're getting Arches from a particular source. Anyone could go set up an alternate archesproject.org and offer a slightly different copy with some different features or code changes and that they will be perfectly within their rights to do that. So that means that the people who are governing the copy that comes from archesproject.org have to do a good job because after all, anybody can just copy the project and offer an alternate version any time.

So self governance, if the people running the project are competent, then they know what things are likely to fly and what things are going to cause grumbling and eventual rebellion. And they just don't do the latter things. So in the spectrum, from informal or formal, there's actually a lot less formal governance, a lot less procedure. You don't often need to have a vote for it for example. And a lot of things are just done informally because everyone in the discussion knows that a majority wants X or Y and if we do Z instead, that's going to start causing potential rebellions. There's also a big difference between the governance of day to day decisions like, OK, the text is blue, but there's a, you know, a typo. You don't need to have a vote about that. On the other hand, changing all the text from blue to red, maybe that's something that you want to have a discussion on the forums about because different people have different feelings about user interface readability. So there's a smooth continuum from day to day development decisions up to kind of more roadmap or more larger scale directional decisions. Questions about who is taking part in the discussions: generally, anyone in the forum can, but if it comes to an actual vote, then there is a

designated electorate. And that's usually the people who have been contributing code. And so there sometimes is a backstop formal governance structure. It is not invoked often, but it is when necessary. It can be invoked. Your position in that structure and your position, even in the informal questions, is generally proportional to how much you have contributed. That works well at both a formal and informal level because everyone kind of knows who's doing the most work and who's not. And so when someone who does a ton of work says, I think the B is bad we should do Q instead, people tend to listen. The most important thing I can say is when you're designing governance in an open-source project, be very clear about what assets are being governed. There might be a domain name, a website, some trademarks. It's OK. you can have a trademark of an open-source project. In fact, Arches does, that's fine. Trademarks licenses are different from copyright licenses and means something different. And the other thing your governing is this copy of the code. Not all copies in the world. This copy that comes from archesproject.org. And once you have that clarity, a lot of governance decisions kind of become more obvious. Licensing can affect governance and also sort of community behavior. And James is going to talk a little bit about that.

JV: Yeah. So the Arches license in particular uses the Affero GNU General license, usually just called the APGL because Affero GNU General license is quite a mouthful. The big thing to know about the APGL is that it is this kind of copyleft license, which is to say that whenever you modify a version of a copy of Arches and throw it up on a website and start using it, you are obligated under the license to take the changes you made to Arches and provide them to the public, or anyone who's using your software your version of the software. You're required to provide them with your modifications. That is to say, if you download Arches, you're downstream from Arches and you add a ton of new features - anyone else who wants those new features should be able to get them. And that's not really that big a problem. I don't know anyone using Arches who is philosophically opposed to sharing their changes to the world. But the sticking point is really just that your vendor needs to know that when they're developing software on your behalf that they're modifications have to be made available to other people under the terms of the APGL, which is say that those other people can take it and modify it and use for their purposes. And not only does your vendor have to be able to write code that they are willing to share in this manner, they have to be willing to only rely on code that can be shared in this manner. So if your vendor brings in an additional software library, an additional feature component that they didn't write, that they got from somewhere out in the world, maybe under open-source license, they have to have the legal ability to then share it with everyone else. And the big effect of that is that your vendor cannot bring in proprietary dependencies, proprietary libraries, proprietary components because they will not have, generally speaking, the permission to then take those proprietary components and relicense them under a free license like APGL. And we see this a lot where vendors don't quite understand this particular notion and they start incorporating changes into the product and they deliver it to you. And then you can't actually go and deploy it without violating the upstream Arches license. And so there needs to be, again, contractual provisions with your vendor, but also a healthy conversation with your vendor about what that means.

And the kinds of things you can't bring in is actually just everything that would be part of Arches has to be licensable or distributable under the AGPL. And this is a point of education, both for folks who are who are implementing downstream, but really for their vendors. So we always say that the best practice, the best thing you can do is to explain to your vendor that there needs to be a public repository where anyone can download the version of the code that you are running, because once that happens, you are going to be compliant with the upstream license. Anyone can come and get it. And your vendor will know that the point where they're putting in the repository that they have to have the legal ability to do that, that they are not causing themselves a legal headache with whoever they are getting their software components from. So the best practices always have a public repository where people can come get your software. They're free on GitHub. It's very easy to do. And it is, in fact, the standard way of collaborating in our industry.

Once you have that repository, you want to be really sure it matches what's on your website. What you're actually running. And so we really highly recommend using automated processes, push-button processes that allow deploying to your website from that public repository. Don't deploy from a private repository and then have a separate public copy, because what we almost always see happens is that the public and the private diverge and then you're not really complying with the license by offering people the version that that is actually running. So if you are downstream from an open-source project, best practices have your open-source repository and deploy from that repository. And I think that's it for our introduction to open-source. We really want to say thank you to everyone for listening. We know we went through a lot of things very quickly. We tried to condense things down to the bare minimum. You need to sort of navigate the information in the space. But we also recognize that there's a lot of things that we didn't cover. And so we invite questions now. And not only that, we invite you to contact us, David knows how to get us – we are @opentechstrategies.com, please send us questions. We are more than happy to take a little bit of time and talk to you about any confusion you might have about anything we said or that you're encountering in the open-source world. These particular slides are available at the URL: https://opentechstrategies.com/files/presentations/2023-gci-arches-oss/index.html. These slides are also available at open-source repository. They change over time. So you might see additional information if you come back and check them out of future date. But we're really happy to have presented them here and we hope this has been useful to you. Thank you, everybody, for listening. And I think David has another couple of slides that he wanted to talk through. So, David, if you want to take over the screen share, you're welcome to do that.

DM: That's right. Thank you both Karl and James for that really interesting and informative presentation. And let's open it up for questions now. So as I mentioned before, you can type your questions in the Q&A panel. And we already have our first question here. So let me put this out to James and Karl. The question is: “if you build an extension following the official extension pattern for Arches, it is not part of the Arches code base, how does the licensing apply? Can you install it to run in Arches? But the source is not part of Arches or a fork of the repo.”

JV: So the AGPL would say that you would put that in a public repository just like anything else, and license it under terms that allow people to take it. Anything that Arches is pulling in, even if it's pulling it in at runtime. The standard interpretation of the license for an Arches extension would be that that would be as well.

KF: I would say the best practice- our strong recommendation would be put a clear AGPL license on that extension and put it in a public repository. There are some other things, like some you could even there are some non copyleft licenses you could put it out under, but you might as well just make everything very clear for people by putting it under the same license as the thing that it is an extension for, which is Arches.

DM: All right. Thank you. We have the panel open for more questions and while we're waiting for our next question, was there anything, Karl or James, that you presented that maybe you would just like to add a touch to?

KF: I mean, we were working so hard to compress years of open source into forty minutes. It's hard to know, where to expand their choices.

JV: I think one thing that we would encourage folks to do is to think of not think of open source in a highly technical manner, but start from the base understanding, which is that the point of Arches being open-source is to build a commons of technology resources that people can use in many different ways. And the central bargain of the AGPL is that if you are benefitting from that commons, from that wealth of engineering, from that many years and many millions of dollars of investment, that you are receiving for free, the only thing that we are asking of you is that when you make improvements to it, you also put those in the comments that other people can benefit from it as well. And that collaborative spirit emanates not just from the license, but flows through to everything we do in this space. And so when we talk about building resources around Arches we're always looking to build them in ways that they can be taken up by other people and turned into further resources that get used well beyond their original intention. And that's how we kind of build a community of resources, a wealth of resources benefits through well beyond the original investment in the project. So bring that collaborative spirit to everything you do around Arches. And if you do that, everything else will work out. Everything else is just technicalities. And if you make a misstep, we will find a way to fix it. And everyone will find a way to reach a solution that works for everyone. Don't think of the license as a tricky set of legal obligations, because the truth is, we will never, almost never resort to legalities to solve any of these problems. They are a articulation of principles and the rules are designed to serve the principles. But the way we enforce those principles is not actually the mechanisms of law. The way we enforce those principles is person to person conversation and increasing collaboration to make sure it all goes right. So don't be scared by the licensing aspect of all this. Just

realize that that is the technical expression of the spirit of collaboration. And if you are in that spirit of collaboration, you're going to do just fine.

KF: I will say 30 plus years of involvement of open source. I have been involved in an actual license enforcement situation exactly once and it never, never went to court or anything like that. It was just we had a lawyer write a letter and that was in a very extreme situation with someone who really, really was not cooperating. You'd have to work pretty hard to be in that situation.

JV: Yeah. And the other aspect of it is that we have a term in open source called pre flighting, which is to say before you do something you do a bunch of pre-flight checks before you commit to it. And those pre-flight checks usually involve talking to the other folks who are in the ecosystem, talking to the upstream project. And if you do a lot of that, if you let people know what you're planning to do and how you're planning to do it, you will get enough feedback that you will avoid missteps before you're committed to them, before they become costly to do so, talk to your neighbors. That is so often the solution is so many of life's problems. And that is as true here as it is anywhere else.

DM: We do have a couple of more questions now in the queue and I think at least one of them relates to the point you just made, James. But let me get to them. The first one says: “Arches is growing and gaining more international contributions. What approaches can be taken to improve interactions and negotiations by developers across time zones?”

KF: Ooh, shall I take a stab at that, James?

JV: Yeah, go for it.

KF: So, I mean, the first thing I would say is that technical collaboration on shared pieces of work between people in different time zones doesn't look all that different in open source than it does in any kind of project. But there are some aspects of open-source communications tools and communication styles that will help one. So like a lot of water cooler conversation, open source happens in real time chat forums, apps, things slack. But often, the actual open-source projects are not using Slack but using other things that are very similar. And those are obviously, they give a lot of advantage in many cases to people in North America or South America and in Western Europe who can sort of be on at the same time.

And then often overnight, like, you'll find out that, like, you know all the Chinese developers were having a conversation that no one else was taking part in. Because that was when they were awake. So one principle that products often have is: have any conversations you want. And by the way, also leave messages like you can tell these chat systems to tell so and so when they come in, when they wake up and join the chat room, tell them this message from me, because we're not going to overlap in terms of our bedtimes. So you can use the tools to leave messages for people so that they know that you're there,

even if you don't overlap. But the other thing is many products have a sort of rule sometimes formalized that real decisions don't get made in the chat rooms. They always get made on the mailing list because the mailing list is a non-real time forum. Where you post messages, the messages are in conversational threads so people can follow certain topics. And if they care about a topic, they can participate in it when it is suitable in their time zone. And so the corollary to that rule is that no major decision gets made within less than 72 hours. So everyone has a chance - 70 to 48, whatever you want. But the idea is that everyone who might care has a chance to weigh in in the non-real time forum. So you just you make affordances. I guess the other thing is if you have regular developers meetings or users and adopters meetings and things like that, try not to always put them in like a time zone that convenient for a time slot that's convenient for like US West Coast. Which is a thing that tech companies tend to do, float the time. So that the convenient time moves around the world and always, always, always give your time zones, give your times of events with the time zone attached. So that people are clear on when it's happening. That's some basic stuff. I don't want to spend forever on that question. There's more stuff we could say and we're happy to answer questions outside this forum. What is the second question, David?

DM: The next one is from Alexei: “Great presentation James and Karl! Arches allows users to define scheme (eg: models) and export them as JSON. This would imply that models are not “code” but data and therefore not subject to the AGPL. Is this correct? And if so, how should we manage models in a public repo?”

JV: Yeah, that is quite correct. Traditionally speaking, the license applies to the code, not to the data that it operates on. And you would not need to apply the AGPL to anything the software is treating just as data. For example, all the data in your database does not have to be APGL licensed. In terms of what to do with it, you can take your model if you want to publish it, you can put it in its own repo. You can put it in the repo with a little note that says this is not AGPL. You can kind of treat that however you want as a separate piece of property and publish it in whatever manner you see fit. But the best practice for that would be to put it in an open repo somewhere with whatever license, you want to put on it. And let people download it and see it and copy it and use it to their benefit as well. But yeah, we would highly recommend you publish it somewhere even if it is not legally required by the license.

KF: I wanted to, David, go back real quick and say there's another thing about international collaboration that doesn't get discussed much, and I think we're all familiar with it, which is that everything happens in English. Unless the project originated in, say, France and it's all mostly French developers or something. I wonder if that's going to change now that there are sort of real time and good translation services coming online, just thanks to technological advances. But it might be worth being explicit about what the expectations is for English or other another language usage are and also letting the native speak whatever the language, the project uses in its documentation, in its forms, is going to be, the native

speakers of that language should be explicitly cautioned not to use slang and expressions will be clear to them, but not clear to someone for whom it's a second language. Just seem to be sensitive to in projects.

JV: And if we're talking about language diversity, you really have to design software that is easily localizable, well translatable. And there are there are structures and packages that help you do that all across the land of free software. You can pick one and make sure that you're doing your development. That makes your software text strings plug in to translation engines for easy translation, either automatic or with the help of translators. But that is a step that some projects miss. And when it comes time to make their software available in places where English is not the predominant language, they run into a lot of difficulty. So that has to be thought about relatively early on in a project to avoid some pain.

KF: Yeah. As far as I'm aware, Arches does –

JV: Yes Arches does a good job of that. Absolutely.

DM: That's right. Version Seven of Arches included full internationalization thanks to support from the Arcadia Fund. We're waiting on another question. But could one of you just go ahead say: “why bother making the extra time to answer someone's post on the forum or improve spend time improving the documentation? What's in it for you?

KF: James may also have thoughts. I’ll barge in first here. Aside from the sort of altruistic motivation of helping someone out or improving the project, if you are involved in the project, even if you're just a user, the social currency that you have in the project is an important tool. If you improved the documentation or if you are known for answering questions well, in the forums, other people are more likely to help you out when you need help. And I have observed this many, many times myself. In fact, I've observed it in Arches where I started personally contributing some documentation improvements and other things. And I get faster question answering from people who are more expert than I am since that started. But that's a general rule is your influence is proportional to your investment and your kind of social currency is proportional to your investment.

JV: Yeah. And if you're using Arches, you have a stake in Arches success. You have a stake in it doing well, gaining resources, finding new users, finding new contributors. And when you provide that support and make the community more responsive as a whole to new uses all of Arches gets better. And as somebody who is going to be using Arches and depending on it for years to come, hopefully you will benefit from that improvement. You will benefit from that increased reach of Arches in the world.

KF: I should add that if you're not involved in a hands-on way, coding and open-source collaboration, personally, you might not realize just the collaboration tools that open-source projects use are they're actually amazing. They’re one of the most interesting achievements of the human race to date. They are

really, really good at tracking who has done what, where exactly what they did and exactly how it relates to something else that someone else did. And you passed things that you did. And people in the projects use those tools very expertly to understand when they start talking to someone to understand what that person's activity in and around the project has been. I can only call it like a benevolent surveillance state. You know, it's real. It's truly used really for your benefit and for the project's benefit. But those tools are doing a lot of sophisticated things to help track who is doing what in the project. And everyone involved is using them. And if you get involved will probably start learning to use them too. There are other things I could add on other topics, David, if you want, unless there's a question.

DM: We have about a minute or two left for any questions to come in. We're still waiting on the next one. So, Karl, if you have something you'd like to add in the meantime, go ahead.

KF: I have about a one minute contribution. One thing that organizations that are new to open source sometimes do is, is that if they have either the software vendor that's working with the organization, or maybe the organization has an in-house IT staff that is making technical contributions and standing up to software and testing it and stuff, they will be afraid to have their internal conversations in the public forums of the project. So instead it'll be like, OK, we have a meeting of just the people at our company and they have an internal discussion and they all come to an agreement and then they carry that as a unified front into the project. With one person posting and actually a much better way is whatever discussion you and your vendor and your developers were going to have, not like a business contracting discussion, but a technical feature design discussion: have it in the same forums that everyone else has conversations in the project, that are archived and searchable and allow the people who are all working within or associated with your management hierarchy to disagree with each other and to argue in those forums. Because other people will see that discussion, participate and contribute and the quality, the end result is almost always going to be better. So don't be overly private. Have your discussions in public as often as you can.

DM: Thank you. So I think we are out of time here. And sorry, we do have one more final question: “Using "Free" OSS is an easy sell... its free right?! But how do you ensure that senior management recognize and accept the commitment that also needs to be made to the project?”

KF: Well, one way is that you're often getting your open-source software with some kind of support contract unless you have in-house expertise. It's free in the sense of freedom, like you can do anything you want with this code. And it's also free in the sense that it costs nothing to download. But there's always an effort involved in setting up and maintaining your instance, say your instance of Arches or whatever it is. So it's likely that there's going to be some kind of contract accompanying the usage of that free software and that their contract won't – it’s not necessarily going to be cheaper than going with some kind of proprietary solution where you rent the software, and then and then they have their support staff

over at that company. It's just that there's only one possible vendor in that second scenario. Right. Whereas you're not locked into vendors with open source. So I think it actually, paradoxically, one thing to do is to point out to management, hey, it's yeah, it's free software. But that doesn't that doesn't mean that there are no costs here. There's an organizational investment going on here. And that actually makes management take it more seriously and give us gives it more credibility in a way. James if you have anything to add to that –

JV: No I think that that covered it.

DM: Thank you, Karl. And that is the end of our time here. So I would like to thank both Karl and James for your presentation and everyone for attending, your questions and I think it's really been terrific. Like we said, we'll be putting a recording probably within a week. We will send that notice out to everyone who registered and share it in our other channels. So thank you again, everyone, and hope to catch you at our next webinar. Thank you, everybody.