TASKTOP

Episode 23: Gail Murphy
Episode Transcription

Mik Kersten: (00:15)
Hello, and welcome to the Mik + One Podcast, where I sit down with industry leaders to discuss the Project to Product movement. I'm Mik Kersten, Founder and CEO of Tasktop, and best-selling author of Project to Product: How to Survive and Thrive in the Age of Digital Disruption with the Flow Framework®. I'm very happy to have Gail Murphy joining us today. Gail is a Professor of Computer Science and Vice President of Research and Innovation at the University of British Columbia.

Mik Kersten: (00:38)
In addition to being my PhD supervisor, Gail also co-founded Tasktop with me almost 14 years ago. I continue to look to her for insight and updates on the state of the art in the research community, and I'm thrilled to have her share those with us today. There's no better authority that I know of on research and the developer productivity or as you'll note on the podcast, perhaps on productivity that we've been studying all these years. So, with that, please join me with my longtime mentor, colleague, and friend on the final episode of 2020.

Mik Kersten: (01:10)
Welcome, everyone. I'm here with Gail Murphy, the Vice President of Research and Innovation at the University of British Columbia. Gail was my PhD supervisor. She's on the Tasktop Board of Directors, taught me computer science, taught me my first programming languages classes. So, we have a very long history. I'm thrilled to have her here today to tell us about some of the history in the research that she's done, that she's led, that she's supported on productivity, on understanding how tooling and the intersection of tooling and organizations and development teams really works. I think she's elevated a lot of that research. I've learned a lot from her. A lot of the work I've done, my entire career has been based on what she's uncovered. So, with that, welcome, Gail.

Gail Murphy: (01:49)
Thanks a lot, Mik. It's always fun to have a conversation.

Mik Kersten: (01:52)
It's really nice to actually be looking through the bigger picture right now. So, I know that some of the themes have been near and dear to you and I think are just becoming very front and center for a lot of the listeners of the podcast, organizations. We're trying to understand how to improve, how to get better, how to remove friction from the work of individuals, the work of teams, and really the work of organizations. I'll never forget when I was doing my thesis with you, I had all these ideas for how to do that, right? I was a bit more self-centered back then.

Mik Kersten: (02:21)
As a developer, I want to remove friction from my day-to-day work. I kept bringing in new ideas. You said, "Well, how are you going to measure that any of this makes a difference, right? This is not just about how great these ideas are. It's about proving the ideas." In fact, the Software Practices Lab that you created, the reason I always had so much respect for all the work leading up to getting to work with you and then afterwards is this very empirical approach of actually

measuring and understanding. So, the empirical, the ground truth and being able to measure outcomes.

Mik Kersten: (02:49)
So, you challenged me to demonstrate how some of these ideas could provide a measurable increase in productivity. And then I think a really large portion of my work actually ended up and my work with you was around figuring out how to measure productivity, because I realized I didn't know how to do it. Either you or I took much stock in things like lines of code. So, can you tell us a bit about why one of the most basic and important things about understanding any discipline has been so challenging in software development? Why are we still even talking on this podcast about this topic now a couple decades later almost on measuring productivity?

Gail Murphy: (03:24)
It's really interesting how there was actually a lot of work on productivity back in the '70s and '80s as people started to try to think about lines of code and whether those were an indication of the amount of work that was being done. If you had an indication of the amount of work being done, then you can apply standard productivity formulas and get somewhere. Over time, people started to realize that you really can't use a measure like that, because of the lines of code you might write in C++ or some other language is really different than if you tried to express the same thing in APL, if it's an appropriate problem for APL. So, lines of code weren't such a good thing.

Gail Murphy: (04:02)
And then people got stalled, I think. They really didn't know how to think about productivity, these function points. They tried to abstract over lines of code, but it was also hard for people to think about, "How do you describe the functions of your software in a way that was systematic enough to really think about productivity?" Yet at the same time, and I had been a developer for a while, you would see people working really hard. You knew that the organization wanted to know something about, "Are we a productive organization? Are we a productive team? Is this particular individual a productive programmer or not?" Yet, we had no basis on which to do that.

Gail Murphy: (04:40)
So, I think when you started to be interested in those ideas, we had a lot of fun whiteboarding different approaches to thinking about how to improve development. And then if you're going to improve development and you want to do it in a scientific way, you have to start thinking about how to measure. So, the kinds of approaches you were thinking about when you did the Mylyn tool and coming up with edit ratio and measures like that were really interesting approaches to say, "Well, this isn't productivity per se, but maybe this is a proxy for what productivity might be."

Mik Kersten: (05:10)
Yeah, I guess that where we got to as the quickest way to measure is measuring what wasn't productivity, right? This realization, I remember being in your office, what we're realizing is if someone is clicking around looking for things and not generating new information, not writing code, well, we know that's not productivity. We don't know exactly which parts of their coding are productivity or whether that can be measured with whether they wrote more lines of code or

less lines of code, made bigger changes or bigger refactorings, but we realized what wasn't productive work. It's interesting that you say that, because now I realize, one of the main things that we're doing with an organization level with flow metrics is actually identifying the wait states, right?

Mik Kersten: (05:46)
We know that when flow efficiently is low, when there's lot of wait states, that's non-productive. So, if we can just actually highlight the non-productive work and look at strategies for removing that, then that will help. So, okay, that sounds like there's two things, because one of the things you mentioned, the lines of code or the function points. I will not forget, you taught me a lot around the history of this, when I was in a Fortune 100 organization, one of our customers - who are actually most forward looking on their agile transformation - they had basically, lean thinkers in-house, some people have a lot of respect with, some authors.

Mik Kersten: (06:22)
I just will never forget this meeting where the way that they were measuring productivity was through function points. This is not that long ago, right? This is a few years ago, not the '70s or '80s. So, in terms of actually measuring, and I guess, maybe we should at this point start dissecting this, right? Because there's the individual level of productivity, and you and your students have done just a ton of research on that. I'd love for you to take us through some of that. And then the team level of productivity, all the way up to organization productivity.

Mik Kersten: (06:50)
But I think some of the mistakes that this organization was making was taking this one measure that made sense for something and actually betting their IT strategy on it and their transformation strategy on it. So, could you take us through some of the journey that you've seen maybe a little bit more recent than the '70s and '80s? But notably, a lot of companies still seem to be stuck there on the individual productivity of software development.

Gail Murphy: (07:13)
Yeah, absolutely. A lot of this is I'm André Myers' PhD work. Andre was a PhD student at the University of Zurich and supervised by Thomas Fritz. Tom Zimmerman from Microsoft Research has also been involved in this work. It's been a great collaborative effort to try and understand a little bit about, "When do individual developers think they're productive?"

Gail Murphy: (07:36)
So, if we take the premise that it's really hard to have a universally accepted notion of what productivity is that we can measure, then one question you can start to ask is, "When do individual developers think they're productive? Are there any trends across what they think? Would that allow them away to try to start thinking about when they are productive and whether or not and be able to direct more of their activities towards those times they think they're productive?"

Gail Murphy: (08:06)

So, we did a number of different kinds of studies. We started with a survey. We asked developers in the survey questions like, "I have a productive day when?" and have them select and fill in information about when they might be productive. So, not surprisingly, you get results that you might expect. Individual developers think they're productive when they complete tasks or goals, when they're not being interrupted, when they have no meetings, and when they have really clear goals. So, it's interesting to see that trend that you might expect. And then you can ask them, "What are productive activities, and what are unproductive activities?" Not surprisingly, again, coding is a really productive activity, but meetings start to show this tension between the individual and the team.

Gail Murphy: (08:55)
Meetings came up as both the second most productive activity overall and the topmost unproductive activity. So, you can see this causing some challenges for individuals, because they don't really want to go to the meetings apparently or don't find them productive. But on the other hand, if it's the right meeting with the right agenda with clear goals, then they really see it being something useful to them. We then ask them questions like, "How do you measure your own productivity?" Up at the top of the list were typically number of work items, the time I spend on work items, the time I spend on code reviews, the time I spend writing code.

Gail Murphy: (09:36)
Way down at the bottom are the number of code elements I change. That was not seen as being very productive or a measure of productivity. The number of lines of code was the second lowest. The number of emails written was the third lowest. So, you see that tension between thinking about the organizational activities to some extent with work items versus the individual bashing with the code and the emails that you might need to actually do the work being attention between what's productive and what's not.

Mik Kersten: (10:07)
I'm reflecting right now. So, my days are very unproductive right now, because I measure them by emails written mostly or PowerPoint slides generated. As a developer, it was the more work items I got done. So, the more features I added that the happier I was, right? The less weight and friction and deployment issues or whatever they were that were in my way, the more productive I felt and engaged I felt as well. So, you're saying this is a common ranking in terms of how developers feel about their own productivity.

Gail Murphy: (10:38)
Yeah, an overall trend across about 370 developers that participated in the survey, but the challenge is there was no predominant, really dominant features that everybody agreed on were good measures of productivity. So, we then asked the developers, "How would you actually like to measure your productivity?" Only 27% said it was by activities that they do. 18% said it was by achievements. 70% said it was the value they produced, but you're not seeing 50 or 70% say the same thing. So, it really came across that everybody wants to measure productivity differently, which is a challenge then when you want to take it beyond the individual to the team or to the organization, because you're not going to have any consistent view of what productivity really is.

Mik Kersten: (11:32)
How much consistency was there in unproductive activities?

Gail Murphy: (11:36)
Unproductive activities, the most consistency was meetings. People think meetings are not productive.

Mik Kersten: (11:41)
Okay. So, I'm thinking we shouldn't do this, but just for this particular podcast about measuring unproductivity. So, there's more consensus on that. I'm not reflecting that what we've been doing, what I've been doing with customers is actually measuring their waste, which is obvious, rather than measuring the actual value, which then leads to the flow of value, which then leads to outcomes. But what do you take away from that, the fact that you've got at least three broad buckets that there isn't that much agreement on in terms of what makes a developer feel productive?

Gail Murphy: (12:11)
Well, what it made us do is say, we've actually got to go watch what people do and try and break it down into something that's just not a survey, because obviously, maybe we don't know the right questions to ask in order to get a holistic view across lots of developers. So, we then went. André sat behind 12 developers at three really different kinds of organizations and watched everything that they did for four hours. So, he literally wrote down every tool that they used and what they were trying to perform, how often they switch between tools. It turns out that developers switch around a lot. They work on multiple tasks per day, which other research has also shown. They switch between those tasks really frequently around.

Gail Murphy: (13:00)
So, about 13 times per hour this switching between tasks. That means they're only ever spending six minutes at any given time on a particular task. They spend a lot of time coding, but they also spend a lot of time testing. When they do that, they're always switching between different tools. So, they spend a lot of their time moving context at least from their brain between, "How do I use these tools? What tool do I need to go to?" That's a certain overhead in a sense that they have to spend when they're actually doing development.

Mik Kersten: (13:36)
I think you and I spent a lot of time on this topic of context and task switching and the rest. So, the premise was if we could reduce that, we could actually improve the flow of value that the developers are creating. But what do you think right now? Are you still seeing what we were seeing 17 years ago when we started doing this scale? If you're still seeing that in more recent research results, is it just something we should give up on or what are your colleagues' recent conclusions on this?

Gail Murphy: (14:06)

I think it means that context switching is actually a really interesting problem still. We haven't quite figured out for the many different scenarios that developers work in, how to reduce that context switch. So, what's really interesting to me is always the fact that the technology we use for software development is always racing ahead, where we might be able to help developers deal with the environments in which they're working in. So, a case in point might be eons ago, people build integrated development environments. That was supposed to reduce a lot of the friction you were referring to earlier, because all the tools would be together and you wouldn't have to move from one tool set to another to get your job done. That's been great.

Gail Murphy: (14:50)
If you want to do structured navigation of the code, you're often in maybe a development environment. Maybe you can run your tests from there. Maybe you can't. But now, developers are often having to move out of their development environment to use newer tools that might be required to do their job, maybe go test on the cloud or maybe run some other kind of specialized analysis tool. As soon as you move out of the environment, you're then again, causing them to switch the tools, which is a kind of context switch, but the bigger context, which is the fact that they work on multiple tasks per hour.

Gail Murphy: (15:28)
So, they're changing from maybe a task on building a new feature, perhaps they get stalled out, because they need to talk to someone else on the team who's not available. They are very productive people. They say, "Okay, I'll put that aside. I'll go fix this bottom." They start fixing the bug, which requires them to think about a whole new problem. So, when they move from the feature to the bug, they're actually changing their entire cognitive model of the code that they're thinking about, how it goes together, how it's structured. We know from other research studies that developers do have models in their brain of how the code works.

Gail Murphy: (16:08)
So, they're always switching between all those things, which is a lot of overhead. So, there's a ton of ability to start thinking about, "How do we help people make that switch easier? How do we reduce the number of times they have to switch? How do we ensure that the tool sets they're using carry information between them? So, they don't have to manually enter things, they don't have to think about different models of the code, but they can actually focus their cognitive effort on what they need to do and have the tools support them."

Gail Murphy: (16:41)
One of the phrases I often use is I think our tool sets for development still cause developers to do too much of the work that the tools should be doing. So, how do we flip that equation? How do we keep the developer focused on whether cognition's the only thing that could solve that problem and just make the tools be there to support them? We haven't figured out that balance correctly.

Mik Kersten: (17:03)

Yeah, that's amazing to me. Your curious ideas about how AI will replace developers, we haven't even gotten to the point where just normal tooling provides the value offloads enough of the developers workday, let alone completely replacing the ability to write the code.

Mik Kersten: (17:19)
The amount of things that have not been automated is mind boggling to me, because yeah, the trend that we were on a couple decades ago, even some of what was so much easier in the very early environments of... I'm going to go way back now, but a small tech of interest, some of those things have gotten worse. Our ability to deploy and scale code has increased dramatically, but some of these things actually have gotten harder, not easier, just because of the heterogeneity and the richness of the tool sets, the API's and the frameworks, and the cloud runtimes.

Gail Murphy: (17:47)
Yeah, the very first development environment I've used was on a Xerox Lisp machine. The debugger was amazing. We still don't have debuggers that I think are as good as those debuggers were. They do a lot more complicated code and a lot more complicated things. But if there's any developers listening to the podcast, I'd challenge them to spend a half an hour of their day as they're working and just think about, "What was hard?" Where did you actually have to do something where you think, "Oh, I should never have to take that in," or "I should never have to look up that information. It should just be there, because the tooling knows enough about what I'm going to do that it's ready to present the information that I need"?

Mik Kersten: (18:29)
This is super interesting, because I think in the end, it's these wait states as these points of friction. So, I'll share with you something that I'm seeing over and over in the flow diagnostics that we're seeing in the very large enterprises. So, you've shared some really important studies of actual individual developers. This is going all the way to the top of these organizations. So, what we're seeing is with the Flow Framework, we're able to measure flow efficiency, right? Which is similar to what you and I did for Mylyn, individual developers edit ratio, but it's basically just the ratio of a work item and a flow item being actively worked on versus waiting.

Mik Kersten: (19:03)
The flow efficiencies in large enterprises are horrendously bad, just to put it mildly. When we measure a highly effective small teams, they'll have very high flow efficiency, say over 50% or more, that work is flowing and there's just less context switching. But these large organizations get basically into these death spirals, where the amount of multitasking at the value stream, at the team, at the individual level gets so high. A lot of work waits, and then your flow efficiency tanks. So, I remember as a developer, you just reminded me, I would ask myself and try to do that bit of introspections, "Why am I waiting?"

Mik Kersten: (19:42)
Sometimes I'm waiting for a large test suite to run. Sometimes I'm waiting for a slow deployment. Sometimes I'm waiting on someone else to do a code review or submit something. The amazing thing about these waiting is to me they make sense at the individual, at the team,

and at the organizational level. So, if we can just, I think, again, reflect on and measure, "What's causing the wait states?", I do think this is a very key piece of the puzzle all the way up to the organization level.

Gail Murphy: (20:11)
Yeah, I totally agree. I think if we do think about these places where we're getting stalled and how do we get across that stall point, it can be really useful. If you think back to how organizations are structured, you've got individuals who are doing their development work as part of a team and the team is part of the organization. The individual has to make a lot of the choices to begin with of what they work on, at what time, what information they need from other people. So, they're always trying to optimize to some extent their own productivity. We have found that when they're provided with dashboards to just show them what they've been working on, that can be a really useful reflection tool for them to start thinking about some of these friction points, but we don't have a lot of really good tools yet.

Gail Murphy: (20:59)
I mean, Tasktop, I think, is a leader in this area about the team and then about the organization. So, as a team, you need to do the same reflection. Things like retrospective pieces of sprints, I know, are supposed to provide that. It's not clear how well they're working and really making people think about, "What is it that's causing challenges to our flow efficiency?" A lot of the choices the individuals are making are going to affect the team and the choices they're making for the team are going to affect them. So, you have to find somehow the right balance. If I'm an individual developer, I might choose not to answer that query from a teammate that might be stalling them, but maybe it's better to optimize the team at that point than the individual. How do you possibly make that trade off?

Gail Murphy: (21:46)
And then you think about the organization being a collection of teams, often, you get into the same kind of challenges. So, the more that we're able to visualize, the more that we're able to allow people to reflect on what they're doing and encourage them to do so, the more we might be able to get people to think about some of these trade-offs and start making them in a more systematic and thoughtful way.

Mik Kersten: (22:10)
Yeah, something that I think is so important is that just the act of reflecting on productivity, on what's slowing you down, as you just challenged our listeners to do, it in itself starts making a difference.

Gail Murphy: (22:22)
Yeah, I think so. I mean, if I remember back to some of our early Mylyn days, using Mylyn was one of the first times that I systematically wrote down the tasks I needed to do. I really like seeing them checked off, right? So, that getting things into the right granularity to check them off. I still work a lot that way now. I think it makes such a difference to have that visual reminder, what you need to do in a visual progress meter to some extent that things are either happening or not. But I would love to do that much more over my total flow than I do right now. So, how

could I optimize my use of email? How could I make sure that the information is collected together before I work on something?

Mik Kersten: (23:05)
Yeah, I think there's something to this whole approach, right? Because what we're saying is the multitasking comes... I ended up multitasking, I guess I probably still do. I obviously still work on this very task focused, as we know, when I'm waiting on something, when I'm waiting on some external dependency, on something to happen, on another team member or on a separate team. So, the act of making that explicit and visible and asking, "What's causing that multitasking? What's causing us to switch tasks six times now?" or "What's causing a team to have taken on five times a little they should have, because the 10 things they have in progress are all stuck on some security review or some legal review, some code review, some dependency?"

Mik Kersten: (23:43)
So, I'll quickly share just a couple stories, because I think the interesting thing to me is that in the end, the individual, the person and the team, they want those wait states, those bottlenecks out of their way, right? So, I remember when one of your students, Thomas Fritz, came into the Tasktop office and more or less, I mean, more structured than this asked, "Can we monitor a whole bunch of your developers and put devices on the wrist to see how their stress levels are interacting with their productivity, what they're getting done?" The quick reaction would be like, "Well, people might not like that, Thomas," but I think the three of us knew better, that people love that. Our developers love the fact that we're trying to help them get the friction out of the way.

Mik Kersten: (24:21)
So, their response to that was unanimously positive and more people want to sign up to that than he can handle. So, to me, there's this inherent positivity about that very tangible when what you're trying to do is remove that friction out of the day by removing those wait states or that need to multitask. So, how have you seen that? Because in the end, there's this way of looking at productivity that's this more big brother, we need to get more, squeeze more juice out of these lemons. Yet the approach of your entire group and all the researches that you've been doing has been very much, I think, focused on the benefit to the individual. I think there's been a very positive [inaudible 00:24:59] from that. So, can you speak to how you actually think about productivity in the larger context?

Gail Murphy: (25:04)
Yeah, great question. I mean, I think it's a question I've struggled with for a really long time, because it really does seem like we should be able to somehow measure software development productivity. It just feels like we should be able to figure out how to do that. On the other hand, if you think about knowledge work, of which software development is one kind of knowledge work, so much of it is about decision making and making the right decision at any given point. So, in a way, I've started to think about software productivity a little bit more as maybe a sequence of decisions that you're making. How do we help people do two things? One is to make the best

decision that they can at a given point in time, and second, make it as easy as possible to go back and remake the decision in a different way.

Gail Murphy: (25:54)
So, concrete things that we're thinking towards doing we haven't quite got there yet is, "How do you start surfacing all the decisions that you had to make during even writing a piece of code?" If you sit down and think about even writing, let's say, a Java class, the number of actual decisions you make is huge. So, how do you start recording a decision tree? And then allow people to maybe later say, "That decision wasn't a good one. Let me roll back to that point, remake that decision, and roll me forward as far as you can in the code."

Gail Murphy: (26:29)
So, an example would be you're writing, you have to choose a library to do some function. You choose some library. You get two hours later, and you're like, "I should have used the other library." So how do you go back and just change that decision and all the functional code that you wrote that doesn't depend on that library per se stays, but you're able to just make a simple decision change or move forward? That's a code-based example. You can almost imagine doing the technology to make that possible, because you stored in a version repository. You do all sorts of stuff. You do static analysis.

Gail Murphy: (27:07)
Well, think about that as you work as a team. You're making lots of decisions that interrelate with each other. How do you bring the right information together, so people are making the right decisions or the best decision? And then how do you share that information, so people know why you ended up somewhere at the end of the week, that you didn't expect what's your overall project?

Mik Kersten: (27:29)
Yeah, this part is so fundamental. This is really the part that I think makes so much sense and works at the individual level and all the way up to the team and the organization. So, I think, main things I've learned is the importance of flow, because it gives you a fast feedback loop. The fast feedback loop gives you a fast learning cycle or a continuous learning cycle. So, in the end, this is just the Gene Kim's Phoenix Projects, three ways of DevOps, right? Flow, feedback, and continual learning. But the thing that I see over and over in terms of... I'm really glad you're piling this. In the end, organizational productivity and business productivity is about the speed of decision making. I think what we're seeing is that for lower performing organizations, the bottleneck on that is the flow through development, right?

Mik Kersten: (28:15)
The fact that not enough is getting done. When your flow efficiency is 1% or 10% or 20%, the developer, the team, and the business is not getting feedback from what's being deployed, what's actually adding value, right? I think a beautiful way of thinking about this is something that Adrian Cockcroft taught me, which is to think about this as applying the theory of constraints to your OODA loop. So, basically your observe, orient, decide, act, loop, which is

what you're saying. I think that is a more enlightened way to think about productivity, because somehow it does feedback into the speed and effectiveness of decision making.

Mik Kersten: (28:48)
If you've got less friction because you can't roll back, or as a back end developer, some small changes that you made just broke your entire build infrastructure or just basically the test suite fails to catch a change that you made. You can't move fast, you can't have fast flow. But have you or your students studied that at all? How decision making factor basically factoring the speed of decision making and learning?

Gail Murphy: (29:15)
Not the speed of decision making per se. We've been trying to work on, "How do you capture some of the decisions and how do you find them later?" So, we've done some recent work to try to automatically locate design decisions in things like Git repositories. Okay, so if you look at Git pull requests on some projects, that can be really long streams of comments, that provide really interesting information about the design of the system. It's really hard for developers to follow all those chains of comments. So, a lot of the design decisions which are really critical get hidden, right? They're never documented. They talked about, they're written about.

Gail Murphy: (29:56)
So, Giovanni Viviani has been looking at, "How do you actually locate those automatically and develop a machine learning approach that can distinguish a piece of text in a series of Git pull requests comments? Is it about a design decision, or is it not?" So, once you can start to distinguish the text, now you have an opportunity to leverage that information in different ways. So, one thing you could think about doing is pulling out those pieces of design and being able to warn individuals on an open source system. You need to respect these kinds of designs. You may be able to start encoding them as design rules that are making them more explicit. Those are places where just by the flow of work that happened, you can go in post hoc and try and pull out the relevant information.

Gail Murphy: (30:47)
So, that the next person who makes a decision is making it in the context of information about the team. So, now you're starting to change the dynamic. Instead of a bunch of individual decisions, how do you allow people to make a decision in the context of what the team thinks? We want to look at, "How that over time could actually start helping people balanced this individual versus team tension that we've been talking about?"

Mik Kersten: (31:14)
Yeah, that sounds amazing. So, this goes back to, I think, the view that... Well, I learned it from you, but I've certainly kept this perspective is that these repositories, these tool repositories that developers that others in the value stream work with have just an incredible wealth of information around the flow of work, right? That for us, I think, you were doing this for years with the Software Practices Lab, that basically mining that information and understanding that information and tapping into those flows, augmenting them, as you were just pointing out, really

is how we can empirically learn and study how to understand, improve, measure productivity and unproductivity. So, what's your view on that now?

Mik Kersten: (32:00)
We've continued to spend time understanding basically these value stream networks, right? These different networks of relationships and work items and linkages to code and design and other document runtimes. How's your thinking about these repositories evolved? I feel like I'm still very much executing on what we understood back then when you first had us as your students, mining Bugzilla, the precursor to JIRA, for information that would help us understand how developers worked, the automatic classifiers that you were doing right back then, back in early 2000.

Mik Kersten: (32:33)
That, by the way, I just learned recently, they're now cutting edge. It's the state of the art in some of the enterprise tools to automatically classify incidents and defects based on plain text, NLP matching, natural language processing on just the descriptions and things and so on. It is actually amazing to me by the way, that those things that were so obvious back then are only being applied in industry now. So, what's your view on this and really the things you've learned over the last decade or two around repositories and flow?

Gail Murphy: (33:02)
Well, there's been a conference called Mining Software Repositories for quite a long time now. I've forgotten what edition it's on, but it's a long time. There's a lot of really interesting work, looking into whether it's your Git commits and what you can learn and automatically generating change logs to being able to understand Stack Overflow and provide better pointers and recommendations for developers who are facing a problem into that kind of repository of information. So, as, we develop things, what's always been interesting, and you and everyone else in the lab has worked on a lot of these problems, is that unlike a lot of knowledge work, software developers record a lot of stuff. They leave these digital trails behind.

Gail Murphy: (33:48)
So, those digital trails are available for doing analysis, doing recommendations, doing machine learning, doing classification, doing natural language processing. We're still trying to figure out how to actually extract the right value from all of what can be noise and understand what's relevant now from the past, because some of the past is just not relevant. So, there's a lot of work in trying to suss that out in these kinds of repositories, but it's really fairly easy in a way to create a prototype to show things are possible. It's really hard to make it work at the enterprise scale. So, even if you think about a classifier to determine who a new feature request or a new bug should automatically be assigned to based on the history of the repository, which is a classifier we built in 2006.

Gail Murphy: (34:34)
John Anvik did that work. It was already hard then to tune the classifier to produce good results, because you had to know, "Who was still available on the project? Who was going to be on vacation? What was still relevant of who was working on what kind of code and their expertise?"

So, actually making them work at scale, automatically being configured to work well for a project has taken a long time. So, it's not surprising to me that something you can prototype on one or two projects and make sure there's value takes quite a while to actually make it to enterprise deployment.

Mik Kersten: (35:10)
Yeah, I think that's a fair point. There is this interesting almost tension I've observed, where so much of the research that I've followed here and there, so much of its around Git just pull requests and a lot of what's out there in open source. So, much of what's out there is, from open source, publicly available. Some open source projects and the patterns that we're seeing, because we've been able to measure things like your group have studied, Moodle, when these large open source JIRA repositories or GitHub repositories.

Mik Kersten: (35:40)
The data that we see, the values from network data that we see and the flows that we see are pretty different than what we see in enterprise. I mean, I think the key thing is that how much of trail developers leave behind because of how rich these tools are and how effective right now these agile and DevOps tools are and adding value to developers actually capturing some of this work around the various workflows of coding and code review and the like.

Mik Kersten: (36:07)
I'm seeing two interesting things. One is that a lot of the work outside of the dev tools are also increasingly visible, a lot of the operations work to support all those flows of information. They're actually there. They're captured in a very rich way, as well. And then some of the design work, the product management work, the requirements management work, which actually you spent a ton of time on, it's there as well. So, I feel like those things have been maybe ignored a little too much from my point of view. I think it really is time as we see that data, the complete picture of the value stream is so much different than just the depth picture of the value stream.

Gail Murphy: (36:39)
That's a place where it's really hard from the software engineering research community to sometimes connect to what's really happening in enterprise software development, because there are those differences that you talked about. Open source is not a mimic in any way of some of the enterprise flows. It means that there's questions that you might want to ask in open source that don't apply to enterprise. There are questions you might want to know an answer about enterprise, you can't do an open source. I do worry about that gap, because it can end up being that the research might only really affect the open source community, which is still a large, really important community, but not be addressing a large part of the software development that's also happening in the world.

Mik Kersten: (37:21)
Now, on the flip side, something we've noticed in our flow diagnostics over the course of the last year as well is that because there's so many people in the dev tools and the dev tools are so rich... Let's say, we've got a large customer, where they do not have any proper project management, product management, or requirements management tool right. Quite a few

customers have shifted to agile without [inaudible 00:37:40] that part of the work. So, it's actually living in Excel or in PowerPoint or some insane place like that. But the really interesting thing that we've noticed is that in the flow metrics, we don't have a full picture of the value stream. However, we have a picture of these wait states still, because anything external to development that's upstream will have a wait state.

Mik Kersten: (38:00)
If we actually just formalize a bit more of the wait states for the organization, so when they're blocked on UX design, because they don't have the screens they were supposed to get when they were supposed to get them in the sprint, as long as they're actually setting that workflow state and the tool, we at least see that there is some Blackbox or semi-Blackbox of wait states over here or on some business analysis or some such thing.

Mik Kersten: (38:22)
So, I still think all that research still completely applies. Just the more of the value streams we get modeled and we get into this value stream network, the more meaningful the results are. But it still is amazing to me that the very dev centric things that you and the other researchers have done, I still think there's a lot to get out of that in terms of understanding where the bottlenecks are.

Gail Murphy: (38:42)
The interesting thing that you said too is, "What are the actual critical breadcrumbs?" Right? So, knowing that it's a wait state on UX can be a critical breadcrumb for understanding a whole picture of how parts of development are happening. If we had a better idea of, "What are the breadcrumbs we need?", we could do a better job of taking some of the research results and applying them across more contents.

Mik Kersten: (39:04)
Yeah, exactly. So, my hope, we've chatted about the sum here and there is that by having these end-to-end value streams modeled, we'll be able to apply the last 20 years of research, which again, we haven't quite effectively done yet. And then we'll actually be able to apply some of what's happening now around, let's say, applying machine learning to patterns of flow or to bottlenecks of flow and so on.

Mik Kersten: (39:28)
But my view on this, I think we see organizations trying to do this now, right? Just trying to dump all of their data into a data lake and point some ML algorithms added. The interesting to me is I've not seen a single instance of meaningful results from that, even though I've seen a lot of instances of really significant hype around it. So, what's your view on applying some of what's happening with AI to understanding, measuring software development productivity?

Gail Murphy: (40:02)
So, I mean, pretty much everybody's trying to use AI or ML, really ML to solve problems in development. There's some really cool research that's doing a lot more learning of patches to automatically fix bugs, which is really quite remarkable that you can automatically generate

code to fix bugs just based on how they've been fixed in the past. Also, it shows the repetition we have in how we code that maybe we could get rid of, but I think we have to spend a lot more time thinking about, "What are the true problems" Where there's those friction points?", those seem to me good places to see whether or not we could remove the friction point with using some machine learning.

Gail Murphy: (40:42)
But you really focus on, "What are the high value places that we could solve some flow wait state or flow problem and get a huge result of improved value flow as a result?" If we were more thoughtful about that, we would probably make more headway than when we just throw out a bunch of data and say, "Tell me what you see."

Mik Kersten: (41:05)
I think a big part of that struggle is that data... Even some of the things that you touched on, auto applying patches or fixing bugs, because it's been done before, right? Again, your point, I think, is a pretty profound one. It's probably shouldn't have been done the same way over and over before, but that is the state of the, practice out there. So, I think the interesting thing is that the state of the data, understanding of the data and the models, and I think we do know that the effectiveness of ML approaches have to do with the quality of the data, because that affects the training.

Mik Kersten: (41:37)
I think we've got this interesting scenario where data in these enterprises on the different tool data is just a complete mess, because tools are using JIRA differently than JIRA might not be connected to the downstream tools like ServiceNow or upstream tools like Planview or Clarity or some other tool, Jama. And then there's new tools being brought in. As you know, they're bringing in a tool like Tricentis and so on. None of that is connected.

Mik Kersten: (42:03)
So, my view on this is that any ML approach is just doomed to fail, because of the quality of the data that has to do with the heterogeneity of the tools, as well as the different ways that teams work. Because the fascinating thing is and this is so interesting, to me, the consistency that we see in the open source repository is so much higher than what we see in the enterprise repositories. I know, you're not going to agree with that, Gail. So, why don't you actually just tell us some of it? Because these have been interesting, the use of labels that you see, let's say, in some GitHub repositories. But before we get back to that point I was trying to get you to or ask you to comment on, what consistency are you seeing out there?

Gail Murphy: (42:44)
So, I think what we have to remember is that software development is an incredibly complex task, right? Sometimes, I'm totally blown away. If you think about, sitting down to watch Netflix during a pandemic, you think about like the layers of software that it takes to deliver that in high definition seamlessly to a device. It's pretty phenomenal when you think about how much software is there. So, we're building incredibly complex things. We're doing that in a way where there's still a lot of creativity and rightfully so and how we go about doing that. The tool chains,

software structure, the architecture, just that whole environment is very different, depending on the kind of software that you're building.

Gail Murphy: (43:31)
So, given that complexity, people apply different processes and they use the tools they have in different ways. So, examples from GitHub would be if you take a look at Kerberos, right? They have 250 tags or something they use in the repository. They use it like really systematically, but that's a lot of tags to learn if you're a new developer into it versus some other system, which might use only a fraction and doesn't actually record anything about their process, right? Some of them might only have open a bug, close a bug, right? They won't have any of those wait states. You won't know who it's really assigned to.

Gail Murphy: (44:10)
So, the range of processes that we're able to do to build this really complex thing means that the data that comes out that you might want to apply machine learning on, you have to understand that data. You have to understand what produced it and what the process was that got you to that point, if what you're trying to do is learn something about that process. Since all of that is just invisible in those tool chains, it makes it really difficult to easily apply machine learning in a meaningful kind of way.

Mik Kersten: (44:43)
Yeah. So, that to me, it's an intractable problem in the sense that let's say, you would have to have that entire process that was specified with those 200 some labels by that team as actually understood by whatever was analyzing the data or trying to train off the data, right? This is what we're seeing where it's down to... I guess, you're right though. It is similar in open source, the teams have their own ways of working.

Mik Kersten: (45:06)
That has to do with the particular tech stack they're using, the architecture they're using, the way that they interact with each other and with their end users. Those are customers. So, my view on this is there is no hope unless the team models their process into a common value stream model, right? No one else has that information. Reverse engineering it, I can't wrap my head around after the fact if the team doesn't model it.

Gail Murphy: (45:31)
Yeah, I mean, we've looked at a couple systems to try to say, "Okay, we should be able to look at the JIRA or whatever and figure out what's a feature request and think about the value chain and trace it through." It's almost impossible for most systems, right? You just cannot figure out what the real important feature requests were, what the value might have been for that. Where it even went through the system is impossible to track.

Mik Kersten: (45:53)
Okay, well, in that case, we're going to keep on our current track of saying, the team, that organization has to do the modeling into the common model. And then all these wonderful things like better analytics and at some point, machine learning for bottleneck detection, all

these great things that lie ahead, those will be possible. Until those models are done, they won't be, because the data in JIRA or the data in GitHub or Azure DevOps or wherever is actually very tied into these various specific processes.

Mik Kersten: (46:21)
By the way, I do so commonly see this desire by the organization to say, "Well, no, we need everyone using our agile tool this way, everyone using JIRA this way." But I think you nailed the point, which is I'm pragmatic about this and I just say, "Well, you won't. That's a nice thing to wish, but you'll be saying the same thing two years from now." But I think you're actually hitting on the fundamental issue, which is these teams are working with such complex code bases and legacy and new frameworks and services that they're consuming, that there's a reason for this, right? There's a reason they've really tailored their work to those tech stacks and processes. So, that level complexity is not going away.

Gail Murphy: (46:59)
No, sometimes they're creating something new. They're also responding to the particular strengths and weaknesses of people on their team. There's just such a large set of parameters in a way in any software development that I think, as you said, it's too simplistic to say just ‘work this way’.

Mik Kersten: (47:16)
Yeah, that's to say that the need for the teams to have autonomy over the way they work is fundamental because of the complexity of their work. To tell them to all the work the same way is not effective. I think we've seen that in the evolution of tools over the last few decades, right? The ones that enable this are the ones that have been adopted and pulled into organizations. The ones that make everyone work the same way, have one of you what Scrum is, one of you of how you should work actually haven't. Those tools are being ripped out of organizations today.

Gail Murphy: (47:46)
So, one of the cool things about all the kinds of roles in software development is I think people are fairly quick to get rid of a tool if it's not working for them. So, much of what we sometimes provided to development teams has the developers working for the tools more than the tools working for the developers. So, they do vote with their feet. They move to a new tool if it's going to serve more of their needs and allow them to focus on what they need to do and not all those points of friction that we've talked.

Mik Kersten: (48:16)
Yeah, I think that's so key exactly is that the developers know what makes them productive, right? I guess that's almost to say that there's a fairly high degree of that introspection that you talked about earlier, which is, "If this thing's not helping us, we're not using it," right? There's constant pressure on development teams to deliver more. There's constant sense of people being behind. That's just to say, I guess, the tool sets that we have today and the poll for those tools has been to actually aid with productivity. I mean, maybe that's a stretch, but those 200 some labels that you saw, that those were helpful to that team.

Gail Murphy: (48:47)
Yeah, I mean, people put things in because they're helpful. I mean, we are very good at cleaning up afterwards. There's that problem. But overall, if we allow developers to search for what process works for them, then the challenge, like you said, is, "How do we surface enough of that process to allow all of this analysis, prediction, classification to happen?" I don't think there's a really big gap there. The problem is we just haven't quite figured out how to let them just start something and say, "This is the thing that's important."

Mik Kersten: (49:19)
So, especially for the next generation of tools, those things might get easier, getting the entropy out of the schemas and then still add the labels that actually can probably be automated in some forms. But we have what we have today. Yeah, I think the key thing right now is that we actually do get those teams who have their own way of working to make the more abstract models, so we can understand the flows and understand where their bottlenecks are, right? Understand what's making them switch context with so frequently or have so much work that backlog. So, it seems like at least we're on a reasonable path there. Hopefully, we can learn more and more of these flow diagnostics and maybe get some of those developers, Gail, to go from six tasks which is an hour to four.

Gail Murphy: (49:58)
Yeah, four meaningful ones.

Mik Kersten: (50:00)
Yeah, four meaningful ones, exactly. Okay, amazing. So, yeah, I'm going to have to definitely think more about this, because you're now making me realize, I'm reflecting back on the last couple decades and maybe I've been trying to measure unproductivity, not measure productivity. So, that'll be a longer topic of reflection for me. Anything else that you want to leave with our listeners in terms of... I think, [inaudible 00:50:22], ways of thinking about helping organization, helping your teams, helping individual developers, by, for example, makes you reflect on, "What are productive victories and what aren't at every one of those three levels?" Anything else you want to leave our listeners with?

Gail Murphy: (50:36)
No, I think you had all the really high points. I think it's about identification of friction and reflecting on the decisions that you're making and what you need to make better decisions.

Mik Kersten: (50:49)
Excellent. Well, thank you so much. Yeah, let's do this again before too long. Thanks for sharing all of those learnings. Again, I think the next decade of what we're seeing in tools, a lot of that will come from what you and your teams and colleagues have been doing. So, thank you for that.

Gail Murphy: (51:04)
Always a blast, Mik. Thanks.

Mik Kersten: (51:10)
A huge thank you to Gail for joining me on this episode. For more, follow me on my journey on LinkedIn, Twitter, or using the #MikPlusOne or #ProjectToProduct. You can reach out to Gail on LinkedIn or her Twitter handle, which is @Gail_Murphy. We're taking a break for the holidays. We'll be back in January 2021 to continue the conversations with leaders and innovators on moving from project to product. I'd like to thank you all for your continued support of the past 22 episodes. If you haven't had an opportunity to listen to them, now is probably a great time to catch up. So, with that, happy holidays and see you in the new year.