Human Factors in Software Engineering

featuring Andrew Mellinger, Suzanne Miller, and Hasan Yasar

Welcome to the SEI podcast series, a production of Carnegie Mellon University's Software Engineering Institute. The SEI is a federally funded research and development center, sponsored by the Department of Defense and operated by Carnegie Mellon University. Today's podcast is going to be available at the SEI website at sei.cmu.edu/podcasts.

Suzanne Miller: Hello. My name is Suzanne Miller. I am a principal researcher in the Software Solutions Division working in the Agile-in-Government team. I am here today to moderate and participate in a panel on the topic of beyond coding. The beyond coding we are talking about here is human factors in software engineering.

I have my colleagues Hasan Yasar and Andrew Mellinger with me. We are all going to talk about different aspects of this. I am going to ask them to introduce themselves in a minute. I am going to start with myself, and we will move on to Hasan and then to Andrew.

My educational background in undergrad was actually human factors in ergonomics, but it was at a time—because I am the oldest here—when we were on the cusp between punch cards and video terminals. When I started, we were doing punch cards. When I graduated, video terminals were starting to come in.

I had the opportunity to see what human factors, from the viewpoint of both the developer—because I did go into development after that— and users meant when you are talking about IBM 3277 Green Screens, which some of you may remember, and some of our audience may not. That informed my early thinking. Organizational human factors has been a lot of my work since I have been at the SEI. That is focused on developers. We can also talk about the users, the usability piece. I come at this from a couple of different perspectives.

Hasan, why don't you give us a little bit of background about how you fit into this?

Hasan Yasar: Thanks for having me, Suzanne. My background is double E, electrical and electronic engineering. I have been writing code since 1988. When I compare it to 25 years ago, things change a lot in terms of technology, in terms of society, but the human is in the center all the time. It is really the human affecting the quality of the work and affecting what needs to be done. Also, the user is different. The profile seems to be using different environments, but at the end of the day, the engineering concept is a similar concept. It does not change a lot. But the environment got changed. So something (like tools and techniques) got changed, but the human did not.

Suzanne: Humans are humans, but we have different behaviors that come out when different things are in place. So we will talk a little bit about how that changes. Andrew, tell us a little bit about what you bring to this.

Andrew Mellinger: I started writing software back when I was 14. I was paid to write a small finance app for a company that they ended up using for a couple of years until they acquired a spreadsheet, and they were able to replace that with a spreadsheet. That started off with just sitting beside one of the secretaries and watching what she did and coding it up. There was really no process except that if it made her happy, and it got the right answers, then it was the right software. That kind of colored my career, such that I decided that kind of software is boring. So I went in to physics as an undergrad. I got out, and software pays better than physics. I have been doing software ever since.

Suzanne: There is that.

Andrew: You are right. Obviously, the customer is deeply involved at every step. I think we lose sight of that a lot. Obviously, the developers are very involved. So that relationship that I had with Darlene, to be able to talk to her, that is what made that piece of software work. That is what we need to nurture the whole way throughout. In short, software is a team sport, and if you can't build those teams, it's not going anywhere.

Suzanne: I have a brother—actually I have three brothers—I have a brother who went into software because he thought it would allow him to live in a little office, code his stuff, hand it over to somebody, and not have to talk to anybody all day. We all know somebody like that. In the early days, in the 70s and 80s, you could kind of get away with that to a certain extent. You would have to talk to somebody, whether it was the user or the boss, about what was wanted. As you said, there wasn't very much in between, *Here's what I want* and *Here's what I gave you*.

One of the things that I see that has changed a lot in the 40-some years since then is that there is a lot in between, *Here is what I want* and *Here is what I gave you* when we are talking about

complex systems because they are not as simple as the almost spreadsheet that you built for Darlene or the FORTRAN displays that I built for my early customers.

Talk about how the complexity of the work has changed what software engineering is about.

Hasan: I started first time writing a COBOL application, which is a similar thing to what you did for Excel files by making some accounting software. The technology at the moment was enough to develop something that people can use, which was easy. *Easy* means specific to the target and goal, *Get something done*. Right now, most of the tools are dependent on each other. So going back to as you describe your brother, he was able to go in a corner and write the code. But it is not the same anymore because it all depends on each other. It is really complex. I was teaching a class at Heinz College, I said, *Show me how you how you say "Hello, World!" in CC++*. There are multiple lines versus looking for "Hello, World!" in Python is one line.

It makes it easy to write the code, but actually it is making us more dependent on each other. You cannot really go to your cave and write the code anymore. You have to be dependent on the other team members because you are building such big systems that depend on each other. In the old days we just go and sit down and write the code by ourselves or with the individual persons who have impact on it because really there was no internet like in '84 where we were talking about. You don't need to really copy the code somewhere else. You just spend the time and learn and bring something on the table, but it has changed a lot now.

Andrew: Certainly, when I first started writing code, the instruction manual I had was about that thick. It was maybe three quarters of an inch thick. All the programming languages, all the system was in there. Even the operating system they had printed in back, and I could look up things and change what I wanted to do. With that in mind, your space is limited. You can basically hold the whole thing in your head at once, and now we can't. We are dependent upon libraries [that] are coming and going like mad. Specific domain specializations, whether it be machine learning or AI [artificial intelligence] or genetic algorithms or who knows what. The ability to actually do it all yourself anymore is rapidly decreasing, and also the easy problems have been solved.

Suzanne: I think that's a really good point. The problems that people are trying to attack today…. There is a subset of easy problems, but I think that for a lot of those easy problems, in my observation, user programmers are doing that. People that are actually using it build it on their own. We have a generation of people that are way more computer savvy on average than the people that grew up in my generation.

The idea that I have to hire someone to build a spreadsheet-like thing or that I have to hire someone to get my game to work, I don't have to do that. Tools are also opportunities for people

that have ideas but don't necessarily have all the technical detail that we had to have way back in the day. They can still do something. The professional software engineers are the ones that actually have to deal with the hard problems.

I did a study. Actually, you just reminded me of a study that I did with some folks in 2006. What we were looking at is why doesn't the software engineering community embrace modeling in the way that some other kinds of engineering have?

At the time—and it has gotten better since then—but at the time, getting people to actually do models of the software, not the system but models of the software, you actually didn't see that happening a lot. We did a whole bunch of interviews and things. One of the things that we came up with is that software engineers are trained to hold a lot of complexity in their heads. Especially from the earlier years, I was trained to hold a lot of complexity in my head when it came right down to it. That complexity makes us think that we can do anything. I know some people that fit this pattern. Until it is so complex that I can't keep it all in my head, I don't want to be bothered with modeling and some other kinds of techniques. I have it all up here. It is really hard to have it all up here anymore because of dependencies, because of lots of connections out to the world. Do you see that?

Hasan: I think that because of the domain specific information that is required, you may not know everything. In software engineering if you know the concepts, you can write it. If you know the techniques and stuff, you can really write. But when it comes to domain, it is required to learn. We cannot learn everything, it is impossible. Maybe you are focusing just on accounting software example, but now you need to get knowledge on health systems. You may have to look for other net systems. So it is really requiring the deep knowledge of the domain specific environment. You need somebody else to come and tell you what the requirements are.

Suzanne: And constraints, right? I mean a lot of what we deal with in the areas that we are dealing with are standards that you have to follow, certification, that you have to achieve; they all have sometimes conflicting requirements. You have to navigate your way through all of that. Those constraints make it more complex. It's not just, *I'm just building this thing that is going to do X. I have to do X in the context of all of these constraints that I have to live within*.

Andrew: I was thinking about when you were talking about complexity and holding it all in your head. So, certainly, we have better tools now, conceptual tools for decomposition and interfaces and things like that to be able to manage that. But I wonder how much of it ties back to the fact that we have better hardware, better compilers, better tool sets. Before I had plenty of time while I was compiling to think about my problem, and I don't now. It compiles much more quickly, or it just runs, right? There are a lot of things that were natural like the idea of how the QWERTY keyboard is laid out as it is to slow people down.

We had these natural affordances in the development space that slowed us down. We can only have so many bytes. Your problem could not exceed that 640K or whatever it was. There was no way you could deal with big data.

Suzanne: You had 640?

Andrew: That's enough for anybody, right?

Suzanne: I had to start with 48.

Andrew: I started with 4. I wonder how much of that is artificial because we don't know how to build those smaller environments for ourselves. Things like the Raspberry Pi make programming much more accessible, make systems much more accessible to people again. That might be one reason why those things take off.

Hasan: The demand got changed though because when you try to build up some unit, it depends on what the user wants, what the business wants. There is a big pressure as you said like we have a lot of time for compilation. If you think about it now, there is so much demand. There is no wait basically. You have to get something out the door quickly and easily. That is a big change. That is one of the big changes for the developer. Everybody is rushing. Everybody has to do something quick and quick and quick, because there is a demand from the business. You have to survive in this world. To survive, you have to get something out the door, either a product, which every product has software right now, which is pretty much the reality. I don't see anything that doesn't have software on it. Everything is software driven now. The engineer has to be so quick. Most of the time we either we have to delegate to each other or maybe lower quality maybe. So much stress. There are a lot of artifacts or a lot of problems that are happening that the business motivates, the business drives.

Suzanne: I was actually talking to someone who has been in the aerospace business for a long time. We were talking about how the pace has changed. We have all the Agile team stuff and incremental development and produce something every two weeks and everything. He says, *How different does agile look when you have a four-hour compile in between every time you submit something*? I had to think about that. There are some implications of that—and I'm not saying you still couldn't do it—but you would have to be thinking differently about how you formulate your work. Those gap times, those thinking times. If you don't use them productively, they don't do you any good. There are some things that you could do differently if you had some of those wait times in there, but we don't.

Hasan: We don't. Look at the new technology right now because every technology makes it easier to eliminate any type of compilation and environment virtualization. These are the new

ecosystems in software engineering that we are living with right now, which is really forcing to have quick work and quick results.

Also, the user wants to see the feedback. The updates that came out from iOS or other mobile systems, everybody wants to update. For what? I think they would like to see new features. So everyone's eager to look at that feature. *Do we have that problem before?* No, actually. *If it is working don't change anything*. Now even though everything is working, it's a better version that is updated. So even though it's kind of like a marketing business driving engineers to get much, much faster and much quicker development.

Suzanne: And much more iterative.

Hasan: Right. We talked about it before. It's causing a lot of the problem on the software engineer side though. Some people are OK with the fast pace. Some engineers aren't. Your brother, for example, maybe he cannot go in his corner and write code anymore.

Suzanne: He's learned.

Hasan: I am curious. How did he learn that?

Suzanne: He had to work on some teams, and he figured out how to do it. But it was not something that he was really taught in school. In the 70s and 80s, you didn't learn how to be on a team when you went into computer science. That wasn't part of the curriculum. By the 90s it was.

Andrew: Arguably, no. I would still say even now a lot of the CS [computer science] folks that I get are not engineers really. They are still just developers. They might take an engineering class. But you think about how those classes are structured, right? Your goal is to submit a piece of code, which generates the right answer. Your code doesn't get reviewed. It doesn't iterate over time. You get group projects, but even those code bases are not well examined over time. It's not like at the graduate level like some classes like the one you have [gestures to Hasan] in which we start to see actual long-standing code bases and people thinking about commenting and actual production code.

Suzanne: And what is different about toy projects versus the business.

Andrew: Yes, the idea of actually looking at requirements ahead of time besides, *Here is what the professor said I have to write*. Developing my own requirements and understanding those and owning those. So it is interesting that we have been talking mostly about *software engineering*, but I think we've been talking mostly about *development*.

Suzanne: Yes, I think you are right.

Andrew: And not as much about engineering.

Suzanne: Yes. In the early days, you had a CS degree or double E [electrical engineering] degree. Either one of those would qualify you to work as a software engineer or developer. Most of the time they called you a software engineer because that is a better title, but I will agree with you. Certainly, when I first started developing you know in the early 80s, I wasn't an engineer. I didn't have the same perspective that I do today on understanding the whole ecosystem of what I am building. That is one of the big shifts when you start trying to put engineering into this.

Andrew: Since we only have four hours for this webcast, I am going to avoid the whole discussion of the PE professional engineer for software engineering. We can do that later.

Suzanne: That can be another one. I hear you.

Let's talk about teams. That is one of the things that we see in much of the literature. The ecosystem that engineers live in today is about teaming and about being good team members and those sorts of things. Where do people learn how to do that if they are not getting it in school?

Hasan: I guess they have to learn the hard way. When they start to develop a project, which is requiring teamwork. Either they have survived as a part of the team, which your brother learned somehow as an example, or they are going to be a part of the team. If you think about for the daily scrum for example, if the person is not able to show the work to a team, if the visibility of the work is done through JIRA or anything else in the system, how can the person survive in this team culture? There is no way. Also, a lot of people may be producing code. If the other person is not able to produce enough, maybe producing bad code as an example, which is causing a bad team culture behavior and social behavior. It is really challenging for whoever is leading the team to get the harmony amongst team members. Either they have to learn, or they have to do something or maybe go to another project. Maybe change the carrier because there is no way you can go to your corner and write the program and come back a month later. It's kind of a daily activity.

Suzanne: That is one of the things that attracted me to moving out of development and into looking at team consulting and organizational things because I had team members that were sometimes frozen. They just didn't know how to behave outside of the small space that they were comfortable in. People have to learn this on the job. If they have to learn on the job, I think one of the things that that we have been able to do at the SEI is provide some of the resources for them, both in terms of our podcasts and blog posts and things. We do go beyond just the technical development aspects. Because if you don't learn to work together, you don't produce as much and as good of a product.

Hasan: One of the biggest failures actually for software engineering or for the product is if the team is not able to produce well or sacrificing from the code and quality. If the team is not talking to each other. If they are not sharing what they have, either they cannot develop a good algorithm for a given margin of the software, which then can fail the quality requirement, which eventually means the product will fail. It is really requiring a lot of team attitude behavior. They may be a super star for this specific feature of the code. Then when we try to get together, if it is not glued together now, there is no way. You may have a shiny perfect individual engine, but that engine cannot work anymore.

Suzanne: It's what it has to do to sit within the car. Right.

Andrew: It can't scale.

Over the course of the years, as I have been sitting on the admissions committees and working more with software engineering programs and software engineering groups, we see a lot more emphasis on internships and a lot of other adjacency experience. I think there is a lot more respect now for that kind of group mentality when a couple of decades ago people went into computer science because they were like math folks. They like that, *Computers are deterministic, and I know exactly what they are going to do. They're not going to do crazy things, and I don't have to deal with people, right.* That obviously changed. As we see now, almost every real team has an embedded ability UX design or user interface designer folks that are really steeped much more and talking with customers. I think that whole landscape has really changed over the last decade or so.

I think that the answer is *How do they get it?* Well, people are recognizing that is a bigger part of the job, and they're more forced into it I think.

Hasan: Another element, earlier you said that getting the team together is also requiring other elements of the tooling example. In the old days, we weren't really communicating or talking so much. Now the tool is also important to open up that communication channel. Look at the current generation of people. They are talking with a text. They are not talking verbally anymore.

If you have a similar environment when they are coding, like a chat example, they will like to talk to the chat, even if they are in the same room, they keep chatting each other. Isn't that same as with kids? They are texting each other. That is another cultural shift in this society that is affecting that programming side of it. That tool is also important. If you have enough tools, it makes it easy for them to communicate. They can talk. They can share.

Suzanne: What I am waiting for is enough bandwidth and enough compression in video that there is more video. Because text gives very limited affect.

Andrew: But that is a benefit.

Suzanne: Yes, you are right. It is to some. It keeps people from having to see...

Andrew: Well, I also have had a variety of co-workers in the past who are not very good speakers. They have extremely eloquent writing, but if they want to talk to you, it takes a long time. Certainly people are trying to do software because it is textual-based for the most part. Even now I have trouble getting folks go up the whiteboards and draw pictures. So we are going to continue to gravitate towards text. Think about all the case tools that we're all pictorial-based modeling and so forth. Why do they not go anywhere?

Suzanne: Software through pictures.

Andrew: I think the main [thing about] software through pictures is because we haven't had to write since we were you know in kindergarten, right? We are text-based people. We like visual things, but our brains work in text all the time.

Hasan: It is changing though, Andrew. It is changing. Look at the kids right now. They are using the touch pad. No one is using any pen or pencil it seems to me. Naturally now they use their fingers. Most of the kids they don't know how to use the mouse.

Andrew: But they still type. It's text.

Suzanne: It's just is a different keyboard than what we are used to.

Andrew: But about the text part. There is also that the delay, right? It gives me a chance to think about it. A lot of people now are studying that whole idea still of how do we get significant chunks of time to think deeply about what we are doing [See here, here, and here for more on attention management. It also relates to flow]. Even dealing with issues like the various online tools and how much time I spend chatting and not actually thinking deeply. It gives you a more of an opportunity to disconnect if it is text. If it is a text message, the person is not staring at me waiting for a response.

Suzanne: I can wait here all day, Andrew, but I'm trained to do it.

Andrew: With the text, you are free.

Suzanne: Yes, that is right. If I don't respond right away, there is not an immediate assumption that I am ignoring you or maybe there is. But there is a set of assumptions that could be reasons, and I have to go through those and decide which one I'm going to ascribe to for this.

Andrew: Yes, I think actually it was somebody here at the SEI—I thought it was you— did a study on they had these pods: four developers or four folks per pod. They checked the correlation between data flow within the pod and data flow across the pod.

Suzanne: It wasn't me, but I need to get that study.

Andrew: They saw how the tasking…If folks from different pods worked on the same task, then the information mapped in terms of information flow. Basically it was that forced interaction that caused the information flow, but there still was information flow between the folks in the pod.

Suzanne: Yes, the study that I wasn't part of, but I still remember, is back in the 90s. There was a study about conflict resolution in different media. They went from lowest affect, which is email, to highest affect, which is face-to-face, where you get the most information. The results from that study were that if you couldn't resolve a conflict within two rounds of communication at one of the lower levels of affect, you needed to go up a level. If it is a really deep conflict, you are going to have to go face-to-face, but that knowledge, *Don't go more than two rounds*, was important. I don't even think they call them that anymore, but we used to have in email what we called flame wars where somebody would say something provocative in email. You would have days of, *No! Not like that!* Everybody would get all up in arms about it. They were classic examples of nobody went to a higher-level bandwidth communication style to be able to resolve it.

I am very conscious of that. I probably favor face-to-face communication a lot more than team members that may be younger. But I don't know how that is going to evolve if we move away from the face-to-face as our primary way of communicating. How does that affect our ability to resolve conflict?

Hasan: I'm not saying that we are going to get away from the face-to-face. We should have face-to-face still. It doesn't mean it has to be in the same room but something that uses high bandwidth connection. At least let people see each other on their daily meetings, the project meetings, and not hearing a phone call and maybe some video conferences if that is available.

When we start chat mechanisms or texting for the part of their daily routines, If they are stuck in any problem, they should ask right away and not wait. That is requiring some chat or texting capabilities. For the project meetings, or any team meetings or getting to the meeting, they have to see each other. They have to open up the cameras. If they are in the same office they have to get in the same room and talk.

Suzanne: The same virtual room.

Hasan: Yes.

Suzanne: Interesting things…We added video phones into all of our offices here at the SEI. You have a choice as to whether you have it on or off. I am interested in these kinds of things. I always leave mine on. It is interesting to see how different people interact when they are on video versus on phone alone, and how I interact differently with some of those folks when I am on my cellphone on travel instead of seeing them on the video.

There are people that recognize me that I have only talked to them on the video phone. I see them in the hall, and there is recognition that wouldn't be there. There is actually a little bit of a relationship that wouldn't have been there if it was just phone. Does that make sense?

Andrew: Absolutely, yes.

Suzanne: As the complexity increases, we have got to figure out how to preserve enough of that relationship building, so that we can do the cross-team information flow. We have enough trust, so that people will share information and ask questions. But we are going to be using different tools as we move away from co-located kinds of environments.

Hasan: Or maybe there is somebody on the other side of the country. Maybe they're in different geographic zones. Maybe they are in different countries. In one of my DevOps journeys, I was talking to people that are really outside the company. Maybe some are in India or some are in Europe. How are they going to get together and talk and share in the same business vision? One successful company did it. They were getting a virtual-room capability because when they tried to have audio capabilities, people weren't able to express their concerns or problems or joys. When they started to have video capabilities, and using different locations and connecting in the same virtual room, their productivity increased in terms of code quality. Also communication was interesting because they said, *What was the problem?* The problem was the developers. They were talking, but they didn't understand each other. When they see some of the body language it is helping to understand each other.

Also another psychological effect of the programming. If one person is basically criticizing the code that the person knows well. Maybe the other person doesn't know of the coding quality. When they criticize each other...when they turn on the video, it is becoming more friendly. The person's attitude is, *It's OK*. But voices can reflect different type of attitudes. But the face was really showing the attitude of, like "*I'm not really criticizing you. I'm just basically helping you to correct the problem."* So that got changed, and that is one reason the productivity went up. It was very interesting to hear that.

Andrew: Right, so you are saying it's more of the fact that the video increased the engagement rather than it actually changed the technical content.

Hasan: Correct. That's right.

Suzanne: And it changed your ability to judge the affect. *Voice only, I perceived hostility. But when I saw the facial expressions that went along with the voice, I perceived that there wasn't as much hostility as I was hearing.*

Andrew: You say *affect.* I use *bandwidth* or *latency higher bandwidth.*

Suzanne: It's all good.

Andrew: I am going still back to your question, if we continue on the current mode, when is the indicator now to bump it up, three or four email exchanges? I think now most people don't think about it consciously like you do. They are probably like, *It just doesn't seem like it's working*, and what is their threshold for that? *How do I suddenly get more confidence if I'm used to talking in text all the time? Is that a higher barrier to entry for me to suddenly pick up the phone and call you when we're not used to that?* What is the impact of that long term?

Suzanne: I know. I am watching because I am watching how different teams of different age groups are interacting together and trying to see whether…One observation that I have out in the wild with customers is that the older generation of engineers is very mentor-focused. Especially the ones that are getting closer to retirement are conscious of, *I have this knowledge I need to pass on*. I know this one guy, and he just started writing white papers and giving all of the younger team members these white papers. They are like, *What?*

Then he started doing whiteboard sessions. He would have lunch-and-learn sessions. Everybody would come, and they got a lot more out of that personal engagement with him than out of the white papers. I am not sure if he hadn't started with the white papers if they would have shown up to the lunch-and-learn. Do you know what I'm saying? He had to kind of establish that there was something here to know. He basically communicated, *You need to know more than what's on the paper to really get this*. It was an interesting thing I saw.

Andrew: A big part of mentorship, from my understanding of mentorship, is that there has got to be a mutual understanding that both people are interested in that improvement. You don't get that from a white paper. *If I am in a room with you, I am invested in you, and you are invested in this thing, and we're together*. It might go back to that engagement piece.

Hasan: It goes back to it. I'm sure you guys heard about digital natives, digital immigrants. The younger generations know what type of tools or techniques or the socialization they can use to talk. I always learn to develop something as a kind of silo and transfer to somebody. *I've done my part. That's your part. It's not working anymore. Let's get together and talk. Yes, let's get together and somehow in the same room.*

Suzanne: I have a little game that I use in one of my classes. It is actually an adaptation of a board game called Carcassonne. It is tiles that you match up like Dominoes with landscapes.

There is one group that has two teams, and they each build their components. I give them some requirements. They build their components. Then they integrate the components. They bring them together physically and have to make them all follow some rules. Then the other group has two teams, but they start from the very beginning in the integrated environment, and they have the rules of what they are supposed to end up with.

It is interesting. I don't prevent the teams that are separated during component level—I don't prevent them from talking to each other, but they don't. They don't until it is time for them to actually integrate. Whereas on the other team I have created the environment where they are expected to communicate from the very beginning, and they act differently. They communicate from the very beginning.

Hasan: That goes the back to the analogy. You should have the right environment so that people can communicate.

Suzanne: It's interesting how just a very simple task like that generates different behaviors when you change just one thing. That is the only thing I changed is, *You guys start out at either end of the table, and you guys start out together.*

Andrew: Well, they're different people, right?

Suzanne: Well, they are different people. Yes.

Andrew: How did you remove the, *This person happens to be good chatter, and he likes this guy over here*.

Suzanne: No, I didn't deal with that. I mean it's in a classroom environment. It's not really an experiment in that sense.

Andrew: This goes back to human factors part. Every team is different. You are going to get teams that work and teams that don't. I mean, you think about all the case studies you read, about all these projects that fail because the requirements were not well written or they churn. Then you talk to people who were like, *Oh, he was a jerk* or *The project manager didn't like the technical lead*. That is really why they failed. That's why the requirements were bad.

I don't think we study enough how to actually build these software teams. We talk about general teams. There is lots of lore about shared mental models but not about taking the weird software folks and how do you actually get them to form a good team. Then bring in like a normal person like a UX designer or something like that, a normal person.

Hasan: That is one of the good things for the DevOps movement, I have to say, because I have a passion for it. It brings all the stakeholders together. That is the reason it is successful in software development practices because it gets all the stakeholders, including UX designers in your example, getting users as part of this mixture. We never really met the users before. It was kind of siloed because always we were looking to write code based on what the paper gives us as the requirements. Now it has changed because users are going to be part of the team.

Suzanne: Well, and, yes. That allows us to have a more fluid response to changes in the environment. We said that at the very beginning, but we haven't talked so much about that. The whole ecosystem—not just development but the use of our products and how quickly they are expected to evolve—that has changed a lot. Whether you are in health insurance or finance or in military applications and flying, we have a lot of environmental changes. I think they were always there, but we weren't as cognizant of them in the past, or we put more constraints on it. I know when I was working on satellites in the 80s, we had a very constrained set of things that we could change. There was an envelope that we had to stay within for pretty much everything, whether it was power or memory or anything else. Putting all those constraints on it actually made the problem simpler. The way things are evolving with technology, the environment can actually have more effect because we have got fewer of those constraints.

Andrew: Are you familiar with the term *wicked problem* for sociology?

Suzanne: Explain that to our viewers.

Andrew: For folks that are listening, wicked problems are problems that have a variety of different, weird qualities, such as there really is no right answer. There is no end. More importantly, they have a tendency to change over time as you interact with them. I think that is what we are seeing here. It used to be before, in financial program aid, they are going to do it the exact same way every time. There is nothing that is going to change.

They send the bill out. It might be years until somebody decides that they are going to contest a bill and see it a little bit different. Here, the fact that you can release a piece of software to millions of people at once and suddenly society says, *Here is how we are going to react to that. We'll start sending more or less messages or are we going to do whatever?* Suddenly that changes the problem space. Our interaction through software is changing the problem on a routine basis. If you wait too long, that's the problem. Is it that we are given these larger, more complex systems again and that rapidity has to happen just because of the nature of the problem space?

Hasan: You have to ask the right question. *Why do people or users want to change?* Are they looking for more simplicity, more usefulness, maybe more fashion? Because that's really driving

the complexity. It's not as it is before. You may be getting the same bill 100 times, but it's not happening anymore. People want to change.

Andrew: Security, privacy.

Hasan: Security compliance is driven from the domain, but I don't think the user is going to care so much.

Suzanne: The level of sophistication of the user community, certainly in the environments that we work in, it's a lot higher than it was 20 years ago. The fact that 20 years ago my user had no idea that we could actually have some kind of a last-mile link via cell connection or via some kind of wireless outpost…We didn't know what we didn't know as users. Now the user community knows a lot more about what's feasible. When we translate, especially in some of the environments we work in, they translate expectations from the general world to what they should be able to do in their mission world. We have to keep up with that. That is sometimes difficult in some of the regulated environments. We have had that conversation that we are working.

The users themselves changing I think is one of the things that changes the demand and changes the perception of what is desirable. Some of my users are willing to deal with more complexity to get some richer kind of result. Others want more simplification. It really depends on what their mission is and where they fit into the overall milieu. When you're doing things in the airframe part of the business, the joke is if you get two pilots in the room, you get three opinions as to what you should do. Because the variety of user experiences of people flying airplanes is actually quite different, even if they're all flying the same kind of airplane. We're seeing that in other places too.

Hasan: What I want to understand from you, you are saying engineers or developers should be much, much closer to the users.

Suzanne: I subscribe to that, yes.

Hasan: There is a way to success.

Suzanne: It is also part of our engineers becoming aware of the fact that if there are two pilots in a room, there are three opinions. You are going to have to adjudicate. Someone has to adjudicate what the result is going to be. Maybe you can accommodate all of them.

Hasan: Or maybe the engineer will have to find the solution to please everybody.

Suzanne: Sometimes that's possible, not always. Conflict resolution is one of the things that I know. A lot of software engineers, computer science people, did not sign up for doing conflict

resolution among multiple users and stakeholders. I guess the message that I would bring out for that is, our software teams need people that have those social skills and that can help with that.

In an Agile world, you have got scrum master roles. You have got different roles that are not necessarily as technically focused, but this is part of what they bring to the table, is the ability to help make that interaction feasible. Because I can deconflict these three opinions into something that somebody who isn't as good at conflict resolution can actually use the results.

Andrew: I think it's a differentiation in many ways that gets back to engineering versus developer. I mean you don't need to hire everybody on your team to be able to talk to customers. You don't want all of your developers out talking to customers all the time. To be able to actually build that team that has the right blend of skills and understand that there is not just a one-size-fits-all developer. Then getting longevity on that time is a lot different than it was 20 years ago.

Suzanne: I'll go back to my brother that started out thinking he was going to sit in the room coding. The thing that turned the corner with him is—I've never asked him why—but he ended up getting a master's in business administration. He was in a place where he needed to understand the business. As a result of that, he learned a lot of other things. I actually saw a huge change in his ability and his willingness to deal with different kinds of problems than he had before. It was very interesting.

Hasan: Another thing, it may be important to mention that if your brother probably can see the impact of what he does into the business, it makes a big enjoyment. I have been writing programs almost 25 years. When I see my app is being used somewhere else, it makes me happy. It is creating some impact as a by-product and progress. It is an impact. It's great. It's enjoyment. Every software engineer would like that enjoyment. If they develop code, if the code is not being used somewhere else, people say, *Why am I spending my time? It's a waste*. *I'll do something else.*

Suzanne: I would say that he has gotten to where he enjoys the impact on the business. I would agree that he has gotten to that point. It was just an interesting journey.

Hasan: So things are changing.

Suzanne: Yes. Any other final points that you guys want to bring out related to this topic and the way we wander around it?

Andrew: I want to go back to reference one thing from before. Maybe here's a business opportunity for you to make an IM [instant message] application that after you have a message

go back three and four times, it offers to let you open a video chat up and encourage you to do video chat.

Suzanne: Oh, that's an excellent idea.

Andrew: That says, *Hey, maybe you schedule a face to face meeting*. So where's that little clippy for your social etiquette avatar?

Suzanne: No more clippies. Hasan, anything else from you?

Hasan: I would like to see that people are more visible and more sharing, especially on the same team and more transparent. It may be very hard for a new programmer or newcomers. Usually sharing should mean more learning capabilities because they are going to learn from each other. That really depends on the team management or team lead. If they open up that environment, the society, or the group culture and maybe the ground rules. If they share that they have a problem, it's going to create quality. That is how engineers learn. They are not learning that at the school. They will learn in an environment, so it really depends on the managers or depends on the team, depends on the stars to share an experience. That is the way we learn, so I strongly believe we should share continuously what we have.

Suzanne: I worked in a commercial software company for a few years, a small company. The boss of the small company had a refrigerator in the office, and there was beer in the refrigerator. And the heuristic in that company was, after 3 o'clock, you know, if you want to have a beer, grab a beer, but not by yourself. It was a very interesting way to kind of get the sharing going. And you know it's like, OK. And it worked. People built relationships pretty quickly in that room. It's not something we can do at the SEI, but you know some companies can do that. Yeah.

Hasan: You've got to find another way. Get some ice cream.

Suzanne: That ice cream place closed. All right. I'm going to say we're done for today with this conversation. I hope that our viewers have gotten some insights. Certainly this is the kind of podcast that we really invite comments from the community. If you have a different opinion or you've seen something different than what we've talked about, that is one of the ways we learn as we learn from you. Thank you for joining us today. You'll be able to find this podcast wherever you get your podcasts. It's not just sei.cmu.edu/podcasts. We've got a YouTube channel. There is SoundCloud. There is Stitcher. There are things I have never heard of that you can get the podcast on. So we welcome you to do that. As always, if you have questions, please send them to info@sei.cmu.edu, and we'll happy to try and answer. Thanks for viewing.