Conversations with technology leaders: Erik Meijer

Article *in* Communications of the ACM · May 2017

DOI: 10.1145/3080143

Erik Meijer
Delft University of Technology
127 PUBLICATIONS **4,287** CITATIONS

Conversations with Technology Leaders: Erik Meijer

GREAT ENGINEERS ARE ABLE TO MAXIMIZE THEIR MENTAL POWER.

KATE MATSUDAIRA

There are smart people in the world. And then there are *really* smart people. You know the ones I am talking about—those who are so impressive that it doesn't matter what they do (academia, programming, engineering, or management); you know if they are doing it, then they are doing it well.

For this column, I wanted to share with you some of my favorite engineering and leadership lessons from one of the smartest people I know: Erik Meijer.

Whether you are a leader, a programmer, or just someone aspiring to be better, I am sure there are some smart takeaways from our conversation that will help you grow in your role. Oh, and if you read to the end, you can find out what his favorite job interview question is—and see if you would be able to pass his test.

What qualities make someone into an amazing engineer?

There is a paper called "The Humble Programmer,"[3] and even though it was written in 1972, after all these years it is still super-relevant. In the early days of computing, programming was perceived as puzzle solving and optimizing the computational process—it is astonishing how those archaic ideas are still there when it comes to interviewing developers. Our world today is very complicated—we are dealing with distributed systems, all kinds of models, neural nets, frameworks, new languages. We don't have the mental power to keep on top of every new innovation and idea. Mental power is our most precious resource.

Part of this is being able to leverage the *power of abstraction*—focusing on what is important and leaving out the unnecessary details. Sometimes details are important; other times they are not. We cannot talk about everything in absolute terms. Compared to assembly code, C is declarative. But compared to transistors and gates, assembly code is declarative. Developers need to recognize these levels of abstractions.

A good engineer knows how to handle leaky abstractions and can swiftly go up a level or dive deeper down when needed. But that same engineer also has to accept that you can never understand everything.

We need to be asking, "How can we design systems so that computers can handle more of the work for us?" For example, a lot of developers are still creating programs as text. A lot of the tools we use to manipulate programs are still too primitive because they treat code as sequences of bytes. We need to be much more mindful of how we can use computers to help do our jobs.

The whole point of "The Humble Programmer" is that your brainpower is your most limited resource, so using smart tools is a good thing. Good developers understand that they can't do everything, and they know how to leverage tools as prosthetics for their brains.

Kate's Takeaway

You should read (or reread) "The Humble Programmer." And always be on the lookout for ways to work smarter—better tools, intelligent systems, and enlisting help. Focus your mental energy on the task with the most dividends.

You have to be able to under-stand other people, and you have to be able to steer people and move people.

What qualities make someone into an amazing engineering manager?

First, you have to have deep technical knowledge. But it's also important to have self-awareness, empathy, and emotional intelligence. You have to be able to understand other people, and you have to be able to steer people and move people.

In management, there is a communication feedback loop. In one direction, a manager interacting with his or her reports requires emotional intelligence. He or she has to know what drives the other person to get optimal results. A great manager will help people do their best work.

The second part of the loop is the reports back to their manager, and the skill that matters here is empathy. You have to understand what they are trying to say despite the noisy channel between you and your reports.

Each direction of the loop introduces uncertainty. You might hear something, but that isn't what was said, and vice versa. It is your job as the manager to make sure this communication is optimized. It is on you to make sure that the feedback loop is a virtuous cycle—the better you understand your reports, the more empathy and emotional intelligence you have in that relationship.

By taking a Bayesian approach, you can increase your empathy by performing error correction on what you hear and increase your emotional intelligence by inserting redundancy into your communication. One way you get that error correction and redundancy is through peer feedback and 360 reviews to train your neural net continuously.

Kate's Takeaway

The feedback loop is an interesting way to think about your interactions and relationships. If you want another lens on a similar topic, Erik wrote a paper on "The Responsive Enterprise" that talks about these loops in an organizational context.[4]

In your mind you create a model of someone. When something happens, you hear something or observe something; then you are updating your prior assumptions. This is where you must watch your biases. In the beginning, you don't know anything about someone, but the more interactions you have over time, the more the uncertainty in your model diminishes.

What book do you wish all software engineers would read, and why?

How to Win Friends and Influence People.[1] That book gives you really complete ways of thinking about human relationships and how you interact with other people. It is written in a way that makes you consider the lessons by putting yourself in other people's shoes. How do they think or feel in these situations? And what can you do differently?

I print out the Wikipedia summary of the book and glue it into the notebooks I carry around everywhere to keep notes. Every two weeks I reread the rules and refresh myself into doing the right thing.

The books I recommend for managers are ones by Jeffrey Pfeffer and Robert Sutton (professors at Stanford) since they are more evidence driven. A lot of business books are about what people believe, but there is hardly any proof.

Kate's Takeaway

Whether it is *How to Win...* or another book, figure out your own rules and revisit them regularly. Without some sort of external stimulus, most of us will fall back into our default modes of socially awkward introvert, and so a paper taped to the inside of your planner or notebooks is a smart idea.

What is the best piece of career advice you have ever received?
When I did my Ph.D., afterward in the celebration, my advisor, Kees Koster, said to focus at the intersection of theory and practice. There is no progress without friction.

It is easy to dive into theory, or all the way into just practice—but the real interesting work happens between theory and practice. Try to understand both sides. The safe spot is to retreat to one of the extremes.

There are so many online courses these days, so many blogs, and white papers that it is easier than ever to stay up to speed on both sides. You can subscribe to Adrian Colyer's The Morning Paper,[2] go through the ACM Digital Library, read the Research for Practice column in *acmqueue*—a lot of people are making it easier to bridge gaps. Going back to "The Humble Programmer," understand that you can't keep up with all of the knowledge that is produced. You don't have to throw your hands in the air and say it is too much—you have to hone your Google skills.

Kate's Takeaway
➡ It is never enough just to do what is obvious. You have to dig deep. Devote time in your schedule to learning new things. Try to read a white paper per week, or per month.

What is your team process? How does work get done? How do you communicate status?
A lot of what you read about process and agile has very little evidence behind it. I don't believe a lot of process is scientific. Instead, I define general guidelines about what I want to see happen, and within those I don't care how things happen.

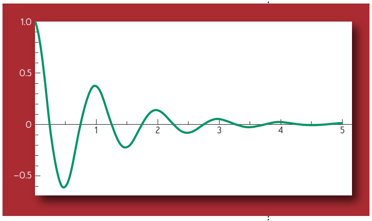
My thinking has two main sources of inspiration: the military and the hacker way.

Over thousands of years, armies have figured out how to get things done and achieve their goals in an environment that is really chaotic and completely unpredictable. That is the environment we live in as developers as well. If you read the U.S. Marine Corps *Warfighting* manual, and replace the word *war* with *software,* everything in there holds true.

So how do you deal with uncertainty? When people attempt to solve with process, they are trying to fight or control uncertainty. For example, someone can say just adopt zero inbox and your life will be awesome. In reality, though, that isn't really the case.

One of the things I like about Facebook is "the hacker way."[6] It is an approach to creating software that involves continuous improvement and feedback. It is about computational thinking: how do you program the system, and how do you make the system do things that no one thought was possible?

Being agile is about communication. The process needs to change with the situation. You have to have a big picture of where you want to go, but any plan or process will shatter immediately when you hit your first bug or something happens out of your control.



In most projects there are two phases: an explorative phase and an execution phase. Your project should progress like a damped sine wave, where the amplitude gets smaller over time. You have to figure out what to build, and figure out what question you are trying to answer. In the beginning you want to increase the vertical velocity to get uncertainty under control, and then you want horizontal velocity to increase when you get into execution.

With prescriptive processes, people are looking for a silver bullet to solve problems, but it doesn't exist. It comes back again to "The Humble Programmer." The world is super-confusing, and you have to embrace it and work with it.

Kate's Takeaway

You have to make your process work for you. Imagine your projects progressing on a damped sine wave—first focus on finding the right questions, and then the answers.

Who is the best manager you ever worked for? What made this person so great?

William Adams. He was my manager at Microsoft. He is an inspiration, and I am still trying to emulate him in my work.

There are several things I like about him. One is the importance he sees in diversity. For example, when dealing with feedback loops and prior assumptions, you need diversity to challenge your thinking. You have to actively put energy into creating a diverse environment so you are always challenging the status quo and maybe resetting your accumulated state. Don't get stuck in a local optimum.

The other thing is that he always focused on people first. You want to create the circumstances where everyone can focus on their strengths. Always find the best job for the person. Try to get a sense of the progress and circumstances so you can get ahead of what is next. For example, if the project is winding down, make sure there is always a pipeline of new

Kate's Takeaway

Think about the people around you. Do you have enough different opinions to keep your team out of a local optimum? How can you get more diversity?

ideas. You have to make sure the pipelines are set up so they never stall—keeping things innovative.

What are the common mistakes that even good engineering managers sometimes make?

Your prior assumptions are not higher order—that is, you are not making assumptions about your assumptions. I keep reading *How to Win Friends*... because I understand it is easy to fall back into my default behavior. That is the big thing: your work is never done. You never know and you aren't perfect. There is always stuff to learn. You have to keep up with your trade and keep learning. You have to keep pushing yourself to get better.

Once you get stuck and stop pushing yourself, you are toast.

Kate's Takeaway

Think about some of the past lessons you've learned. What could you use a refresher on? What are some new things you want to learn?

What is your best interview question?

Given a generic type `Cont r a = (a -> r) -> r`, prove that this type forms a monad.

If you try to solve this question by brute force, you are going to fail. But if you look at it from the right level of abstraction, it is easy. So it forces you to problem solve your own problem-solving skills.

The particular formulation using monads and type sounds really theoretical, but it is super-practical. When you are using JavaScript to write event handlers for button clicks, you are using continuations. It is a microexample of everything above in one single type.

Being a great developer is hard. It requires constant learning and a passion for technology and science. The same thing is true for great technical leaders. There are a lot of smart lessons, but perhaps the most important one is always to be pushing yourself, and to be smart about your brainpower and energy (working smart).

Hopefully you enjoyed this interview and learned a few things that you can incorporate into your work and life. See you next time!

Related articles

Lean Software Development: Building and Shipping Two Versions
Kate Matsudaira
Catering to developers' strengths while still meeting team objectives
http://queue.acm.org/detail.cfm?id=2841311

It Probably Works
Tyler McMullen
Probabilistic algorithms are all around us. Not only are they acceptable, but some programmers actually seek out chances to use them.
http://queue.acm.org/detail.cfm?id=2855183

A Conversation with Erik Meijer and José Blakeley
The Microsoft perspective on ORM
http://queue.acm.org/detail.cfm?id=1394137

References

1. Carnegie, D. 1936. *How to Win Friends and Influence People.* Simon and Schuster.
2. Colyer, A. The Morning Paper; https://blog.acolyer.org.
3. Dijkstra, E. W. 1972. The humble programmer. *Communications of the ACM* 15(10): 859-866; http://dl.acm.org/citation.cfm?id=361591.
4. Meijer, E., Kapoor, V. 2014. The responsive enterprise. *acmqueue* 12(10); http://queue.acm.org/detail.cfm?id=2685692.
5. U.S. Marine Corps. 1997. *Warfighting*; http://www.marines.mil/Portals/59/Publications/MCDP%201%20Warfighting.pdf.

6. Zuckerberg, M. 2012. Mark Zuckerberg's letter to investors: "the hacker way." *Wired*; https://www.wired.com/2012/02/zuck-letter/.

Erik Meijer *has been working on "democratizing the cloud" for the past 15 years. He is perhaps best known for his work on, among others, the Haskell, C#, Visual Basic, and Dart programming languages, as well as for his contributions to LINQ and the Rx (Reactive Extensions) framework.*

Kate Matsudaira *is an experienced technology leader. She worked in big companies such as Microsoft and Amazon and three successful startups (Decide acquired by eBay, Moz, and Delve Networks acquired by Limelight) before starting her own company, Popforms (https://popforms.com/), which was acquired by Safari Books. Having spent her early career as a software engineer, she is deeply technical and has done leading work on distributed systems, cloud computing, and mobile. She has experience managing entire product teams and research scientists, and has built her own profitable business. She is a published author, keynote speaker, and has been honored with awards such as Seattle's Top 40 under 40. She sits on the board of acmqueue and maintains a personal blog at katemats.com.*