Michael A. Cusumano is a professor at the Massachusetts Institute of Technology’s Sloan School of Management in Cambridge, Massachusetts. For more than 30 years, he has conducted research and taught classes on software company strategy, product development and entrepreneurship, as well as (early in his career) manufacturing and product development methods in the automobile and consumer electronics industries.

He has published 13 books and more than 120 articles and columns on these topics. His latest book (Strategy Rules: Five Timeless Lessons from Bill Gates, Andy Grove and Steve Jobs, published with David Yoffie in 2015) has been translated into 17 languages and received widespread media acclaim. He has also lectured and consulted to nearly 100 organizations around the world.

In the last two years, Professor Cusumano had been on leave from MIT and served as VP and Dean at the Tokyo University of Science. Because of his extensive studies over the last four decades on software development, product development, manufacturing, and (more recently) software business strategy, he possesses unique insights on technology and technology-based business strategy and product development.

---

TCS: Your research on lean management practices dates back more than 30 years. Can you tell us about the genesis of lean thinking?

Michael Cusumano: As part of my doctoral thesis on the Japanese automobile industry, and eventually for a book I published (The Japanese Automobile Industry: Technology and Management at Nissan and Tokyo, Harvard University Press, 1985), I analyzed the Toyota production system in the 1980s. I studied their productivity at the company level—units [outputs] per company as well as value-added—and I adjusted for things such as labor hours, capital investment, and capacity utilization. Toyota was producing vehicles with half the number of people [used by the American companies], but with the same amount of capital equipment. That was interesting.

So what was ‘lean’ at that time was the number of people.

Then, a student of mine—John Krafcik, who was working on his master’s thesis here at the MIT Sloan School—studied the Japanese production system at a more abstract level. John took this thinking to the assembly plant level: What was productivity like there? What degree of automation did they have in the assembly plants? What management practices did they use?
A number of us here at MIT worked on this research and debated what was the right term to describe these management practices we saw at Toyota. John came up with the term ‘lean.’ Today, he is CEO of Google’s automobile self-driving subsidiary, Waymo. At that time (1988), John published a companion article to mine (on Japanese manufacturing innovations) in the *MIT Sloan Management Review* called ‘The Triumph of Lean’.6

That started the whole thing. Since then, lean has been applied to everything under the sun. But the original idea was doing roughly the same amount of work in terms of having the same number of outputs or products, with much leaner staff.

A book that came out in 1990, *The Machine That Changed the World*7 [by the three co-directors at the time of MIT’s International Motor Vehicle Program, James Womack, Daniel Jones, and Daniel Roos] documented that the Japanese car makers were assembling cars in less than half the time the U.S. manufacturers were—12 hours vs. 25 hours.

---


TCS: By the mid-1990s, your research began focusing on product development in the automotive sector. Tell us more about that.

Cusumano: One of my doctoral students at the time (Kentaro Nobeoka) and I studied automotive product development. An earlier study by two of our colleagues (Kim Clark and Takahiro Fujimoto, who published the research in their book *Product Development Performance*), found U.S. automobile companies were generally taking 60 to 64 months to develop a new vehicle. The Japanese were pretty close to a 48-month schedule and doing it with about two-thirds the engineering hours.

But we had identified a problem in our research in the mid-1990s: The Japanese automakers had become in many ways too lean. They weren’t sharing much technology across projects within the same company.

Their heavyweight projects might appear lean if you look at them individually. But if you look at a company with a portfolio of products, the Japanese heavyweight project management teams were too self-contained. They weren’t sharing very much. So it was actually becoming expensive for them to develop new automobiles. Nobeoka and I wrote more about that in our 1998 book (Thinking Beyond Lean).

TCS: So then you shifted your research on automotive manufacturing and product development to the software industry. Tell us more about that.

Cusumano: I actually shifted my major research emphasis to software engineering even before I published my first book in 1985 on the Japanese automobile industry. Large-scale software development became my main focus from 1984.
When I wrote my doctoral thesis at Harvard in the early 1980s, it was clear to me that the next future challenge for Japanese companies would be writing software. In a post-doctoral project I did at Harvard Business School from 1984 to 1986, half of my time I spent studying consumer electronic projects with a Harvard professor, Richard Rosenbloom, and the other half I spent studying Japanese ‘software factories’—i.e., largely how the big Japanese computer manufacturers such as Fujitsu, NEC, and Hitachi developed their software.

Then I published a book in 1991—called *Japan’s Software Factories*—on who started the factory idea, how they evolved, and so on. The Japanese essentially copied how U.S. computer manufacturers like IBM ran their own software development centers in the 1970s and 1980s, because the Japanese companies were mostly making IBM-mainframe compatible hardware and software. I got interested in the question of whether Japanese companies could apply the expertise they had accumulated in manufacturing and engineering—quality control, project management, and so on—to large-scale industrial software.

I found most were taking IBM approaches to software development. There was nothing lean about it, by the way. It was almost the complete opposite of lean or what we would today call agile.

**TCS:** *So it was true waterfall approaches to software development?*

**Cusumano:** It was very waterfall.

However, Japanese software developers were good at quality. But they didn’t have a good business model for making money from the software itself. They were essentially selling hardware like IBM had done for many years, building operating systems and some applications. But they didn’t really know how to make money from software.

After this study came out in 1991, from being very interested in software and the new personal computers, I decided to ‘follow the money,’ as I like to say. Who was actually making money from software? It was Microsoft. Yet nothing had really been written about how Microsoft developed software.
In 1992 or so I suggested to an IBM person (Stan Smith), who was doing a master’s thesis under me at MIT, that he write his thesis comparing mainframe and PC software development. So he compared IBM and Fujitsu with Microsoft and Lotus, and got an interview at Microsoft, and I went with him there. When I heard the story of how Microsoft was developing software, I immediately knew this was very, very different from the waterfall world of software development that I had learned.

Almost all the principles of agile development—Microsoft was doing back then. They just didn’t call it that at the time.

So I immediately pitched a book to Microsoft, which eventually had to be approved by Bill Gates, which he did. I co-authored it with a professor of computer science, Richard Selby, because Stan went back to IBM. [Cusumano and Selby published the book, Microsoft Secrets, in 1995.]

**TCS:** So what did you learn about the ways that Microsoft developed software in the 1990s?

**Cusumano:** Since this interview is about lean and agile management approaches, I think you’ll find this interesting: Almost all the principles of agile development—Microsoft was doing back then. They just didn’t call it that at the time.
They did certain things differently. But the basic tenets of daily builds, continuous integration testing, working around small chunks of code, features, small feature teams—Microsoft developers were all doing those things. I must note that the company wasn’t doing this in all of their [product] groups. But they were doing it in their best groups—Excel and Word, among them. The Excel group pushed these techniques the furthest. Eventually all development groups in Microsoft adopted these practices, with some variations.

We never called it ‘lean,’ but they thought of themselves that way, especially when they compared themselves to IBM.

**TCS:** So now we circle back to agile development. When you look at your research over the last 30-plus years, starting with manufacturing and product development processes in the auto industry, and then moving to software development and software business strategy more recently, are there any principles in developing and making cars that transcend to software?
Cusumano: At a higher level of principles, there are a half-dozen points that I make. The most critical one is about waterfall’s sequential approach to development vs. doing things concurrently or in parallel. In both cases, when you do things sequentially it’s going to take a longer time.

So why was GM’s product development cycle 64 months back then? It was because of the way they scheduled things. They had certain bottlenecks and couldn’t do certain things in parallel because they had never tried to do them in parallel. When we looked at how the Japanese auto companies got product development down to 40-42 months, they were doing a whole bunch of things in parallel. For example, they were starting their manufacturing preparations and die manufacturing almost concurrently. They were sharing information with the different teams. As soon as they could figure out the dimensions of a car door, the die guys would start working. They didn’t wait until the whole car was designed. So they started overlapping a lot of those phases.

If you look at waterfall vs. agile/lean methods, or what Selby and I called the ‘synchronize-and-stabilize’ approach, which is the name we used in the book about Microsoft, the company was doing a lot of things in parallel. They would start writing code before they had a complete specification. Sometimes they never got a complete spec. They would start integration testing as early as possible; they didn’t wait to test a system until the end.

Another thing we found was that there could be a lot of wasted effort if you have to redo a lot of work. If you built a prototype of a software product and showed it to customers early, you could eliminate a lot of those changes later on. Even if you were doing a lot of what seemed to be re-work, or fixing bugs, early by throwing code into integration tests early, you were still saving yourself a lot of time in the end.

Firms that did early prototype-driven development—and not just throw-away prototypes—but working prototypes, and then did small incremental but frequent builds, and continuous design
and coding reviews ended up with higher productivity and better software quality than software developed through waterfall-ish approaches.

**TCS:** So the last thing we come to is the topic of building software for Internet applications. You’ve written about this too, starting 20 years ago, comparing how one of the early Web browser companies (Netscape) and Microsoft competed against each other in the early days of the Web.

**Cusumano:** When Netscape came to market with one of the first Internet browsers (Navigator), they had basically copied some of Microsoft’s practices. They were doing daily builds, for example. But one of the things that was very different with Netscape is that they weren’t printing software on disks and putting disks in boxes and shipping boxes to stores and that kind of thing. They weren’t delivering software that way. They also weren’t necessarily delivering software to PC manufacturers. That’s the way that Microsoft delivered software; that slowed the cycle.

Netscape was just posting new versions of Navigator on its website and you downloaded it. So we saw at Netscape that they would have new releases every week, or every other week. That was a completely different way of developing and releasing software. In Microsoft, those would all be internal releases. In Netscape, they were public releases.

Microsoft came up with its own browser (Explorer), but the first couple of versions weren’t very good. But by version 3 they had caught up and Netscape got bogged down. Netscape had added all these features around the browser and they were not as disciplined as Microsoft. So eventually the daily builds wouldn’t work. They had around 30 million lines of spaghetti code; those are the phrases Netscape engineers used internally.

So Netscape did things too fast and they were not very disciplined at testing. We think Microsoft won the browser wars because they actually had a better browser by the time you got to version 3 or 4. And then, of course, Netscape sold itself to AOL, although the code lives on in Firefox.
TCS: And of course, since those early Web days of browser wars, we’re seeing whole industries being upended by the Internet: media, retailing, banking, the taxi, and hotel industries, and many more. What do you think that means for the way companies develop software, and whole Internet-based businesses that of course require software?

Cusumano: I must admit that I don’t know much about how these companies develop software these days. But I don’t see software development as a problem for them. Whereas 30 years ago software development was a huge problem, the bigger problem now seems to be their strategy or business model.

Perhaps software development is much less of a problem today because people have more knowledge of what good practices are, and that they’ve largely abandoned, although not completely, the old waterfall style. They have become much more careful about how and when they use the waterfall and are more eclectic in methods.

In addition, a lot of big companies today have outsourced their software development. That’s another reason why it’s not a problem. These companies may have some lean in-house staffs, but outside specialists are building a lot of their software, although that might be different for defense companies or large banks, or small Internet startups.

In general, once companies stopped trying to force square pegs into round holes, or force a methodology that might work for space shuttle software but is not necessary for a smartphone app, then software development got a lot easier for a lot of companies.