Editor: **Mik Kersten**
Tasktop
mik@tasktop.com

The End of the Manufacturing-Line Analogy

Mik Kersten

I **RECENTLY VISITED** the BMW Group's Leipzig plant. My goal was to brainstorm with BMW Group IT leaders on how we could seamlessly integrate production lines with the software lifecycle. The visit involved a 10-km walk along the plant's production lines, with plant leadership explaining each system, process, and tool involved in car production. That visit impacted my understanding of lean manufacturing more profoundly than all the books I've read on lean processes.

The plant is an incredible facility that leads the industry in technology and sustainability, producing a BMW 1 or 2 Series car every 70 seconds. It also houses the amazingly innovative i3 and i8 production lines. Walking into the Central Building (see Figure 1) combines the sense of watching a starship construction facility with the feel of a large tech startup. Open offices sit below an exposed part of the production line that moves cars from the body shop to the bottleneck of the plant (more on that later) and then to the assembly building.

As I asked the plant leadership hundreds of questions, my mind raced trying to draw parallels between how cars and software are built. The combination of robots and humans working in unison was a glimpse into the future of how human skill will be combined with AI and automation. But what impressed me the most was the plant's architecture, which demonstrates an elegance and modularity any software architect would envy.

In Figure 2, the assembly line's key stages are visible as the "knuckles" of the five "fingers" growing out to the right. Each finger is a key fixed point in the production line's architecture, with the buildings growing outward as manufacturing steps are added and as technologies and customer demands evolve. I had never imagined that the principles I associate with software architecture could take such a physical, monumental form.

I spent the months after my visit thinking about how to apply the plant's manufacturing innovations to rethinking how software is built. How do we emulate the visibility that the rework area provided? How do we align our software architectures with the value stream the way the Leipzig plant has done, from the building's structure to the placement of each stage and tool? How do we blend automation and human work this seamlessly? And, most important, how do we make the key bottleneck of a software process as visible to our staff as the plant has done?

Then I had a lengthy talk with Nicole Bryan, Tasktop's vice president of products, who convinced me that I was thinking about this all wrong.

Software Development Isn't a Manufacturing Process

One of the most impressive things about the Leipzig plant is the large-scale implementation of just-in-time inventory. Even more interesting is that the cars are manufactured just-in-sequence: cars come off the production line in the same order that the customer orders come in. While the many stages are impressive, seeing the optimizations in the end-to-end process was nothing short of mind blowing.

The concept of *pull* is core in any lean system. For manufacturing, pull is the sequence of customer orders for physical widgets. At the BMW Group, the widgets are cars that meet market demand once they delight the customer with "sheer driving pleasure," a phrase posted throughout

0740-7459/17/$33.00 © 2017 IEEE

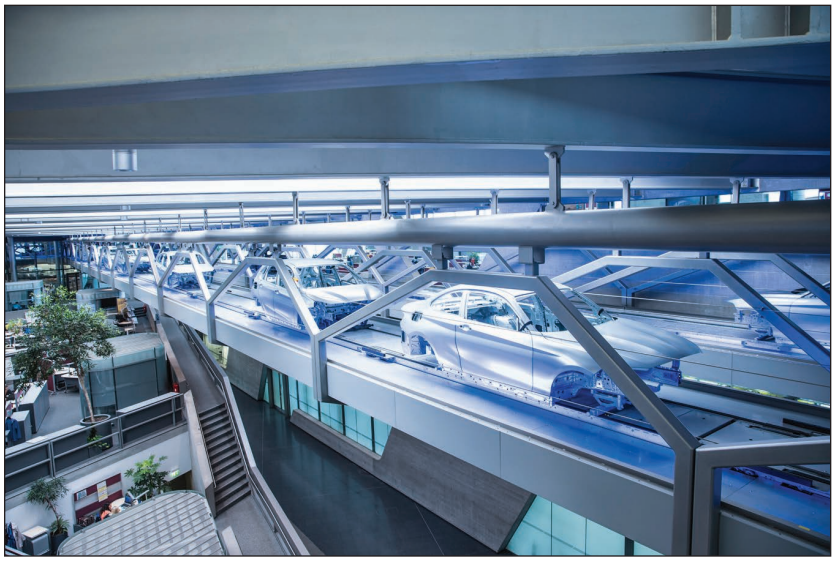


FIGURE 1. The Central Hub Building of the BMW Group's Leipzig plant. The plant produces a BMW 1 or 2 Series car every 70 seconds. (Source: The BMW Group; used with permission.)

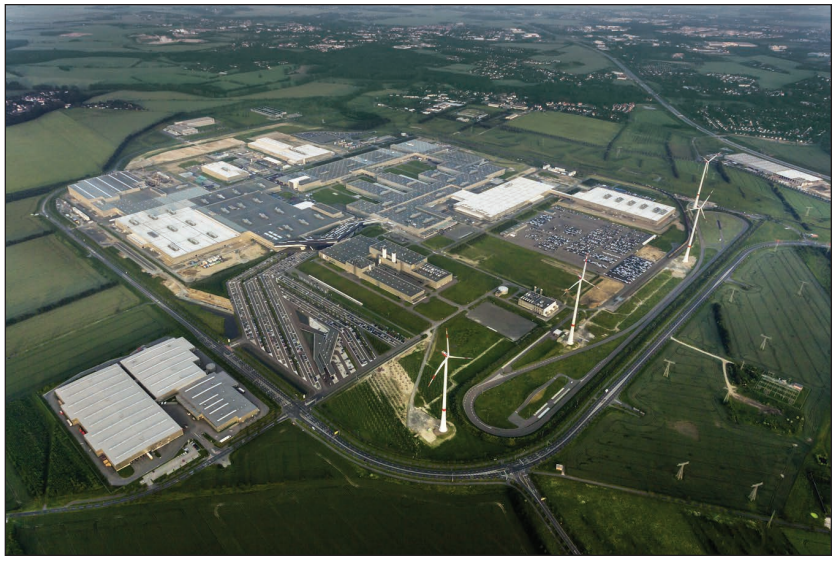


FIGURE 2. A drone photo of the BMW Group's Leipzig plant. The manufacturing line's key stages are visible as the "knuckles" of the five "fingers" growing up from and to the left of the middle of the plant. (Source: The BMW Group; used with permission.)

the plant for staff to see. If more 1 Series than 2 Series cars are delighting users, more 1 Series cars come off the line, and the line's tooling and processes adapt to the new demand.

As I walked the factory floor, the Zen koan stuck in my head was trying to figure out what these "flow units" would be in an amorphous software delivery process. Taking inspiration from the BMW Group's emphasis on "sheer driving pleasure," we might conclude that those flow units should be something that delights our end users. Yet we know that with the days of shrink-wrap and compact-disk stamping far behind us, developers delight nobody by shipping the same piece of software again and again.

Lean thinking is about letting the customer pull value from the producer. So, widgets in software should be units of business value that flow to the customer, producing some combination of delight, lack of annoyance, and revenue. I'll tighten the definition of these flow units in a future column; for now, consider them to be features added, defects fixed, security vulnerabilities resolved, and similar units of business value that customers want to pull. Yet no two of these flow units are ever the same.

And here we see a core difference. Whereas a car-manufacturing plant aims to churn out the same widget in various configurations with the highest speed, reliability, and quality possible, software development organizations crank out a different widget with every feature delivered. Determining what those features should look like is similar to the BMW Group designing its next car. But in high-efficiency software shops, it happens at a weekly or an hourly, not a yearly, cadence.

If you have a constrained set of inputs and want to produce high-quality widgets, your best bet is to create a completely linear batch-style process, the ultimate example of which is a car production line. But if you're cranking out a different widget every time, and defining that widget's size and shape is a creative process, a linear process is a wrong and misleading model.

Pitfalls of the Wrong Mental Model

As scientists, engineers, and technologists, we do well by reducing complex problems to simpler ones. But consider some of the missteps we've taken in past attempts to improve large-scale software delivery. Waterfall development looked great in theory because it made linear the complexity of connecting all the stakeholders in software delivery. Agile development came to the rescue but oversimplified its view of delivery to exclude upstream and downstream stakeholders such as business analysts and operations staff. DevOps addressed that by embracing operations, automation, and repeatability of deployment processes. But I now fear that by overfocusing on linear processes rather than the DevOps tenets of end-to-end flow and feedback, organizations are about to make similar mistakes by adopting an overly narrow and overly linear view of DevOps.

The ability to stamp out frequent releases in an automated, repeatable way can be a great starting point for DevOps transformations. But that's only a small step in optimizing the end-to-end software value stream. The theory of constraints[1] tells us that investing in just one segment of the value stream won't produce results unless that segment is the bottleneck. But how do we know it's the bottleneck? Even more important, what if we're looking for a linear bottleneck in a nonlinear process?

Software development comprises a set of manufacturing-like processes. Taken in isolation, each can be thought of as batch flow in which automation and repeatability determine success. For example, in the 1970s, we mastered software assembly, with compilers and systems such as GNU Make providing batch-style repeatability for building very large codebases. In the following decade, GUI builders and code generation became an automation stage we now take for granted when building mobile UIs. Now, we're in the process of mastering code deployment, release, and performance management, making frequent releases a reliable and safe process. However, each of these is only a single building block of an end-to-end software value stream, analogous to the various stages of robots that form, weld, and assemble a car. But with software, these various stages don't combine to form the simple one-way batch flow of a production line.

If we could take a virtual MRI of the workflows in a large IT organization, similarly to viewing a moving x-ray of the BMW Group plant from above, what underlying structure would we see? I've done this for my own organization and for our clients' organizations, and the resulting visualizations look nothing like an assembly line. But they do bear a fascinating resemblance to the airline network maps at the back of in-flight magazines. If you imagine the visualization of the flow of airplanes over time, adapting to route changes or bottlenecks due to severe weather and delayed crews, you're starting to get the picture.

If we try to map an IT organization like an airplane network, what are the nodes? The routes? How do we map the flows of features and fixes across projects, products, and teams? I'll examine this more closely in an upcoming column. For now, I propose that this network-based model is more representative of software development, and that by reducing software development to linear manufacturing paradigms, we're pursuing the wrong approach. The process of identifying a linear batch flow's constraints differs greatly from optimizing a network's flow.

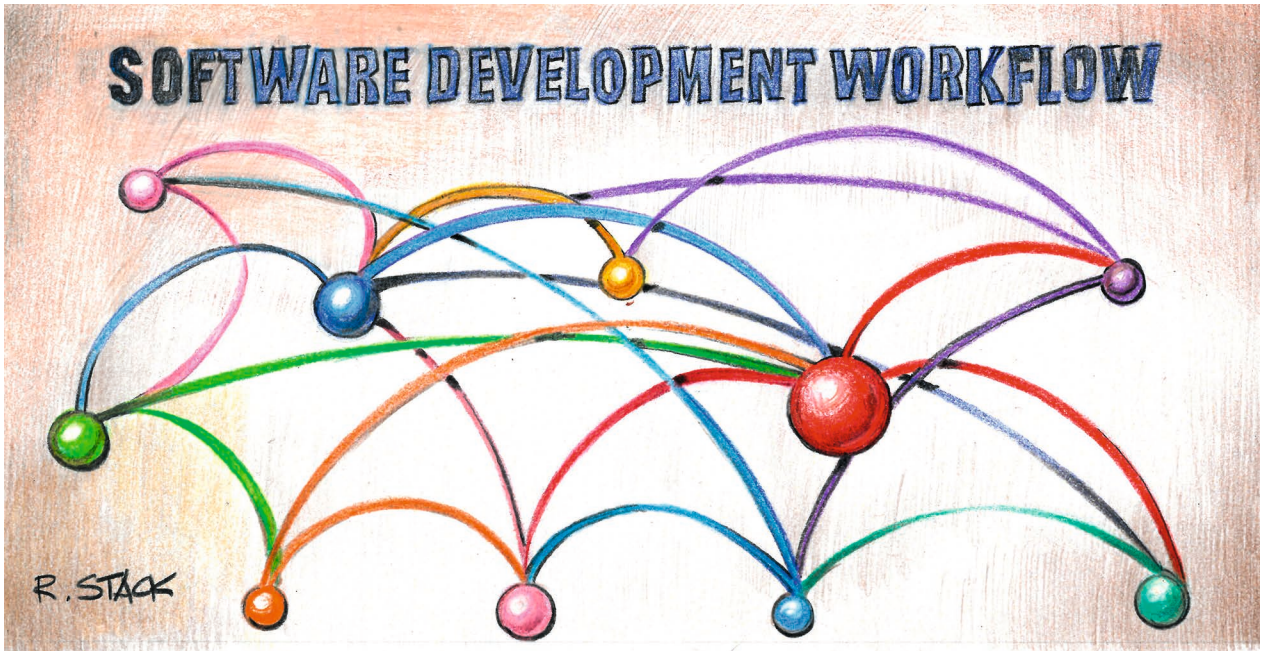
More Like Routing Airplanes Than Manufacturing Cars

At its core, the end-to-end software lifecycle is a business process that delivers value to end users. As such, the principles that James Womack and Daniel Jones listed in their summary of lean thinking very much apply:

> *Lean thinking can be summarized in five principles: precisely specify* value *by specific product, identify the* value stream *for each product, make value* flow *without interruptions, let the customer* pull *value from the producer, and pursue* perfection.[2]

Many lean concepts are relevant when we're shifting our thinking of flow from an assembly line to a network, such as small batch sizes and one-piece flow to minimize work in progress. However, to avoid overapplying manufacturing analogies—or worse, continuing down the path of the wrong mental model—we must more clearly define the key differences between managing the iterative and network-based value streams of software development and managing the linear value streams of manufacturing:

- *Variability.* Manufacturing has a fixed, well-defined set of variations for what will emerge from the end of the line, whereas new software features are open ended. Manufacturing needs to minimize variability; software development needs to embrace it.



- *Repeatability*. Manufacturing is about maximizing throughput of the same widget; software is about maximizing the iteration and feedback loops that drive innovation. We need repeatability at each stage of software delivery, such as reliable automated deployment, but we're trying to optimize more for flow and feedback than for repeatability.
- *Planning frequency*. Cars are designed up-front in waterfall cycles spanning years. Modern software organizations usually plan delivery using a two-week sprint cadence. This means we must design our value streams for frequent planning and change.
- *Creativity*. Manufacturing processes aim to achieve the highest feasible level of automation, which is facilitated by removing any creative and nondeterministic work from the production process. Creative work shifts to defining and tuning the production process itself. We see some of this in software. For example, defining the value stream from planning through deployment can be a bigger technical challenge than coding a new feature. However, even with the upcoming major advances in automation AI, we'll still be left with creative work and collaboration at each step of the software value stream.
- *Visibility*. What makes software so interesting is that it's not subject to physical manufacturing constraints, making it almost indefinitely malleable. This means that adaptation to a market's needs can happen at a dramatic pace. However, the lack of physical bits makes gaining visibility of flow and output a fascinating challenge, in contrast to how explicit this is in a car-manufacturing plant. Just as we had to invent microscopes to understand the inner workings of a physical world our eyes couldn't see, we now need a new set of tools to understand and manage intangible software value streams.

If you buy into the notion of software value streams forming a network and into the airplane traffic analogy, we must also consider what makes for robust, efficient networks, ranging from route optimization to flow control. For example, to optimize a network, we must consider the following:

- *Throughput*. We can measure a network's effectiveness as throughput—for example, how many passengers can be transported along certain routes. Where in an IT organization should we invest to gain the highest increase in overall throughput?

- *Latency*. Latencies are easy to deal with in a linear process, but what about the scenario in which one feature must be implemented by both a front-end and a back-end team? Does outsourcing to distant time zones increase latency? How do we measure overall network latencies and end-to-end lead times to increase time to market?
- *Resiliency*. A robust network assumes that nodes can fail while flow remains. How does this relate to a failed product or an insurmountable technical debt?

Finally, Metcalfe's law tells us that a network's value grows with its connectedness. If our value stream network has insufficient connectedness, is there any point in optimizing any particular stage? For instance, assume that no formalized feedback loop exists between operations and service desk staff working with an IT service management tool such as ServiceNow and developers coding in an agile tool such as JIRA and planning releases in Microsoft Project. In this case, will investing millions into continuous delivery produce any measureable business benefit? When a company's competitiveness in the market is on the line, ad hoc answers to these questions don't suffice. We need a more robust model for software value stream networks; this is something I'll explore in an upcoming column.

The Leipzig plant's bottleneck is the Paint Shop. Although the station employs cutting-edge high-voltage curing, changing the paint color and drying time in big ovens takes well over the 70 seconds I mentioned earlier. The resulting need to re-sort the cars by the desired color, batch them into the dreaded inventory, and reorder them into the just-in-time sequence is the incredible mechanical ballet that takes place above the plant's lunchroom in the ultimate tribute to value stream visibility.

As I walked out of the Leipzig plant, my perspective was transformed by the ingenuity, innovation, and managerial sophistication that the BMW Group has attained. It's now time for us to lay down the groundwork and new mental models that will let us attain this kind of precision, perfection, and flow for scaling how software is built. As long as we continue viewing software delivery as a linear manufacturing process, we'll remain stuck in the age before flight.

Acknowledgments

I'm grateful to Frank Schaefer and Rene Te-strote for arranging the visit and reframing my perspective by teaching me how the BMW Group has achieved such excellence in production.

References

1. E.M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*, North River Press, 2014.
2. J.P. Womack and D.T. Jones, *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, 2nd. ed., Productivity Press, 2003.

ABOUT THE AUTHOR

MIK KERSTEN is the Founder and CEO of Tasktop. Contact him at mik@tasktop.com or follow @mik_kersten.