Sonatype

Software Supply Chain Automation:

GOING BEYOND AGILE, LEAN AND DEVOPS

By Mike Hansen, Sonatype Senior VP of Products

WHITEPAPER

EXECUTIVE SUMMARY

While the term "supply chain" is not common in the lexicon of the software development discipline, it is a ubiquitous trait of modern software development life cycles. Software development has borrowed much from the learnings of Toyota. However, we still have yet to fully embrace the most fundamental element of *The Toyota Way*: The underlying supply chain where the extraordinary benefits of Toyota's perennial continuous improvement programs are evident.

Agile has brought the concepts of iteration, routine inspection and adaptation along with a focus on continuous improvement. Lean introduced the notion of the build-measure-learn loop. Most recently, DevOps, like the combination of agile and lean, has bridged the gap between the operational and development disciplines. However, nearly all optimization has been done around the production of software, without real consideration for the supplier dimension, primarily the open source ecosystem. This unchecked diversity of supply, where development teams typically choose whatever technology is deemed appropriate, as well as whatever version might be in vogue at the time of selection, has introduced a significant degree of unnecessary complexity. While empowering development teams to choose their own suppliers (i.e. software components) has enabled speed, increased throughput and unlocked innovation, there is much hidden inefficiency and risk.

Software supply chain automation will unleash the next level of application development efficiency, driving extraordinary increases in innovation, productivity and cost savings while enabling far greater control of risk.

Mike Hansen

Senior Vice President
of Products

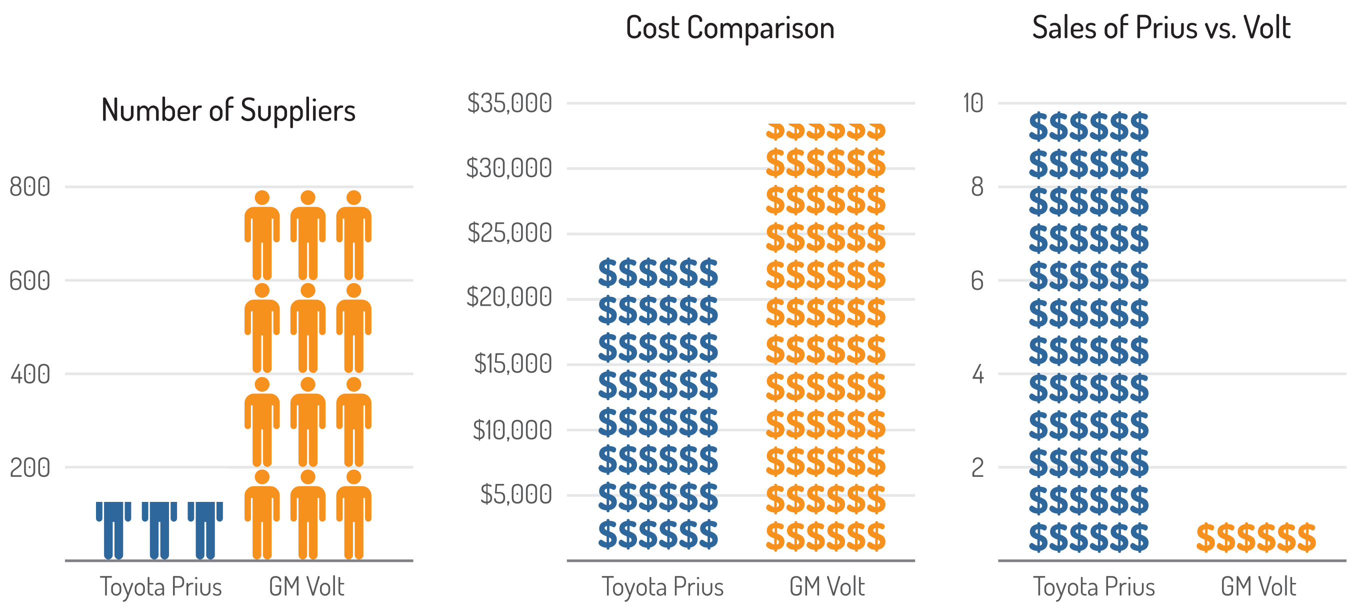
ABOUT MIKE HANSEN

As the Senior Vice President of Products, Mike brings nearly 20 years experience building and leading global product development organizations ranging from start-ups to the Fortune 100. Most recently Mike was Vice President of Development for Hanley Wood where he led development across five product lines, including the creation of advanced business analytics platforms for the residential construction industry. Prior to this, Mike was Vice President of R&D for Bantu where he led product development efforts that drove one of the worlds largest private instant messaging networks with the U.S. Department of Defense.

SUPPLY CHAIN OPTIMIZATION

To produce the Prius, Toyota leverages 125 total suppliers. To produce the Chevy Volt, General Motors uses over 800 suppliers. Ironically, 73% of the Prius is sourced from Toyota's suppliers, whereas only 46% of the Volt is outsourced. The visual comparison is somewhat striking.

Impact of supply chain optimization in automobile manufacturing



Supplier complexity impacts cost and quality, two important factors for competitive differentiation. The same is true of software.

That is an extraordinary difference in complexity for GM, leading to significantly higher overhead costs, variability in the flow of supplies and greater challenges in quality management to cite only a few of the inherent disadvantages. It is no wonder the Prius outsells the Volt better than 10 to 1. Interestingly, the software development practices of the majority of organizations today tend to have more in common with GM than you might think.

RECENT PROGRESS

Over the last 15 years, software development has benefited greatly from practices and tooling created as part of the agile, lean product development and DevOps movements. Agile has brought the concepts of iteration, routine inspection and adaptation along with a focus on continuous improvement. This has led to marked increases in development efficiency relative to approaches like waterfall. However, we occasionally ended up building the wrong thing.

Lean emerged in response to these product management challenges and has introduced the notion of the build-measure-learn loop, tightening the relationship between the product management and development disciplines along with the end users of the software. Borrowing from the routine inspection and adaptations of agile, lean has enabled the notion of "failing fast" providing teams with a more direct route to building the right solutions.

Most recently, DevOps, like the combination of agile and lean, has bridged the gap between the operational and development disciplines. This has shifted important operational thinking and practices to much earlier points in the development process and vastly reduced the time it takes to deploy new code, ultimately introducing significant gains in productivity.

The inclusion of these key disciplines involved in the production of software has led to significant increases in development effectiveness as silos have been torn down, communications streamlined and waste removed. However, despite the revolutionary nature of the changes in supply that have occurred throughout this same 15 year period, there has been little focus on the suppliers that now feed software development—primarily the open source software ecosystem.

Today's component supply chain

30+ critical or severe license or security issues in an average application	**43** versions of the same component are downloaded by an average organization	**60%** of organizations don't know what components are used or where

A Software Supplier Sea Change

At the turn of the century, the vast majority of a typical application was written by the organization producing it. Today, with the huge and rapidly expanding body of freely available open source software, the opposite is true. The largest portion of modern applications is typically open source, upwards of 80-90% in many cases. While the term "supply chain" is not common in the lexicon of the software development discipline, it is a ubiquitous trait of modern software development lifecycles.

This shift happened somewhat gradually and kind of snuck up on the software development discipline. Organizations continued to witness incremental gains in productivity by leveraging more and more of the diverse specialization that the open source ecosystem offers. Nearly all optimization has been done around the production of software, without real consideration for the supplier dimension. This unchecked diversity of supply, where development teams typically choose whatever technology is deemed appropriate, as well as whatever version might be in vogue at the time of selection, has introduced a significant degree of unnecessary complexity. In fact, the situation is arguably far worse than what were explicit and intentional choices made in building the Chevy Volt and other GM models. It leads to a dark matter version of technical debt creating a pernicious drag on overall development throughput.

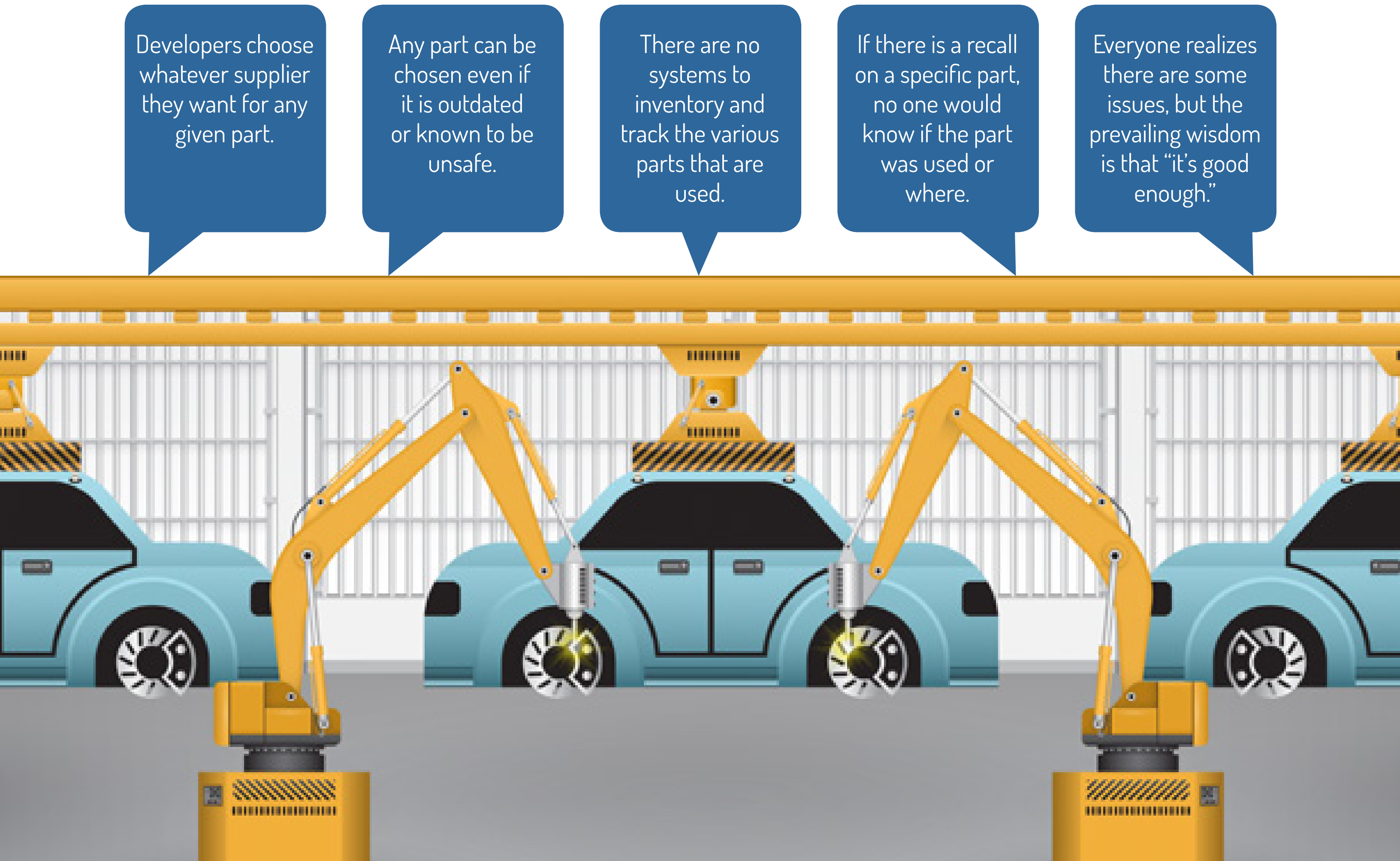
If Developers Built Cars

For a provocative thought experiment, let's transpose modern software development practices into the production of an automobile. In this universe, BigAuto teams independently choose whatever supplier they want for a car's transmission and whatever revision that might be available at the time. Word on the street is that a lot of people have looked at how the transmission was made and no one is complaining much, so it is presumed to be good.

This transmission also has numerous other parts inside it that were chosen from even more suppliers that supposedly have good reputations and produce good parts. There are ways to see what parts are being used but no one really bothers with that. Each team just wants a transmission and worrying about the internals is not part of their job descriptions. "Someone else tested the transmission, and we just need to test the car. If something is wrong that the supplier isn't willing to fix, they are happy to share the designs so we can fix it ourselves," the teams rationalize.

The team that is responsible for each model car is also given complete autonomy to choose pretty much any supplier they want for any part they want and change them whenever they want. This is done without any systems for tracking the corresponding activity. BigAuto sensed this might not be optimal and tried to institute a tight set of controls on their supply chain several times. However, without understanding the benefits that the current level of autonomy and unobstructed flow was yielding, each ill informed attempt crushed productivity and innovation and undermined their competitive position. Forced to contend with this generally ad hoc approach to sourcing, BigAuto was left without the ability to perform any kind of orderly recall, not managing the corresponding risks.

Due to a lack of software supply chain visibility, processes and automation...


Developers choose whatever supplier they want for any given part.
Any part can be chosen even if it is outdated or known to be unsafe.
There are no systems to inventory and track the various parts that are used.
If there is a recall on a specific part, no one would know if the part was used or where.
Everyone realizes there are some issues, but the prevailing wisdom is that "it's good enough."

The process used by BigAuto seems like it is working for the most part. There isn't too much news about people getting stranded, and there haven't been too many catastrophic outcomes. Everyone realizes there are some issues, but the prevailing wisdom is that it's good enough.

However, a few internal champions have recently started to think investments in modern tooling here could lead to significant competitive differentiation in conjunction with risk reduction.

Ripe for Optimization

This is basically the way most software is written today. It does in fact work and teams are more effective than they ever have been despite the abridged list of shortcomings depicted in the allegory, all of which have a very real analog in software development.

However, good is the enemy of great. While empowering development teams to choose their own suppliers (i.e. software components) has enabled speed, increased throughput and unlocked innovation, there is much hidden inefficiency and risk. Attempts to centralize control and decision making —often instituted in reaction to some previously unidentified risk manifesting—have failed to produce positive results and have often made situations far worse. This is an intuitive and commonly attempted response, yet incorrect. Instead, solutions must allow for the appropriate constraints (rules) to be established up front, the testing for conformance automated, and the information needed to remediate issues provided in the context of the existing tooling used by development and operations teams —trust but verify.

Like the waste cut out by agile, lean and DevOps practices, there is substantial waste that can be removed with automated tooling and process innovation throughout the software supply chain. Additionally, significant benefits are achievable regardless of the specific methodologies in use.

By providing visibility and empowering teams with the knowledge necessary to make the right choices as early as possible (data+rules+context), we can go even faster, further driving down costs and simultaneously managing or eliminating elective risk. The current state of the art is ripe for optimization.

Good is the enemy of great. While empowering development teams to choose their own suppliers (i.e. software components) has enabled speed, increased throughput and unlocked innovation, there is much hidden inefficiency and risk.

Supplier Bloat

Complexity is the enemy of many things but especially of efficiency and risk. It leads to symptoms like burdensome technical debt, quality issues and challenges in keeping a system secure. The magnitude of unnecessary complexity in modern software supply chains serving even a small portfolio of applications can be significant. In a large portfolio it can be enormous, and it typically is.

We worked with one organization that upon deeper inspection was found to be using 81 out of the 85 available versions of a popular web framework, spanning years of release history. While that is somewhat of an extreme case, this anti-pattern of greater than necessary version dispersion is actually quite common. After all, the average open source project releases a new version four times each year and it is fairly common practice to update only in response to a known issue impacting a particular application. Given the average application also has hundreds of open source components, any given day can see multiple new versions across the inventory of components in use. Also, issues that may be known to a particular open source project are often unknown to its consumers—i.e. the "no recall" problem mentioned earlier.

In addition to version dispersion, there is also technology dispersion, where different components that perform the same basic functions are used somewhat indiscriminately and often due to personal tastes or familiarity (e.g. logging frameworks, web frameworks). Viewed in the context of a single application, such dispersion isn't usually much of an issue, but notable impacts can become apparent with even as few as two applications. For example, subject matter expertise must be spread more thinly across the organization concentrating expertise locally within a team and potentially limiting future flexibility in resource allocation. The effective surface area of supply is also greater correlating directly with reduced efficiency and the potential for problems.

More recently, open source vulnerabilities have become exploit targets as commonly used components with known vulnerabilities are often the path of least resistance for attackers. These vulnerabilities are now getting catchy monikers like HeartBleed, ShellShock, POODLE and GHOST. Unfortunately, a larger than necessary set of suppliers with no real visibility of them leads not only to greater risk potential but incremental challenges and greater costs in securing the supplier dimension.

There are significant opportunities for performance gains as investments optimizing these supply chains can markedly improve efficiency and control risk, unleashing the full potential of an organization's capacity for innovation.

ANSWER: SOFTWARE SUPPLY CHAIN AUTOMATION

As software development evolves, we continue to create situations where complexity blocks further increases in productivity and innovation per unit of investment. These problems are often the impetus for rethinking and reshaping the status quo.

For instance, the introduction of high-level languages, object oriented programming, agile, lean, DevOps and dependency management are all examples of how we broke through barriers when reaching the limits of scale. Soon, the industry will begin to broadly demand solutions in response to software supply chain complexity. But there is no need to wait.

As the use of automation in areas of testing, build and deployment has provided significant performance benefits, so can further automation throughout the software supply chain. This is done by providing non-intrusive guardrails that consider the need for both autonomy and acceleration.

The manufacturing industry was transformed with three basic principles: Use fewer and better parts, limit the variety and quantity, and track where they are used. Software supply chain solutions address these principles using repository management, automated open source policy enforcement and up-to-date component data feeds. At a high level these solutions facilitate:

- **Quality** - Easily avoid known open source license issues and security vulnerabilities. Use better, up-to-date open source component types and versions.
- **Visibility** - Integrate component insight and policy automation into popular development tools.
- **Traceability** - Instantly identify out-of-date and defective components across the SDLC.
- **Remediation** - Effectively prioritize responses to new issues using a combination of visibility and context to rank risk.

Solutions that facilitate comprehensive software supply chain automation are poised to usher in the next wave of development productivity, on par with the gains possible with agile, lean and DevOps. In fact, the gains are likely to be even greater.