Velocity

in Software Engineering

FROM TECTONIC
PLATE TO F-16

TOM KILLALEA

Software engineering is necessary in all modern companies, but software engineers are expensive and in very limited supply. So naturally there's a lot of interest in the increase of velocity from existing software-engineering investments. In most cases, software engineering is a team activity, with breakthroughs typically achieved through many small steps by a web of collaborators. Good ideas tend to be abundant, though execution at high velocity is elusive. The good news is that velocity is controllable; companies can invest systematically to increase it.

Velocity compounds. It's also habit-forming; high-velocity teams become habituated to a higher bar. When velocity stalls, high contributors creatively seek ways to reestablish high velocity, but if external forces prolong the stall, soon they'll want to join another team that has the potential for high velocity. High velocity is addictive and bar-raising.

DIRECTION

Velocity is a function of direction and speed; you can't focus on only one of these. Of the two, direction is more easily overlooked. The most common reason that projects fail, however, is that the team was building the wrong thing. As Thomas Merton more eloquently put it, "People may spend their whole lives climbing the ladder of success only to find, once they reach the top, that the ladder is leaning against the wrong wall."

Amazon's Working Backwards product-development process seeks to compensate for the difficulty of determining direction (i.e., predicting product/market fit). The explicit artifacts of the Working Backwards process—a press release and an FAQ—have been widely discussed,⁸ and inherent in the process is the clear identification of who the customers are, then working backward from their needs to a product definition that would viably meet those needs. Frequently it's about paying attention to the voice of the customer, or, as Intuit cofounder Scott Cook put it, "Success is not delivering a feature; success is learning how to solve a customer's problem." Teams often lament that their customers use only 20 percent of what they shipped. Ideally, we would like to listen to customers and meet their needs while shipping only the 20 percent that most interests them.

Even for the best listeners and most visionary innovators, it's difficult to predict what customers need. Because there's some guesswork involved in choosing a direction, flexibility and course correcting become crucial. Flexibility might show up as openness, maximizing the rate of experimentation, learning quickly, reducing

Velocity is a function of direction and speed; you can't focus on only one of these.

commitment to any given plan, rapidly evolving products, and distinguishing between one-way (irreversible) and two-way (reversible) doors in decision-making. As to course correcting, Amazon CEO Jeff Bezos said, “If you are good at course correcting, being wrong may be less costly than you think, whereas being slow is going to be expensive for sure.”

SPEED

In 2003, at a time in Amazon's history when we were particularly frustrated by our speed of software engineering, we turned to Matt Round, an engineering leader who was a most interesting squeaky wheel in that his team appeared to get more done than any other, yet he remained deeply impatient and complained loudly and with great clarity about how hard it was to get anything done. He wrote a six-pager that had a great hook in the first paragraph: “To many of us Amazon feels more like a tectonic plate than an F-16.” That nobody responded defensively to this statement reflects well on the culture at Amazon at that time. Rather, the response was one of recognition: “He nailed us. That's us! A tectonic plate!”

Matt's paper had many recommendations that by now have become broadly adopted industry-wide, including the maximization of autonomy for teams and for the services operated by those teams by the adoption of REST-style interfaces between highly decoupled components, platform standardization, removal of roadblocks or gatekeepers (high-friction bureaucracy), and continuous deployment of isolated components. He also called for the extension of the definition of “complete” to include the

achievement of low ongoing maintenance costs, and for an enduring performance indicator based on the percentage of their time that software engineers spent building rather than doing other tasks. Builders want to build, and Matt's timely recommendations influenced the forging of Amazon's technology brand as "the best place where builders can build."

There have been many attempts to directly observe the velocity of software teams, but measuring such velocity is notoriously difficult. To compensate, companies can use engagement surveys to ask questions relating to velocity. Such surveys have become widespread, but too often they are limited to high-level measures of employee engagement and alignment with the company's goals. Some companies use their surveys as opportunities to determine whether they are great places for builders to build at high velocity, asking software engineers questions about how much time they spend actually designing and writing software, the adequacy of their tools, the effectiveness of their processes, and the impact of technical debt.

Software engineers can be cynical. Prior to embarking on surveys with questions such as these, companies should commit to acting on the results, so those actions positively impact both current velocity and future responses to such surveys.

AUTONOMY

The challenge of scaling software-engineering projects so that the addition of engineers results in greater throughput has been much discussed since the publication

of *The Mythical Man-Month*³ by Fred Brooks in 1975. Brooks examined the lack of increased throughput in software-engineering projects as more engineers are added and contrasted it with activities such as reaping wheat or picking cotton. He finds blame in the cost of coordination and communication.

Scalability can be improved by organizing into autonomous teams that have a high degree of internal cohesion around a specific and well-bounded context or area of responsibility. Teams, and the services that they're accountable for, expose APIs (application programming interfaces) to each other; in an ideal world no cross-team communication occurs since the APIs are all that are needed to interact with the business logic that is the responsibility of a team behind a remote service.⁵

The implementation details of the service are not typically shared, and there is no backdoor access to the data store on which a remote service depends. Coordination becomes unnecessary; even if a service needs to change in a backward-incompatible way, the new and old versions of the APIs will typically be made available for an overlapping period of time, so clients can migrate before the old version is deprecated. Round went so far as to argue for the measurement of crosstalk between teams in order to get an objective read on their level of independence.

Service owners can evolve and release changes at their teams' own pace, independent of and decoupled in time from other changes that may be taking place around them. Permissionless innovation, "the ability of others to create new things on top of the communications constructs

that we create,” as defined by IETF chair Jari Arkko,¹ can flourish. The work of identifying the seams between areas of responsibility, however, is challenging, and inevitably those seams will evolve over time. Perfect autonomy will always be illusory.

A set of software services evolves constantly, not unlike a living organism. New interfaces are released, whole services may split or merge, and individual services may go through significant redesign or deprecation. Ideally, teams within a company have a high level of autonomy and are “highly aligned, loosely coupled,” to quote from the Netflix Culture document.⁶

By extension through Conway’s law, the services operated by those teams should be independent. A lofty target is that any given team can implement 80 percent of the items in their backlog without needing any changes in the services on which they depend. Of the remaining 20 percent, a simple request to the owners of the remote service might result in a response indicating that the requested additional or altered interface makes sense, and it will be available by the time the requestor plans to start consuming it.

In the remaining cases the service owner may agree that the requested change makes sense and fits with the service’s roadmap, but its position on that roadmap is low compared with the priority that the requestor places on it. In such cases, the requestor might offer an “away team” to implement the requested change. An away team might consist of a pair of developers from the requestor team that temporarily joins the team that owns the service. The away team designs, tests, implements, and releases the

Even in a world of rapidly evolving requirements, it's OK for a team's well-ordered backlog to change constantly, provided the latest version is used for sprint planning.

requested change, all with stage-by-stage approval by the service owners who will be the long-term owners of the changes; when they're done they return to their "home team." A side effect of this away-team approach is cross-pollination of best practices, which can be particularly fruitful in a world where otherwise there is minimal communication between teams.

AGILITY

In an agile approach to product development, it's possible to establish a healthy balance between course correcting and optimizing for speed.

Even in a world of rapidly evolving requirements, it's OK for a team's well-ordered backlog to change constantly, provided the latest version is used for sprint planning. The team makes an explicit commitment to a set of tasks from the backlog and in return gets an uninterrupted window of protected time, a *sprint* in which to work with as much speed as possible. Following the conclusion of this blissfully uninterrupted and churn-free period, the sprint demonstration shows the commitments that the team met.

Before the cycle continues with the next sprint planning meeting using a course-corrected product backlog, the team holds a retrospective. This is an introspective session in which the team assesses the velocity that it reached and identifies ways to increase velocity in subsequent sprints. An honest retrospective, grounded in trust and self-awareness, can be used to figure out how to "sharpen the saw" before moving on to the next sprint.

FOCUS

Focus is necessary for achieving high velocity.

While Round was dreaming of a time when his team might be able to deploy their software to the Amazon website independently in less than a minute without needing to gain the approval of or even to notify anyone else, Andy Jassy was working on a vision document for a new business that would serve the needs of developers. Over time, Jassy's AWS [Amazon Web Services] vision would coalesce around the need to help developers avoid "undifferentiated heavy lifting."

Teams want to focus on solving their customers' problems and on implementing, at high velocity, the business logic that is uniquely their responsibility. The heavy lifting of procuring, provisioning, and operating data centers, servers, and networks is a burden that they would rather not bear. They also want to avoid if at all possible being blocked by any people or processes that they don't control (i.e., that lie outside of their own team). As Bezos put it, "Even well-meaning gatekeepers slow innovation."² Cloud computing is an enabler for permissionless innovation and for moving toward software architectures that have a marked absence of gatekeepers and in which gatekeeping controls, such as access controls and compliance assertions, are programmatically enforced.

CULTURE

A high-velocity team pays attention to fostering a culture that encourages the team's talent to flourish and deliver results. This is self-reinforcing: teams with a culture that enables high velocity tend to disproportionately attract

top talent. It's important to start with the presumption that people are talented, aligned with the mission, and want to work at high velocity. Some aspects of culture that positively impact velocity include diversity and inclusion, humility, trust, openness to learning, willingness to move with "urgency and focus,"⁷ ownership, autonomy, and willingness to collectively commit to delivering results.

ENABLEMENT

To achieve high velocity, it's necessary to invest in systems that enable engineers to work at speed and to maximize the percentage of their time spent working on their area of unique responsibility. The obvious starting points are the tools and processes that they use to build, integrate, and deploy their code, and those used to operate their code after it has been released to ensure that it meets its requirements for availability, reliability, performance, and security.

Less obvious is the need to enable observability; while a services-based architecture may bring the benefits of autonomy and velocity, failures across service boundaries can be much more difficult to troubleshoot. It's helpful if metrics collection and propagation, monitoring, alarming, and issue tracking are common across services. Observability capabilities should enable distributed tracing, facilitating the precise detection of critical signals and indicators, and the progressive refinement of the search space, leading to pinpointing the root cause.

EXPERIMENTATION

In the race to increase the rate at which they innovate,

many companies actively seek to reduce the cost of running experiments so that they can do more of them. A higher rate of experimentation can facilitate more frequent course correcting. It's worth noting that a high rate of experimentation can be viewed as a high volume of discarded ideas, dead code, and failures.

Successful teams embrace failures, knowing that their models may be incomplete and that most of the incorrect choices they make are easily reversible. Ed Catmull, cofounder of Pixar, said, "Failure, when approached properly, can be an opportunity for growth. But the way most people interpret this assertion is that mistakes are a

necessary evil. Mistakes aren't a necessary evil. They aren't evil at all. They are an inevitable consequence of doing something new and, as such, should be seen as valuable; without them, we'd have no originality."⁴

CONCLUSION

Software engineering occupies an increasingly critical role in companies across all sectors, but too many software initiatives end up both off target and over budget. A persistent myth is that effective delivery involves a perfect vision of what is needed combined with a plodding and unblinking march toward that

Related articles

➡ A Conversation with Werner Vogels
Learning from the Amazon
technology platform
Jim Gray

<https://queue.acm.org/detail.cfm?id=1142065>

➡ Conversations with Technology Leaders:
Erik Meijer

Great engineers are able to maximize
their mental power.

Kate Matsudaira

<https://queue.acm.org/detail.cfm?id=3092954>

➡ Meet the Virts

Virtualization technology isn't new, but it
has matured a lot over the past 30 years.

Tom Killalea

<https://queue.acm.org/detail.cfm?id=1348589>

vision, blind to all distractions or new information. A surer path is optimized for speed, open to experimentation and learning, agile, and subject to regular course correcting.

References

1. Arkko, J. 2013. Permissionless innovation. IETF; <https://www.ietf.org/blog/2013/05/permissionless-innovation/>.
2. Bezos, J. 2012. Annual letter to Amazon shareholders.
3. Brooks Jr., F. 1975, 1995. *The Mythical Man-Month*. Addison-Wesley.
4. Catmull, Ed. 2014. Creativity Inc. Random House.
5. Killalea, T. 2016. The hidden dividends of microservices, *acmqueue* 14(3); <https://queue.acm.org/detail.cfm?id=2956643>.
6. Netflix Culture. Netflix Jobs; <https://jobs.netflix.com/culture>.
7. A quick guide to Stripe's culture. Stripe; <https://stripe.com/us/jobs/candidate-info?a=1#culture>.
8. Vogels, W. 2006. Working backwards. All Things Distributed (Nov. 1); https://www.allthingsdistributed.com/2006/11/working_backwards.html.

Tom Killalea *was with Amazon for 16 years and now provides advice to technology-driven companies and sits on the boards of Akamai, Capital One, Carbon Black, and MongoDB.*

Copyright © 2019 held by owner/author. Publication rights licensed to ACM.