DORA
DEVOPS RESEARCH & ASSESSMENT

Forecasting The Value Of DevOps Transformations

Measuring ROI of DevOps

IT as a Value Driver and Innovation Engine

Traditionally, IT has been viewed as a cost center and, as such, was expected to justify its costs and return on investment (ROI) up front. However, IT done right is a value driver and innovation engine. Companies that fail to leverage the transformative, value-generating power of IT risk being disrupted by those who do. What has been missing is an analytical, data-driven framework to forecast the value and justify investment in DevOps transformations. This white paper helps to fill that gap. While the methodology is not exhaustive, it does outline important considerations.[1]

Using key metrics from the 2016 *State of DevOps Report*[2] and industry averages, we will forecast the value of implementing DevOps practices for High, Medium, and Low IT Performers—important characterizations that are described in this report. We will also show how you can use these metrics to calculate your productivity and estimate the potential ROI of your transformation initiative by increasing your capabilities and improving your IT performance.

The information presented is particularly well-suited for technology leaders and executives and/or finance partners to help drive technology transformation within an organization. You should be able to make a strong business case for undertaking a technology transformation by quantifying the costs and returns possible, using your own numbers and the industry benchmarks provided. This guide also provides insight into the gains possible as you continually improve and progress. If you are a Low or Medium Performer, take note of the benchmarks set by the High Performers, and be aware that the industry is improving every year. If you aren't improving, you will be left behind. If you are a High Performer, see how you compare to other High Performers and strive to continually improve and raise the bar, noting that we report the median benchmarks, and the industry continues to improve year over year, particularly among High Performers.[3]

SOFTWARE DEVELOPMENT SPEED AND STABILITY

High IT Performers

Realize the highest benefits from superior software delivery, such as low unnecessary rework and high employee satisfaction.

Medium IT Performers

Have the most to gain by burning down technical debt and optimizing for speed and value over cost.

Low IT Performers

Have the most opportunities for improvement by addressing low-hanging fruit and setting measurable goals.

Companies that fail to leverage the value-generating power of IT risk being disrupted by those who do.

IT and Organizational Performance

The *State of DevOps Reports*, coauthored by DORA, classify technical patterns of software development and delivery teams along the dimensions important to the core disciplines of DevOps. These include agility (or throughput) of development and reliability for operations. We captured agility by measuring how often code was deployed and how long it took code to be deployed. We also captured reliability by measuring mean time to restore service (MTTR) and change failure rate (i.e., how often changes to code or infrastructure need to be rolled back or hotfixed).

These measures were selected for several key reasons. Measures of agility capture the goals of developers well, and help to emphasize the importance of moving fast to deliver features to customers. Similarly, measures of reliability capture the goals of IT operations well, and help to emphasize the importance of reliable code and infrastructure The advantage of using both approaches is that these measures are in tension with one another, keeping teams from "gaming" the metrics, and providing a good holistic view of the overall ability of the team to develop and deliver software.

Statistical analysis shows that teams fall into distinct groups based on these measures: High, Medium, and Low IT Performers. (More detailed information can be found in the 2016 *State of DevOps Report*, but basic information is outlined in Table 1.[a]) High Performers show high achievement in terms of both throughput and stability, demonstrating good performance in software development and delivery without tradeoffs. That is, they apply principles and practices that enable them to improve both throughput and stability in tandem.

One important note about IT performance: Each team in an organization is on its own journey. Therefore, different teams within a single organization can—and often do—have different IT performance profiles. By identifying where your own team falls, you can see where you are in your own journey for continuous improvement and set goals for the future. In the context of this ROI exercise, you can use these IT performance profiles for data points from industry benchmarks if you do not have the data easily available within your own team or your own organization. For example, later in the report we will use percentage of unnecessary work in calculations of waste. If you don't have those numbers readily available for your own engineers, you can use the industry benchmarks provided and select the one based on the IT performance profile that best fits your current technical performance. However, we point out that there can be wide variation in these measurements and teams may vary greatly from these benchmarks; therefore, we strongly encourage teams to provide their own measurements.

[a] In addition to the 2016 report, we strongly recommend readers refer to the 2014 and 2015 *State of DevOps Reports*, which contain additional information and guidance on IT and organizational performance, and the technical, managerial, and cultural practices important for improvement work.

TABLE 1 Statistics from the 2016 *State of DevOps Report*

	High IT Performer	Medium IT Performer	Low IT Performer
DEPLOYMENT FREQUENCY *For the primary application or service you work on, how often does your organization deploy code?*	On demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every six months
LEAD TIME FOR CHANGES[b] *For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code commit to code successfully running in production)?*	Less than one hour	Between one week and one month	Between one month and six months
MEAN TIME TO RESTORE (MTTR) *For the primary application or service you work on, how long does it generally take to restore service when a service incident occurs (e.g., unplanned outage, service impairment)?*	Less than one hour	Less than one day	Less than one day[c]
CHANGE FAIL RATE *For the primary application or service you work on, what percentage of changes result in degraded service or subsequently require remediation (e.g., lead to service impairment, service outage, require a hotfix, rollback, fix forward, patch)?*	0 - 15%	31 - 45%	16 - 30%

High IT Performers were superior in all four measures at statistically significant levels. They deployed code most often and in the fastest cycles, and had the shortest MTTR when they did have failures, which were also the lowest.

Medium IT Performers were middle-of-the-road on most of the measures at a statistical level, lower than the High Performers group and higher than the Low Performers group. Change fail rate is the exception[d].

Low IT Performers were inferior in three of the four measures at statistically significant levels. They deployed code the least often and took the longest to release. They report the longest MTTR on average, but report a change fail rate lower than Medium Performers.

[b] We focus on the point of time from code commit to code deploy because the point when changes are introduced into version control represents the dividing point between different parts of the value stream.

The first phase of work includes design and development and is akin to Lean Product Development. It is highly variable and uncertain, often requiring creativity and work that may never be performed again, resulting in highly variable process times.

In contrast, the second phase of work, which includes testing and operations, is akin to Lean Manufacturing. It too requires creativity and expertise, but we expect testing and operations to be predictable, fast and mechanistic, with the goal of achieving work outputs with minimized variability (e.g., short and predictable lead times, near zero defects).

[c] Low IT Performers were lower on average (at a statistically significant level), but had the same median as Medium Performers.

[d] Our research doesn't conclusively account for why this is observed, but there may be an explanation, which is supported by other data: Low Performers spend more time on new work and less time on rework when compared to Medium Performers. We believe this greater amount of new work could be occurring at the expense of ignoring critical rework, thus racking up technical debt.

This would match scenarios where Low Performers see short-term efficiencies by ignoring technical debt, but at some point (as they start to mature and progress), they have to pay for their shortcuts. These inefficiencies are paid for as Low Performers undergo technology transformations, becoming Medium Performers. These Medium Performers then deal with greater complexity and resulting failure rates in their systems.

What Makes up ROI?

When organizations and technology leaders evaluate whether to undertake a technology transformation initiative with a focus on continuous improvement, they often ask about the return on investment. This exercise requires two sequences of numbers:

- **THE INVESTMENT,** or how much money and resources (converted to a dollar amount) will be devoted to the technology, process, training, and cultural improvements
- **THE RETURN,** or how much money and resources can be expected from their investment

While this white paper focuses on calculating the return aspect of ROI, remember to include costs beyond the technology acquisition in your investment calculations. Important considerations include training, lost productivity from learning and integrating a new technology or way of working, long-term maintenance costs, and any lost time spent re-architecting and replacing existing systems. Which costs are included in these investments will depend on the team and the organization, and where they are in their journey.

When calculating return, organizations have two categories of costs and resources they should always consider. The first is value-driven; the second is cost-driven.

Value-Driven Categories

High-performance organizations have demonstrated that a value-driven approach should take priority (or at least have equal importance with cost-reduction efforts), with a strong appreciation for market pressures and the ability to respond to those pressures—such as customer demands, the availability of new technologies, and competitor pressure—quickly and reliably, and without requiring heroics from their technology teams. Visionary technical leaders understand this and are notably optimizing for speed over cost, which is a significant shift in mindset (a strategy cited by DevOps leader Courtney Kissler[4]).

Value lost can include opportunity cost or the resources you are currently spending on non–value-added work (such as unnecessary rework and manual testing) but which you could be spending on value-added work (such as new features or additional automated testing).

Value lost from postponing new products or features is also a key concern, but is often skipped because it is difficult to estimate. This lost value can include the revenue and customers that an organization does not earn, but would have, if it had released software more quickly. This can be thought of as an opportunity cost, or cost of delay: the costs incurred from not releasing features in a timely manner.

The ability to more rapidly discover and deliver value to customers and your top line is a key benefit of the lean / agile paradigm, and is a true competitive advantage that remains relevant year over year and quarter over quarter. Furthermore, just because something is difficult to estimate doesn't mean it shouldn't be done. A high level of precision is not required in order to calculate return on investment, and we show how to calculate useful values for this number later on.

Cost-Driven Categories

In a cost-driven approach, the focus is on cost savings and efficiencies realized by implementing DevOps—for example, time savings from implementing a technology, time and cost savings from automating manual processes, etc. Cost savings, such as time and efficiency-based savings, are easy to identify and are often the only category used when justifying investments in IT. These can include the cost of downtime and the cost of manual vs. automated work. These savings can be achieved by adopting lean practices and continually improving your work to achieve efficiencies, such as eliminating sources of waste and unnecessary rework. Lean thinking is a strong foundation for improved economics and ROI arguments. However, considering these expenses exclusively is insufficient and rarely yields systemic, long-term gains—efficiencies that are realized in year one "no longer count" beyond year two as the organization adjusts to a new baseline of costs and performance. Worse, only focusing on cost savings signals to technical staff that they will be automated out of a job rather than being liberated from drudge work to better drive business growth, which has additional negative effects on morale and productivity.

EXAMPLE

In 2008, AOL was struggling with installs that were taking longer and longer to deploy to production. Gene Kim was working with Eric Passmore, who was the Senior Vice President of Global Engineering at AOL at the time. Gene says of the project, "It took [months] for the ops team to update the Linux kernel from 2.4 to 2.6, and the Dev teams required the multi-threading support that the 2.6 kernel provided. For the company, the absence of multi-threading support was as debilitating to the company as a code freeze." In other words, the development team had completed the new software features, but customers couldn't use it or get value from it until Ops finished the kernel upgrade.

Gene and Eric realized this was much more than a Dev or Ops problem – the delay of getting software functionality to customers was a business problem. This translated into real money lost for the business.

By improving the software development and delivery process, Eric and his team were able to improve deployment time from six hours to 45 minutes, removing bottlenecks in the process to allow AOL to deliver features and value to the customer faster[5].

Calculating Return Using Value and Cost

Let's see how ROI calculations break down in terms of both value and savings, keeping in mind that all costs that a business avoids are considered returns to the business.

We used conservative estimates for these calculations. Your numbers may be higher or lower based on your specific circumstances. We present the complete methodology for the calculations so you can calculate return using your own numbers. We also supply industry benchmarks and estimates to help you fill in any numbers you may not have on hand.

KEY IDEA **Costs avoided by a business are considered returns because any changes in costs and revenue are compared to a starting budget, which acts as a baseline for comparison.** *For example, if the baseline budget has accounted for $100 million in expenses for the year in IT spend, but through technology improvement initiatives that spend is reduced to $80 million, there is now an "additional" $20 million available that was not previously planned for. Therefore, this additional $20 million is a return to the business.*

Value Calculations

The best, most innovative companies undertake their technology transformations with an eye to the value they can deliver to their customers and the business in addition to the cost savings and efficiencies they can realize. However, many companies focus only on cost savings, because the concept is generally well-understood and commonly used to justify investments in technology.

EXAMPLE

"By installing a rampant innovation culture, we performed 165 experiments in the peak three months of tax season. Our business result? Conversion rate [in our customer acquisition funnel] is up 50%. Employee result? Everyone loves it, because their new ideas can make it to market."
—Scott Cook, *Founder Intuit*[6]

While a focus on cost savings is a good first step, it is not sufficient on its own. Cost savings can have good impacts early, but provide diminishing returns in future years. In addition, treating cost savings as valuable in and of itself is shortsighted. Pioneering companies that use technology to win in the market focus on value: They reinvest the returns they see from these savings to discover new customers and increase the value they deliver to existing customers. By leveraging superior software development and delivery capabilities, they are able to continuously deliver valuable new products and features, delighting customers, employees and investors.

Pioneering companies that use technology to win in the market focus on value.

We include two types of value in our calculations of return. The first is the value gained from reducing inefficiencies in work. This comes from continuous improvement initiatives, where teams reduce waste and increase efficiency. Many organizations categorize this type of improvement work as cost savings, but we make the case for this to be a value calculation instead. The second type of value included in our calculations of return is the value gained from new development work that contributes to revenue. These are discussed in detail below.

Value Gained from Unnecessary Rework Avoided per Year

The amount of time, and therefore money, spent and lost on unnecessary rework each year is a significant hit to productivity and the technical economy[8]. And yet, many organizations overlook this cost. All costs avoided represent returns to the business and can generate significant value. Because unnecessary rework represents work that can be avoided through improved processes, some organizations calculate gains in efficiency simply as cost savings. However, we point out that these cost savings are only realized if costs are fully avoided; that is, a reduction in workforce equivalent to the accumulated time savings. However, we strongly recommend organizations do not adopt this strategy, which has a negative impact on morale and organizational culture, can reduce efficiencies, and even incentivize workers to not improve their work processes. Because hiring and retention in the technical sector is a serious challenge right now, companies can instead recoup this time and reinvest it in the business, essentially getting “free” headcount. Retaining and training existing talent is more cost-effective, preserves institutional knowledge, and gives organizations an advantage by having a strong technical workforce that is engaged and continuing to learn.

By retaining your workforce and utilizing the time recovered by decreasing inefficiencies, organizations gain value through additional manpower hours. Therefore, we categorize this as the value gained from unnecessary rework avoided, and accumulate it per year. While the exact steps undertaken to improve processes and become more efficient will differ for each organization and even each team, using lean thinking and continuous improvement can enable teams to reduce waste and achieve efficiencies.

KEY IDEA **Recognize the value of labor hours recovered by reducing inefficiencies.**

Organizations are essentially getting additional capacity without having to recruit and hire – just by improving processes. Our research also shows that improving DevOps practices leads to higher employee satisfaction and employees in high-performing teams were 2.2x more likely to recommend their organization as a great place to work. This is a huge win where current competition for technical talent is fierce and costs of turnover far outstrip costs of retaining talent.[e]

EXAMPLE

charles SCHWAB

"[In the beginning], we brought prices down, down, down, so they are now essentially commodities. [Now...] to succeed in the business, we had to move in a direction of adding other value to the relationship with our clients."
—Charles Schwab[7]

Retaining existing talent is more cost-effective, preserves institutional knowledge, and gives organizations an advantage by having a strong technical workforce that is engaged and continuing to learn.

[e] A study by the Center for American Progress found that the typical cost of turnover is 21% of an employee's annual salary. https://www.americanprogress.org/wp-content/uploads/2012/11/CostofTurnover.pdf

To calculate Value Gained from Unnecessary Rework Avoided per Year, we use the following equation:

Cost of Unnecessary Rework Avoided per Year = Technical Staff Size x Average Salary x Benefits Multiplier x Percentage of Time Spent on Unnecessary Rework

Technical Staff Size

Organizations should include the total number of technical employees they have, since unnecessary rework affects everyone along the value chain, from development, QA, and test, all the way to operations. For illustrative purposes, we use the following groups for different-sized organizations:

- **For large organizations** whose primary business relies on software largely created in-house (e.g., financial services), we estimate 8,500 technical employees.
- **For medium to large technical organizations,** we estimate 2,000 technical employees.
- **For small to medium businesses and non-technical enterprises,** we estimate 250 technical employees.

Of course, when calculating the cost of unnecessary rework for your own organization, you should use the number of technical staff involved in software development and delivery at your company.

Average Salary

According to a 2015 report by Incapsula, the overall median salary for DevOps professionals is $105,600.[9] While this number increases for larger teams and varies based on geographic location and cost of living, we use this number in our calculations. When performing the calculations for your own purposes, use a typical salary appropriate for the technical staff in your organization.

Benefits Multiplier

Employee benefits such as insurance, vacation, and retirement cost money beyond base salary. While we have seen benefits multipliers range from 30% to 110% of salary costs (resulting in a benefits multiplier of 1.3 to 2.1), we use a conservative 1.5 multiplier for our calculations.

Percentage of Time Spent on Unnecessary Rework

For our purposes, we reference the reported percentage of time spent on unnecessary rework, on average, reported by 2016 *State of DevOps* survey respondents. This number represents the amount of time spent on non-value-added work – labor hours that are essentially wasted through inefficiencies.

Of course, not all unnecessary rework can be eliminated but teams should set goals to continuously improve on unnecessary rework. We suggest a goal of 20%, based on two sources. First, research reports that between 19% and 40% of code is reworked prior to final release[10]. Second, our own research in the 2016 *State of DevOps Report* finds that High Performers report 21% unnecessary rework. Therefore, 20% unnecessary rework appears to be a goal in line with the best performance studied.

For High IT Performers, the amount of unnecessary rework reported is 21%. Because we believe that even High Performers have improvements to make in their work and should be continuously striving for progress, we use the **1%** difference between reported rework and goal in our calculations. However, teams working on more static projects, such as mature project maintenance, may set more aggressive goals for unnecessary rework. While there is always some unplanned work to be done, catching errors early and having fast feedback loops helps to minimize this for High Performers. The best news here? By catching errors early, this group is also able to spend the most time on new work compared to all groups, reporting approximately 50% of their time spent on new work, such as design, new features, and new patch deployments.

For Medium IT Performers, the amount of unplanned rework reported by the industry is 32%. Subtracting the 20% goal gives us **12%** for our calculations. Medium Performers may not have the level of automated tests and other mechanisms in place to catch many defects as early as the High Performers, so they spend more time on unnecessary rework. Medium Performers report spending the least amount of time on new work (approximately 35%), likely because they are doing the hard work of cleaning up their technical debt. In fact, we see this pattern quite often: The journey from Low to High Performer involves the hard work necessary to catch up on the tech debt accumulated in the past and get to a point where you are catching defects early and often. Note that Medium Performers are still deploying more frequently and pushing code through the pipeline faster, and are doing it more reliably than Low Performers.

For Low IT Performers, the amount of unnecessary rework reported by the industry is 27%. Subtracting the 20% goal, gives us **7%** to use in our calculations. In all of these estimates of unnecessary rework, Low Performers are most likely to have immature and unreliable measurement practices, and therefore have less visibility into how much time they are spending on unnecessary rework. Therefore, we suggest this estimate may be low because Low Performers just don't realize how much time they are wasting. Based on the reported number, Low Performers spend more time on unnecessary rework than High Performers but less than Medium Performers. We believe this is because Low Performers are overwhelmed with the total amount of work at hand, or they may not care to keep up with the unplanned, reactionary work and may disregard it in favor of shipping new code at any cost. This is often the case when the business prioritizes new features and functions in order to gain a strategic position in the market, but this strategy is not sustainable. In fact, we see in our data that Low Performers spend approximately 40% of their time on new work— that's more time spent on new work than Medium Performers. While doing new work and delivering new features is good, ignoring defects and unnecessary rework is a losing strategy in the long run— technical debt adds up, increasing the costs of maintaining existing systems and reducing the rate at which new functionality can be delivered.[f]

The 2016 *State of DevOps Report* found:

High IT Performers spend 50% less time on unnecessary rework than Medium and Low Performers.

High IT Performers spend 66% more time on new work than their lower-performing peers.

Low IT Performers were inferior in three of the four measures at statistically significant levels. They deployed code the least often and took the longest to release. They report the longest MTTR on average, but report a change fail rate lower than Medium Performers.

[f] This post by Greger Wikstrand outlines how technical debt adds up over time and decreases throughput. http://www.gregerwikstrand.com/technical-debt-reduction/

Using the formula and inputs given above provides the following estimates for cost of unnecessary rework per year:

TABLE 2 Yearly Returns Possible from Cost of Unnecessary Rework Avoided

Technical Staff Size x **Average Salary** x **Benefits Multiplier** x **Percentage of Time Spent on Unnecessary Rework** = **Cost of Unnecessary Rework Avoided per Year**

	High IT Performer	Medium IT Performer	Low IT Performer
LARGE ORGANIZATION that relies on in-house software (8,500 technical staff)	8,500 staff x $105,000 salary x 1.5 benefits x 1% rework **= $13.4M**	8,500 staff x $105,000 salary x 1.5 benefits x 12% rework **= $160.7M**	8,500 staff x $105,000 salary x 1.5 benefits x 7% rework **= $93.7M**
MEDIUM TO LARGE TECHNICAL ORGANIZATION (2,000 technical staff)	2,000 staff x $105,000 salary x 1.5 benefits x 1% rework **= $3.2M**	2,000 staff x $105,000 salary x 1.5 benefits x 12% rework **= $37.8M**	2,000 staff x $105,000 salary x 1.5 benefits x 7% rework **= $22.1M**
SMALL TO MEDIUM BUSINESSES AND NON-TECHNICAL ENTERPRISES (250 technical staff)	250 staff x $105,000 salary x 1.5 benefits x 1% rework **= $393.8K**	250 staff x $105,000 salary x 1.5 benefits x 12% rework **= $4.7M**	250 staff x $105,000 salary x 1.5 benefits x 7% rework **= $2.8M**

While the Low Performers see lower yearly costs of unnecessary rework, this likely comes at a cost of letting technical debt accumulate. If true, this strategy will create problems in the future. In addition, Medium and Low Performers have greater unpredictability in their software development and delivery environments when compared to High Performers, which creates uncertainty. Managing this uncertainty translates into far greater overhead and unnecessary rework downstream that they are unable to foresee.

Undergoing a technical transformation with an eye toward continuous improvement in terms of building quality into the product results in a reduction of unnecessary rework and its associated costs. This is a waste-reduction strategy, and a key goal of the technical practices of continuous delivery. Note that these costs, if avoided, represent significant returns to the business. A reduction in these costs will be categorized as returns in our calculations shown in Table 2. Organizations may choose to realize these costs through headcount reduction, however adopting this strategy will have negative implications for morale and the gains cannot be utilized to create value; indeed, often the best people to make contributions and innovations to your product and technical environment are those who are already experts in it.

Similar business value calculations can be done for other improvement initiatives, such as automation, by using the percentage of time recovered through automation efforts across several initiatives, such as testing, infrastructure, workflow, and compliance. We don't include these calculations in our analysis because there are not yet good estimates of the savings and value available through automation improvement initiatives, but you should consider including these in your own calculations.

Potential Value Added from Reinvestment in New Features

While more difficult to forecast, lost revenue is just as important to consider when calculating savings and efficiency returns from technology investments, if not more so. These lost opportunity costs, if avoided, have the potential to continue adding value to your product and your portfolio year over year and catapult you over your competitors. The best organizations understand this, and include the value of technology transformation in their ROI calculations. However, since this concept is tricky to estimate and communicate, we have provided a framework to help you quantify it here. We use the ongoing value realized from delivering features to customers as our proxy. By delivering customer value, we hope to create the conditions to generate revenue or create our desired business value.

While delivering new features to customers brings revenue, not all features are winners: Only about one-third of well-designed, well-researched features in mature products deliver top-line value to organizations. The statistics are considerably worse for new products and business models[11]. Therefore, we see high performing companies such as Amazon leverage their ability to deploy frequently to run experiments in production. They do this so they can avoid building and maintaining features that don't deliver value. For our calculations, we base the revenue potential of new features on the current revenue of the business. This revenue potential represents potential return to the business from embarking on a technology transformation.

KEY IDEA **Leverage time recovered from reducing inefficiencies, and turn that into value by using it to generate revenue through new features for your customers.**

We calculate Potential Value Added from Reinvestment using the following equation:

Potential Revenue from Reinvestment = Time Recovered and Reinvested in New Features X Revenue Generating Features

[WHERE]

Revenue Generating Features equals

(Frequency of Experiments per Line of Business) X (Lines of Business in the Organization) X (Idea Success Rate) X (Idea Impact) X (Product Business Size)

Time Recovered and Reinvested in New Features

This is captured as the percentage of time recovered from reduction in unnecessary rework and reinvested in new features.[g] Frequency of experiments (below) assumes that all of a team's time is spent working on and delivering new features. While that may be possible for a new dedicated team, this analysis will focus on the gains possible through a technology transformation initiative and therefore only the portion of time that is recovered through improvement. This is an estimate, and each team's results may vary depending on their organizational and technical maturity.

We use the same methodology as above to estimate the amount of time that can be recovered by improving inefficiencies and use our stated goal of 20% rework.

These particular gains in value are only possible when the efficiencies realized from reduction in unnecessary rework are reinvested in the business. That is, by allowing your technology professionals to take their newly discovered free time and use it for work that is devoted to features that have the potential to create revenue for the business. If, for example, this recovered time is spent on work such as documenting processes or automating tests, the organization still benefits from the additional labor hours recovered (accounted for above), but it does not have the potential to realize revenue.

For this number, we also refer to the 2016 *State of DevOps* industry benchmark data.

High Performers are able to reduce unnecessary rework, and therefore redirect their efforts to value-add work, by 1%. (Reporting 21% originally, this group can realize a 1% increase in value-add work by redirecting technical staff's efforts to value add work by hitting the suggested goal of 20% of time spent on unnecessary rework.)

[g] Additional time may be recovered from the elimination of other types of non-value-add time, such as coordination time, transaction time, and queueing time. We do not include these categories because industry benchmarks were not available. Activities such as Value Stream Mapping can help teams identify and eliminate these inefficiencies.

- **Medium Performers** are able to redirect their efforts to value-add work by 12%. (This group reported 32% of their time spent on unnecessary rework; aiming for a goal of 20%, the difference is 12% of technical staff's time that can now be spent on value-add work.)

- **Low Performers** are able to redirect their efforts to value-add work by 7%. (This group reported 27% of their time spent on unnecessary rework; by reducing their unnecessary rework to 20%, they recover 7% of their time for value-add activities.)

Frequency of Experiments

The ability of an organization to test out features on customers through A/B tests or through other kinds of user research, both quantitative and qualitative, is a huge benefit to organizations seeking an objective test. However, this feedback from customers is much harder for software products if the team cannot deploy code regularly. That is, deployment frequency creates a constraint to their ability to experiment and test features with customers. Conservatively, we suggest an experiment frequency of one experiment per week per line of business, because this is the locus of experiments in organizations for this calculation. We refer to the 2016 *State of DevOps* industry benchmark data to verify if it is possible for each group:

- **High Performers** are able to deploy code on demand, or multiple deploys per day. Therefore, an experiment frequency of twice per day (or 730 times per year) is achievable. This is the number we use for our calculation.

- **Medium Performers** deploy between once per week and once per month. For this group, we use the high end of these two durations for experiments, or once every month, for our calculation.

- **Low Performers** deploy between once per month and once every six months. For this group, we use the high end of these two durations for experiments, or once every six months, for our calculation.

Lines of Business in the Organization

Organizations create and deploy software in strategic business units, or lines of business. Every line of business has a core software product or service that allows it to serve its customers. This core software product or service is the locus of experimentation in organizations. Large technology organizations have more products (which support lines of business), and therefore can run more experiments. There is a high amount of variability in how many lines of business each organization has, depending on industry and company structure. While you should insert your own numbers, for illustrative purposes, we use the following numbers for different-sized organizations:

- **For large organizations** whose primary business relies on software largely created in-house (e.g., financial services) with an estimated 8,500 technical employees, we assume 20 lines of business.

- **For medium to large technical organizations** with an estimated 2,000 technical employees, we assume 8 lines of business.

- **For small to medium businesses and non-technical enterprises,** with an estimated 250 technical employees, we assume 1 line of business.

Idea Success Rate

While the time spent on innovation and value-added work is generally a win to organizations, and definitely time better spent than unnecessary rework, not every piece of work will generate revenue. Numerous experiments have shown that only one-third of well-designed features improve key metrics,[12] so we use this in our calculations. Note that this metric applies to products with a strong, existing user base—for new products, the odds of building something that delivers value to the business may be considerably lower. Because this estimate may be optimistic for your context, use rates that accurately represent your environment.

Idea Impact

Each idea or feature has the potential to contribute to our bottom line. For our calculations, we assume that each successful idea or feature contributes an average of 1% to revenue[h] based on conversations with industry experts working on established web software properties that are undergoing incremental feature improvements and not significant changes. You will want to base your idea conversion on rates seen in your own products.

Product Portfolio Business Size

For many organizations, the revenue potential of new features is a function of the current revenue of the current product or business. We perform these calculations for a product portfolio with $100M in revenue.

While difficult to forecast, lost revenue is important to consider when calculating savings and efficiency returns from technology investments.

h In reality, this will be a distribution of percentages, where some ideas contribute 0.01% to revenue, while other ideas contribute 200% to revenue. For our calculations, we use 1% as an average contribution to revenue across all ideas.

Based on the formula and inputs above, we summarize the potential value added to the business by recovering time lost in unnecessary rework and reinvesting it in value-add activities (see Table 3). This can also be thought of as value lost from the business by not improving work processes and reinvesting in new features each year, as the best and most innovative companies do.

TABLE 3 Potential Value Added from Reinvestment in New Features[i]

Time Recovered and Reinvested in New Features x **Revenue Generating Features** = **Potential Revenue from Reinvestment**

(**Frequency of Experiments per Line of Business**) x (**Lines of Business in the Organization**) x (**Idea Success Rate**) x (**Idea Impact**) x (**Product Business Size**)

$100M Product Portfolio Business Size	High IT Performer	Medium IT Performer	Low IT Performer
LARGE ORGANIZATION that relies on in-house software (8,500 technical staff)	1% time recovered x 730 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $48.7M return**	12% time recovered x 12 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $9.6M return**	7% time recovered x 2 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $933K return**
MEDIUM TO LARGE TECHNICAL ORGANIZATION (2,000 technical staff)	1% time recovered x 730 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $19.5M return**	12% time recovered x 12 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $3.8M return**	7% time recovered x 2 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business **= $373K return**
SMALL TO MEDIUM BUSINESSES AND NON-TECHNICAL ENTERPRISES (250 technical staff)	1% time recovered x 730 experiments/year x 1 line of business x 1/3 success rate x 1% idea impact x $100M product business **= $2.4M return**	12% time recovered x 12 experiments/year x 1 line of business 1/3 success rate x 1% idea impact x $100M product business **= $480K return**	7% time recovered x 2 experiments/year x 1 line of business 1/3 success rate x 1% idea impact x $100M product business **= $47K return**

[i] These numbers may seem high for organizations not used to estimating returns based on value. We urge readers to consider current revenues and extrapolate potential returns from this; the results may surprise you.

[j] Bessemer Venture Partners' 2016 study shows this type of growth is possible: higher revenue multiples are achieved in cloud companies with faster revenue growth, an indication of high IT performance. http://www.slideshare.net/ByronDeeter/bessemers-10-laws-of-cloud-computing

Cost Savings Calculations

Savings calculations start with cost savings from time and effort avoided. From a business standpoint, any costs that are planned or usual expenses that are then avoided represent returns to the organization. That is, even though it is not new money coming into the business, it is categorized as such. We will highlight this throughout the report.

Any costs that are planned or expenses that are then avoided represent returns to an organization.

Cost of Downtime per Year

Application and infrastructure downtime carries significant costs, with a recent report by Steven Elliot and the IDC team suggesting yearly downtime costs can range from $1.25 to $2.5 billion dollars for a Fortune 1000 firm[13]. Downtime costs are highly variable depending on the nature of the business, with high-volume financial transaction businesses seeing much higher costs of downtime than a small brick and mortar business that simply maintains a web presence to notify customers of its operating hours. In addition, the ability to recover from an outage depends on the architecture. While we provide these calculations as an example, we strongly suggest that you calculate these costs with your own composite costs and IT architecture in mind.

Downtime numbers highlight the importance of a team's ability to restore service quickly and (as much as possible), avoid failure in the first place by designing resilient systems. An elimination or reduction in downtime costs represents returns to the business. This section identifies the amount of downtime that High, Medium, and Low IT Performers may be able to avoid each year.

KEY IDEA **Find a way to estimate outage costs, because when these are avoided, they can represent savings to a business.** *This section provides an example.*

To calculate Cost of Downtime per Year, we use the following equation:

Cost of Downtime per Year = Deployment Frequency x Change Fail Rate Percentage x Mean Time To Restore (MTTR) x Outage Cost

Deployment Frequency

The frequency with which a team deploys will affect how often it has a chance to introduce changes that can cause an incident. However, remember that less frequent deployments result in releasing much larger, more complex bundles of code into your production environment, making integration and support of that new code challenging and identification of any failures increasingly difficult.

We refer to our 2016 *State of DevOps* industry benchmarks for these statistics:

- **High Performers** are able to deploy on demand or multiple deploys per day. For this calculation, we will code this as 2 deploys per day, or 730 deploys per year. While two deploys per day may seem high, Etsy reports 80+ deploys per day and Netflix and Amazon deploy thousands of times per day, making our estimate quite conservative.
- **Medium Performers** deploy between once per week and once per month. For this calculation we use the average of these two, or 32 deploys per year.
- **Low Performers** deploy between once per month and once every six months. For this calculation we again used the average of the two, or 7 deploys per year.

Imagine your code base and infrastructure as a Jenga tower. Frequent releases are like adding a single Jenga piece onto the tower. It is manageable to support and easy to identify which addition caused an outage if there is one. We can also continue to strengthen and support the underlying infrastructure as we go, seeing how the small additions affect the tower. Infrequent releases are like adding a giant ball of hundreds of Jenga pieces, glued together, on top of your Jenga tower. That tower is much more likely to topple from that single large addition, and now you must figure out which piece or pieces in that ball of Jenga additions caused the outage.

Change Fail Rate

Every change introduced into production has a chance of causing a failure, incident, or service degradation. These interruptions in service must be addressed by the team, and have the potential to lead to larger outages. We refer to the 2016 *State of DevOps* industry benchmarks for these statistics, but suggest you use your own if they are available:

- **High Performers** report 0% to 15% of changes result in a degraded service or require remediation. For our calculation we will use the average of these two numbers: 7.5%.
- **Medium Performers** report 31% to 45% of changes result in a degraded service or require remediation. For our calculation we will use the average of these two numbers: 38%.

Low Performers report 16% to- 30% of changes result in a degraded service or require remediation[k]. For our calculation we will use the average of these two numbers: 23%.

Mean Time to Restore (MTTR)

We work with complex systems, and some failure and downtime is inevitable. The key is the ability to restore systems quickly. We again refer to the 2016 *State of DevOps* industry benchmarks for these statistics:

High Performers report being able to restore service in less than one hour when an outage occurs. Because high performers are so sensitive to outages and prioritize system uptime, we will use the midpoint of this range for our calculation: .5 hours.

Medium Performers report being able to restore service in less than one day when an outage occurs. For our calculation we will use a point in the range reported: 4 hours[l].

Low Performers report being able to restore service in less than one day when an outage occurs[m]. For our calculation we will be conservative and use the upper end of the range: 1 day.

Outage Cost

Outages are costly to organizations. However, the cost of outages is highly variable and depends, in particular, on the "blast radius" of the outage (has it taken out your entire infrastructure or just a single non-mission-critical application?) and the level of service degradation (is the whole system unavailable, or are we seeing a long tail in response times for certain kinds of requests?). You will need to gather your own data in order to refine these calculations.

At a low level of precision, a recent report from Stephen Elliot and the IDC team put the average hourly cost of an infrastructure failure at $100K, and the average hourly cost of a critical application failure between $500K and $1M[14]. Because DevOps is involved in developing and delivering core application functionality, we will use the numbers supplied for critical application failures. We will also remain conservative and use $500K in our estimates. It should be noted, however, that some businesses, such as retailers and financial institutions, report outage costs of millions of dollars per minute, so these costs should not be overlooked. We suggest you use your own average per-hour outage costs if they are available.

k Why are Low Performers reporting less change failures than Medium Performers? It could be that they are choosing to remediate less service incidents, allowing more work to pile up later. This is similar to the behavior we are seeing related to unnecessary work.

l We use four hours because Medium Performers were statistically faster than Low Performers on average, and because four hours follows a logarithmic curve often seen in performance improvement.

m We note that the Low Performers have a significantly lower average MTTR than the Medium performers. However, for the purposes of these calculations, we use the median MTTR, which was the same as the Medium Performers.

Using the formula and the numbers identified above, we calculate the cost of downtime per year to be:

TABLE 4 Returns Possible from Cost of Downtime Avoided

Deployment Frequency	x	Change Fail Rate Percentage	x	Mean Time To Restore (MTTR)	x	Outage Cost	=	Cost of Downtime per Year

High IT Performer	Medium IT Performer	Low IT Performer
730 deploys per year x 7.5% change fail rate x ½ hour MTTR x $500,000/hr outage cost **= $13.7M Downtime Cost per Year** **= $18.75K Downtime Cost per Deployment**	32 deploys per year x 38% change fail rate x 4 hours MTTR x $500,000/hr outage cost **= $24.3M Downtime Cost per Year** **= $760K Downtime Cost per Deployment**	7 deploys per year x 23% change fail rate x 24 hours MTTR8 x $500,000/hr outage cost **= $19.7M Downtime Cost per Year** **= $2.8M Downtime Cost per Deployment**

According to our model, High Performers have somewhat lower total downtime costs than Low Performers, but much lower per-deployment costs. In reality, these numbers should be lower, since High Performers will typically architect systems so that outages will be localized rather than systemic, and will result in service degradations rather than completely taking systems down. These important architectural characteristics substantially reduce the business impacts—and costs—of downtime. The solution to decreasing downtime costs is not to decrease deployment frequency but to decrease change failure rates, reduce MTTR, build resiliency into the system, and contain failures so that the system gracefully degrades rather than leading to cascading, global outages. The hidden costs of not deploying frequently include the lack of feedback from customers, a factor that gives the best companies the edge as they experiment, adjust, and continue to win in the market. Note that all downtime costs saved represent a return to the business; we categorize them as such in our calculations moving forward.

Adding it All Together

Now that we have identified the primary cost and value components of technology transformation and improvement work, we will combine them to find the potential returns of a technology transformation such as DevOps. Keep in mind that all costs saved represent a return to the business.

TABLE 5 Potential Return of Large Product Business ($100M)

Value of Rework Recovered + **Value Lost from New Features** + **Cost of Downtime** = **Potential Return**

$100M Product Portfolio Business Size	High IT Performer	Medium IT Performer	Low IT Performer
LARGE ORGANIZATION that relies on in-house software (8,500 engineers)	$13.4M value of rework recovered + $48.7M value lost from new features + $13.7M cost of downtime **= $75.7M return**	$160.7M value of rework recovered + $9.6M value lost from new features + $24.3M cost of downtime **= $194.6M return**	$93.7M value of rework recovered + $933K value lost from new features + $19.7M cost of downtime **= $114.4M return**
MEDIUM TO LARGE TECHNICAL ORGANIZATION (2,000 engineers)	$3.2M cost of rework + $19.5M value lost from new features + $13.7M cost of downtime **= $36.3M return**	$37.8M cost of rework + $3.8M value lost from new features + $24.3M cost of downtime **= $66M return**	$22.1M cost of rework + $373K value lost from new features + $19.7M cost of downtime **= $42.2M return**
SMALL TO MEDIUM BUSINESSES AND NON-TECHNICAL ENTERPRISES (250 engineers)	$393.8K value of rework recovered + $2.43M value lost from new features + $13.7M cost of downtime **= $16.5M return**	$4.7M value of rework recovered + $480K value lost from new features + $24.3M cost of downtime **= $29.5M return**	$2.8M value of rework recovered + $46.7K value lost from new features + $19.7M cost of downtime **= $22.5M return**

The yearly returns are much larger than most people estimate, illustrating that investments in technology—if done with true transformation and continuous improvement in mind—can deliver worthwhile results.

Now consider the additional gains available that we haven't included in the above calculations. One example is the value organizations could realize by reinvesting resources elsewhere: for example, taking the time saved by reducing unnecessary rework and reinvesting that time to new projects, creating value for the company. In this example, the calculations could be imagined as a straight investment, almost like "free work" or additional headcount. Alternatively, they could be analyzed as a capital investment, using the excess resources as an input in traditional reinvestment calculations, evaluated by hurdle rate and internal rate of return. In our discussions with forward-thinking companies, they do this exercise routinely, planning to leverage their gains in efficiency to realize innovation and value. While we won't include these calculations in this exercise, we encourage you to consider them in your own thinking.

Finally, the benefits to employees and organizational culture should not be ignored. Consider the morale improvement of teams spending less time in rework and more time in value-added development. Studies have shown that engaged, happy employees contribute to IT and organizational performance[15] and correlates to company growth[16]. Furthermore, it helps teams attract and retain additional good talent, creating a virtuous cycle.

Engaged, happy employees contribute to IT and organizational performance and correlates to company growth.

Demonstrating Return on Investment

Armed with a monetary representation for the return of your technology transformation, you are almost ready to demonstrate your return on investment. You also need to calculate the cost of investment in this transformation. While this white paper will not go into the details of these costs, remember to include the costs of:

- **Technology,** including acquisition, licensing, etc.
- **Training,** including the costs of productivity lost while your technical staff is in training (include the benefits multiplier)
- **Downtime while new technology and processes are learned** (including the cost of salary and benefits)
- **Consulting services**
- **Other related expenses,** such as refactoring or re-architecting

Sample Calculation

Using an investment value of $5.6M (which is inclusive of all acquisition, training, and personnel costs) for a large technical organization's technology transformation with a product line valued at $100M, we will demonstrate two methods: payback period and return on investment.

An example $5.6M investment breakdown could look like:

ITEM	SPEND AMOUNT
Consulting: assessment and roadmap development for technology transformation initiative	**$200,000**
Cloud subscription services	**$65,000**
Automation software	**$1,000,000**
SREs and DevOps engineers to augment team (5 x $180,000 x 1.5 benefits multiplier[n])	**$1,350,000**
Training and DevOps/Kanban/ agile coaching for teams	**$200,000**
Dedicated time and resources of existing workforce (equivalent to 18 FTE x $105,000 x 1.5 benefits multiplier)	**$2,835,000[o]**

RETURN	$75.7M, Rounded

(Large technical organization with $100M product business)

[n] This calculation uses a higher salary number than that used earlier because hiring and retention is a challenge for organizations, and finding senior SREs and DevOps engineers will likely require paying a premium.

[o] This number may seem disproportionately high, but it is likely much higher; technology transformations rely heavily on labor. Research from the 2000s suggests the cost of labor is 2x the cost of technology[17]. In a more recent example, Forrester's Cloud App Migration Cost Model also finds that labor costs far exceed service and infrastructure costs[18].

Patterson: Patterson, D. (2002, Nov 3-8, 2002). A Simple Way to Estimate the Cost of Downtime. Paper presented at the Large Installation System Administrator's Conference (LISA '02), Philadelphia, PA.

Forrester: https://www.forrester.com/report/Brief+The+Cost+Of+Migrating+An+Enterprise+Application+To+A+Public+Cloud+Platform/-/E-RES132801

Payback Period

One of the simplest methods of talking about return on investment is payback period. Simply put, this method asks how long an investment takes to pay itself back in terms of profit or savings. In terms of our calculations, how long it takes our investment to cover the returns[p]. The output of the equation is in years. Given the example above, we are considering an investment that will cost $5.6M and will generate $75.7M per year in returns. If we assume equal cash flow each year, we calculate the payback period by dividing the investment by the returns:

$$\text{Payback Period} = \frac{\text{Investment}}{\text{Returns}} = \frac{\$5{,}600{,}000}{\$75{,}741{,}666.67} = .074 \text{ Years}$$

The payback period is .074 years, or about 27 days, meaning this investment will "pay itself back" very quickly. In this calculation, faster is better. Payback period is considered useful from a risk analysis perspective because it reveals how long the investment will pose a risk to the firm. It is particularly relevant in industries such as technology where investments can become obsolete quickly. The benefit of this analysis is that it is easily understood and communicated. The reader should note that this method for calculating payback period assumes that cash flows are equal; if they are accelerated or uneven, your calculations should take that into account.

p Payback period ignores the time value of money and reinvestment and is often done "on the back of a napkin." It is generally done with cash based calculations but can also be used with all investment and returns for estimation purposes, as we show here.

Profitability

Return on investment calculates the profitability of a project and reports the return as a percentage of the investment[q]. The output of the equation is a ratio. This ratio is meaningful to investors and people in business who compare it to other investments.

Given the example above, we are considering an investment that will cost $5.6M and will generate $75.7M per year in returns (rounded). To calculate the return on investment, we subtract the investment from the return and divide that number from the investment:

$$\text{ROI} = \frac{\text{Return - Investment}}{\text{Investment}} = \frac{\$75{,}741{,}666.67 - \$5{,}600{,}000}{\$5{,}600{,}000}$$

The ROI for this investment is 12.525. You may be asking: Is this a good ROI? That depends on what an organization considers "good" and what it is comparing it to. However, we can say that the organization made ~$12.53 for every dollar it invested in its technology transformation initiative. You can also think of an ROI ratio in comparison to other investment assets: What kind of returns are available from investments outside the firm, such as stocks and bonds? While investments in a diversified stock portfolio are less risky, investments in your own company that have a large ROI can be a good way to increase your opportunity for returns. That is, if you can achieve similar returns from investing in your own technology transformation (or even better returns, which is likely in the example above), and those internal investments will also help you win in the market, why wouldn't you choose that strategy?

[q] ROI is another estimation method that ignores time value of money.

Technology Transformation Pays Off

As we've demonstrated, undertaking a technology transformation initiative can produce sizeable returns for any organization. Of course, when undertaking any cost-estimation exercise, there are risks that costs may be over- or under-estimated, as well as risks that returns may not be realized in the expected timeframe or that market conditions may shift, leading to changes in customer preferences or interest rates. That said, cost and value estimations are still worthwhile, providing team members and leadership a basis for decision making. For each type of IT performer, there are lessons to be learned.

The data suggests that Medium Performers have the most to gain by continuing to burn down technical debt and optimize for speed and value over cost. We urge Medium Performers to continue this work and not reach a point where, after a time of doing hard work, they think they are not making progress and shift back to their old ways, settling for short-term improvements and building up technical debt again. Medium Performers must continue making progress toward operational efficiency, implementing smart technical practices of continuous delivery such as continuous integration, automated tests, and version control to achieve sustained high performance in both throughput and stability.

Low Performers face a paradox. On the one hand, they lag well behind competitors, often due to complex legacy systems and conservative cultures. However, in these organizations there is typically plenty of low-hanging fruit, provided the political will exists to seize it. As with all initiatives, it's essential to set measurable business goals for your initiatives and work with stakeholders throughout the organization to experiment with bold ideas to achieve results. Start with teams that have the capacity and desire for change and have support at the senior leadership level, and look for quick wins that will deliver measurable results in weeks, not months, even if the impact is limited.

For any team starting a technology transformation, remember that many improvement initiatives follow a "J-curve," so be prepared for early disappointments. The J-curve is the performance hit teams often experience when a new member joins a team or when new processes are put in place and there's an initial negative impact on performance before things get better. As Julia Wester notes, the size of the change often affects the depth of the negative impact[19]. A technology transformation initiative is a big change, so don't give up if (realistically, when) there is an initial hit to performance or productivity. This pattern is seen in our data, with the path taken from low performance to high performance taking a dip through higher rates of unnecessary rework as teams tackle their technical debt. When teams stick with it, they are rewarded with superior software development and delivery capabilities, and the lowest rates of unnecessary rework, on par with those reported in other studies.

High Performers are doing well, but still have room for improvement in many areas. In fact, the 2016 *State of DevOps Report* shows that High Performers continue to improve their technical, managerial, and cultural practices, as well as their IT performance year over year. Indeed, this is the hallmark of High Performers, and it pays off for their organizations in terms of market share, productivity, and profitability. In order to stay competitive, High Performers must continuously push themselves to improve, or they will be overtaken by their competitors—and the data shows it can happen in just a year's time.

For more information on what steps you can take and what technical practices you should implement to truly improve your IT and organizational performance, visit our website at www.devops-research.com

Authors

Dr. Nicole Forsgren is an IT impacts expert best known for her work with tech professionals and as the lead investigator on the largest DevOps studies to date. She is a consultant, expert, and researcher in knowledge management, IT adoption and impacts, and DevOps. Nicole is the CEO and Chief Scientist at DORA. In a previous life, she was a professor, sysadmin, and hardware performance analyst. She has been awarded public and private research grants (funders include NASA and the NSF), and her work has been featured in various media outlets and several peer-reviewed journals and conferences. She holds a PhD in Management Information Systems and a Masters in Accounting.

Jez Humble is co-author of ***The DevOps Handbook, Lean Enterprise,*** and the Jolt Award winning ***Continuous Delivery.*** He has spent his career tinkering with code, infrastructure, and product development in companies of varying sizes across three continents, most recently working for the US Federal Government at 18F. He is currently researching how to build high-performing teams at DevOps Research and Assessment LLC and teaching at UC Berkeley.

Gene Kim is a multi-award winning CTO, researcher, and author. He has been studying high-performing technology organizations since 1999. He is the founder of Tripwire and served as CTO for thirteen years. He has co-authored four books including ***The Phoenix Project: A Novel About IT, DevOps,*** and ***Helping Your Business Win (2013), The DevOps Handbook (2016),*** and ***The Visible Ops Handbook (2004).***

About DORA

DevOps Research and Assessment (DORA), founded by Dr. Nicole Forsgren, Jez Humble, and Gene Kim, conducts research into understanding high performance in the context of software development and the factors that predict it. DORA's research over the last four years and over 23,000 data points serves as the basis for a set of evidence-based tools for evaluating and benchmarking technology organizations.

Are you evaluating your own technology transformation? We offer assessments directly to organizations. **Request a demo at sales@devops-research.com**

Are you a consultancy? We offer a turnkey assessment, backed by trusted names in DevOps. To learn more about a co-branded scorecard and partnership opportunities, **email sales@devops-research.com**

Learn more at **www.devops-research.com**

Acknowledgments & References

The authors thank Puppet, Inc. for its collaborative support on the 2016 *State of DevOps Report*, which inspired this paper and provided many industry benchmarks for the analysis. We would like to thank Alanna Brown in particular for her thoughtful guidance and insights throughout the process.

The authors also thank reviewers for their thoughtful reading and insights, which made this manuscript much stronger: Katharine Wright Adame, Michael Blaha, Jeff Gallimore, Sam Guckenheimer, Courtney Kissler, Scott Prugh, Walker Royce, Mark Schwartz, and Nate Shimek.

1 Kim, G. (n.d.). The Amazing DevOps Transformation of The HP LaserJet Firmware Team (Gary Gruver). Retrieved from http://itrevolution.com/the-amazing-devops-transformation-of-the-hp-laserjet-firmware-team-gary-gruver/

2 Puppet, Inc., & DevOps Research and Assessment, LLC. (n.d.). 2016 State of DevOps Report (Rep.)

3 Puppet, Inc., & DevOps Research and Assessment, LLC. (n.d.). 2016 State of DevOps Report (Rep.)

4 DevOps Enterprise Summit 2014. (2014, October 29). DOES14 – Courtney Kissler – Nordstrom – Transforming to a Culture of Continuous Improvement. Retrieved from https://www.youtube.com/watch?v=0ZAcsrZBSlo

5 Earnshaw, A. (2013, July 18). DevOps Solves Business Problems: Gene Kim's Top Aha Moments. Retrieved from https://puppet.com/blog/devops-solves-business-problems-gene-kim%E2%80%99s-top-aha-moments

6 Divine, C. (2011, April 20). Leadership in an Agile Age: An Interview With Scott Cook. Retrieved from https://web.archive.org/web/20160205050418/http://network.intuit.com/2011/04/20/leadership-in-the-agile-age/

7 Chron 200 / Interview with CEO of the Year Charles Schwab. (2007, April 9). Retrieved from http://www.sfgate.com/business/article/Chron-200-Interview-with-CEO-of-the-Year-2603664.phphttp://dspace.mit.edu/bitstream/handle/1721.1/83541/REP_0101_Ippo.pdf?sequence=1

8 Ippolito, B., & Murman, E. (2001, December). Improving the Software Upgrade Value Stream. In 43rd AIAA Aerospace Sciences Meeting and Exhibit (p. 1252).

9 Harpaz, D. (2015, April 23). DevOps Salary Survey 2015: Facts About DevOps Careers and Salaries. Retrieved from https://www.incapsula.com/blog/devops-salary-survey.html

10 Morozoff, E. (2009, September 4). Using a Line of Code Metric to Understand Software Rework. Retrieved from http://ieeexplore.ieee.org/document/5232799/

11 Kohavi, R., Crook, T., Longbotham, R., Frasca, B., Henne, R., Ferres, J., Melamed, T. (2009). Online Experimentation at Microsoft. Retrieved from http://ai.stanford.edu/~ronnyk/ExPThinkWeek2009Public.pdf

12 Kohavi, R., Crook, T., Longbotham, R., Frasca, B., Henne, R., Ferres, J., Melamed, T. (2009). Online Experimentation at Microsoft. Retrieved from http://ai.stanford.edu/~ronnyk/ExPThinkWeek2009Public.pdf

13 Elliot, S. (2014). DevOps and the Cost of Downtime: Fortune 1000 Best Practice Metrics Quantified. Retrieved from http://info.appdynamics.com/DC-Report-DevOps-and-the-Cost-of-Downtime.html

14 Shimel, A. (2015, February 11). The real cost of downtime. Retrieved from http://devops.com/2015/02/11/real-cost-downtime/.

15 Puppet, Inc., & DevOps Research and Assessment, LLC. (n.d.). 2016 State of DevOps Report (Rep.)

16 Reichheld, F. F. (2003, December). The One Number You Need to Grow. Retrieved from https://hbr.org/2003/12/the-one-number-you-need-to-grow

17 Patterson, D. (2002, Nov 3-8, 2002). A Simple Way to Estimate the Cost of Downtime. Paper presented at the Large Installation System Administrator's Conference (LISA '02), Philadelphia, PA

18 Rymer, J. R., Bartoletti, D, Martorelli, B, Mines, C, Tajima, C. (2016, March 9). Brief: The Cost Of Migrating An Enterprise Application To A Public Cloud Platform. Retrieved from https://www.forrester.com/report/Brief The Cost Of Migrating An Enterprise Application To A Public Cloud Platform/-/E-RES132801

19 Wester, J. (2016, February 6). Why improvement initiatives fail. Retrieved from http://www.everydaykanban.com/2013/02/26/why-improvement-initiatives-fail/

White Paper layout and design by Bullseye Creative, Seattle • BullseyeCreative.net