# The ROI of DevOps Transformation

# 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 Accelerate: State of DevOps Report[2] and industry averages, we will forecast the value of implementing DevOps practices for Elite, 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 in the form of investing in DevOps tooling 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, Medium or High Performer, take note of the benchmarks set by the Elite Performers, and be aware that the industry is improving every year. If you aren't improving, you will be left behind. If you are an Elite Performer, see how you compare to other Elite 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 Elite Performers.[3]

## Software Development Speed and Stability

↑ **Elite IT performers** Realize the highest benefits from superior software delivery, and are delivering software at the highest levels. They experience the most value-add time out of their days and spend the least amount of time doing non-value-add work of all groups.

↗ **High IT performers** Still have room for improvement while being statistically better than medium performers. Excelling at all aspects of throughput and stability, yet must continue to improve in these areas to gain the most benefits from improved IT performance.

→ **Medium IT performers** Have the most to gain by burning down technical debt and optimizing for speed and value over cost. Doing well in terms of stability but fall behind high performers when it comes to speed.

↓ **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.

# Contents

# 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 stability 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 need a period of 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: Elite, High, Medium, and Low IT Performers. (More detailed information can be found in the 2019 State of DevOps Report, but basic information is outlined in Table 1.[a]) Elite Performers show the highest achievement in terms of both throughput and stability, demonstrating superior 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 the 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 2019 report, we strongly recommend readers refer to the 2017 and 2018 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 2019 Accelerate: State of DevOps Report

| Aspect of software delivery performance | Elite IT performers | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|---|
| **Deployment Frequency:** For the primary application or service you work on, how often does your organization deploy code to production or release it to end users? | On-demand (multiple deploys per day) | Between once per day and once per week | 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 committed to code successfully running in production)? | Less than one day | Between one day and one week | 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 or a defect that impacts users occurs (e.g., unplanned outage or service impairment)? | Less than one hour | Less than one day[c] | Less than one day[c] | Between one week and one month |
| **Change Fail Rate:** For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)? | 0-15%[d,e] | 0-15%[d,f] | 0-15%[e,f] | 46-60% |

[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).

Medians reported because distributions are not normal.

All differences are significantly different based on Tukey's post hoc analysis except where otherwise noted.

[c,d,e] Means are significantly different based on Tukey's post hoc analysis; medians do not exhibit differences because of underlying distributions.

[f] Means are not significantly different based on Tukey's post hoc analysis.

↑ **Elite 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 at less than one hour.

↗ **High IT performers** are performing better than most of their competitors. They are deploying code and fixing their errors very quickly but must continue to improve in order to match the level of their Elite counterparts.

→ **Medium IT performers** represent the largest proportion of IT performers as low performers continually improve and high performers succumb to the increased complexity of the industry. Medium performers must continue to improve in areas of throughput, but are doing well in terms of stability.

↓ **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. Low performers have the most to gain financially from advanced IT improvements.

# 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

Elite-performing 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.

# Aol.

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].

## 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.

# 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.

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.

## INTUIT

"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]

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.

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

# 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[7]. 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.

"[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[8]

## 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.[g]*

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.

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

$$\text{Cost of Unnecessary Rework Avoided per Year} = \text{Technical Staff Size} \times \text{Average Salary} \times \text{Benefits Multiplier} \times \text{Percent of Time Spent on Unnecessary Rework}$$

---

| Cost of Unnecessary Rework Avoided per Year | = | Technical Staff Size | × | Average Salary | × | Benefits Multiplier | × | Percent 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.

| Cost of Unnecessary Rework Avoided per Year | = | Technical Staff Size | × | Average Salary | × | Benefits Multiplier | × | Percent of Time Spent on Unnecessary Rework |
|---|---|---|---|---|---|---|---|---|

## Average Salary

According to a 2019 report by Glassdoor, the overall median salary for DevOps professionals is $143,000[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.

| Cost of Unnecessary Rework Avoided per Year | = | Technical Staff Size | × | Average Salary | × | **Benefits Multiplier** | × | Percent of Time Spent on Unnecessary Rework |
|---|---|---|---|---|---|---|---|---|

## 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.

| Cost of Unnecessary Rework Avoided per Year | = | Technical Staff Size | × | Average Salary | × | Benefits Multiplier | × | **Percent of Time Spent on Unnecessary Rework** |
|---|---|---|---|---|---|---|---|---|

## 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 2018 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 18%, 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 2018 Accelerate: State of DevOps Report finds that Elite Performers report 19% unnecessary rework. Therefore, 18% unnecessary rework appears to be a goal in line with the best performance studied.

For **Elite IT Performers**, the amount of unnecessary rework reported is 19%. While Elite Performers demonstrate the gold standard of the industry, there is always room for improvement. So we use **1%** as their goal for unplanned work; the difference between the amount of rework reported and the goal of 18% rework. They perform the best in every metric but still have reactive unplanned work because of interruptions, errors and reactions to bugs in code. Still Elite performers get the most value-add time out of their days and are spending the least amount of time doing non-value-add work of all groups.

For **High IT Performers**, the amount of unnecessary rework reported is 19.5%. Because we believe that High Performers still have improvements to make in their work and should be continuously striving for Elite status, we use the **1.5%** 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 about 10% more time on new work compared to medium and low performers, reporting approximately 50% of their time spent on new work.

For **Medium IT Performers**, the amount of unplanned rework reported by the industry is 20%. Subtracting the 18% goal gives us **2%** 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 Elite or High Performers, so they spend more time on unnecessary rework. This is likely to do to the time consuming work Medium Performers must implement to clean their technical debt. Note Medium Performers are still deploying more frequently and pushing code through the pipeline fast, and are doing it more reliably than Low Performers.

For **Low IT Performers**, the amount of unnecessary rework reported by the industry is 20%. Subtracting the 18% goal, gives us **2%** 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 most of their time on unnecessary and unplanned work, with only about 30% of time spent on new work. **The lowest of all other groups**. Low Performers are overwhelmed with the total amount of work at hand, and they may not care to keep up with the unplanned, reactionary work - disregarding 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. 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[h]. The journey from Low to Elite 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.

---

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

The 2019 Accelerate: State of DevOps report found:

↑ Elite IT performers have nearly tripled, growing from 7% to 20%, showing that excellence is possible - it just requires execution.

↗ High IT performers similar to their Elite counterparts, have grown year over year and report superior availability, which is significantly correlated with software delivery performance profile.

→ Medium IT performers are doing well in terms of stability, on par with the High Performers, but fall behind in speed of delivery.

↓ Low IT performers were inferior in all 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.

| Cost of Unnecessary Rework Avoided per Year | = | Technical staff size | × | Average salary | × | Benefits multiplier | × | Percent of time spent on unnecessary rework |
|---|---|---|---|---|---|---|---|---|

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

| | Elite IT performers | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|---|
| **Large organization that relies on in-house software**<br><br>(8,500 technical staff) | 8,500 staff x $143,000 salary x 1.5 benefits x 1% rework = $18.2M | 8,500 staff x $143,000 salary x 1.5 benefits x 1.5% rework = $27.3M | 8,500 staff x $143,000 salary x 1.5 benefits x 2% rework = $36.5M | 8,500 staff x $143,000 salary x 1.5 benefits x 2% rework = $36.5M |
| **Medium to large technical organization**<br><br>(2,000 technical staff) | 2,000 staff x $143,000 salary x 1.5 benefits x 1% rework = $4.3M | 2,000 staff x $143,000 salary x 1.5 benefits x 1.5% rework = $6.4M | 2,000 staff x $143,000 salary x 1.5 benefits x 2% rework = $8.6M | 2,000 staff x $143,000 salary x 1.5 benefits x 2% rework = $8.6M |
| **Small to medium businesses and non-technical enterprises**<br><br>(250 technical staff) | 250 staff x $143,000salary x 1.5 benefits x 1% rework = $536K | 250 staff x $143,000salary x 1.5 benefits x 1.5% rework = $804K | 250 staff x $143,000 salary x 1.5 benefits x 2% rework = $1M | 250 staff x $143,000 salary x 1.5 benefits x 2% rework = $1M |

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 and Elite 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:

$$\boxed{\text{Potential Revenue from Reinvestment}} = \boxed{\text{Time recovered and reinvested in new features}} \times \boxed{\text{Revenue generating features}}$$

Where

$$\boxed{\text{Revenue generating features}} = \boxed{\text{Frequency of experiments per line of business}} \times \boxed{\text{Lines of business in the organization}} \times \boxed{\text{Idea success rate}} \times \boxed{\text{Idea impact}} \times \boxed{\text{Product business size}}$$

## 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.[i] 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 18% 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.

---

[i]  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.

For this number, we also refer to the 2018 Accelerate: State of DevOps industry benchmark data.

↑ **Elite performers** are able to redirect their efforts to value add work by **1%.** (This group reported 19% of their time spent on unnecessary rework; aiming for a goal of 18%, the difference is 1% of technical staff's time that will be spent on value add work.)

↗ **High performers** are able to reduce unnecessary rework, and therefore redirect their efforts to value-add work, by **1.5%**. (Reporting 19.5% originally, this group can realize a 1.5% increase in value-add work by redirecting technical staff's efforts to value add work by hitting the suggested goal of 18% of time spent on unnecessary rework.)

→ **Medium performers** are able to redirect their efforts to value-add work by **2%**. (This group reported 20% of their time spent on unnecessary rework; aiming for a goal of 18%, the difference is 2% 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 **2%**.
(This group reported 20% of their time spent on unnecessary rework; by reducing their unnecessary rework to 18%, they recover 2% 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 State of DevOps industry benchmark data to verify if it is possible for each group:

**Elite performers** are able to deploy code on demand, multiple deploys per day. Therefore an experiment frequency of twice per day (or 730 times per year) is achievable. We will use this number for our calculation.

**High performers** are able to deploy code between once per day and once per week. For this group we use the high end of these durations, or once per week, 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.

## 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.
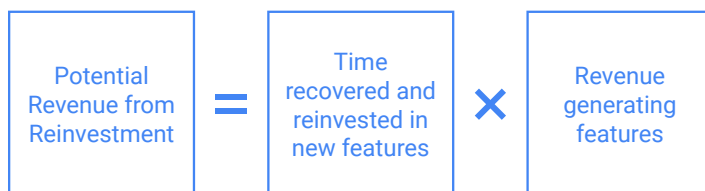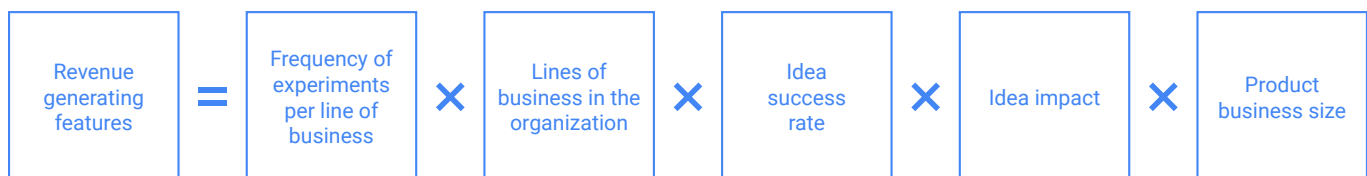
## Idea Impact

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[j] 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.

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

| Potential Revenue from Reinvestment | = | Time recovered and reinvested in new features | × | Revenue generating features |
|---|---|---|---|---|

Where

| Revenue generating features | = | Frequency of experiments per line of business | × | Lines of business in the organization | × | Idea success rate | × | Idea impact | × | Product business size |
|---|---|---|---|---|---|---|---|---|---|---|

[j] 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[k]**

| $100M product portfolio business size | Elite IT performers | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|---|
| **Large organization that relies on in-house software**<br><br>(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<br><br>= $48.7M return | 1.5% time recovered x 52 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $5.2M return | 2% time recovered x 12 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $1.6M return | 2% time recovered x 2 experiments/year x 20 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $267K return |
| **Medium to large technical organization**<br><br>(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<br><br>= $19.5M return | 1.5% time recovered x 52 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $2.1M return | 2% time recovered x 12 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $640K return | 2% time recovered x 2 experiments/year x 8 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $107K return |
| **Small to medium businesses and non-technical enterprises**<br><br>(250 technical staff) | 1% time recovered x 730 experiments/year x 1 lines of business x 1/3 success rate x 1% idea impact x $100M product business<br><br>= $2.4M return | 1.5% time recovered x 52 experiments/year x 1 line of business 1/3 success rate x 1% idea impact x $100M product business<br><br>= $260K return | 2% time recovered x 12 experiments/year x 1 line of business 1/3 success rate x 1% idea impact x $100M product business<br><br>= $80K return | 2% time recovered x 2 experiments/year x 1 line of business 1/3 success rate x 1% idea impact x $100M product business<br><br>= $13.3K return |

[k]  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.

## 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 hourly 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 Elite, 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 the 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 | × | Change fail rate percentage | × | Mean time to restore | × | 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 2019 Accelerate: State of DevOps industry benchmarks for these statistics:

↑ **Elite 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.

↗ **High performers** are able to deploy between once per day and once per week. For this calculation we use the average of these two, or **209 deploys per year**.

→ **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 2019 Accelerate: State of DevOps industry benchmarks for these statistics, but suggest you use your own if they are available:

↑ **Elite 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%**

↗ **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 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%.**

↓ **Low performers** report 46% to 60% of changes result in a degraded service or require remediation. For our calculation we will use the average of these two numbers: **53%.**

# 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 2019 Accelerate: State of DevOps industry benchmarks for these statistics:

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

↗ **High performers** report being able to restore service in less than one day. For our calculation we will use the midpoint of this range: **4 hours.**

→ **Medium performers** report being able to restore service in less than one day when an outage occurs. For our calculation we will use the upper end of this range: **8 hours.**

↓ **Low performers** report being able to restore service between one week and one month when an outage occurs. For our calculation we will use the midpoint of one month, or 15 days (equivalent to **120 hours**)

# 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.

| Cost of Downtime per year | = | Deployment frequency | × | Change fail rate percentage | × | Mean time to restore | × | Outage cost |
|---|---|---|---|---|---|---|---|---|

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**

| Elite IT performers | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|
| 730 deploys per year x 7.5% change fail rate x ½ hour MTTR x $500,000/hr outage cost | 209 deploys per year x 7.5% change fail rate x 4 hour MTTR x $500,000/hr outage cost | 32 deploys per year x 7.5% change fail rate x 8 hours MTTR x $500,000/hr outage cost | 7 deploys per year x 53% change fail rate x 120 hours MTTR x $500,000/hr outage cost |
| = $13.7M downtime cost per year | = $31.4M downtime cost per year | = $9.6M downtime cost per year | = $222.6M downtime cost per year |
| = $18.8K downtime cost per deployment | = $150K downtime cost per deployment | = $300K downtime cost per deployment | = $31.8M downtime cost per deployment |

According to our model, it is clear that Low Performers incur the highest downtime costs both per year and per deployment. High performers have higher downtime cost per year compared to Medium performers, likely due to the High performers deploying nearly six times more than Medium performers and the subsequent costs they make incur to fix those changes. That being said, the model shows that High Performers have a lower spend per-deployment than Medium Performers. In reality, these numbers should be lower, since Elite and 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 down-time 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.
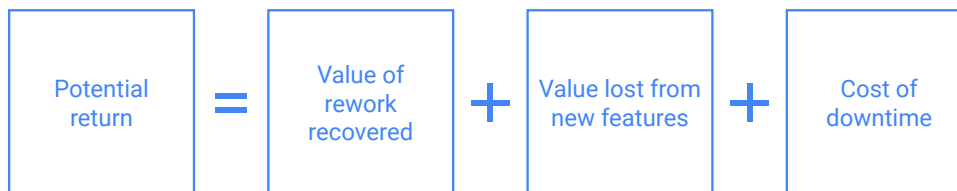
| Potential return | = | Value of rework recovered | + | Value lost from new features | + | Cost of downtime |
|---|---|---|---|---|---|---|

## Table 5.

**Potential return of large product business ($100M)**

| $100M product portfolio business size | Elite IT performers | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|---|
| **Large organization that relies on in-house software**<br><br>(8,500 engineers) | $18.2M value of rework recovered +<br><br>$48.7M value lost from new features +<br><br>$13.7M cost of downtime<br><br>= $80.6M return | $27.3M value of rework recovered +<br><br>$5.2M value lost from new features +<br><br>$31.4M cost of downtime<br><br>= $63.9M return | $36.5M value of rework recovered+<br><br>$1.6M value lost from new features +<br><br>$9.6M cost of downtime<br><br>= $47.7M return | $36.5M value of rework recovered +<br><br>$267K value lost from new features +<br><br>$222.6M cost of downtime<br><br>= $259.3M return |
| **Medium-to-large technical organization**<br><br>(2,000 engineers) | $4.3M value of rework recovered +<br><br>$19.5M value lost from new features +<br><br>$13.7M cost of downtime<br><br>= $37.4M return | $6.4 cost of rework +<br><br>$2M value lost from new features +<br><br>$31.4M cost of downtime<br><br>= $39.9M return | $8.6M cost of rework +<br><br>$640K value lost from new features +<br><br>$9.6M cost of downtime<br><br>= $18.8M return | $8.6M cost of rework +<br><br>$107K value lost from new features +<br><br>$222.6M cost of downtime<br><br>= $231.3M return |
| **Small to medium businesses and non-technical enterprises**<br><br>(250 engineers) | $536K value of rework recovered +<br><br>$2.4M value lost from new features +<br><br>$13.7M cost of downtime<br><br>= $16.7M return | $804K value of rework recovered +<br><br>$260K value lost from new features +<br><br>$31.4M cost of downtime<br><br>= $32.4M return | $1M value of rework recovered+<br><br>$80K value lost from new features +<br><br>$9.6M cost of downtime<br><br>= $10.8M return | $1M value of rework recovered+<br><br>$13.3K value lost from new features +<br><br>$222.6M cost of downtime<br><br>= $223.7M 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 on rework and more time on 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 $6.8M (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 $6.8M investment breakdown could look like:

| Item | Spend amount |
|---|---|
| Consulting: assessment and roadmap development for technology transformation initiative | $400,000 |
| Automation software | $1,000,00 |
| SREs and DevOps engineers to augment team (5 x $180,000 x 1.5 benefits multiplier)[l] | $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 $143,000 x 1.5 benefits multiplier) | $3,861,000[m] |
| **Total Investment** | **$6,811,000** |

---

[l]  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.

[m]  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[n]. The output of the equation is in years.

Using the potential return of a large product business, in the Elite category, we are considering an investment that will cost $6.8M and will generate $80.6M per year in returns. If we assume equal cash flow each year, we calculate the payback period by dividing the investment by the returns:

$$\text{Payback period} = \frac{\text{Investment}}{\text{Returns}} = \frac{\$6{,}811{,}000}{\$80{,}586{,}667} = .085 \text{ years}$$

The payback period is .085 years, or about 31 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.

---

[n] 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.

# Return on Investment

Return on investment calculates the profitability of a project and reports the return as a percentage of the investment[o]. 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 $6.8M and will generate $80.6M 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:

$$ROI = \frac{Return - Investment}{Investment} = \frac{\$80,586,667 - \$6,811,000}{\$6,811,000} = 10.832$$

The ROI for this investment is 10.832. 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 ~$10.83 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?

---

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

# Conclusion:
## 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 Elite 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.

# J-Curve of Transformation

Relentless improvement work leads to excellence and high performance! High and elite performers leverage expertise and learn from their environments to see jumps in productivity

Automation helps low performers progress to medium performers

Technical debt and increased complexity cause additional manual controls and layers of process around changes, slowing work

Teams begin transformation and identify quick wins

Automation increases test requirements, which are dealt with manually. A mountain of technical debt blocks progress

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 cloud.google.com/devops.

# Authors

### Nicole Forsgren, PhD

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

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's currently a Staff Developer Advocate at Google Cloud and [teaching at UC Berkeley]().

### Gene Kim

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).

# Brenna Washington

Brenna Washington is Product Marketing Manager at Google. In her role, she is responsible for developer marketing and crafting the product story behind some of Google Clouds most used developer tools. Prior to working on Google Cloud, Brenna worked with YouTube to drive advertisers' awareness in understanding the critical role YouTube plays throughout the marketing funnel, from awareness to acquisition. Brenna holds a bachelors in Business Administration and Cinematic Arts with a certificate in Big Data Analytics from the University of Southern California.

# Nikhil Kaul

Nikhil Kaul leads DevOps product marketing at Google Cloud, where he is responsible for driving positioning, messaging, and go-to-market for DevOps products and solutions. Prior to Google, Nikhil spent his time in a variety of technology roles, including software engineering and product management. Nikhil holds a master of business administration (MBA) from Georgetown University.

# Dustin Smith

Dr. Dustin Smith is a human factors psychologist and staff user experience researcher at Google. He studies how people are affected by the systems and environments around them in a variety of contexts: software engineering, free-to-play gaming, and healthcare. His research at Google has emphasized identifying areas where software developers can feel happier and more productive during development. Dustin received his PhD in Human Factors Psychology from Wichita State University.

# About DORA

DevOps Research and Assessment (DORA) was founded by Dr. Nicole Forsgren, Jez Humble, and Gene Kim to conduct research into understanding high performance in the context of software development and the factors that predict this high performance. In 2018, DORA was acquired by Google. As part of Google Cloud, DORA continues to create delightful experiences for developers and operators through data-driven insights. Additionally, DORA's research over the last six years with more than 31,000 professionals serves as the basis for a set of evidence-based tools for evaluating and benchmarking technology organizations.

Learn more at cloud.google.com/devops.

Are you evaluating your own technology transformation?
We offer assessments directly to organizations.

Request a demo at camp-info@google.com

Google Cloud

DORA
DEVOPS RESEARCH & ASSESSMENT

# References

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

2.  DevOps Research and Assessment & Google Cloud. (n.d.).2019 Accelerate: State of DevOps Report (Rep.)

3.  DevOps Research and Assessment & Google Cloud. (n.d.).2019 Accelerate: 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.  Ippolito, B., & Murman, E. (2001, December). Improving the Software Upgrade Value Stream. In 43rd AIAA Aerospace Sciences Meeting and Exhibit (p. 1252).

8.  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

9.  Hainzinger, Brittany. "DevOps Salary Report for 2019 Is Here." App Developer Magazine, 22 Jan. 2019, appdevelopermagazine.com/devops-salary-report-for-2019-is-here/.

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. DevOps Research and Assessment, LLC. (n.d.). 2019 Accelerate: 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/