Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformation

Metrics for DevOps Initiatives

DevOps Enterprise Forum

IT REVOLUTION

Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformations

Metrics for DevOps Initiatives

DevOps Enterprise Forum

IT Revolution
Portland, Oregon

Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformations: Metrics for DevOps Initiatives

IT Revolution
Portland, Oregon
info@itrevolution.net

For quantity purchases by corporations, associations, and others, please contact the publisher at **orders@itrevolution.net**.

Cover design by Brian David Smith and Mammoth Collective.
Interior layout generated by O'Reilly Atlas.

Table of Contents

Preface

The DevOps Enterprise Forum, facilitated by IT Revolution, brought together 50 technology leaders and thinkers in the DevOps Enterprise community for three days in Portland, Oregon in May 2015, with the goal of organizing in to task teams and creating written guidance on the best-known methods for overcoming the top obstacles in the DevOps Enterprise community.

We tackled the five key areas identified by the community at the 2014 DevOps Enterprise Summit:

- Better strategies and tactics for creating automated tests for legacy applications
- Addressing culture and leadership aspects during transforming
- Top approaches to organizational design, roles and responsibilities
- Information security and compliance practices
- Identifying metrics to best improve performance with DevOps initiatives.

For three days, we broke into groups based on each of the key areas and set to work, choosing teams, sometimes switching between teams, collaborating, sharing, arguing… and writing.

After the Forum concluded, the groups spent the next six months working together to complete and refine the work they started together.

Our goal was to generate content in the form of white papers, articles, or other resources in time to share that guidance with the technology community during the DevOps Enterprise Summit 2015 in October.

We are proud to share the outcomes of the hard work, dedication, and collaboration of this amazing group of people.

—Gene Kim
October 2015

Introduction

Advancing the state of measurement practice is an important foundation for improving software delivery economics. The DevOps Enterprise community has identified improved measurement as a top priority for 2015. This paper summarizes a position on DevOps measurement that we intend to be a catalyst for accelerating more mature guidance on measuring and steering software delivery.

Principles 1

1.1 Hunger for Better Measurements

There is a hunger for better measurements. DevOps teams and professionals want to push new ideas and new ways of doing things. Metrics objectively help teams distinguish between improvements and unproductive changes. We feel measurements

1. Remove Subjectivity
2. Improve Excellence
3. Focus on Strategy
4. Create Predictability

Measurements work best when they drive crucial conversations and help teams improve. Measurements are undermined when they are driven as an evaluation of performance and impact. People will game the system or distort a metric if it focuses on personal performance rather than team performance. Therefore, a successful transformation to a metrics driven organization is powered by three things.

1. A better work environment more accepting of new ideas
2. Desire for accountable teams
3. Continuous improvement via the scientific method

1.2 What are some key principles of measurement?

Software's most important quality is its adaptiveness and ease of change. Ability to execute on a faster cadence (efficiency) and the ability to freely add more value (effectiveness) demand continuous change cycles. This is different from manufacturing processes. Accepting continuous change in software drives our belief in that we need different thinking to drive measurement and metrics for DevOps.

From experience reports we know there are a set of practices common to successful software delivery. Using these experience reports we have developed a set of guiding principles for a stronger measurement approach for DevOps software delivery.

1. Software's most important quality is its adaptiveness (ease of change), therefore rate of change and cost of change are the most important measures for software
2. The product pipeline—change managed builds, tests, deployments, and customer usage are the primary sources of measurement. Process pipelines are secondary sources.
3. Track at least one measure in each of three dimensions: 1-internal (efficiency or cost related), 2-external (effectiveness or value related), 3-cultural (team dynamics or people related); and no more than a few in any dimension.
4. Metric collection should be largely automated and unobtrusive to practitioners. Automation eliminates subjectivity and manual overhead, and improves consistency.
5. Measures should be useful and transparent to both leadership and engineering personnel in communicating progress and quality in a consistent format.
6. Measurement targets should be captured as distributions of probable outcomes. An honest exchange would include a discussion of the variance and the uncertainty inherent in the forecasted expectation.
7. Measurement precision and fidelity should improve over time along with the improvements in the software development process. Measures are not static; the best teams evolve their measures.

1.3 Which classes of measurements matter most?

We have identified three dimensions (or classes) of measures that are required to steer projects through the uncertainties, tradeoffs and critical decisions of software delivery:

- Internal - Objective measures and trends in technical progress, product pipeline trends and resource consumption. These are inside-out measures of efficiency and resource consumption (cost/time/materials)
- External - Objective measures and trends in quality, usefulness, performance, and business outcomes. These are outside-in measures of effectiveness and value delivered.

- Cultural - Objective and subjective trends in team dynamics, process overhead, trustworthiness, shared objectives, morale, motivation, and team/product/enterprise identity

In this paper we felt that External Measures have been well covered by accounting rules and industry experts. In addition we felt External Measures depend on the stage of the company from startup to steady business and the revenue model of the business. Public companies have standard accounting measures found in GAAP, IAS, and IFRS. Startups and new companies have a wealth of measures available like the “Four Gears” by Geoffrey Moore. For these reasons we focus the paper on Internal and Cultural measures.

1.4 Discussion on Lead Time and vs. Process Time

In the Internal Class of Measurements, Lead Time and Process Time are some of the most powerful measures. Software is at its best when it remains flexible and the team is free to adapt to changes. Lead Time and Process Time effectively measure the dynamic nature of software development across industries and types of projects. When used at an appropriately high level Lead Time and Process Time help to identify the odds of delivering items within expected timeframes, the bottlenecks in the process, quantify handoff efficiency between teams, and enable continued improvement in feedback cycles. Using these definitions across the industry will help establish some consistency and help document improvements in our practice.

Lead Time (LT): The elapsed time from receiving a customer request to delivering on that request. Lead Time = Process Time plus Wait Time.

Process Time (PT): Process time begins when the work has been pulled into a doing state and ends when the work is delivered to the next downstream customer.

Wait Time (WT): The time that work sits idle not being worked.

Lead time, Process time and Wait time metrics reveal how long it takes to deliver value. A faster delivery time allows for faster feedback which in turn increases the product value These metrics can also expose inefficiencies (idle time, non-value-adding process time) in the workflow that create uncertainty. Reducing uncertainty helps organizations become more predictable.

1.5 Discussion on Code Deployment Lead Time

One of the key findings of the Puppet Labs State of DevOps reports spanning three years, which included surveys of over 20,000 IT professionals, was that a significant predictor of IT performance and organization performance is "code deployment lead time," as it divides the two qualitatively segments of the technology value stream:

- The Design and Development value stream begins when product owners come up with hypotheses or customers submit requests, and ends when code is committed into our version control repository.
- The Testing and Operations value stream begins at this point when work is checked into version control, where it is ideally then integrated, submitted to a automated suite of unit, acceptance and non-functional tests (such as performance tests), and then deployed into production in single-piece flow.

The Design and Development value stream more resembles Lean Product Development, which is highly variable and highly uncertain, often requiring high degrees of creativity and performing work that may never be performed again. On the other hand, the Test and Operations value stream more resembles Lean Manufacturing, while also requires creativity, expertise and experimentation, strives to be predictable and mechanistic, with the goal of achieving work outputs with minimized variability (e.g., lead times, process times, defects, etc.).

Process times in the Design and Development value stream are usually longer (e.g., days or weeks to complete) than process times in the Testing and Operations value stream (e.g., minutes, hours). To achieve DevOps outcomes of fast flow and high reliability, we need Testing work to happen simultaneously with Development work (or even earlier, with techniques such as Test Driven Development).

We do this by ensuring work in the Design and Development value stream is frequent and continuous, with developers checking code into version control and integrating frequently (e.g., at least daily), with the Testing and Operations value stream providing fast feedback (e.g., minutes or hours) back to the Design and Development value stream.

One of the findings of the 2014 and 2015 Puppet Labs State of DevOps research was that "deployment lead time" serves as a predictor of performance for both value streams: Deployment lead times not only predicts deployment and reliability outcomes, but it also measures how quickly Design and Development value stream can receive feedback from the customer and how quickly new customer experiments can be performed.

High performers had significantly better outcomes:

- Throughput and agility metrics
    - 200x faster deployment lead times
    - 30x more frequent deployments
- Reliability and stability metrics
    - 60x higher change success rates
    - 168x faster mean time to restore service (MTTR)
- Organizational performance metrics
    - 2x more likely to exceed productivity, market share, and profitability goals
    - 50% higher market capitalization growth over three years.

1.6 Discussion on Work-in-progress (WIP)

Where lead time and process time are trailing indicators (we don't know how long something took to do until it's been completed), Work-in-Progress metrics provide organizations with a rare leading indicator. It's proven that the more items we have on our plate, the longer those items take to complete. The primary cause is due to increased context switching and dependencies. The minute we set work item 1 aside to take on work item 2, work item 1 starts to becomes stale. If we then set work item 2 aside to begin work item 3 before items 1 and 2 are done, a pile of unfinished work (WIP) builds up that provides no real value. Identifying how much WIP is in the system is the best predictor of Lead Time.

Work-in-progress (WIP): The amount of work in a system that has been started but not finished.

A consistent amount of WIP, where the amount of incoming work requests is approximately equal to the amount of outgoing work requests, helps teams to balance their workload in a sustainable fashion and become more predictable.

1.7 Measuring Culture

1.7.1 Why Culture Matters

Organizational culture has been shown to be an important factor in organizational change and organizational impacts for decades. In contrast to other broad measures of culture (such as national culture), an organization's culture can be seen and observed in its formal state (such as mission statements and goals) as well as its informal state (such as shared values and social norms). The importance of shared values and social norms seen in informal organizational structures have been quite apparent, as they can often be stronger than any formal communication, which may be "tossed aside" or "only used for show."

Organizational culture has many facets, and practitioners can choose to focus on aspects of their culture which would benefit their context or project the most. In DevOps transformations, which prioritize work and communication across previously separate silos of technical teams, information flow is paramount. Ron Westrum proposed a model of organizational culture that emphasized the importance of information flow in complex and high risk work environments (2004). For this reason, the State of DevOps Study 2014 and 2015 (published by Puppet Labs and IT Revolution) selected this framework for inclusion in its investigations, and adapted it for use in research. In fact, one could easily argue that the work of software and IT service delivery is high risk and complex, and information flow is crucial, making this framework ideal for the context.

1.7.2 How to Measure Culture

Organizational culture is a perceptual measure, and therefore, captured using survey methods. It is important to use well-designed items and constructs that are rigorous and show good psychometric properties of discriminant and convergent validity and reliability (all statistical measures). Always base survey questions and constructs on well-tested theories and expert input, and use a Likert-type format. As an example, Westrum survey measures are included below.[1]

Westrum items. Note these items have been shown to be highly valid and reliable statistically. They should be presented together (though not labeled or titled), with responses ranging from Strongly Disagree (=1) to Strongly Agree (=7). They are a latent construct, which means their scores can be averaged, thereby providing a single score for one's Westrum culture metric. The items can be slightly altered, if necessary, to fit your context, but only minor changes should be done in order to preserve the statistical properties.

- On my team, information is actively sought.
- On my team, failures are learning opportunities, and messengers of them are not punished.
- On my team, responsibilities are shared.
- On my team, cross-functional collaboration is encouraged and rewarded.
- On my team, failure causes enquiry.
- On my team, new ideas are welcomed.

1 Westrum, R. (2004). A typology of organisational cultures. Quality and safety in health care, 13(suppl 2), ii22-ii27. Retrieved from http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1765804/pdf/v013p0ii22.pdf.

Studies have shown that Westrum culture is a significant predictor of IT performance (in the context of DevOps work) as well as organizational performance. High IT Performers have significantly higher Westrum scores than Medium and Low IT Performers. Higher scores are indicative of a generative (or performance-oriented) culture, while lower scores are indicative of a bureaucratic culture.

One closing question about the Westrum construct: What does it have to do with information flow? Very few of the questions actually talk about information or communication… What this really comes down to is creating and fostering a culture of **trust**. Because once you work in an environment where you trust those you work with, and you know they trust you, sharing information just happens. It is a byproduct of the intimacy that comes from honest communication, egoless collaboration, shared objectives, learning, and welcoming a diversity of thought.

1.7.3 When to Measure Culture

Good practices suggest measuring at regular intervals. This allows for longitudinal analysis and comparisons across periods and quarterly or twice-yearly measurement is ideal. Organizational culture is known to be important for DevOps transformations, and they give us insight into how a team is functioning – or how they think they're functioning. It also can give us a hint into what may be coming: preliminary results suggest certain aspects of organizational culture (such as the Westrum construct) can be a leading indicator for any upcoming challenges.

For example, a dip in a team's Westrum score that corresponds to a major reorg may be expected, or at least would not be a complete surprise.
However, a dip in a team's Westrum score with no known explanation could indicate challenges in team dynamic or information flow that should be addressed. If not addresses, automation and/or tooling challenges are likely forthcoming.

The key takeaway here is that whatever metrics you use, watch for how they are changing. Progression, digression, or stability of your metrics are the key to learning and improving.

Scenarios 2

2.1 Scenario 1: Moving to a SaaS Culture

Shanti is an engineering manager in an independent software vendor (ISV) that is transitioning its product to a software–as-a-service (SaaS) offering. In the past, her organization has already moved to agile teams and schedules and practices such as a common product backlog. Now moving to the cloud, Shanti and her leadership are discovering that their telemetry and engineering practices are inadequate for a SaaS business. They attack three areas of improvement:

1. Improving Live Site Health
2. Understanding Usage
3. Improving Engineering Flow and Reducing Engineering Debt

2.1.1 Live Site Health

Everything starts with Live Site. If the service is not available and performant for its users, no one will use it and nothing else matters. Accordingly, every Live Site Incident (LSI) is treated to very high scrutiny. For each LSI, Shanti measures:

Time to Detect	Shanti believes that *our telemetry should spot any problem before the customer tells us*. How long does it take to identify that which users are affected and how severely? Is this a single user, the worst .001 percentile, or a broad outage?
Time to Communicate	Shanti believes that customers need to know that we know. How long before we notify the affected users via our service blog and Twitter that we are working on the problem and what the expected next steps will be?

Time to Escalate	Shanti's philosophy is, You build it, you run it. How long before a member of the responsible development team is on the troubleshooting bridge working on a mitigation.
Time to Mitigate	Shanti believes that we operate 24x7x365 and cannot have downtime or slow performance. How long does it take us to restore the service in the users' eyes? Shanti watches the cumulative hours or days to solution by customer whenever there is a problem report.
Time to implement Learning, Repair, and Prevention (at Root Cause)	If something went wrong, there is at least one root cause. Shanti believes in using 5 Whys to understand what preventive actions we can take so that this class of problem does not recur. We then need to identify the engineering items, add them to the backlog, and complete them within two sprints at most. And we need to measure our effectiveness in the time to implement the learnings at root cause.
Total Customer Minutes Outside SLA (service level agreement)	This is the performance measure from the customer point of view. How many minutes has the service not responded within its SLA * the number of affected customers within a given week?
Outlier Performance	The most telling cases are the slow ones. Shanti and the team study performance at the 95th, 99th and 99.9th percentile to understand whether there are special cases that make these users experience the service differently than the majority. These instances lead to more 5 Whys and preventive actions.
Alerting Efficiency	An impediment to working on the right problems is the phenomenon of redundant alerts or missing alerts. Accordingly, Shanti pays careful attention to alerting efficiency. Ideally, every LSI should trigger an obvious and unique root cause alert. Shanti and her team have implemented a health model to consolidate redundant alerts and pinpoint root cause to minimize false positives. No human should have to triage alerts. At the same time, they are always scouring for gaps in telemetry which lack alerting, so that no problems go undetected.

2.1.2 Usage

Having a high-quality live site experience allows Shanti and her team to look closely at business metrics and to experiment with improvements in the business. They think of this as a funnel, in which they start with the effectiveness of *acquisition* of customers to first time trial use, then *engagement, retention, conversion* to ongoing subscription or premium offers. In addition, they are very conscious of understanding and minimizing *churn*. To handle the business metrics, Shanti's team puts custom instrumentation around each of the important business scenarios to measure the business transitions.

A goal for Shanti's team is to run the *maximum number of meaningful experiments in the least time at the least cost*. In other words, for each transition point in the funnel, the team works on hypotheses how to improve the customer pull-through and retention. They will experiment by exposing the new version to a cohort, and pivot or persevere based on observed behavior.

2.1.3 Engineering Velocity and Debt Indicators

In the earlier Agile days, before moving to SaaS, Shanti was confident that her engineering team could develop software faster than it could be deployed. She had already implemented unit testing, code coverage, static analysis, and other development practices for clean code. Now with DevOps, the Definition of Done is that everything runs in production according to service level agreement (SLA).

Shanti sees new forms of debt that come from the velocity she needs in the combined development and operations lifecycle. Her team is very conscious that technical debt can trigger unplanned work that derails their goals. They have had to implement a series of new practices to make this possible. *Infrastructure as Code (IaC)* allows a multistage release pipeline, where each stage is true to production and automation can move new code from development to testing to canary to production, with progressive exposure control along the way. *Database as Code* allows the database changes to be treated in the same way, preventing drift between production and test instances. *Feature flags* allow new work to be flighted as experiments to selected rings of users while in development before being released broadly.

Now Shanti watches these indicators of engineering velocity:

Pull Request to Build Available	From the time an engineer commits code, how long does it take for the new build to be available in a production-realistic environment that all engineers can use. This includes the time for the unit tests and code analysis that run as part of the build.
Time to Self-Test and Canary	When the engineering team is satisfied, and pushes the new features to a full automation run, including load testing, how long does it take to declare the service ready to use?

Time to Deploy (by ring)	Shanti's service deploys automatically across multiple data centers, grouped in parallel "rings." Progressive exposure and telemetry check the health of each ring before the next ring starts. Because the database is an integral part of the service, and downtime is not allowed, deployment happens in multiple steps. Shanti carefully watches the deployment time.
# Experiments/Release	Shanti is very conscious that the frequency of experiments the team can run, with exposure control, is a key to the improvement of the service. She carefully monitors the success of the engineering system in allowing experiments to progress and report results against the hypotheses under test.

Upstream, predeployment, Shanti is making sure that her development teams are not allowing any debt to accumulate. There she watches measures such as:

Bug cap per engineer	For each area team, what is the average number of active bugs per engineer? If it ever exceeds an agreed cap, e.g. 5, all new work has to stop, and all bugs need to be paid down before new work can proceed. By the end of the sprint, the team needs to be clean with zero active.
# and % of bugs caught by test automation, pass rate, and coverage by release stage	Tests are code. Ideally all bugs are caught by test automation and all automated tests run reliably. If the tests are flaky, they need to be fixed. If they do not cover code paths previously uncovered, they are redundant.
False Bugs / Missed Bugs due to Environment Drift or Incompleteness	The production realism of the Release Pipeline is essential. If bugs escape because the environments are not set up correctly, then the environment definitions need to be fixed.
# P0 or P1 bugs aged in buckets	All key issues discovered with the service need to be fixed promptly. These are aged. Shanti scrutinizes anything not fixed within the sprint to understand the delay. This is especially true of security items and performance items, although it also applies to anything that will affectthe quality of service.

Shanti and her team know that this is an evolutionary journey, where everything is about trends and improvement. They are never done. They are their own harshest critics, and they are paranoid that if they do not keep improving, they will discover that someone else has beaten them. They collect as much telemetry as they can, and always look for more opportunities to deliver a better service.

2.2 Scenario 2: High Incident Volume

2.2.1 The Problem

With an organizational change, a team inherits several new online services resulting in 3,000 high severity alerts a week. The team is overwhelmed with the alerts. Each of the 3,000 alerts generates an automated phone call which goes to the on-duty team. The on-duty rotation becomes 24x7 job of getting calls and responding to alerts. Even worse while the team is triaging and fixing one issue, several new issues will arise leading to more phone calls. The team feels frustrated and depressed. They are showing signs of being burnt-out.

2.2.2 Solution

The team decided that something must to be done. They decided the current situation was dire, and they called a meeting to decide on a drastic course of action. They needed to downgrade all alerts which were not customer impacting. For example an alarm indicating the worker process for a data ingestion job had timed out did not impact a customer. A new worker process would spawn in 30 seconds and resolve the issue.

First step: Eliminate false alarms by downgrading severity. These are incidents without customer impact. When the team found a noisy alert, they immediately downgrade the severity for all future alerts of that type. **Metric is signal/noise ratio.**

```
signal-noise-ratio = (total alerts - false alarms)/total alerts
```

The team felt good after the first day downgrading alerts. They were doing something. This effort remove the most frequent alarms and reduce daily volume to 2,600 alerts a week.

2.2.3 Eliminate Redundant Systems

As a result of the organizational change the team inherited new online systems with a completely different alerting and incident management pipeline. The on-duty team members were getting confused and making mistakes as they bounced back and forth between systems. The data was spread between two systems making it difficult to do a system wide analysis of problems and resolutions.

The team consolidate to a single alerting system. **Metric is number of alerting and incident management systems.**

By combining systems the team was able to invest in a single set of reports. They could now see the online service would slow down every Monday morning. This step did not reduce alerts that much but it did enable the team to "see the whole playing field".

2.2.4 Better Signal

The team noticed they would ignore alerts from synthetic tests unless the alert repeated several times in an hour. They discussed why this was true. They found they were more responsive with raw metrics alerts. Raw metric alerts are things like 99 Percentile of Requests slowed to 3 seconds or HTTP 5xx errors exceed threshold. The team decided to move away from synthetic probes to raw metrics. The team found synthetic probes sample infrequently, and they underreport or overreport the problem. **Metric number of alerts by origin.**

This improvement led to an other big reduction, the team was down to 2,200 alerts a week.

2.2.5 Repeat Offenders

The team found many spurious alerts were at a high severity due to concerns they will miss a very impactful event. To address this problem, the team decided alert severity should be raised if the alert repeats several times in a given time period. The team lowered all the system alerts to a low severity. If the alert happened 5 times in 30 minutes it would be auto-escalated to a high severity. This allowed the team to set "low disk space" alerts to low priority.

Metric is signal/noise ratio scoped to High Severity Incidents. High Sev signal-noise-ratio = (total High Sev alerts - High Sev false alarms)/total High Sev alerts

Eliminating Repeat Offenders made the biggest impact. The team was now down to 1,200 alerts a week.

2.2.6 Fix Top Problems

The team found a memory leak which cause the services to slow every Monday. The found a quick workaround and started working on a longer term solution.

They repeated the same process for the top 3 alert categories. **Metric is Top Correlation by Incident Volume.**

By fixing the top 3 causes of alerts the team was able to reduce alerts to 600 a week.

2.2.7 Success

By using these metrics the team was able to reduce their high severity incidents from 3,000 a week to 600 a week. A huge improvement. Now the team has 90 alerts a day. Now they have time to look into the real issues and fix the true underlying problems.

- Metric is signal/noise ratio
- Metric is number of alerting systems.
- Metric number of alerts by origin
- Metric is signal/noise ratio scoped to High Severity Incidents
- Metric is Top Correlation by Incident Volume

2.3 Scenario 3: Complex Integration Test Problems

2.3.1 The Problem

A large telecommunications organization with tens of millions of customers has hundreds of retail stores. One of the most critical applications is called CoFEE, which is used by every retail employee to support customers, by customer service representatives in call centers, and by technicians in the field.

This application is been written over decades, and is a client-server system, which interfaces with over 600 backend applications, including store catalog, customer service, support, billing, customer provisioning, etc. Many of these backend systems are decades old, running on mainframes, as part of an acquisition, also decades ago.

To support organizational needs, there would be one major software release every two months — releases were often problematic, requiring long test times and often longer than desired recovery times when something went wrong.

The evidence of the problem were measured as followed:

- Long code integration and test times
- Low deployment and release frequencies (every 2 months)
- Poor release outcomes -- High number of post-change incidents -- High number of major customer impacting incidents -- Low release success rates

2.3.2 The Solution

Decoupling from Constraints

One of the top problem areas identified were the billing systems: this was an important system of record, with long lead times, because everyone was dependent on changes on the billing system. While new offers or capabilities could be designed for customers, attendant changes to the billing and downstream systems required that work to get slotted into an enterprise release calendar where each change drove multiple prioritization conversations - with each change causing cascading scheduling impacts.

As a result, the billing system was a constraint, with commitments and promises made for the next six months, with competition and contention for those scarce cycles.

The team set out to migrate from a client-server application to a .NET web application, and selected 120 interfaces to move to a SOA model, accessed via API gateways. The goal was to enable better architectural isolation, enable better testing and accountability, and decouple the release cadences of the different services.

Accelerating the Changes

The team took the primary user facing application that was now web based and identified the types of changes that could proceed without backend interactions. Based on the need for the changes, additional areas of functionality like the product catalog and caching of bills for display were split out into separate smaller decoupled applications. This left the core billing systems just doing billing and metering.

In addition, this also allowed each individual team to build their applications with technologies that were best suited to their development needs and their skills. Each application team was also able to make architecture decisions independently allowing appropriate scaling patterns.

2.3.3 Success

By using these decoupling actions, the team was able to speed up application deployments. The key indicators of success for the customer facing teams were that the teams:

- moved to weekly releases

- defined features and capabilities in the application that were not tied into enterprise release model
- built automation to deploy these changes faster
- enabled testing in isolation for non-coupled changes allowing for better-quality designed features
- toggles user-facing features supporting 40K call center reps

3 The Value of Measurement 3

Successful software outcomes depend on negotiation of requirements, accurate scoping of work, value judgments, innovations, team collaboration, software architecture, economic tradeoffs and user demand. Success is less dependent on contract quality, Gantt charts, critical path schedules, earned-value measurement, laws of physics, material properties, mature building codes, and certified engineers. Stated another way, steering software delivery projects is more a discipline of economics than it is of engineering. Unlike most mature engineering disciplines, software delivery is a nondeterministic endeavor with more uncertainty to be managed.

What is measurement?

> Scientists define measurement as an observation that reduces uncertainty, where the result is expressed as a quantity (Hubbard, 2010).

Taking action earlier in the project life cycle is a recurring attribute found in most successful software development efforts. Early efforts to reduce uncertainty and seize opportunity validate learning and increase the probability of success. To progress on this front, this paper makes a case for a core measurement foundation of software-based systems. This foundation will propose stronger practices and tooling that specifically address a few key themes:

- measurement of process time and lead time
- quantifying uncertainty or complexity
- forecasting outcomes probabilistically
- evaluating drivers of organizational health

3.1 Improved measurement can increase trust in software delivery.

Quantifying software progress and quality can lead to more honest and trustworthy exchanges among constituencies. Trust is the currency of lean engineering efficiency. Trust is earned when we combine integrity and competence. Measurements with an accepted basis of theory and practice are easier to trust. The software development industry does not have enough accepted theory, and it has an ever changing set of practices.

3.2 Improved Measurement Can Enable Better Decision Making and Steering.

Software delivery involves tradeoffs across multiple competing dimensions. While some decisions can positively affect all dimensions, most decisions are tradeoffs based on quantifiable optimizations. Without objective measurements, decision-making centers on subjective guesswork. Better user experiences and better business outcomes will always require expert judgement. Teams can play better offense (exploiting opportunities) and better defense (managing risks and uncertainties) by incorporating objective evidence from metrics into their judgments in steering.

3.3 Getting Started

We are excited by the possibilities of standardizing metrics for software engineering. Translating this excitement into action raises the obvious question, where do we start and how do we drive better judgment through metrics and measurements? This is a complicated topic which deserves a complete analysis of the challenges and practical tools and techniques that will make an impact in your organization. Adoption of measures and metrics will be covered in future papers and presentations. We look forward to your feedback and continued support in driving standard measures and metrics for software engineering.

References A

- Westrum, R. (2004). A typology of organisational cultures. *Quality and safety in health care*, 13(suppl 2), ii22-ii27. Retrieved from **http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1765804/pdf/v013p0ii22.pdf**.
- Forsgren Velasquez, N., Kim, G., Kersten, N., & Humble, J. (2014). 2014 State of DevOps Report. Puppet Labs and IT Revolution.
- 2015 State of DevOps Report. (2015). Puppet Labs and IT Revolution.
- Martin, Karen and Mike Osterling. *Value Stream Mapping: How to Visualize Work and Align Leadership for Organizational Transformation*, (McGraw-Hill Education, 2013) p. 71.

Collaborators B

- Dominica DeGrandis, Director, Learning & Development, LeanKit
- Nicole Forsgren, PhD, Director Organizational Performance and Analytics, Chef
- Sam Guckenheimer, Product Owner, Visual Studio Cloud Services, Microsoft
- Mirco Hering, DevOps Lead APAC, Accenture
- Mark Michaelis, Chief Technical Architect, IntelliTect
- Chivas Nambiar, Director DevOps Platform Engineering, Verizon
- Eric Passmore, CTO MSN, Microsoft
- Walker Royce, Software Economist
- Jeff Weber, Managing Director, Protiviti

Acknowledgments C

IT Revolution wishes to thank the following sponsors for making the DevOps Enterprise Forum possible.

We wish to thank all the participants of the 2015 DevOps Enterprise Forum

- Steve Barr, Executive Director, Operations at CSG International
- Ross Clanton, Senior Group Manager, Target
- Jason Cox, Director of Systems Engineering, The Walt Disney Company
- Dominica DeGrandis, Director, Learning & Development, LeanKit
- James DeLuccia, Director and Leader for Certification Services, EY Certify-Point
- Jason DuMars, Senior Director of Technical Operations, SendGrid
- Paul Duvall, Chairman and CTO, Stelligent, Author of Continuous Integration and DevOps in AWS

- Damon Edwards, Managing Partner DTO Solutions, Inc
- Nicole Forsgren, PhD, Director Organizational Performance and Analytics, Chef
- Jeff Gallimore, Partner, Excella Consulting
- Gary Gruver, President, Practical Large Scale Agile LLC
- Sam Guckenheimer, Product Owner, Microsoft
- Mirco Hering, DevOps Lead APAC, Accenture
- Christine Hudson, Solutions and Product Marketing, Rally Software
- Jez Humble, Owner, Jez Humble & Associates LLC
- Mustafa Kapadia, DevOps Service Line Leader, IBM
- Nigel Kersten, CTO, Puppet
- Gene Kim, Author and Researcher
- Courtney Kissler, Vice President of E-Commerce and Store Technologies, Nordstrom
- Dave Mangot, Director of Operations, Librato, Inc.
- Mark Michaelis, Chief Technical Architect, IntelliTect
- Heather Mickman, Senior Group Manager, Target
- Chivas Nambiar, Director DevOps Platform Engineering, Verizon
- Steve Neely, Director of Software Engineering, Rally Software
- Tapabrata "Topo" Pal, Product Manager, CapitalOne
- Eric Passmore, CTO MSN, Microsoft
- Mark Peterson, Sr. Director, Infrastructure Engineering & Operations, Nordstrom
- Scott Prugh, Chief Architect, CSG International
- Terri Potts, Technical Director, Raytheon IIS Software
- Walker Royce, Software Economist
- Jeremy Van Haren, Director of Software Development, CSG International
- Jeff Weber, Managing Director, Protiviti
- James Wickett, Sr. Engineer, Signal Sciences Corp
- John Willis, Director of Ecosystem Development, Docker
- Tim Wilson, Solution Architect, IBM
- Elizabeth Wittig, Field Solutions Engineer, Puppet
- Julie Yoo, Vice President, Information Security Compliance, Live Nation`

And we would also like to acknowledge the organizers, scribes, editors, and designers who lent their support and attention to make the event and these artifacts possible:

Alex Broderick-Forster, Alanna Brown, Robyn Crummer-Olson, William Hertling, Aly Hoffman, Todd Sattersten, and Brian David Smith.