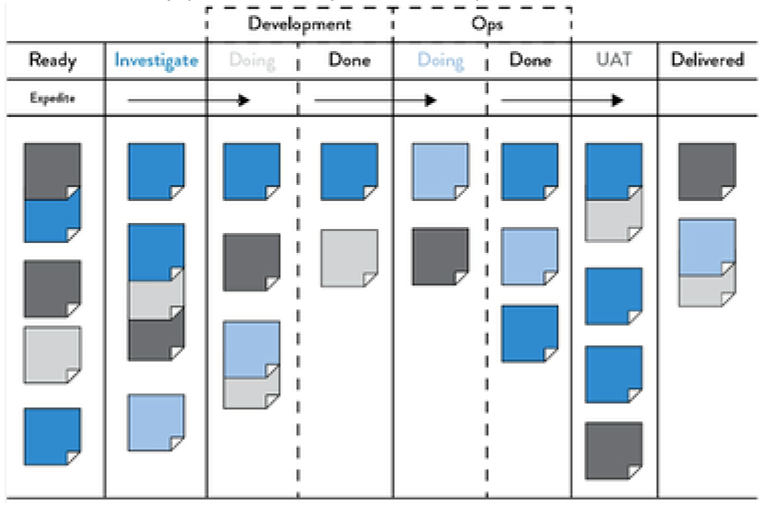
DevOps Handbook

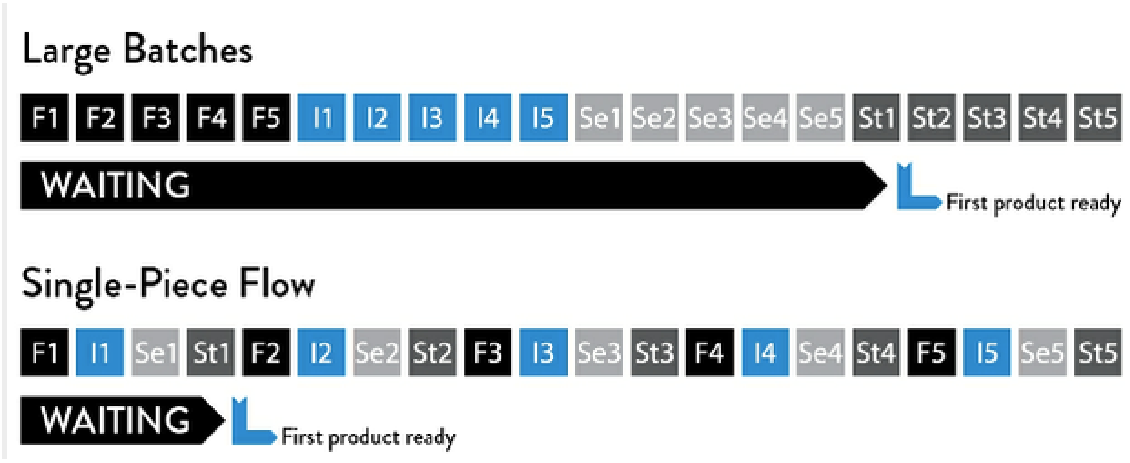
Summary for
The DevOps Handbook:: How to Create World-Class Agility, Reliability, and Security in Technology Organizations
Kim, Gene

Make Work Visible

- Kanban boards are an ideal tool to create visibility, and visibility is a key component in properly recognizing and integrating Ops work into all the relevant value streams. These visual sprint planning boards represent work on physical or electronic cards.
- Enforce WIP limits for each Kanban column or work center that puts an upper limit on the number of cards that can be in a column.
- Visible Telementry (See Telementry section below)
- 

Reduce Batch Sizes & Intervals of Work

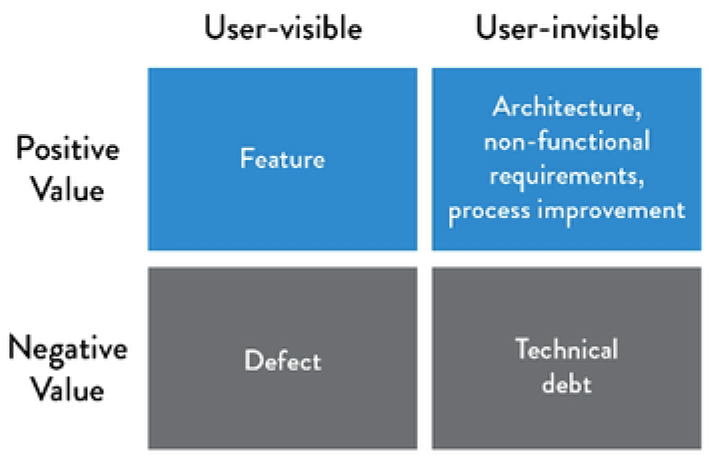
- The secret to smooth and continuous flow is making small, frequent changes that anyone can inspect and easily understand. Deliver work more frequently with incremental releases instead of large waterfall releases. Our goal is to identify the smallest number of incremental steps to achieve the desired outcomes, as opposed to " big bang " changes, which not only take longer to implement, but delay the improvements we need.
- **Reducing Work in Progress (WIP) - "Stop starting. Start finishing."**
  - Controlling queue size [WIP] is an extremely powerful management tool, as it is one of the few leading indicators of lead time.
  - With most work items, we don't know how long it will take until it's actually completed. Lean Methodology is "the use of any material or resource beyond what the customer requires and is willing to pay for." There is a lot of Non Value-Added time that becomes part of work that needs to be understood and removed to reduce WIP time:
    - Handoffs: Reduce # of handoffs through automation and reorganization of teams to be independent
    - Partially done work
    - Extra processes
    - Extra features
    - Task switching
    - Waiting
    - Motion
    - Defects
    - Nonstandard or manual work
    - Heroics
- Single Piece Flow - Perform each operation one unit at a time
  - Four steps to mailing an envelope: fold, insert, seal, stamp. If you had 10 letters, the large batch strategy would be to fold all 10 first, then move to the next step. The problem is once a defect is discovered, you have to go back and start from the beginning. You discover the defect quicker with single piece flow, which results in less WIP, faster lead times, and faster detection of errors & rework.
  - Figure 7: Simulation of "envelope game" (fold, insert, seal, and stamp the envelope)

- 
Large Batches
F1 F2 F3 F4 F5 I1 I2 I3 I4 I5 Se1 Se2 Se3 Se4 Se5 St1 St2 St3 St4 St5
WAITING
First product ready
Single-Piece Flow
F1 I1 Se1 St1 F2 I2 Se2 St2 F3 I3 Se3 St3 F4 I4 Se4 St4 F5 I5 Se5 St5
WAITING
First product ready
    - For software, the easiest batch to see is code. The equivalent to single piece flow in the technology value stream is realized with continuous deployment, where each change committed to version control is integrated, tested, and deployed into production.

<u>Andon Chord (Swarming</u>)

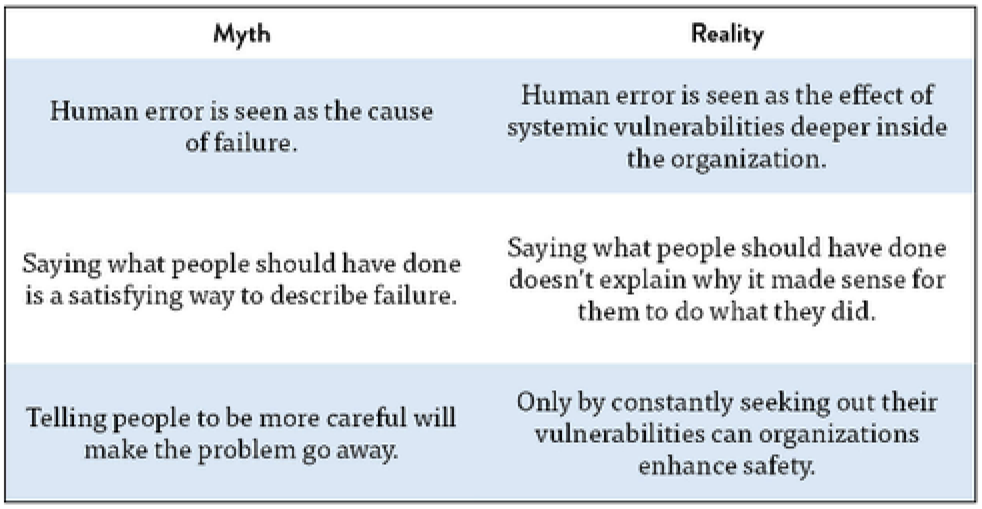
- Instead of working around the problem or scheduling a fix "when we have more time, we "pull an Andon cord", we swarm to solve the problem immediately before it has a chance to spread and prevent the introduction of new work until the issue has been resolved, building new knowledge, pushing quality closer to the source, and preventing the problem from recurring.
- In order to keep our deployment pipeline in a green state, we will create a virtual Andon Cord, similar to the physical one in the Toyota Production System. Whenever someone introduces a change that causes our build or automated tests to fail, no new work is allowed to enter the system until the problem is fixed. And if someone needs help to resolve the problem, they can bring in whatever help they need.
- With all eyes on the problem, the problem will be solved quickly.

<u>Reduce Technical Debt</u>

- **"No wonder then that spiders repair rips and tears in the web as they occur, not waiting for the failures to accumulate."**
- Technical debt describes how decisions we make lead to problems that get increasingly more difficult to fix over time, continually reducing our available options in the future — even when taken on judiciously, we still incur interest. When organizations do not pay their "**20% tax**," technical debt will increase to the point where an organization inevitably spends all of its cycles paying down technical debt.
- Reduce technical debt by building in Quality Checks into daily work, which prevents defects passing downstream.
- Respond to Crises Habitually
    - For high performing organizations, responsiveness is their source of reliability, addressing real and potential crises all the time as opposed to periodically. These organizations can heal themselves and multiply their solutions by making them available throughout the organization. With complex systems, it is important to amplify weak failure signals to avert catastrophic failures (think NASA with rocket launches).
    - It is important to set an organization work routine that makes improvement work habitual. The Toyota Kata institutes an iterative, incremental, scientific approach to problem solving in the pursuit of a shared organizational true north.
    - Improvement kata. Every organization has work routines, and the improvement kata requires creating structure for the daily, habitual practice of improvement work, because daily practice is what improves outcomes.
- Figure 11 : Invest 20 % of cycles on those that create positive, user-invisible value
    - 

<u>Blameless Post Mortems</u>

- Blameless post mortems help create a just, safe, learning-based culture by encouraging everyone to talk about mistakes and accidents.
- "By removing blame, you remove fear; by removing fear, you enable honesty; and honesty enables prevention." Instead of assigning

blame, ask "Why did it make sense to me when I took that action?"
- Place meeting notes and artifacts in a centralized place for the whole company can learn from. Artifacts could include: timelines, chat logs, external communications, & incident Mean Time To Repair ("MTTR")
- Assign corrective countermeasures to someone if it is agreed as a top priority by everyone.
- Table 5: Two Stories
- 

Myth	Reality
Human error is seen as the cause of failure.	Human error is seen as the effect of systemic vulnerabilities deeper inside the organization.
Saying what people should have done is a satisfying way to describe failure.	Saying what people should have done doesn't explain why it made sense for them to do what they did.
Telling people to be more careful will make the problem go away.	Only by constantly seeking out their vulnerabilities can organizations enhance safety.

Game Day Exercises

- Perform Game Day exercises, where we rehearse large-scale failures by putting in place controlled introduction of failures into production to create opportunities to practice for the inevitable problems that arise within complex systems.
- Game Day Exercises come from the discipline of resilience engineering, which is “an exercise designed to increase resilience through large-scale fault injection across critical systems.”
- Netflix
  - Netflix ran Chaos Monkey to gain assurance that they had achieved their operational resilience objectives, constantly injecting failures into their pre-production system. This has evolved into a whole family of tools known internally as the “Netflix Simian Army” to simulate increasingly catastrophic levels of failures, some of which are:
    - Chaos Gorilla & Kong: simulates the failure of an entire AWS availability zone & regions
    - Latency Monkey: induces artificial delays or downtime in their RESTful client - server communication layer to simulate service degradation and ensure that dependent services respond appropriately
    - Doctor Monkey: taps into health checks that run on each instance and finds unhealthy instances and proactively shuts them down if owners don’t fix the root cause in time
    - Conformity Monkey; finds and terminates instances with security violations or vulnerabilities such as improperly configured AWS security groups

Blitzes

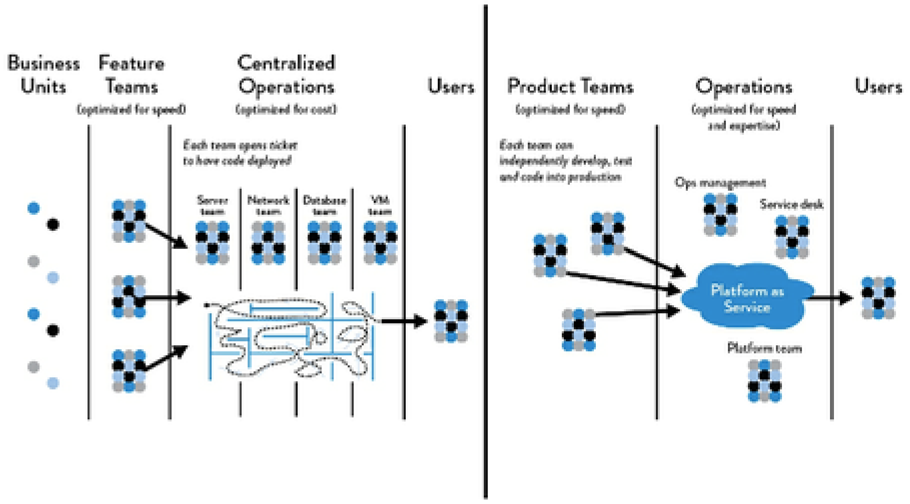
- A group is gathered to focus intently on a process with problems for a few days with the objective of process improvement, and the means are the concentrated use of people from outside the process to advise those normally inside the process.”
- Our goal during these blitzes is not to simply experiment and innovate for the sake of testing out new technologies, but to improve our daily work, such as solving our daily workarounds. While experiments can also lead to improvements, improvement blitzes are very focused on solving specific problems we encounter in our daily work. These blitz squads help remove obstacles — including tools, processes, and approvals — that impede work completion.

Organizational Best Practices

- **The Three Ways**
  - The Phoenix Project presents the Three Ways as the set of underpinning principles from which all the observed DevOps behaviors and patterns are derived. They consist of the delivery flow of work from Development to Operations to our customers, the principles of Feedback, which enable us to create ever safer systems of work, and the principles of Continual Learning and Experimentation, which foster a high-trust culture and a scientific approach to organizational improvement risk-taking as part of our daily work.
    - The First Way enables fast left-to-right **flow** of work from Development to Operations to the customer. In order to maximize flow, we need to make work visible, reduce our batch sizes and intervals of work, build in quality by preventing defects from being passed to downstream work centers, and constantly optimize for the global goals.
    - The Second Way enables the fast and constant flow of **feedback** from right to left at all stages of our value stream. It requires that we amplify feedback to prevent problems from happening again, or enable faster detection and recovery. By doing this, we create quality at the source and generate or embed knowledge where it is needed—this allows us to create ever-safer systems of work where problems are found and fixed long before a catastrophic failure occurs.
      - By creating fast feedback loops at every step of the process , everyone can immediately see the effects of their actions. It requires that we amplify feedback to prevent problems from happening again, or enable faster detection and recovery long before catastrophic failure occurs.
    - The Third Way enables the creation of a generative, high-trust culture that supports a dynamic, disciplined, and scientific approach to **experimentation** and risk-taking (taking as part of our daily work), facilitating the creation of continual

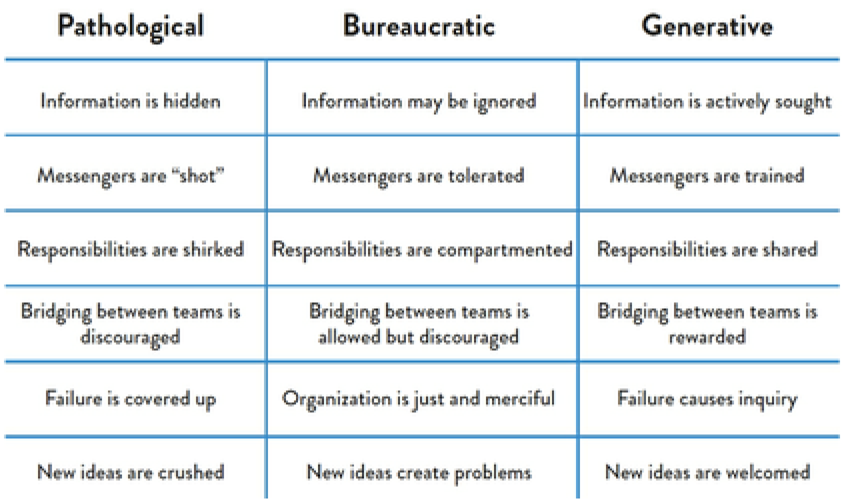
organizational learning, both from our successes and failures.

- Top-Down
  - Leaders of an organization, whether deliberately or inadvertently, reinforce the organizational culture and values through their actions.
  - Conway’s Law states that “organizations which design systems...are constrained to produce designs which are copies of the communication structures of these organizations. The larger an organization is, the less flexibility it has and the more pronounced the phenomenon.”
- Market Orientation
  - Although it is possible to have high-velocity organization around functional orientation (“optimizing for cost”), we typically strive to enable market orientation ("optimizing for speed”) so we can have many small teams working safely and independently, quickly delivering value to the customer. Instead of doing a large, top-down reorganization, which often creates large amounts of disruption, fear, and paralysis, we will instead embed the functional engineers and skills into each service team, or provide their capabilities to teams through automated self-service platforms that provide production-like environments, initiate automated tests, or perform deployments. This enables each service team to independently deliver value to the customer without having to open tickets with other groups.
  - Figure 12: Functional vs. Market Orientation. Left : Functional orientation : all work flows through centralized IT Operations ; Right : Market orientation : all product teams can deploy their loosely - coupled components self - service into production.



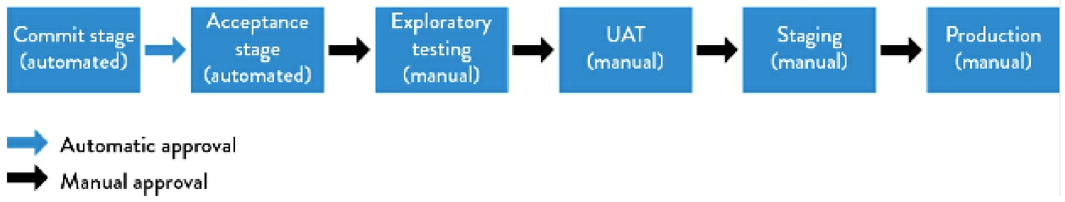
- Guided Experimentation
  - Create “buoys, not boundaries.” Instead of drawing hard boundaries that everyone has to stay within, we put buoys that indicate deep areas of the channel where you’re safe and supported. You can go past the buoys as long as you follow the organizational principles. After all, how are we ever going to see the next innovation that helps us win if we’re not exploring and testing at the edges? As leaders, we need to navigate the channel, mark the channel, and allow people to explore past it.
    - Standardized model, where routine and systems govern everything, including strict compliance with timelines and budgets
    - Experimental model, where every day every exercise and every piece of new information is evaluated and debated in a culture that resembles a research and design (R&D) laboratory.
  - Experimental Development Techniques to out-experiment competition: hypothesis-driven development, measuring customer acquisition funnel, A/B testing

- General
  - Many psychologists assert that creating systems that cause feelings of powerlessness is one of the most damaging things we can do to fellow human beings.
  - Organizational non-functional requirements: stability, scalability, availability, survivability, sustainability, security, supportability, manageability, and defensibility.
  - The top two technical practices that enabled fast MTTR were the use of version control by Operations and having telemetry and proactive monitoring in the production environment .
  - Generative organizations are characterized by actively seeking and sharing information to better enable the organization to achieve its mission. Responsibilities are shared throughout the value stream, and failure results in reflection and genuine inquiry.
  - “The only sustainable competitive advantage is an organization’s ability to learn faster than the competition.” Respond to the rapidly changing competitive landscapes while providing stable, reliable, and secure service to the customer.
  - Fast time to market vs. high service levels vs. relentless experimentation
  - The core chronic conflict is when organizational measurements and incentives across different silos prevent the achievement of global, organizational goals.
  - Optimize for Global Goals. One way is with a unified backlog, where everyone prioritizes improvement projects from a global perspective, selecting work that has the highest value to the organization or most reduces technical debt
  - Consolidate: Decide to massively reduce the number of technologies used in production, choosing a few that the entire organization could fully support and eradicating the rest.
  - Figure 8: The Westrum organizational typology model: how organizations process information

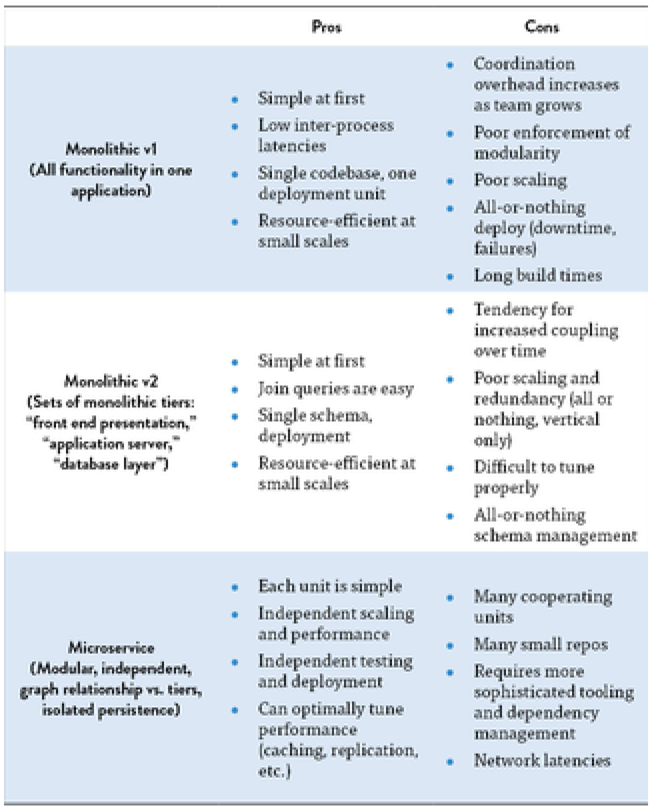
- 

Pathological	Bureaucratic	Generative
Information is hidden	Information may be ignored	Information is actively sought
Messengers are "shot"	Messengers are tolerated	Messengers are trained
Responsibilities are shirked	Responsibilities are compartmented	Responsibilities are shared
Bridging between teams is discouraged	Bridging between teams is allowed but discouraged	Bridging between teams is rewarded
Failure is covered up	Organization is just and merciful	Failure causes inquiry
New ideas are crushed	New ideas create problems	New ideas are welcomed

__Continuous Delivery__

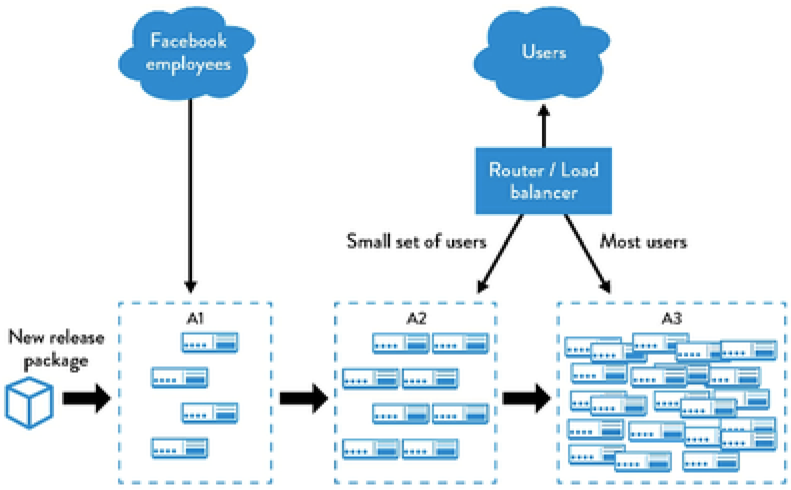
- **Deployment Pipeline**
    - The deployment pipeline, ensures we continually check small code changes into our version control repository, performing automated and exploratory testing against it in a production-like environment, and then deploying it into production. Build, Test, Deploy. The deployment pipeline ensures that code and infrastructure are always in a deployable state and that all code checked in to truck is deployed into production. Developers should see the impact of their work downstream, allowing more informed decisions in their daily work by seeing customer difficulties firsthand.
        - Figure 13 : The deployment pipeline
        - 

        - Table 3 : Architectural archetypes

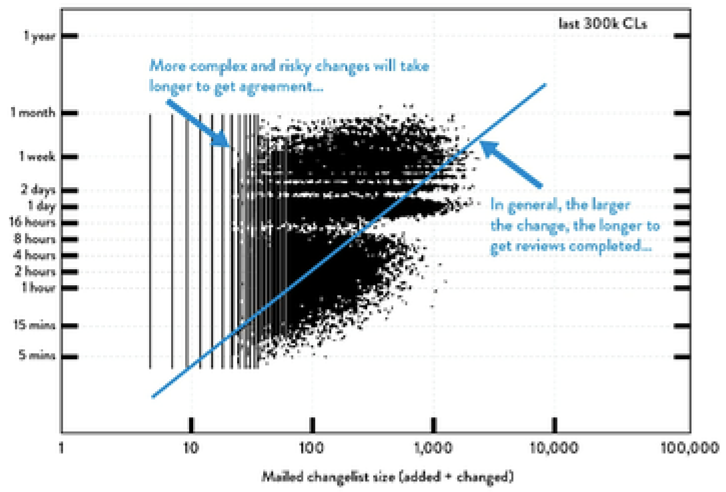


	Pros	Cons
Monolithic v1 (All functionality in one application)	• Simple at first • Low inter-process latencies • Single codebase, one deployment unit • Resource-efficient at small scales	• Coordination overhead increases as team grows • Poor enforcement of modularity • Poor scaling • All-or-nothing deploy (downtime, failures) • Long build times
Monolithic v2 (Sets of monolithic tiers: "front end presentation," "application server," "database layer")	• Simple at first • Join queries are easy • Single schema, deployment • Resource-efficient at small scales	• Tendency for increased coupling over time • Poor scaling and redundancy (all or nothing, vertical only) • Difficult to tune properly • All-or-nothing schema management
Microservice (Modular, independent, graph relationship vs. tiers, isolated persistence)	• Each unit is simple • Independent scaling and performance • Independent testing and deployment • Can optimally tune performance (caching, replication, etc.)	• Many cooperating units • Many small repos • Requires more sophisticated tooling and dependency management • Network latencies

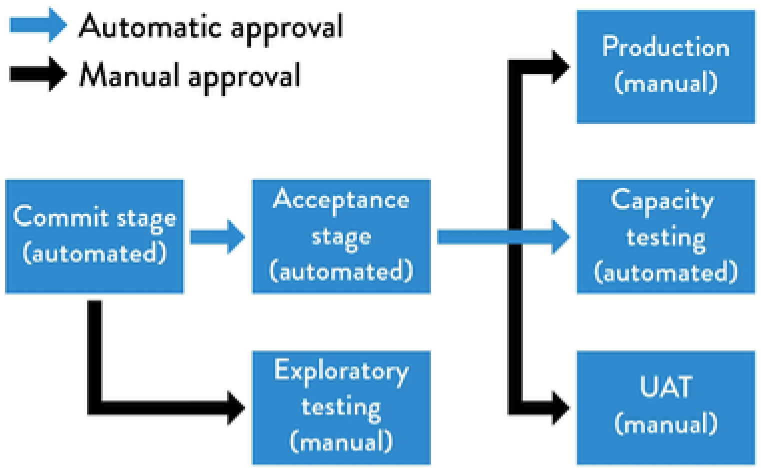
- Figure 21 : The canary release pattern



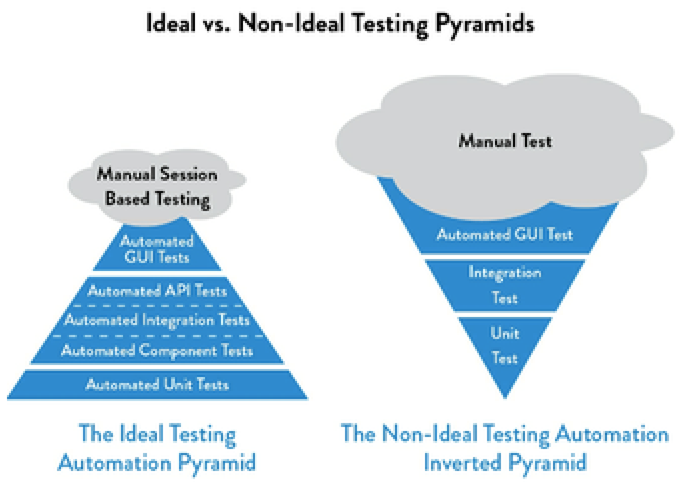
- Daily Trunk Based Commits - GitHub Flow
  - By having developers write, test, and run their own code in a production-like environment, the majority of the work to successfully integrate our code and environments happens during our daily work, instead of at the end of the release. Our goal is to ensure that we can re-create the entire production environment based on what's in version control.
  - When all developers are working in small batches on trunk, or everyone is working off trunk in short-lived feature branches that get merged to trunk regularly, and when trunk is always kept in a releasable state, and when we can release on demand at the push of a button during normal business hours, we are doing continuous delivery. Developers get fast feedback and when these issues are found, they are fixed immediately so that trunk is always deployable.
  - Developers integrate their code into trunk daily produces better throughput and stability with job satisfaction benefits. It forces breaking down work into smaller more more addressable chunks while still having the trunk of the work in a releasable state, allowing continuous integration and making deployment a low-risk process.
  - Continually identify and evaluate our constraints, and eliminating hardships in our daily work, architecting our environments and code to enable low-risk releases.
  - Figure 42 : Size of change vs . lead time for reviews at Google



- Testing
  - Need comprehensive, automated tests (unit, acceptance, integration test) that validate a deployable state, with developers who work on small batches, stopping entire product line when a test fails
  - Reduce manual testing and start with just a few reliable automated tests, growing the number of automated tests over time.
  - Without automated testing, the more code we write, the more time and money is required to test our code — in most cases, this is a totally unscalable business model for any technology organization.”
  - Whenever changes are committed into version control, fast automated tests are run in production-like environments, giving continual assurance that the code and environments operate as designed and are always in a secure and deployable state.
  - Google created a hard line: no changes would be accepted into GWS without accompanying automated tests because a single code deployment error at Google can take down every property.
  - Figure 15 : Running automated and manual tests in parallel

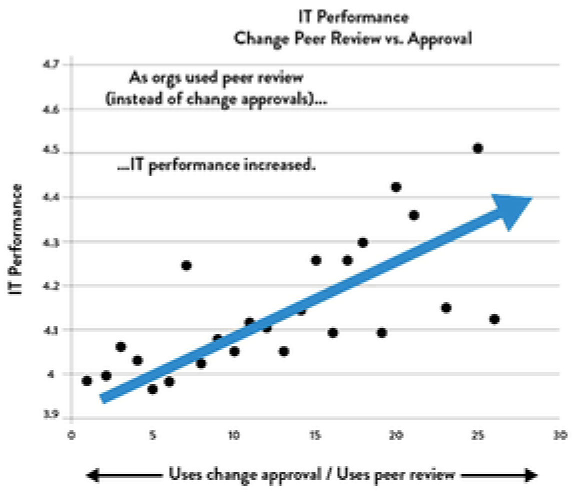


  - Figure 14 : The ideal and non - ideal automated testing pyramids


Ideal vs. Non-Ideal Testing Pyramids
Manual Session Based Testing
Automated GUI Tests
Automated API Tests
Automated Integration Tests
Automated Component Tests
Automated Unit Tests
The Ideal Testing Automation Pyramid
Manual Test
Automated GUI Test
Integration Test
Unit Test
The Non-Ideal Testing Automation Inverted Pyramid

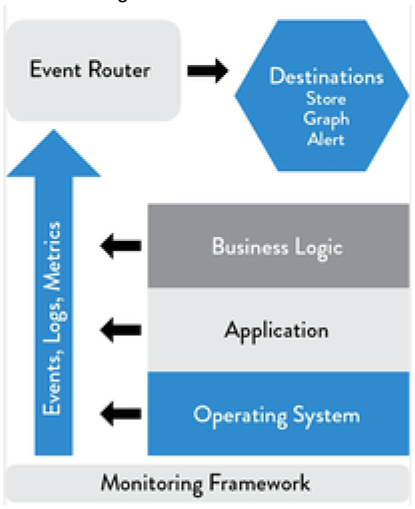
- **CREATE OUR SINGLE REPOSITORY OF TRUTH FOR THE ENTIRE SYSTEM**
  - One of the best ways to make knowledge re-usable is by putting it into a centralized source code repository, making the tool available for everyone to search and use. Encoding knowledge and sharing it through this repository is one of the most powerful mechanisms we have for propagating knowledge, "The most powerful mechanism for preventing failures at Google is the single code repository."
  - Version control also provides a means of communication for everyone working in the value stream — having Development, QA, nfosec, and Operations able to see each other's changes helps reduce surprises, creates visibility into each other's work, and helps build and reinforce trust.
  - Share not only code but artifacts around organizational knowledge and learning because "the actual compliance of an organization is in direct proportion to the degree to which its policies are expressed as code." This acts as a mechanism that helps convert individual expertise into artifacts that the rest of the organization can use. Include:
    - Configuration standards for our libraries, infrastructure, and environments.
    - Deployment, Testing, Monitoring and Security tools tutorials and standards
    - Blameless post-mortem reports.
- **Small, Self-Motivated Cross-Skilled Teams working in a High-Trust Management Model**
  - Large organizations using DevOps " have thousands of developers, but their architecture and practices enable small teams to still be incredibly productive, as if they were a startup."
  - Two Pizza Team Rule
  - When projects are late, adding more developers not only decreases individual developer productivity but also decreases overall productivity.
  - Enable and encourage every team member to be a generalist. Investing in cross training is the right thing for employees' career growth, and makes everyone's work more fun. "
- **Agile Scrum**
  - One of the Dev rituals popularized by Scrum is the daily standup, a quick meeting where everyone on the team gets together and presents to each other three things: what was done yesterday, what is going to be done today, and what is preventing you from getting your work done.
  - Another widespread agile ritual is the retrospective. At the end of each development interval, the team discusses what was successful, what could be improve, and how to incorporate the successes and improvements in future iterations or projects.

- **Peer Review**
  - High performing organizations rely more on daily integrated peer review and less on external periodic approval & inspections of changes because "people closest to a problem typically know the most about it."
  - A logical place to require reviews is prior to committing code to trunk in source control, where changes could potentially have a team-wide or global impact. The principle of small batch sizes also applies to code reviews. This is why it's so essential for developers to work in small, incremental steps rather than on long lived feature branches. Change Controls (the alternative) create unintended outcomes, such as long lead times, reducing effectiveness of feedback, multiplying number of steps and approvals, increasing batch sizes.
  - Code reviews come in various forms:
    - Pair programming: programmers work in pairs. The advantage of pair programming is its gripping immediacy: it is impossible to ignore the reviewer when he or she is sitting right next to you.
    - Over-the-shoulder: One developer looks over the author's shoulder as the latter walks through the code.
    - Email pass-around: A source code management system emails code to reviewers automatically after the code is checked in.
    - Tool - assisted code review: Specialized tools designed for peer code review (e.g.,Gerrit, GitHub pull requests, etc.) or facilities provided by the source code repositories (e.g., GitHub, Mercurial, Subversion, as well as other platforms such as Gerrit, Atlassian Stash, and Atlassian Crucible).
    - Test-Driven development (TDD): Have one engineer write the automated test and the other engineer implement the code.
  - Benefits of Peer Review
    - Programmers are 15% slower than two independent individual programmers, while 'error-free' code increased from 70% to 85%. Since testing and debugging are often many times more costly than initial programming, this is an impressive result.

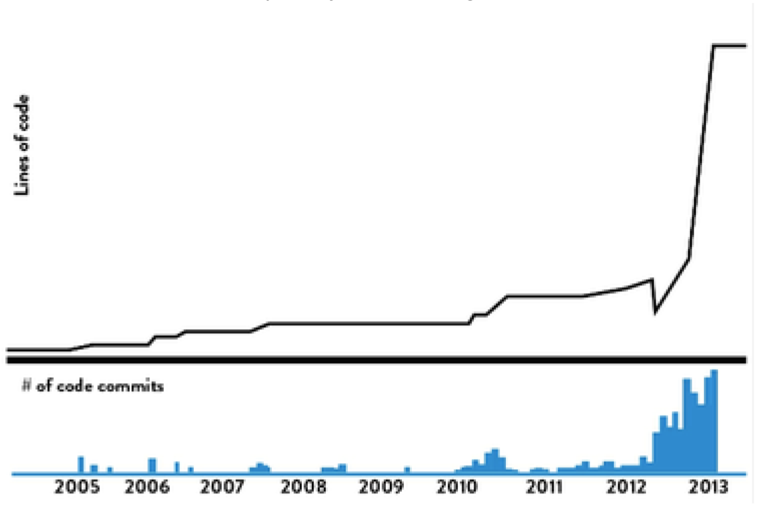
        - Pairs typically consider more design alternatives than programmers working alone and arrive at simpler, more maintainable designs; they also catch design defects early."
        - Dr. Williams also reported that 96% of her respondents stated that they enjoyed their work more when they programmed in pairs than when they programmed alone.
    - Figure 41 : Organizations that rely on peer review outperform those with change approvals
        - 

Telementry

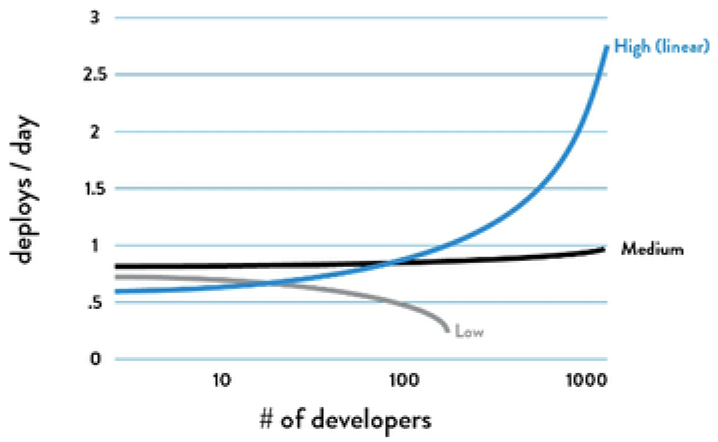
- Overview
    - Definition: "an automated communications process by which measurements and other data are collected at remote points and are subsequently transmitted to receiving equipment for monitoring."
    - Scott Prugh, Chief Architect and Vice President of Development at CSG, said, " Every time NASA launches a rocket , it has millions of automated sensors reporting the status of every component of this valuable asset . And yet, we often don't take the same care with software — we found that creating application and infrastructure telemetry to be one of the highest return investments we've made. In 2014 , we created over one billion telemetry events per day, with over one hundred thousand code locations instrumented."
    - Emitting telemetry that can be analyzed, whether it is in our application, database, or in our environment, and making that telemetry widely available, we can find and fix problems long before they cause something catastrophic, ideally long before a customer even notices that something is wrong.
    - Data collection at the business logic, application, and environments layer: in each of these layers, we are creating telemetry in the form of events, logs, and metrics.
- Levels/Examples of Telementry
    - DEBUG level: Information at this level is about anything that happens in the program, most often used during debugging. Often, debug logs are disabled in production but temporarily enabled during troubleshooting.
    - INFO level: Information at this level consists of actions that are user-driven or system specific (e.g., "beginning credit card transaction ").
    - WARN level: Information at this level tells us of conditions that could potentially become an error (e.g., a database call taking longer than some predefined time). These will likely initiate an alert and troubleshooting, while other logging messages may help us better understand what led to this condition.
    - ERROR level: Information at this level focuses on error conditions (e.g., API call failures, internal error conditions).
    - FATAL level: Information at this level tells us when we must terminate (e.g., a network daemon can't bind a network socket).
    - Test level: Count of automated tests, velocity, incident reports, continuous integration status
    - Business level: number of sales transactions, revenue of sales transactions, user signups, churn rate, A / B testing results, etc
    - Application level: transaction times, user response times, application faults, etc.
    - Infrastructure level (e.g., database, operating system, networking, storage): Examples include web server traffic, CPU load, disk usage, etc.
    - Client software level (e.g., JavaScript on the client browser, mobile application): Examples include application errors and crashes, user measured transaction times, etc.
    - Deployment pipeline level : Examples include build pipeline status (e.g., red or green for our various automated test suites), change deployment lead times, deployment frequencies, test environment promotions, and environment status.
    - Application Events
        - Authentication / authorization decisions (including logoff)
        - System and data access
        - System and application changes (especially privileged changes)
        - Data changes, such as adding, editing, or deleting data Invalid input (possible malicious injection, threats, etc.)
        - Resources (RAM, disk, CPU, bandwidth, network usage, or any other resource that has hard or soft limits)
        - Health and availability Startups and shutdowns
        - Faults and errors Circuit breaker trips Delays Backup success / failure
- Telementry Statistical Alerts
    - A common use of standard deviations is to periodically inspect the data set for a metric and alert if it has significantly varied from the mean. For normal distributions.

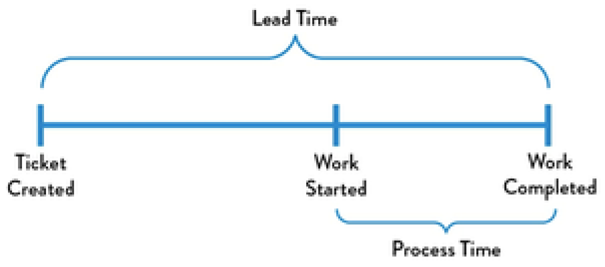
        - Fast Fourier Transform (FFT) and linear regression to smooth the data while preserving legitimate traffic spikes that recur in their data. One of the statistical techniques we can use is called smoothing, which is especially suitable if our data is a time series, meaning each data point has a time stamp.
        - Fast Fourier Transforms, which has been widely used in image processing, and the Kolmogorov-Smirnov test (found in Graphite and Grafan), which is often used to find similarities or differences in periodic / seasonal metric data. These techniques compare two probability distributions, allowing us to compare periodic or seasonal data, which helps us find variances in data that varies from day to day or week to week.”
    - Figure 26 : Monitoring framework
        - 

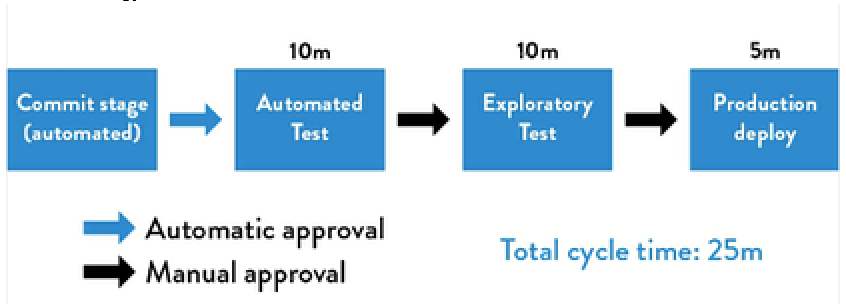
Metrics

- Frequency of code deployments: Lines of code, # of commits, time to go go from "committed" to "successfully running in production."
    - Figure 24: Blackboard Learn code repository: after Building Blocks
        - 

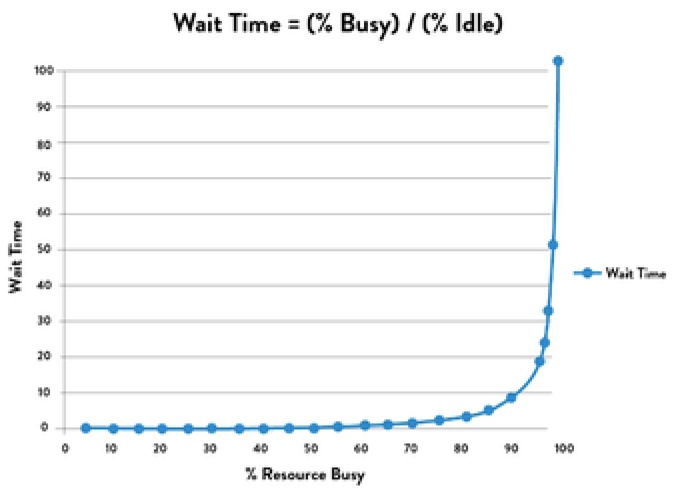
        - The graph above shows the connection between the exponential growth in the number of lines of code and the exponential growth of the number of code commits for the Building Blocks code repositories. The new Building Blocks codebase allowed developers to be more productive, and they made the work safer because mistakes resulted in small, local failures instead of major catastrophes that impacted the global system.
    - Deploys per day to # of Developers (Linear line)

- 

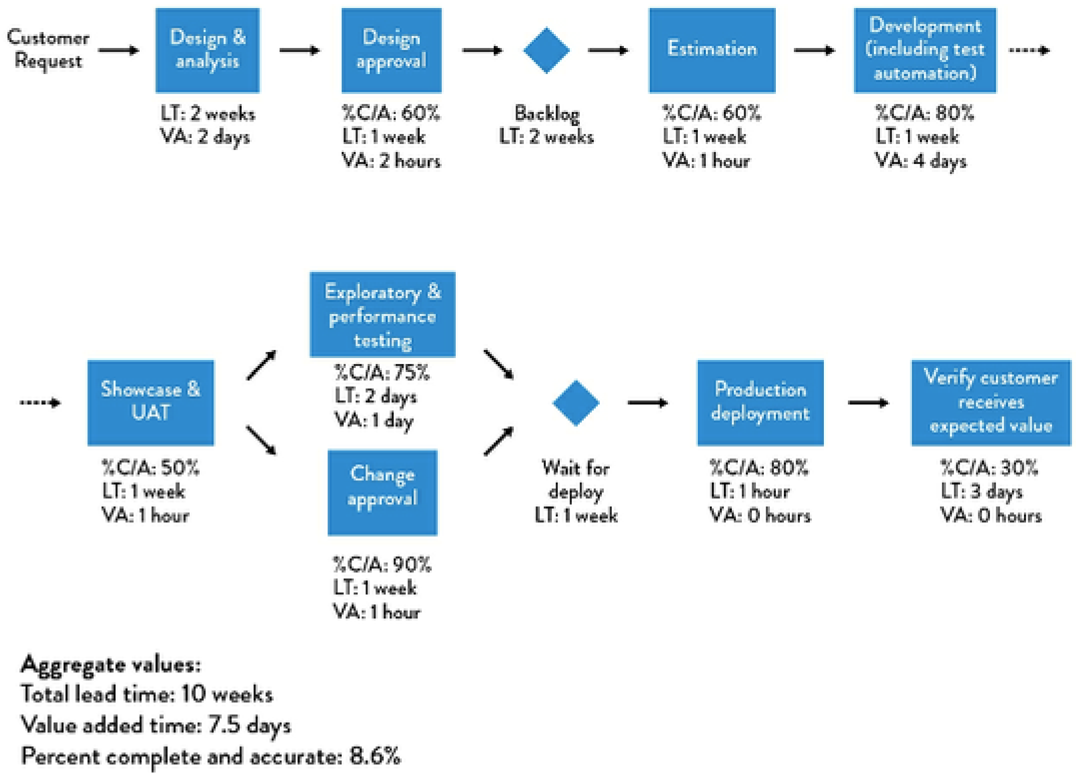
    - Example: 40,000 code commits / day 50,000 builds / day (on weekdays, this may exceed 90,00), 120,000 automated test suites, 75 million test cases run daily 100 + engineers working on the test engineering.
- **Lead Time, Process Time (Value-Added Time), Cycle Time, Percent Complete & Accurate (%C/A)**
    - Lead Time: WIP is biggest indicator
    - Figure 2 . Lead time vs . process time of a deployment operation
        - 

    - Figure 4 : A technology value stream with a lead time of minutes
        - 

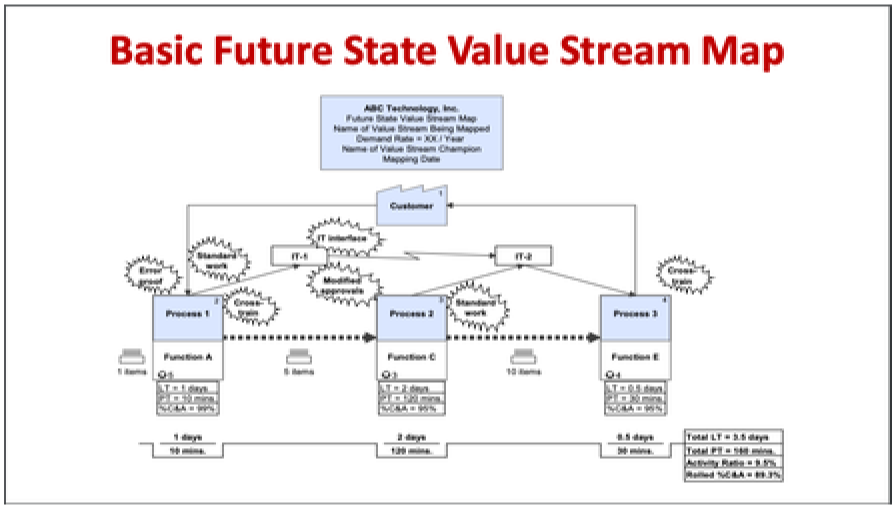
    - Utilization - Queue size and wait times as function of percent utilization
        - As a work center approaches 100% utilization, any work required from it will languish in queues and won't be worked on without someone expediting/escalating.
        - If a resource is fifty percent busy, then it's fifty percent idle. The wait time is fifty percent divided by fifty percent, so one unit of time. Let's call it one hour. So, on average, our task would wait in the queue for one hour before it gets worked.
        - Figure 47: Queue Size and Wait Times as Function of percent utilization: Shape of the line shows is that as resource utilization goes past 80%, wait time goes through the roof.

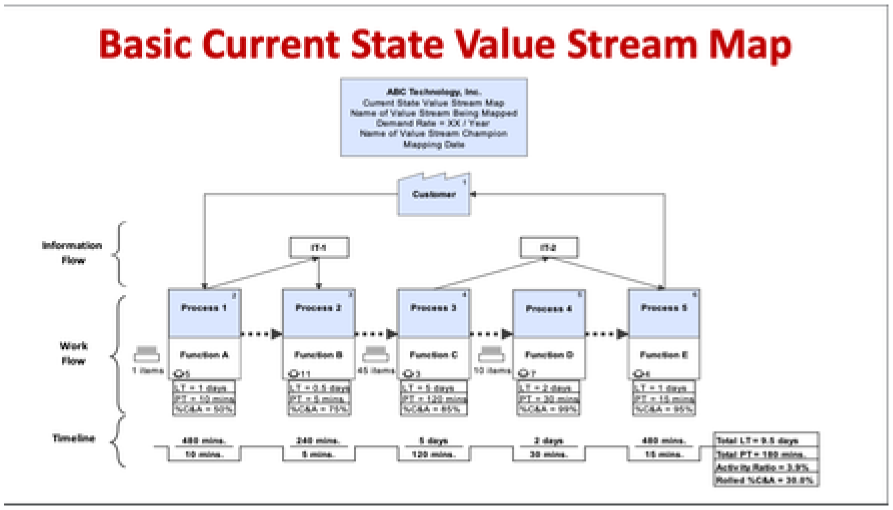


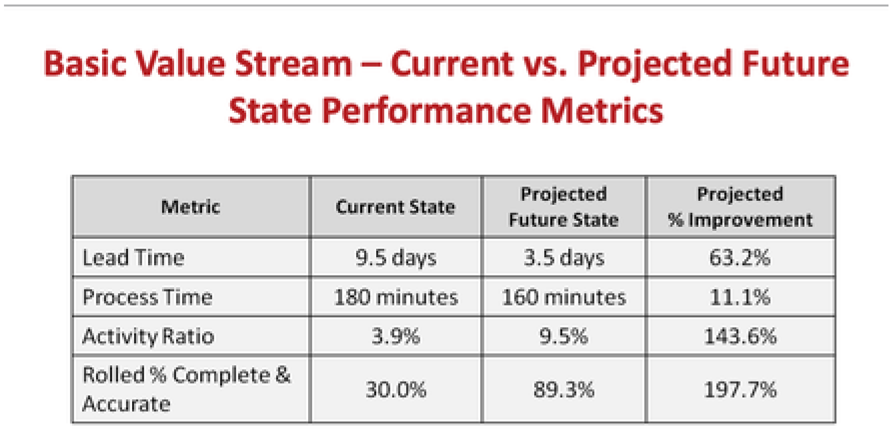
-
- Assuming that all work centers were 90% busy, the figure shows us that the average wait time at each work center is nine hours — and because the work had to go through seven work centers, the total wait time is seven times that: sixty-three hours. In other words, the total % of value added time (sometimes known as process time) was only 0.16% of the total lead time (thirty minutes divided by sixty-three hours). That means that for 99.8 % of our total lead time, the work was simply sitting in queue, waiting to be worked on.
    - 1 Work Center: 90% Busy / 10% Idle = 9hr average wait time before it gets worked
    - Work has to go though 7 work centers = 9hr x 7 = 63hr wait time
    - Process or Value Added Time Taken to Complete: 30 minutes
- Figure 10: An example of a value stream map



- https://www.leanconstruction.org/media/docs/chapterpdf/san-diego/LCI-San_Diego_CoP_Value-Stream-Mapping012116.pdf
    - Note: Not from DevOps Handbook but Relevant in Explaining Above (Figure 10)

- 

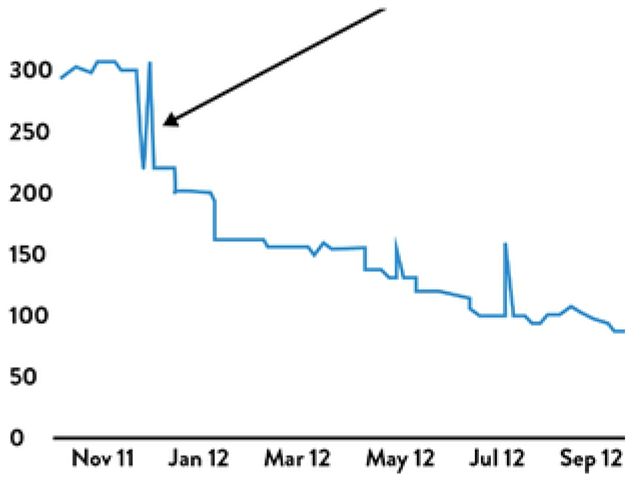
- 

- 

Basic Value Stream – Current vs. Projected Future State Performance Metrics

Metric	Current State	Projected Future State	Projected % Improvement
Lead Time	9.5 days	3.5 days	63.2%
Process Time	180 minutes	160 minutes	11.1%
Activity Ratio	3.9%	9.5%	143.6%
Rolled % Complete & Accurate	30.0%	89.3%	197.7%

- Disruptive Metrics
  - MTTR: Mean Time to Repair
  - service outages
  - service impairments
  - security or compliance failures
    - Figure 44: Number of Brakeman security vulnerabilities detected

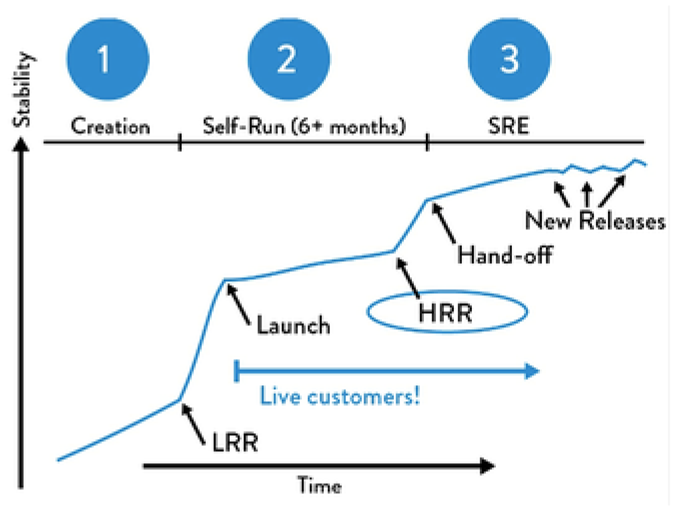


-
    - Development velocity (i.e., speed of delivering features to market)
    - Failed customer interactions (i.e., outages, errors)
    - Compliance response time (i.e., lead time from audit request to delivery of all quantitative and qualitative information required to fulfill the request)."
    - Event severity: How severe was this issue?
    - Total downtime: How long were customers unable to use the service to any degree?
    - Time to detect: How long did it take for us or our systems to know there was a problem?
    - Time to resolve: How long after we knew there was a problem did it take for us to restore service?

- Number of Meetings & Work Tickets
    - " A great metric to publish widely is how many meetings and work tickets are mandatory to perform a release — the goal is to relentlessly reduce the effort required for engineers to perform work and deliver it to the customer."

<u>Tools/Resources/Techniques</u>

- Dark Launches: For feature toggles that deploy all the functionality into production and then perform testing of that functionality while it is still invisible to customers.
- Telementry
    - Ganglia (into Graphite) - aggregating metrics together, everything from business metrics to deployments.
    - Nagios, Zenoss, collectd, AppDynamics, New Relic, Dynatrace, and Pingdom.
    - StatsD
        - " We designed StatsD to prevent any developer from saying, 'It's too much of a hassle to instrument my code.' Now they can do it with one line of code.
        - StatsD can generate timers and counters with one line of code and is often used in conjunction with Graphite or Grafana, which renders metric events into graphs and dashboards.
- Chat Rooms
    - Having this work performed by automation in the chat room (as opposed to running automated scripts via command line) had numerous benefits , including :
    - Furthermore , beyond the above tested benefits , chat rooms inherently record and make all communications public ; in contrast , emails are private by default , and the information in them cannot easily be discovered or propagated within an organization .
    - " There was no physical water cooler at GitHub . The chat room was the water cooler . "
- For instance , we can use tools such as Rundeck to automate and execute workflows , or work ticket systems such as JIRA or ServiceNow .
- Tools such as Gauntlt have been designed to integrate into the deployment pipelines , which run automated security tests
- (e.g . , Remedy or ServiceNow) .
- The LRR and HRR checklists are a way to create that organizational memory . "
    - Figure 39 : The " Launch readiness review and hand - offs readiness review " at Google

- 
1
2
3
Stability
Creation
Self-Run (6+ months)
SRE
New Releases
Hand-off
Launch
HRR
Live customers!
LRR
Time