Continuous Integration, Continuous Delivery, and Continuous Monitoring Overview

David Shepard

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

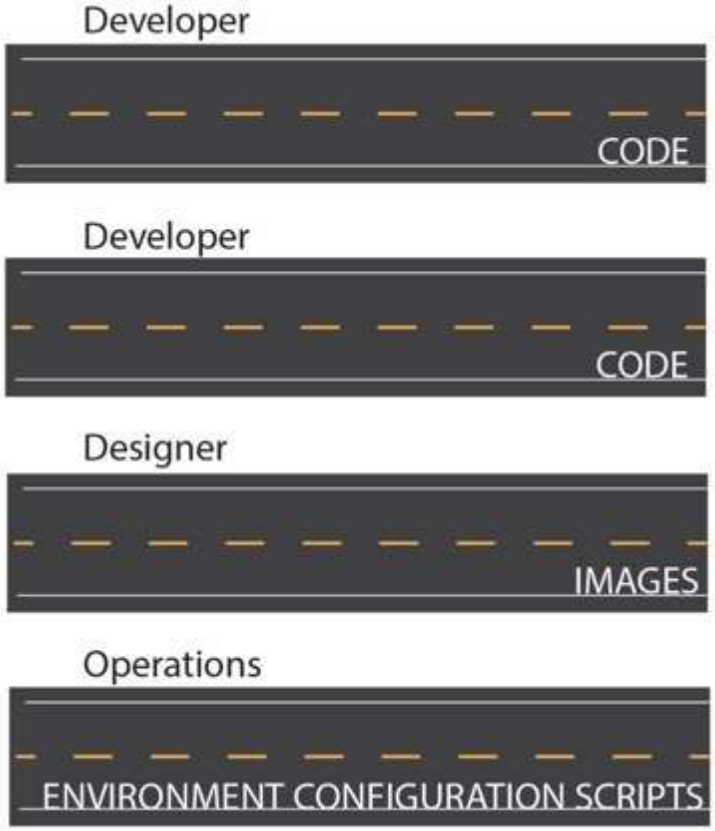
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-1087

Continuous Integration, Continuous Delivery, and Continuous Monitoring Overview

Continuous Integration

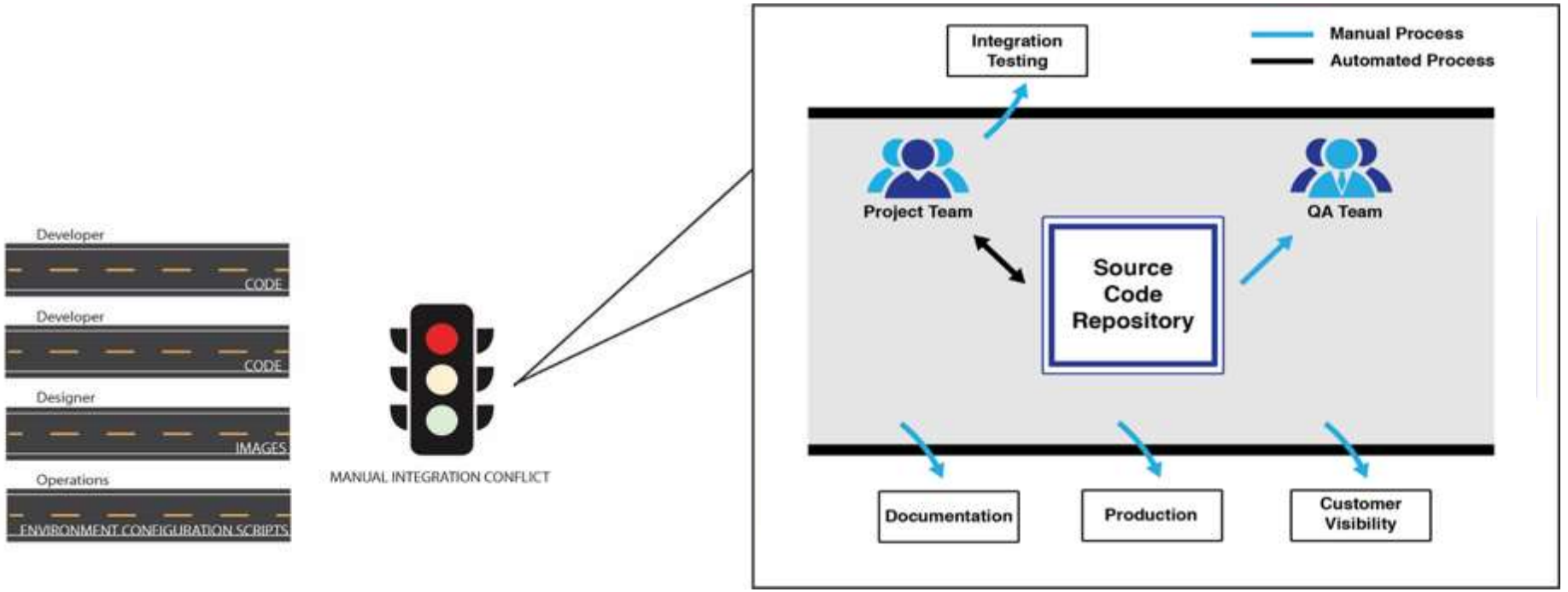
Integration Can Be Challenging



Software projects consist of many artifacts that must be manually integrated.

MANUAL INTEGRATION CONFLICT

Integration Is Often a Manual Process



Manual Integration Is Flawed

Human-driven processes are…

- infrequent
- expensive
- repetitive
- error-prone.

This leads to

- disjointed activities / components
- slow, unreliable, costly reporting and failure recognition
- lack of transparency of problems
- integration Hell.

Automating Integration Fixes These Issues

Automation…

removes inefficiencies due to human-driven process

standardizes the artifact submission process

guarantees consistent results

allows teams to fail fast (and fix fast)

reduces the pain of integration.

Continuous Integration Is Even Better

Continuous integration uses a build server to…

integrate artifacts on every change

give teams immediate notification of failure or success

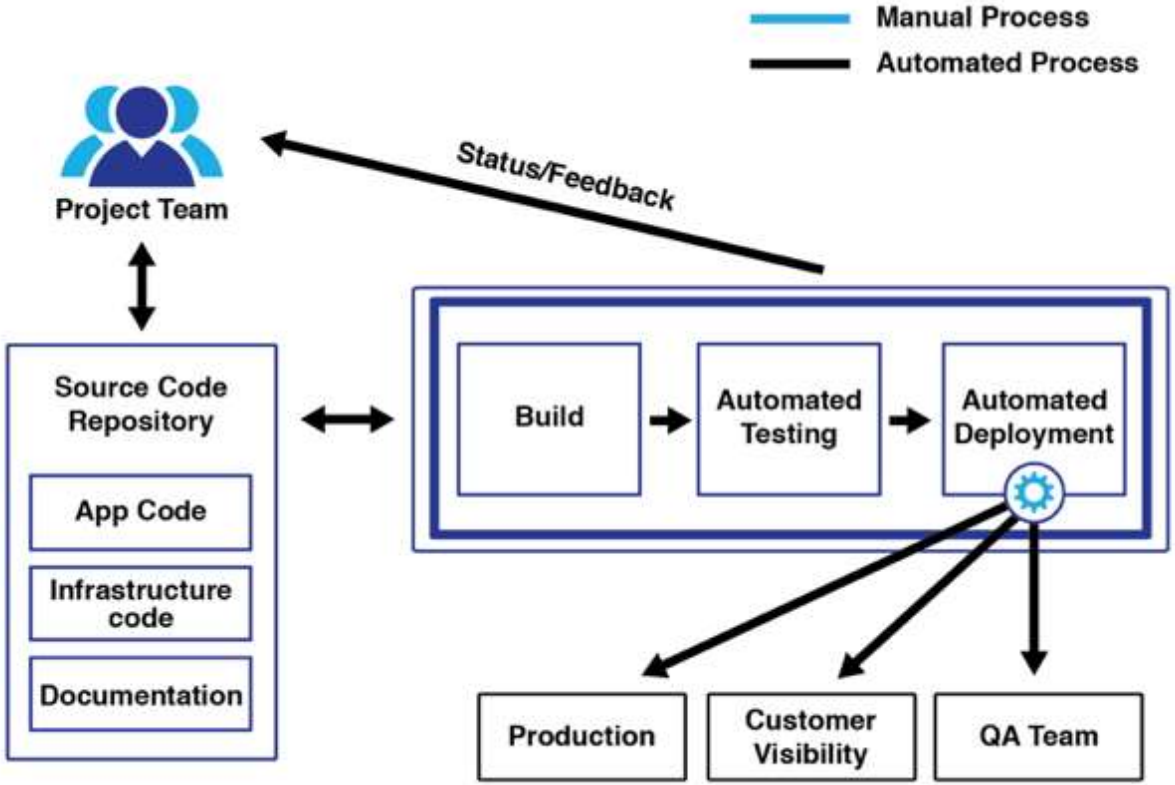
require issues to be fixed before moving forward

enforce standards (can fail based on quality as well as functionality).

What Is Continuous Integration?

Continuous integration is a process that continually merges a system's artifacts, including source code updates and configuration items from all stakeholders on a team, into a shared mainline to build and test the developed system.

Continuous Integration (CI) Model



Fail the Build When Software is Not Good Enough

- Don’t just configure failure for compile/build errors!
- Want 90% test coverage? Fail the build if code base is <90% covered
- Want all DB queries < 2sec? Test them, and fail the build otherwise
- Want to make sure code conforms to style guide? You guessed it…

CI is your best tool to enforce quality standards.

Continuous Integration Requires Some Discipline…

For successful implementation of a CI process:

- Developers must commit changes often
- CI system should build every commit
- Automate every step of the build process
- Automate tests, and fail the build on test failure
- CI system should report results immediately to everyone
- CI system should instantly revert to previous release on failure
- All environments should have 100% parity

…But Yields Significant Rewards

Lower costs

- Immediate detection of problems
- Removal of human integration labor

Greater visibility for everyone

- Developers
- Operations
- Quality Assurance
- Management
- Customers

Increased confidence in your software

Important Points to Remember: Continuous Integration

- **Early integration of any developed code**
- **Build and deliver continuously**
- **Not just code build other artifacts (Documents, IaC, Deployment scripts, test results)**
- **Integrate test automation into build process**
- **Have multiple CI pipeline on various code branch or modules**
- **Share the results across all team members**

Continuous Delivery and Deployment

Clarifying Terms

Continuous deployment is the automated process of deploying changes to production by verifying intended features and validations to minimize risk.

Continuous delivery is a software engineering practice that allows for frequent releases of new software to staging or various test environments through the use of automated testing.

Deployment Is a Hot Topic

2009 Flickr.com famously reported deploying code 10+ times per day

2011 Amazon had a new release to production every 11.6 seconds in

6.2 millisecond in Aug 2020 (~23K per day)

50M deployments per year!

2014 Etsy deployed 80–90 times per day

And others, Target, Walmart, Facebook, Netflix….

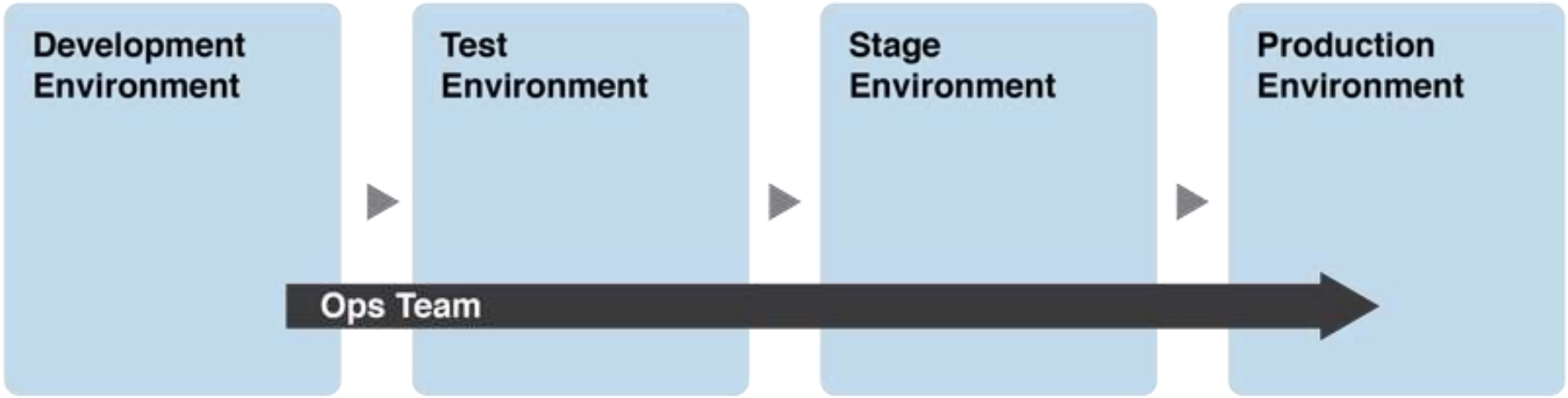
Not Everyone Needs to Achieve Continuous Deployment

Your DevOps goals should be designed around **business needs.**

Do daily deployments give you a competitive edge?

If not, what does?

Shift Operational Concerns Left with Continuous Delivery



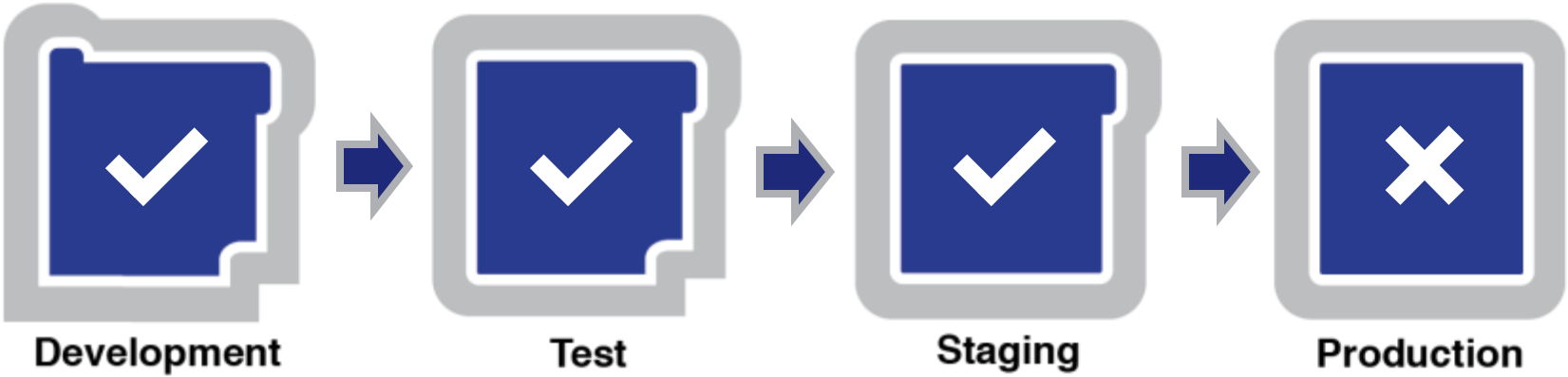
Gold-standard deployment

- Environment Parity
- Process Parity
- Automated
- Perform incremental changes
- Appropriate Testing

Environment Parity

Environment parity: Avoiding inconsistencies between environments will minimize risks

Process parity: Use the same process to deploy to development, test, and production



Automation

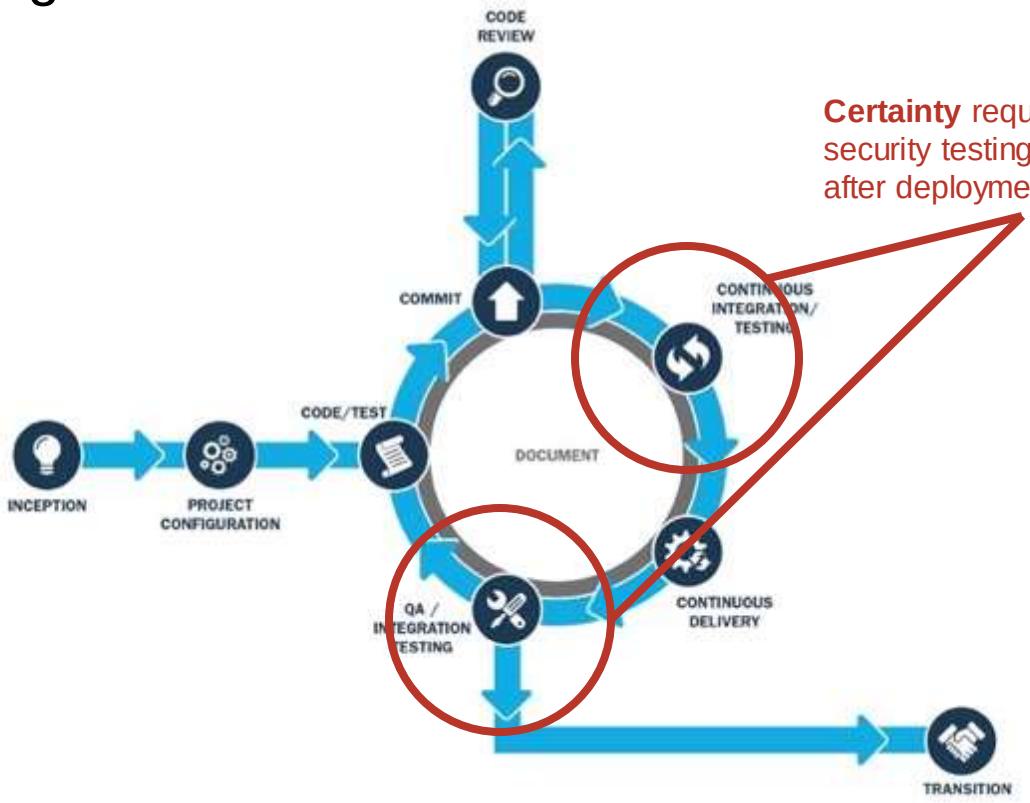
- Reduces manual labor
- Reduces human error
- Enforces parity between processes / environments
- Helps scale
- Enables auditability, visibility, and traceability
- Builds in security via environment parity

Incremental Changes

- Smaller changes to production can be less risky than large ones

- Decrease the number of potential defects that could cause an outage

Continuous Delivery is *really* about Rigorous Security Testing



Certainty requires rigorous security testing before and after deployment

Test Enough that you are sure you could Deploy Successfully:

- **How much is enough?**
- **What factors are important to you and your organization?**
  - Security? Automate a large number of security-focused unit/integration tests.
  - Scalability? Automate high-volume load testing.
- **Design your CI/CD success criteria to enforce your goals.**

Getting Started with Continuous Delivery: *Walk before you run*

- Start with a low risk module
- Start out-of-band with production
- Automate build, testing, and deployment
- Deploy to test environments first
- Measure results

Deployment Strategies

- Two basic all-or-nothing strategies:
  - **Red/black:** Leave *N* instances with Version A as they are, allocate and provision *N* instances with Version B, and then switch to Version B and release instances with Version A.
  - **Rolling upgrade:** Allocate one instance, provision it with Version B, and release one instance of Version A. Repeat *N* times.
- Partial strategies are canary testing and A/B testing.
- Roll back for any deployed services.

Deployment Strategies: Rollback

- New versions of a service may be unacceptable either for logical or performance reasons.
- Two options in this case:
  - Roll back -undo deployment
  - Roll forward- discontinue current deployment and create a new release without the problem

•Decision to rollback can be automated; some organizations do this.

•Decision to roll forward is never automated because there are multiple factors to consider.

- Forward or backward recovery
- Consequences and severity of problem
- Importance of upgrade

Deployment Strategies: Feature toggle

- Place feature-dependent new code inside of an “if” statement where the code is executed if an external variable is true. Removed code would be in the “else” portion.
- Used to allow developers to check in uncompleted code. Uncompleted code is toggled off.
- During deployment, until new code is activated, it will not be executed.
- Removing feature toggles when a new feature has been committed is important.

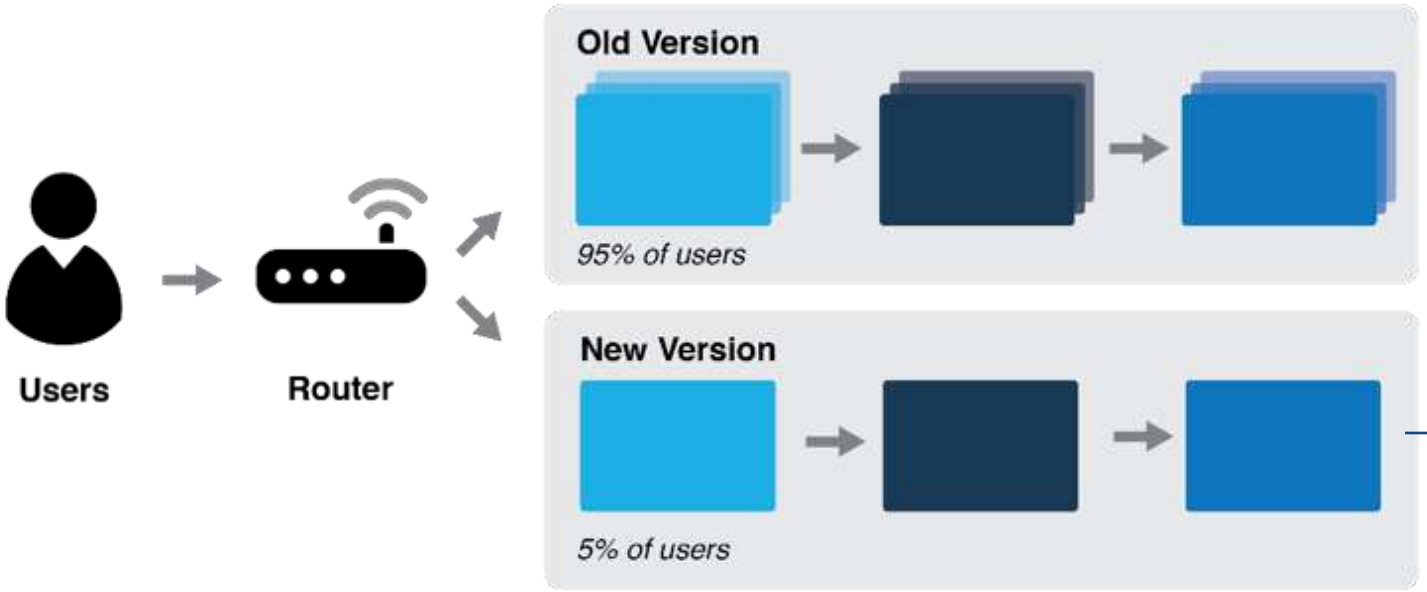
Feature Flag Mitigates Risk

- Mitigating risk of letting developers push out code or releasing more frequently
- Feature Flag pattern
- All code is deployed ASAP
- Feature flags configure which users see what without redeployment
- Allows piloting and experimentation
- Enables “dark launch”
  - A dark launch means that a new product or update is pushed out to a subset of users. Often in a dark launch, users are not aware they are testing a new product.

Canary Release/Testing

- Canaries are a small number of instances of a new version placed in production to perform live testing in a production environment.
- Canaries are observed closely to determine whether the new version introduces any logical or performance problems. If not, roll out new version globally. If so, roll back canaries.
  - Named after canaries in coal mines
  - Equivalent to beta testing for shrink-wrapped software

Canary Release Mitigates Risk



The new version will be observed for logical or performance problems before a global rollout begins.

Important Points to Remember: Continuous Delivery

- **Deliver the build as often as possible**
- **Deploy the build when there is a need**
- **Deliver the build as small as possible with less impact to the other modules**
- **Think about deployment strategy based on technical requirements**
- **Use container (docker) or similar technology**

Continuous Integration, Continuous Delivery, and Continuous Monitoring Overview

Continuous Monitoring

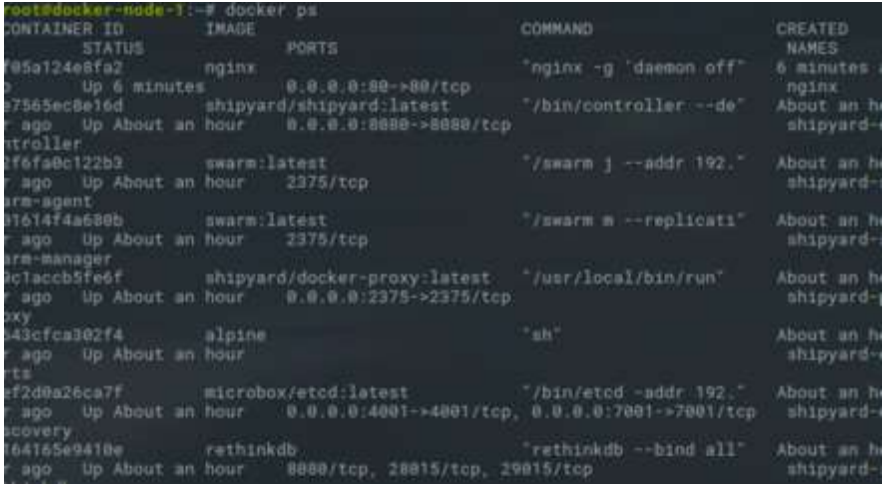
Metrics, Logs, Reports → Data

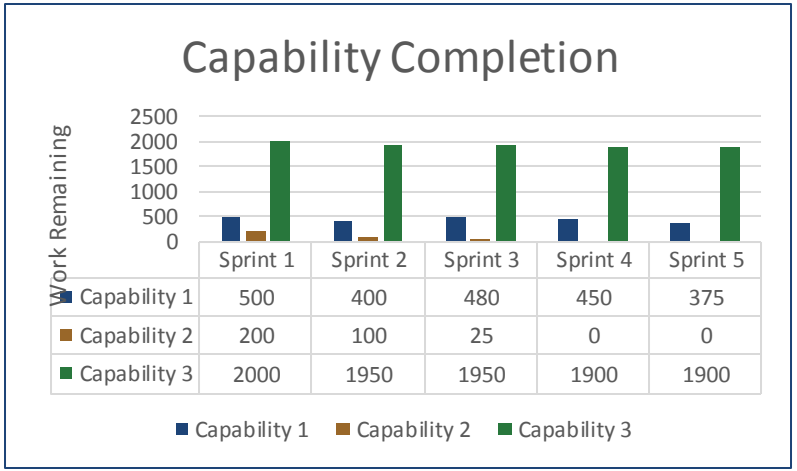
Defining Key Terms

- ***Logs***: a document used to record and describe selected items identified during execution of process or activity (PMBBOK)
- **Metrics**:
  - (1) quantitative measure of the degree to which a system, component, or process possesses a given attribute IEEE 24765
  - (2) defined measurement method and the measurement scale (ISO 14102)
- ***Reports***: information item that describes the results of activities such as investigations, observations, assessments, or tests (IEEE 15289)

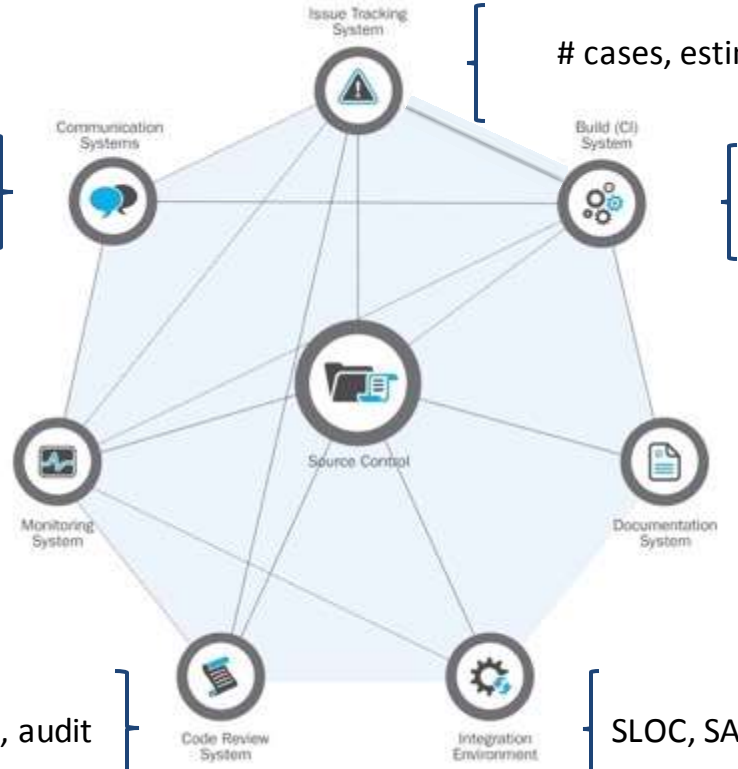
Logs vs metrics

While ***logs*** are about a specific event, ***metrics*** are a measurement at a point in time for the system.





All are in….



cases, estimates, task status, etc

Status report, # of messages

Build ,Integration, test results, status report

Aggregated data visualization

Version, logs, audit

Secure coding, logs, audit

SLOC, SAST

DevOps metrics

- Without metrics there is no way to know if you are improving in your performance of processes to answer :
  - Is the service delivering value to the users?
  - Is the service operating properly?
  - Are we achieving business goals?
  - Is the service secure?
  - Is the infrastructure adequate?
  - Is the service being attacked?
  - Can future needs be supported?
  - Are we able to plan new product? If so how much?
  - Are we compliant?

QUANTIFY THE IMPACT OF INVESTMENT IN SOFTWARE DELIVERY

REGARDLESS OF ORGANIZATION SIZE, the research showed most software delivery teams are not able to quantify the impact of investments they have made to their ability to deliver software.

- Inability to break down cost of delivering software (fast or delay)
- Inability to trace the cost of defects to business impact
- No clarity on benefits of future investments
- Inability to track cost Developing new features or fixing defects

*State of Software Delivery Management Report 2020

Why?

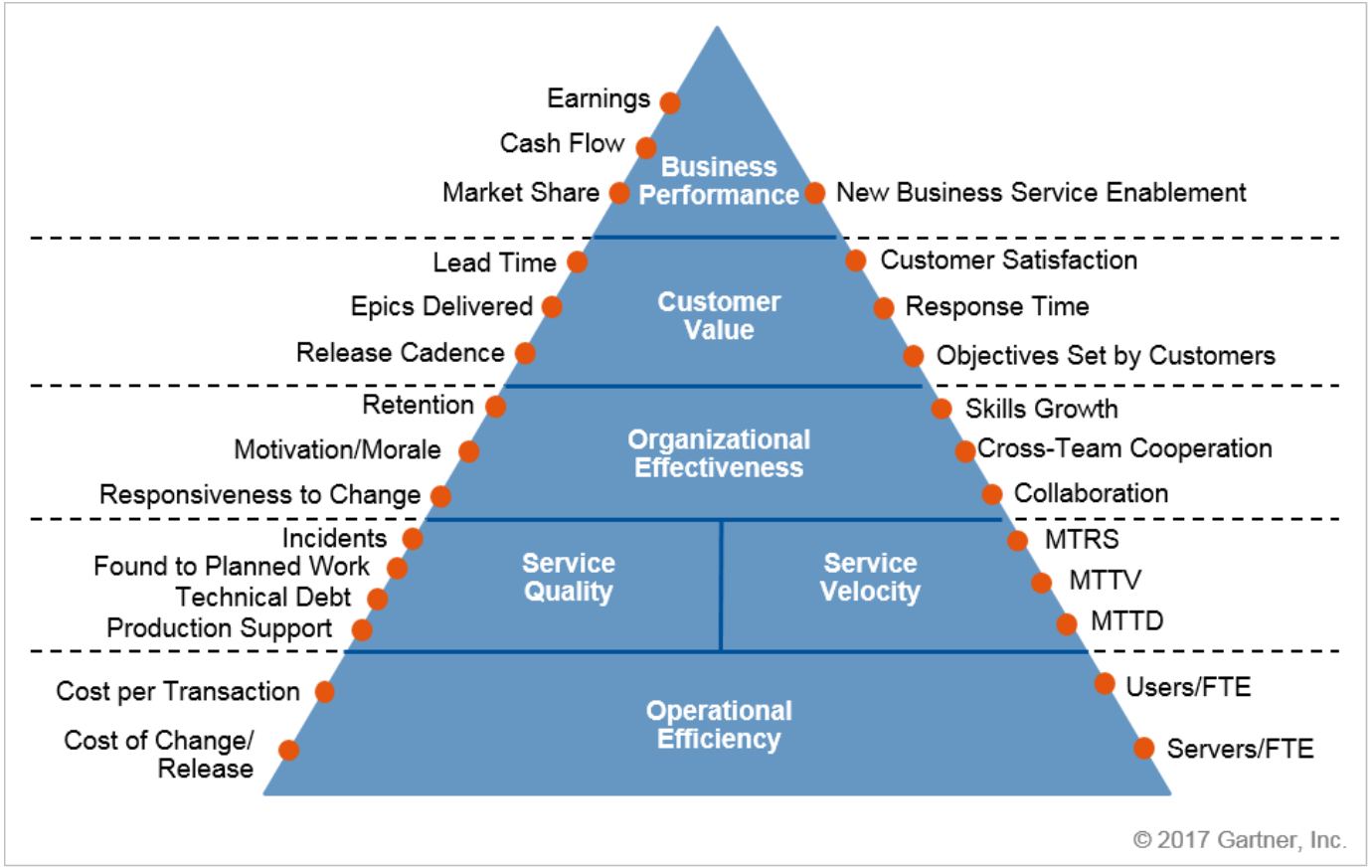
"When you can measure what you are speaking about, and can express it in numbers, you know something about it: but when you cannot measure it, when you cannot express it in numbers, your knowledge is of meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science"

Lord Kelvin, a physicist

Characteristics of good metrics

- ***Relevant***: It must be related to a business goal.
- ***Observable***: A metric that can't be measured is useless.
- ***Actionable***: It should suggest the need for corrective actions or improvements to workflows, policies, incentives, tools, etc.
- ***Traceable***: It should be possible to causally trace the metric to root causes.
- ***Reliable***: It should produce similar results under similar conditions and resist manipulation.
- ***Automatable***: Metrics collection should be built into the system to avoid manual work, errors, and delay.
- ***Auditable***: It should be free from any teams/person influence
- ***Collectible***: It should be rationalized form examining related metrics

DevOps metrics pyramid



Guidelines to Selecting Metrics:

- Avoid relying on a single measure.
- Look for trends, outliers, and level shifts rather than averages.
  - Trends indicate an imbalance.
  - Outliers indicate process misbehavior.
  - Shifts indicate an underlying change.
- Outcome measures directly affect the customer or the business, while
- Process measures help you understand and improve.

When to measure?

- **The pipeline includes key transitions and events**
  - Bug Report Submitted
  - Change Request Submitted
  - Code Commit
  - Build progress
  - Test results
  - Deployment activities
  - Operating Failure or Recovery
- **Continuous monitoring for assurance**
  - Application usage and latency (processors, memory)
  - Network traffic (volume and source)

A typical Measurement-metrics categories

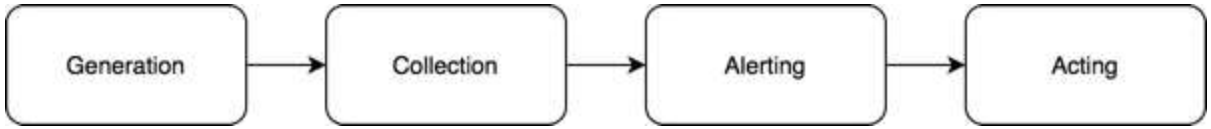
- ***Productivity*** (like: Deployment Frequency, Lead times)
- ***Reliability*** (Like: MTTR, MTTD, Time to Failure)
- ***Quality*** (like: Failed Deployments, # of tickets, Rework)
- ***Security*** (like: Change Request, time to approval, #incidents)
- ***Operations*** (like: resource usage, attack frequency)
- **The measurements are typically used to:**
  - Identify problems or change to a baseline.
  - Compare actual performance to a desired level.
  - Evaluate the effect of a change.

Metrics	Description	Associated Domain (s)
Deployment frequency	Number of deployments to production in a given time frame	Application Deployment; Authority to Operate Processes
Change lead time (for applications)	Time between a code commit and production deployment of that code	Overarching; Authority to Operate (AO) Processes; Patch Management
Mean time to recovery (MTTR) (for applications)	Time between a failed production deployment to full restoration of production operations	Application Deployment; Backup and Data Lifecycle and Patch Management
Change failure rate	Percentage of production deployments that failed	Application Deployment
Availability	Amount of uptime/downtime in a given time period, in accordance with the SLA	Availability Performance and Network Management; Network
Change volume (for Applications)	Number of user stories deployed in a given time	Overarching
Customer issue volume	Number of issues reported by customers in a given time period	Overarching
Customer issue resolution time	Mean time to resolve a customer-reported issue	Overarching
Time to value	Time between a feature request (user story creation) and realization of business value from that feature	Overarching; AO Processes
Time to ATO	Time between the beginning of Sprint 0 to achieving an ATO	Overarching; AO Processes
Time to patch vulnerabilities	Time between identification of a vulnerability and successful production deployment of a patch	Authority to Operate Processes

Monitoring

Monitoring

- Monitoring determining the status of a system, a process, or an activity (ISO/IEC)
- Monitoring is the collection, interpretation, and action on information gathered from a system, from its design and implementation to running in production. It translates metrics into a measurable user experience.



Infrastructure, pipeline or system information is generated based on an observed threshold

Monitoring system collects the generated information

Monitoring system displays information or alerts

Designated person (or possibly monitoring system) takes action

Reasons for Monitoring

Area	Reasons
Development	• Improve DevOps processes
Usability (UX)	• Measure user reactions to various aspects of the system
Performance /Application	• Identify failures • Identify performance problems • Characterize workload for short and long-term capacity planning, as well as for charging purposes
Security	• Detect intruders • Identify data breaches • Identify vulnerabilities
Business Process	• System management • Lifecycle management • KPI, Business Value
Functional Monitoring	• Monitoring of each case or set-of use cases • Various Deployment cases

Types of Monitored Issues

Phase	Issues
Development	• Build failures • Testing failures • Issue monitoring
Operations	• Product monitoring / System expansion forecasting • Outage monitoring • resource usage, attack frequency
Security	• Vulnerability monitoring • Predicting future anomalies • Change Request, time to approval, #incidents

Monitoring Implementation

When implementing monitoring in a system, one must add:

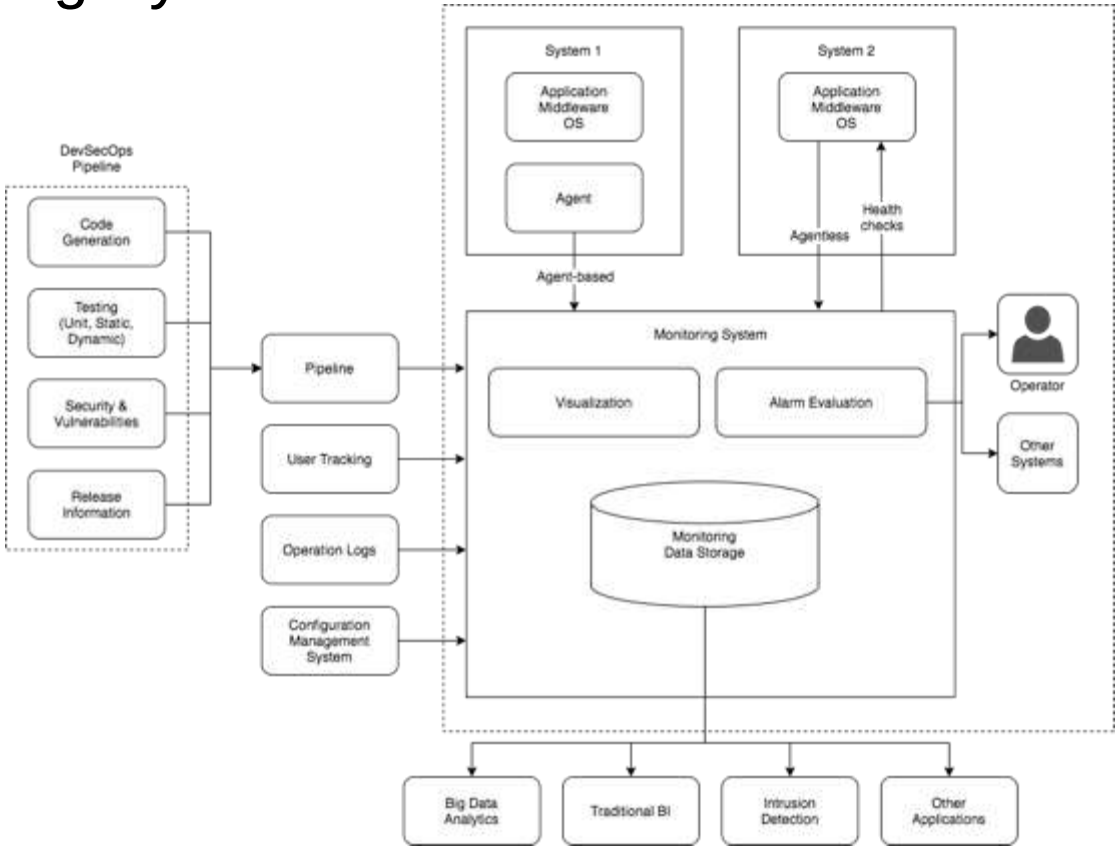
- **Instrumentation**: Code or routine added to main application code, to retrieve relevant information about a certain feature of the system.
- **Telemetry**: An automated communications process by which measurements and other data are collected and transmitted to receiving equipment for monitoring.
- **Storage**: Any mechanism to store the captured information, and pass it on to displaying or alerting capabilities.
- **Displaying**: A mechanism to display the collected information, usually through visual metaphors that allow fast and intuitive understanding of what is happening to the system.
- **Alerting**: Any mechanism to indicate that something is not normal and needs attention.

Below are some useful resources for storing and displaying monitoring data:

COTS	Niche	Custom
Elastic ELK Stack (storing and displaying)	VirtualDevManager* (storing and displaying)	D3 Visualizations (displaying)

*http://virtualdevmanager.com/

Monitoring System Architecture



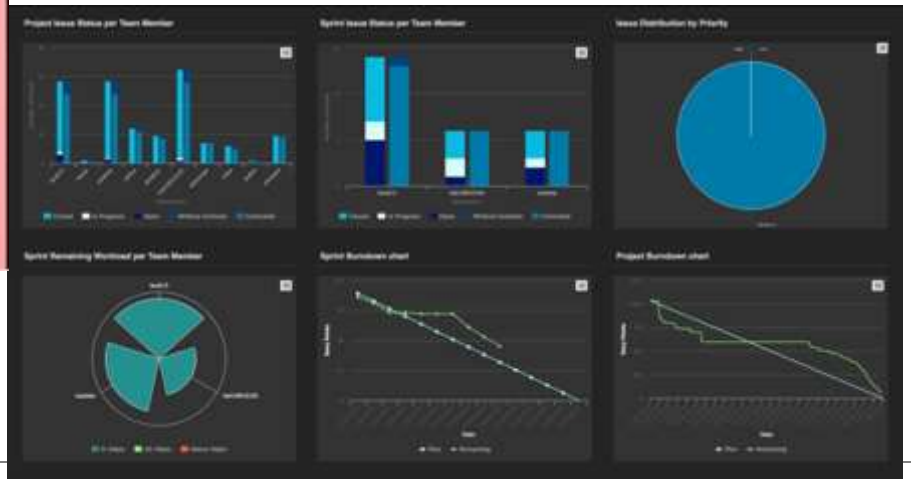
*Partially based on “DevOps: A Software Architect's Perspective” (SEI Series in Software Engineering), Len Bass, Ingo Weber, Liming Zhu

Design Dashboard

- **Valid Information**
  - What's the source of the data?
  - Has the process for collecting or reporting changed?
  - What's changing? What's different?
- **Actionable Choices**
  - What's the significance of the information?
  - Does it represent a risk or problem?
  - What decision is required?
  - What actions are appropriate?
- **Follow-Through**
  - What will you monitor after the action? How often?
  - Can you get the monitoring data you need?
  - Who is responsible?

Different roles need different information

Dashboards can hold a large amount of information and are good in displaying outliers to expected behaviors.



Acquisition, product development, and programs make many assumptions.

Thresholds

- Monitoring systems have "**thresholds**" that cause alerts and alarms.
- Problems with thresholds include false positives and false negatives
  - **False positives** – if thresholds are too often violated alarms will be sent too frequently and operators will ignore them (crying wolf)
  - **False negatives** – if thresholds are not appropriately set, problems will not be detected in time to take corrective action

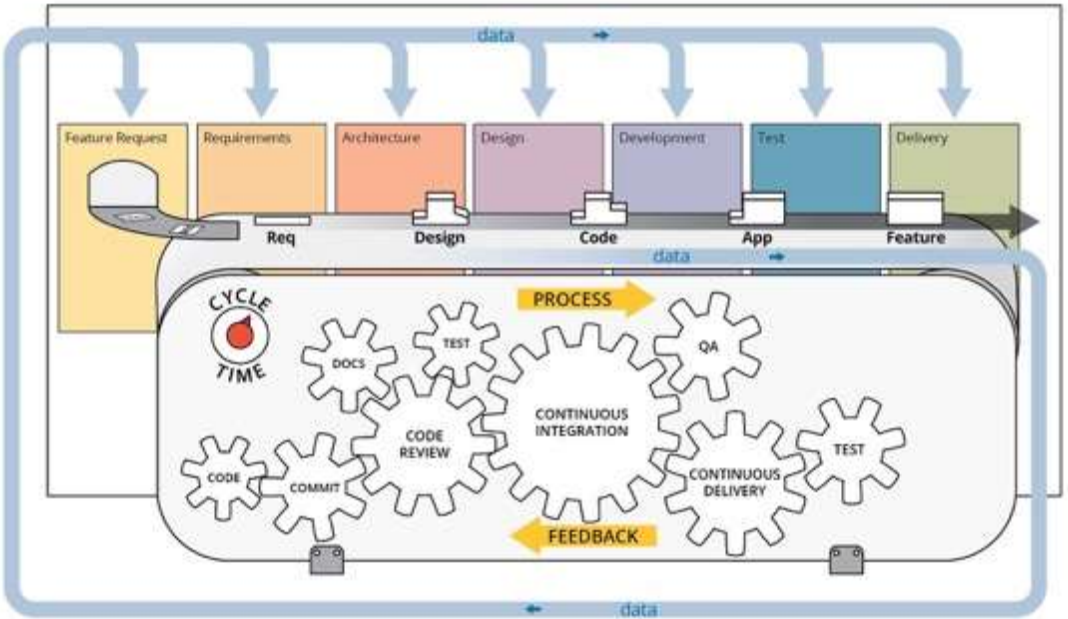
Static and Dynamic Thresholds

- **Thresholds can be fixed at particular values.**
  For example, network segment utilization over 80% triggers an alarm
- **Thresholds can be time varying.**
  For example, CPU utilization over 20% during 9:00pm–5:00am triggers an alarm
- **Thresholds can be based on historical data.**
  For example, given this number of requests, expect the following CPU/network utilization (machine learning techniques are used)
- **Thresholds can be adjusted based on context.**
  For example, adjust thresholds when deployment is ongoing (research area)

Alerts/Alarms

- **The Monitoring system will trigger alerts or alarms**
- **An alert is an indication that something is not normal.**
  - For example, temperature is too high.
  - Alerts do not yet require immediate action but are something to pay attention to.
- **An alarm requires immediate attention.**
  - For example, there is a fire.
  - Alarms may trigger pagers

Summary:



- Map metrics in each DevOps critical Success
- Monitor the Impact of Measurement Systems
- Create a Learning Environment to Process Improvement
- Avoid vanity metrics that promote quantity or speed over quality
- Avoid conflict metrics that promote individuals rather than teams

Important Points to Remember: Continuous Monitoring

- **Collect data from every tools in the DSO pipeline**
- **Monitor data based on metrics matters to you**
- **Learn and estimate accurately based on fact**
- **Think about possibility of using data for any future implementation like Are you ready for AI?**
- **Use any monitoring tech and tools**