

---

# DevOps

These notes (except the last slide) are from:

Jason Baker

Adjunct Instructor

Graduate Programs in Software

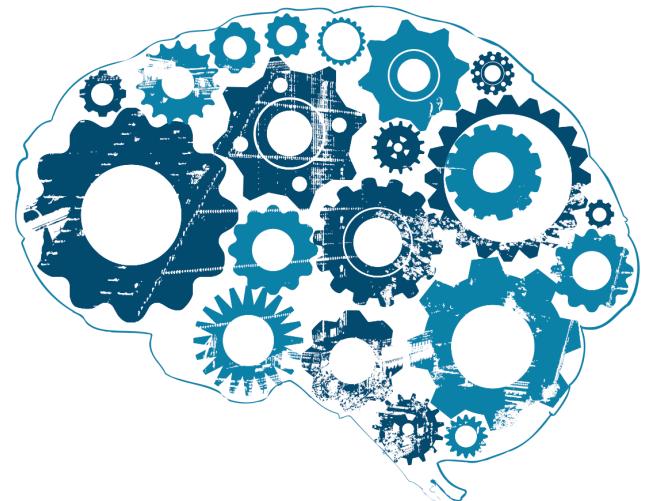
University of St. Thomas

St. Paul, MN

<https://github.com/jasondbaker/infrastructure-class>

Use by permission under the Creative Commons License 4.0

Lightly edited in places by Dr. Gary



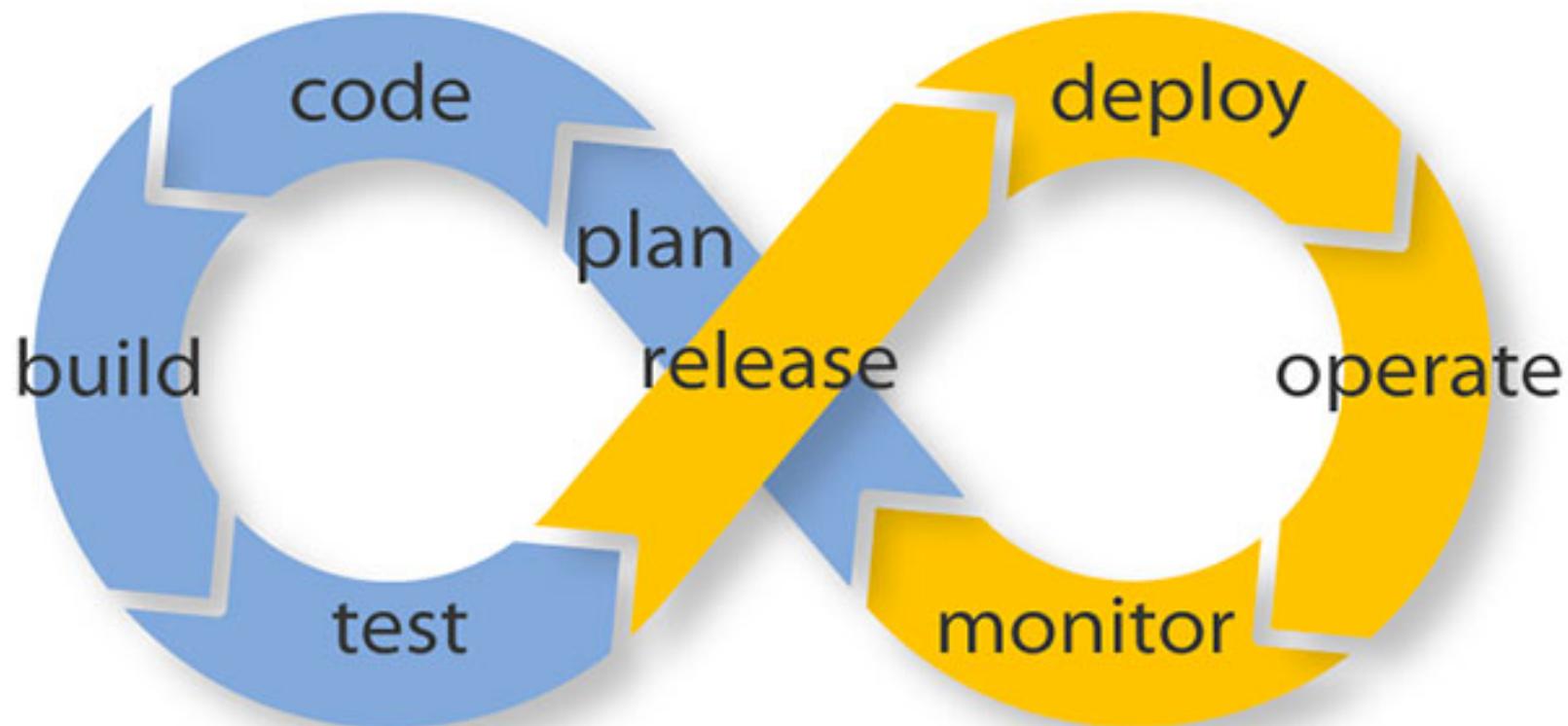
---

# DEVOPS OVERVIEW

# DevOps

---

A **collaborative** effort to improve the **quality** and **velocity** of the service lifecycle.



# Is DevOps really new?

---

Operations staff have been automating system configurations for decades

- Shell scripts
- PXE boots
- VMWare templates

Developers have been automating software releases as well.

- Make, Ant, CVS...



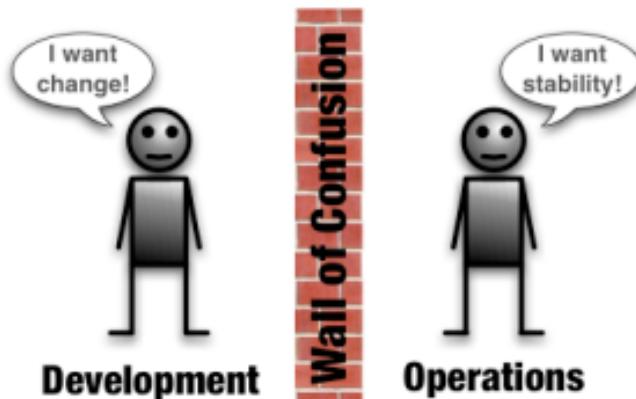
DevOps builds upon decades of practices.

Despite decades of effort, we still have problems...

# Conflicting measures of success

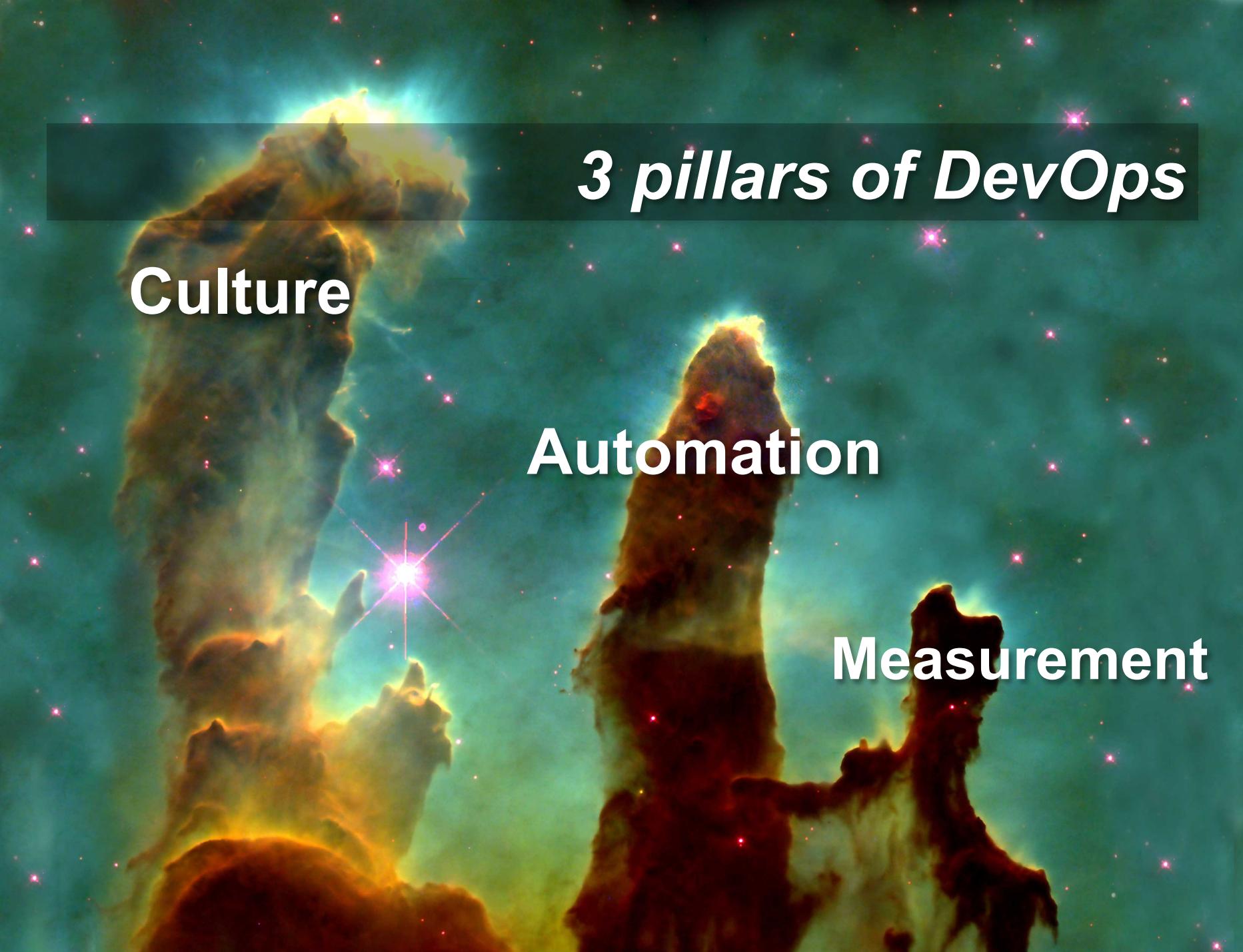
---

- Development success is measured by rate of change during a release cycle: lines of code, commits, scrum stories...
- Operations success is measured by service uptime & performance.
- Traditionally, IT manages the risk of introducing change by:
  - Changing infrequently and only when absolutely necessary.
  - Changing at inconvenient times when staff and vendors are at their lowest performance levels.
  - Batching changes during change windows.



IT'S  
NOT  
FAIR!





# *3 pillars of DevOps*

Culture

Automation

Measurement

---

---

# **DEVOPS CULTURE**

# DevOps Culture & Lean

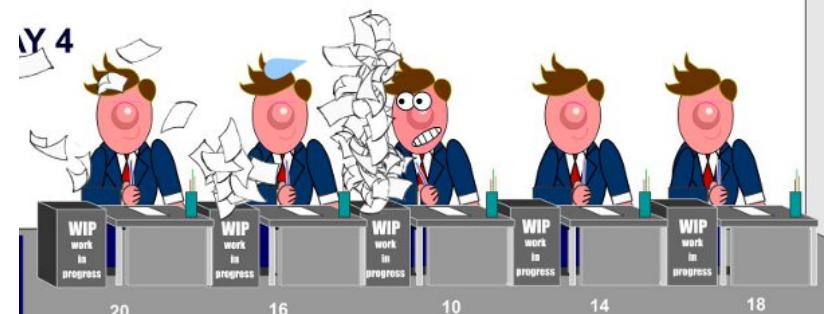
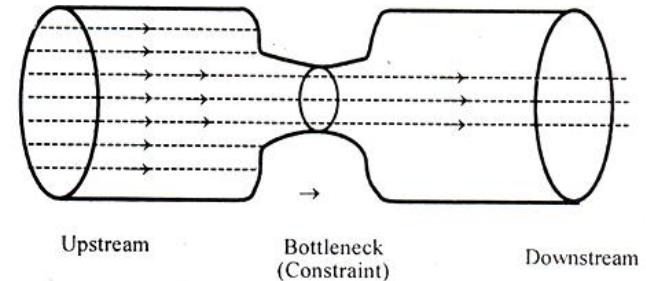
DevOps culture is heavily influenced by **Lean** practices.  
Lean is a wide-spread methodology in manufacturing.

- Commonly implemented as Six Sigma practices
  - Eliminate waste within a system (“Muda”)
  - Improve product quality
  - Respect for people
  - Continuous improvement (“Kaizen”)

Theory of Constraints (Eli Goldratt):

- Think of your process as a chain of links.
- Improving strong links doesn't strengthen the chain.
- Improve the weakest link first.

Example: If it requires 40 hours of effort to test a release and 10 minutes to deploy it, then the testing process is a constraint on improving cycle time.



# Lean Queuing Theory

---

What happens when we push work through a complex system?

**Traditional utilization thinking:** focus on operating resources at peak capacity.

- Optimize locally rather than at the system level.

**Throughput thinking:** focus on identifying the constraints that reduce delivery effectiveness.

Software development is a **stochastic system** with queues (non-linear and non-deterministic).

- Behavior of this type of system defies our instincts.
- Large projects (batch sizes) increase cycle time.
  - Take longer to process leading to variability and more queues
- Increasing worker utilization (just work harder) makes it worse!!

**Little's Law:** The only way to maximize throughput is by minimizing Work In Progress (WIP) through the system using small batch sizes.

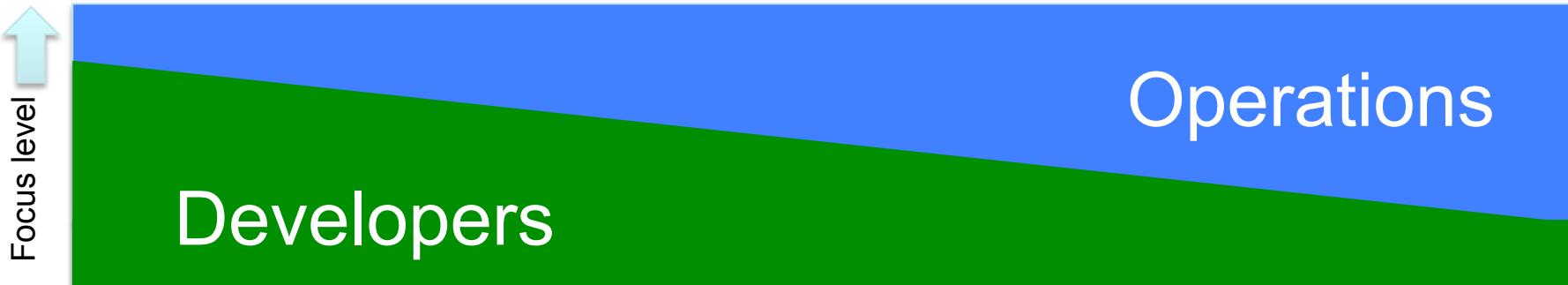
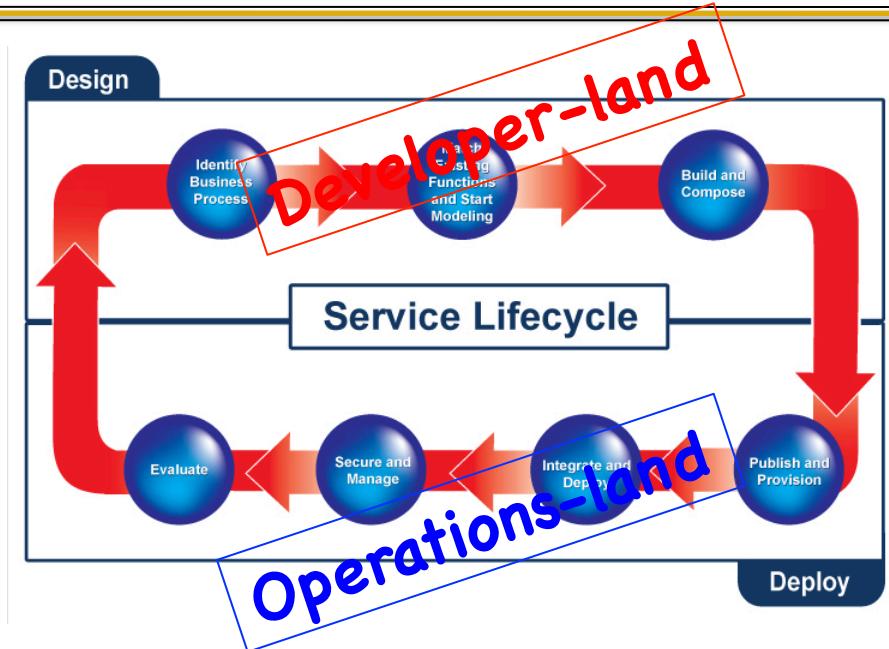
- Example: Small Agile sprints or Kanban tasks

# DevOps Organization Focus

Developers more focused on earlier phases of lifecycle.

Operations more focused on later phases of lifecycle.

**Both** are accountable for service delivery.



# Design For Operations

---

Problem: Software designs oftentimes don't account for deployment, management, and support phases.

- Non-functional requirements:
  - How will we scale this feature?
  - How will we provide high availability?
  - How will our customer service team support this?

In DevOps culture, ops participates in software planning

- Ops ensures non-functional requirements are part of the software design
  - Availability, Scalability, Upgradeability, Supportability, Testability, etc.

Developers participate in service delivery and support.

- Builds accountability: “Eat your own dog food”.
- Developers are part of the on-call support team.

If Developers provide service delivery support do they get root?

- No, but they should get enough access to troubleshoot issues.
- Full access to all infrastructure and application logs.
- Full access to metrics.
- Ability to remediate issues by pushing out corrective updates.

---

---

# AUTOMATION

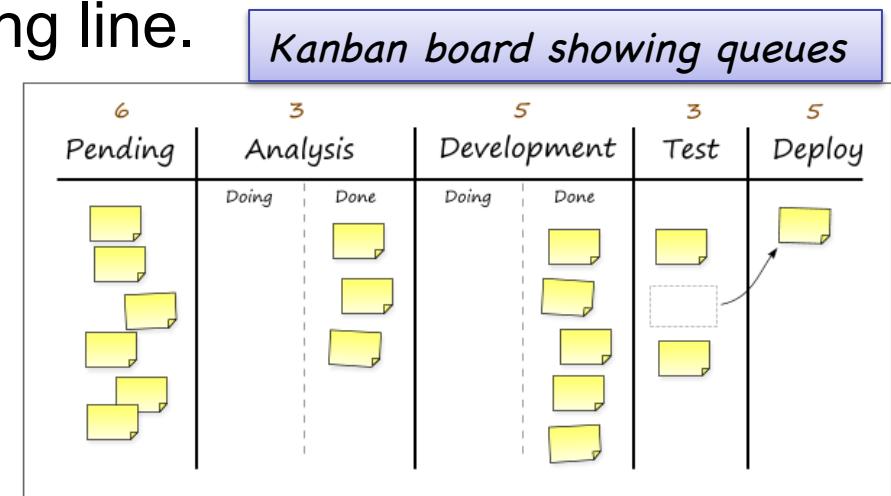
# Automation

DevOps mantra: Automate (almost) everything.

- Humans are brittle.
- Processes must be repeatable.
- Best way to reduce cycle time and improve quality.
- Most cost effective way to scale work.

DevOps treats software development and running a service like an automated manufacturing line.

- Design the product
- Assemble it
- Test it
- Package it for deployment



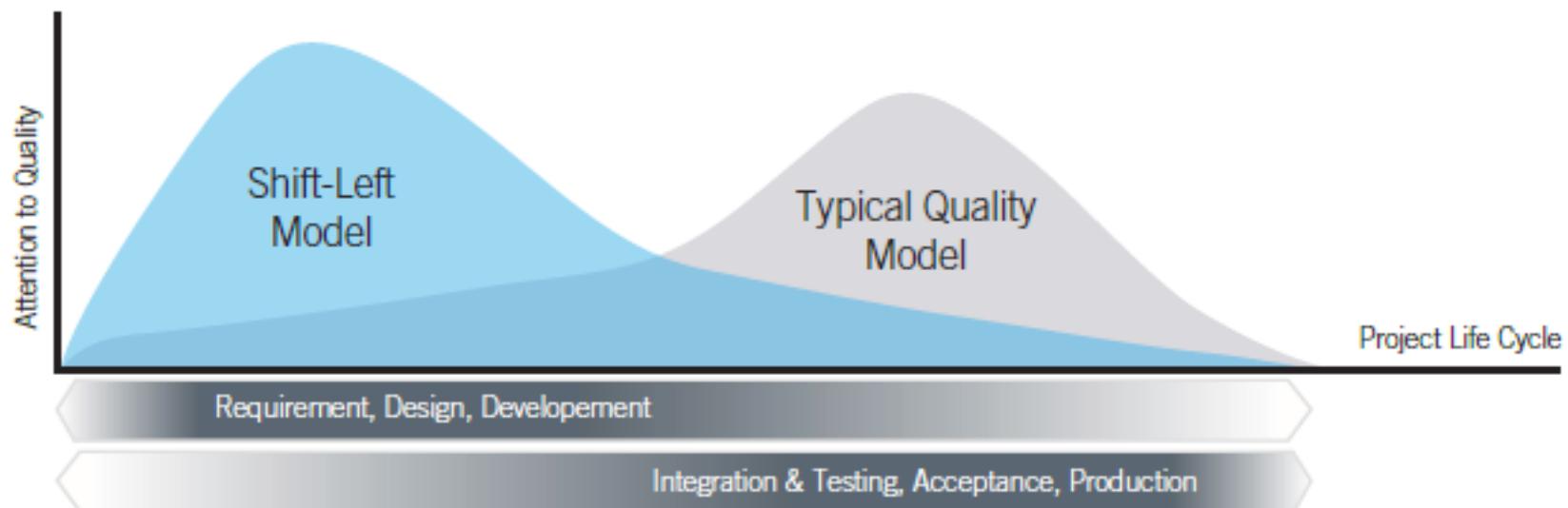
Work moves through the company, queuing at various stations

- Example: product backlog & QA tests are very visible queues.

# Shift Left

## DevOps adopts lessons from Lean

- W. Edwards Deming: “Cease dependence on inspection to achieve quality. Build quality into the product.”
- Oftentimes companies focus their efforts on finding defects versus preventing defects.
  - Manual exploratory testing
  - End-to-end testing
  - User Acceptance testing



# Shift Left

---

Why does *shift left* improve quality & reduce costs in software dev?

- It's easier to automate testing when the test is done as close to the code as possible (bake quality into the product).
- It's much easier for a developer to fix a defect found during unit testing than in integration testing or deployment.

Requirement	Design	Coding	Development Test	Acceptance Test	During Operation
1	3-6	10	15-40	30-70	40-1,000

Frost & Campo Crosstalk 2007, as taken from W. Humphrey *A Personal Discipline for Software* 1995. This table depicts the relative effort for fixing defects that are found out-of-phase.

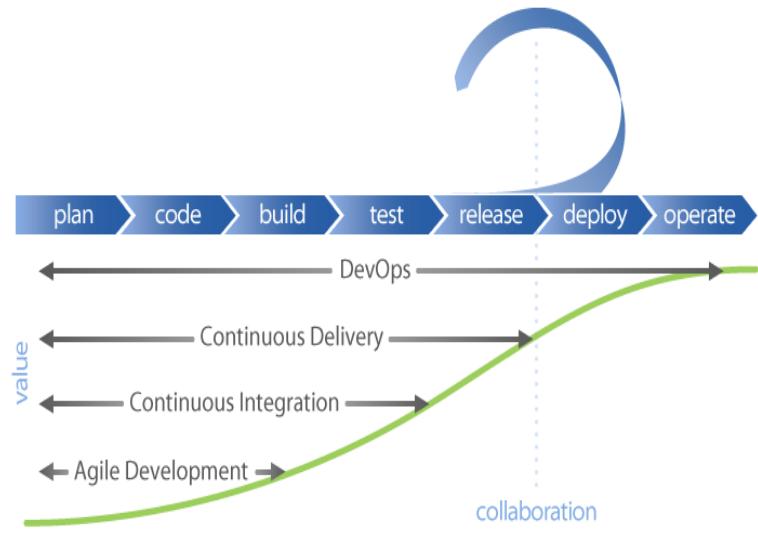
	Coding/Unit Testing	Integration	Beta Testing	Post-Release
Hours to Fix	3.2	9.7	12.2	14.8
Cost to fix (\$)	240	728	915	1,110

Data Source: (*Planning Report 02-3, “The Economic Impacts of Inadequate Infrastructure for Software Testing,” Prepared by RTI for National Institute of Standards and Technology, May 2002, p 7-12.*)

DevOps adopts a “Shift Left” Lean alignment to increase focus on quality issues earlier in the development process.

# Continuous Delivery

- A process that ensures software is deployable throughout its lifecycle.
- Team prioritizes keeping software deployable over working on new features
- All parts of the pipeline are extensively automated including deployment.



## Infrastructure as Code

An approach to infrastructure automation based on practices in software development

- Versionable, Testable, Repeatable
- Infrastructure supports change and is no longer an obstacle.
- Infrastructure change is routine.
- Improvements are made continuously versus big bang releases.



---

# **DEVOPS MEASURES AND SUMMARY**

# Meaurement and Metrics

---

DevOps organizations fanatically collect data.

- Collect data from diverse resources & store in a central repository
  - Time-series and unstructured data
  - Make it easy to search data to identify trends and correlations.

Identify Key Performance Indicators (KPIs) that measure:

- Business: revenue, # orders, # support calls
- Operations: Change error rate, incidents, ptime, response latency
- Generate notifications to alert staff on service health issues.
  - Weed out non-actionable notifications (log but don't email).

Metrics drive decision-making in high performing DevOps orgs

- People perceive the world differently based on experiences & bias
- Oftentimes our technical solutions are educated guesses.
  - "I think if we add more memory app performance will improve."
- DevOps uses metrics to support recommendations.
  - Any recommendation not supported by data is a guess and therefore risky.
- Cloud/container infrastructure gives us a cheap way to run experiments



DevOps is also (like Microservices) the convergence of a community of thought blending theory and ongoing practice

## THEORY

- Theory of Constraints
- Lean Systems Engineering

## PRACTICE

- Agile / Scrum / Kanban
- High profile failures

DevOps is not a light switch you just “flip on”

- You have to set the right cultural environment
- You have to accept new collaborative processes and controls
- You have to challenge long-held assumptions and mantras

However, unlike Microservices, it is built more difficult to incrementally adopt DevOps

- You can certainly pilot on some projects
- But as a process not a technology, it touches on all aspects of how you deliver value