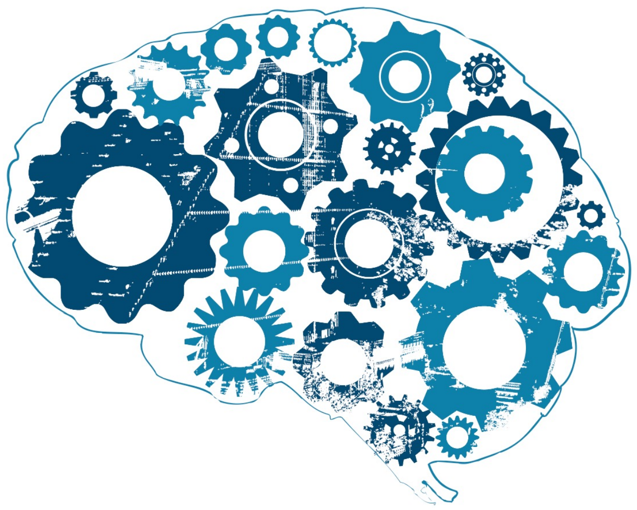
DevOps



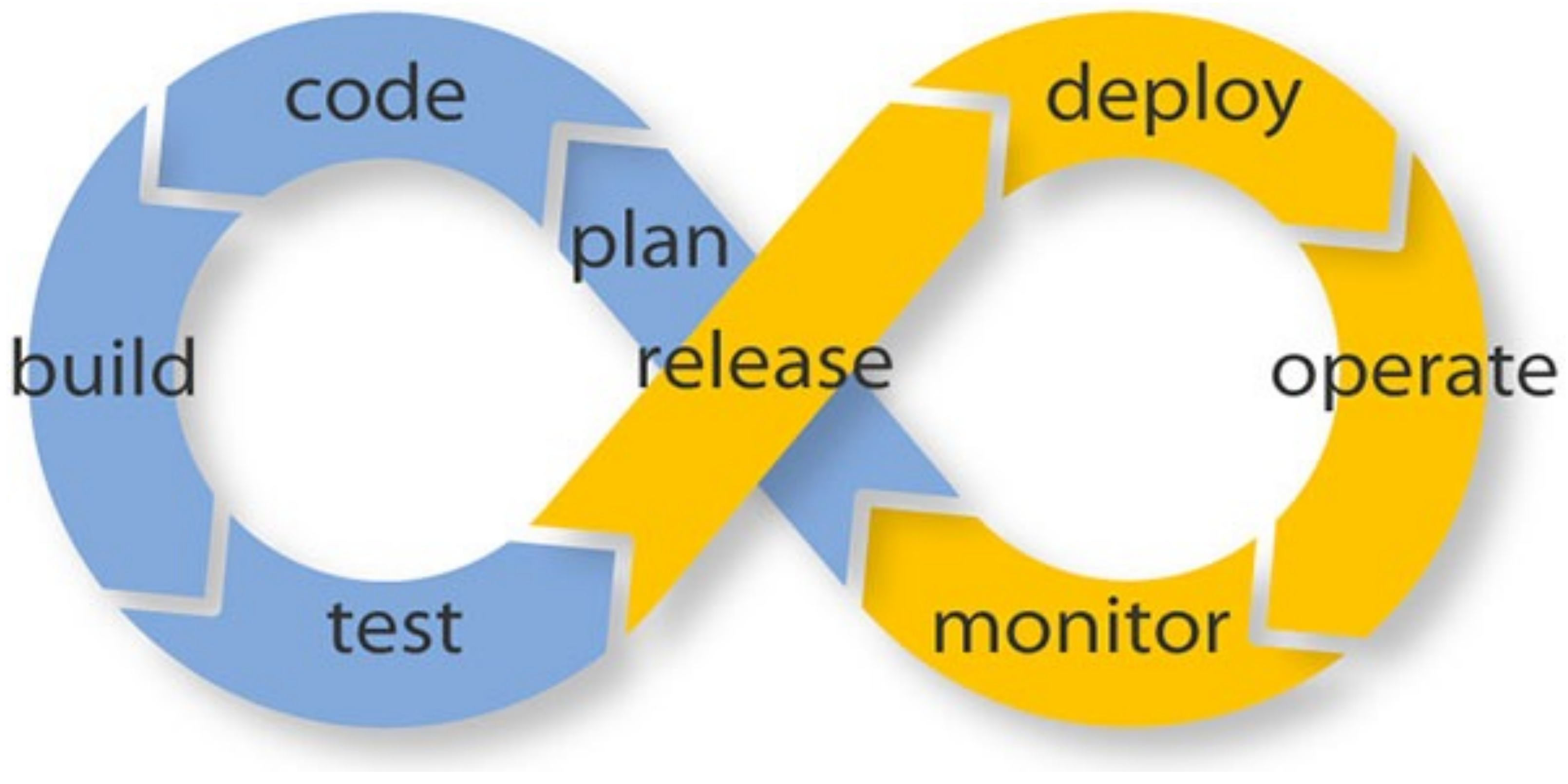
These notes (except the last slide) are from:
Jason Baker
Adjunct Instructor
Graduate Programs in Software
University of St. Thomas
St. Paul, MN

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

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.

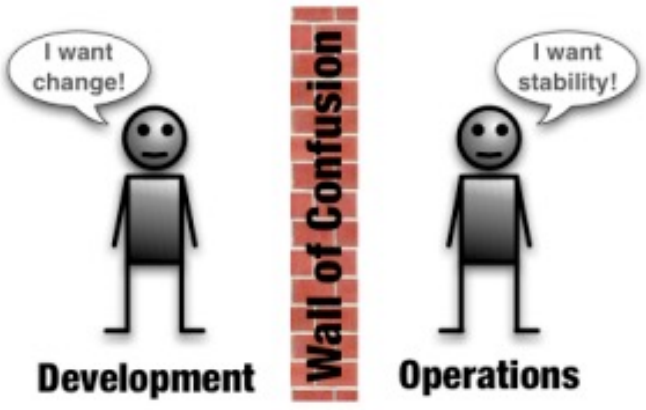
- Make, Ant, CVS…

DevOps builds upon decades of practices.

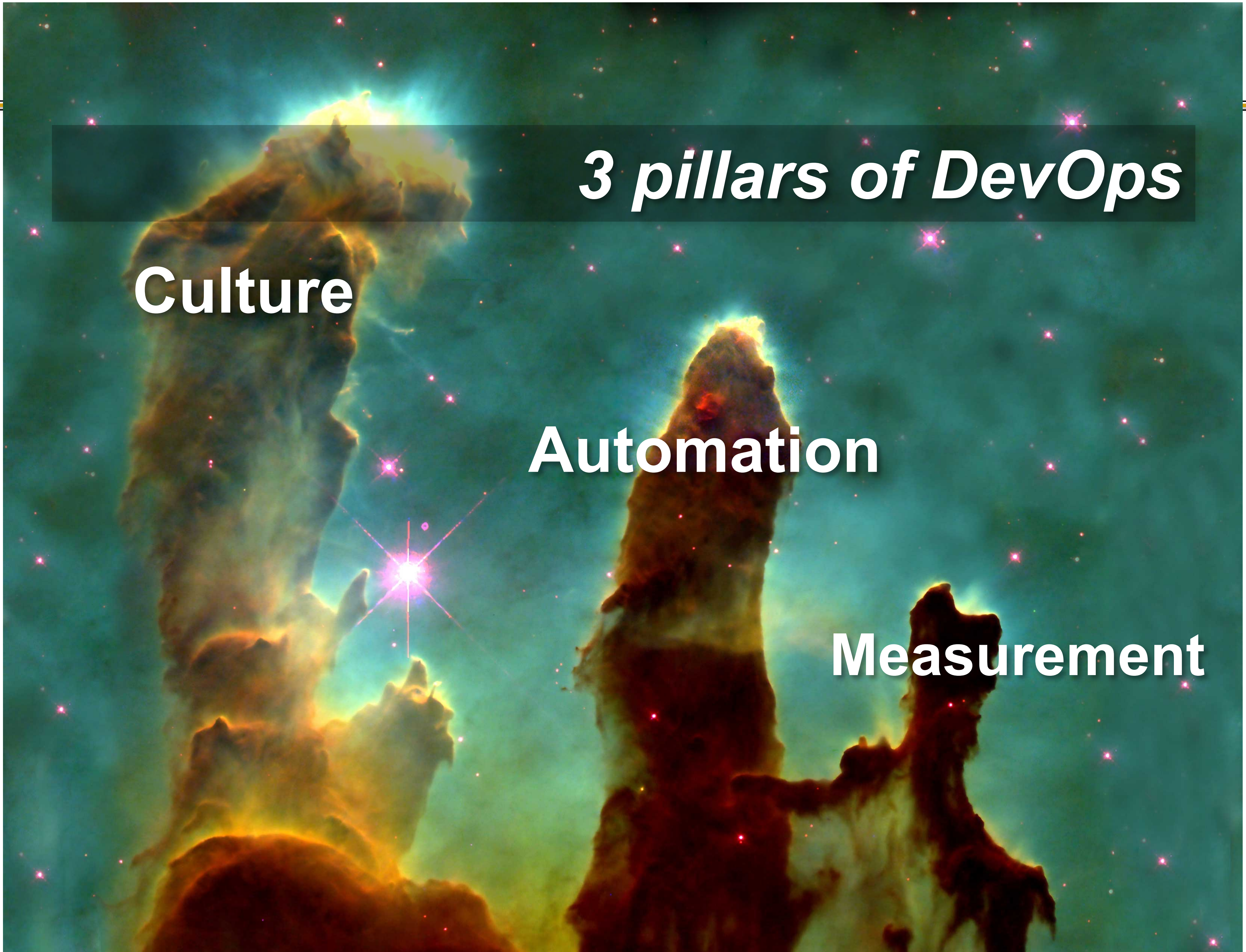
Despite decades of effort, we still have problems...

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 <u>infrequently</u> and only when absolutely necessary.
  - Changing at <u>inconvenient times</u> when staff and vendors are at their lowest performance levels.
  - <u>Batching changes</u> during change windows.



3 pillars of DevOps



Culture

Automation

Measurement

DEVOPS CULTURE

Cultural priorities

Developing a DevOps culture is the hard part!

Collaborative

- Teams are multi-disciplined and focused on service delivery (the greater good).

Transparent

- People share information with one another.
- Blame-less environment

Change oriented

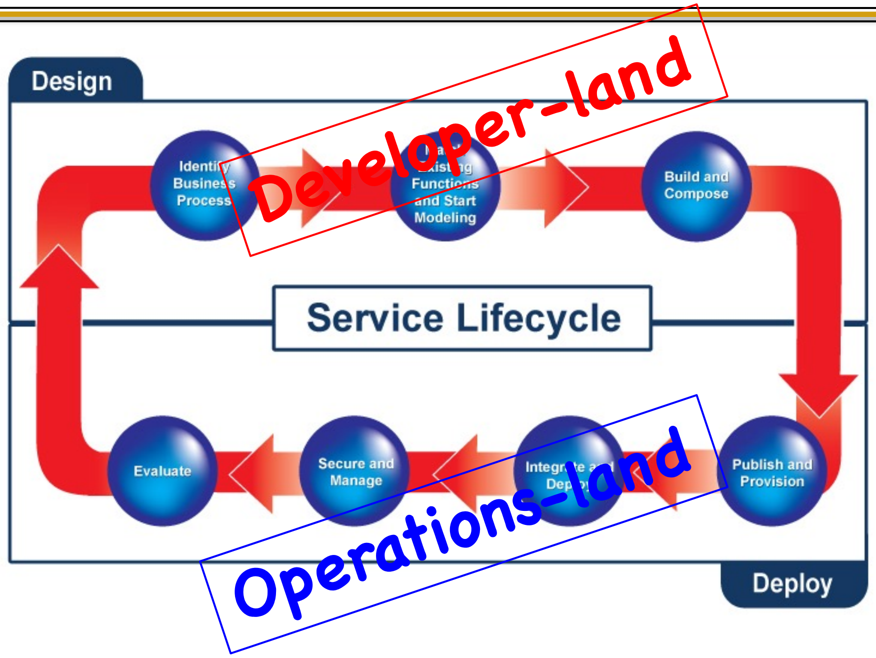
- Change is expected and embraced.

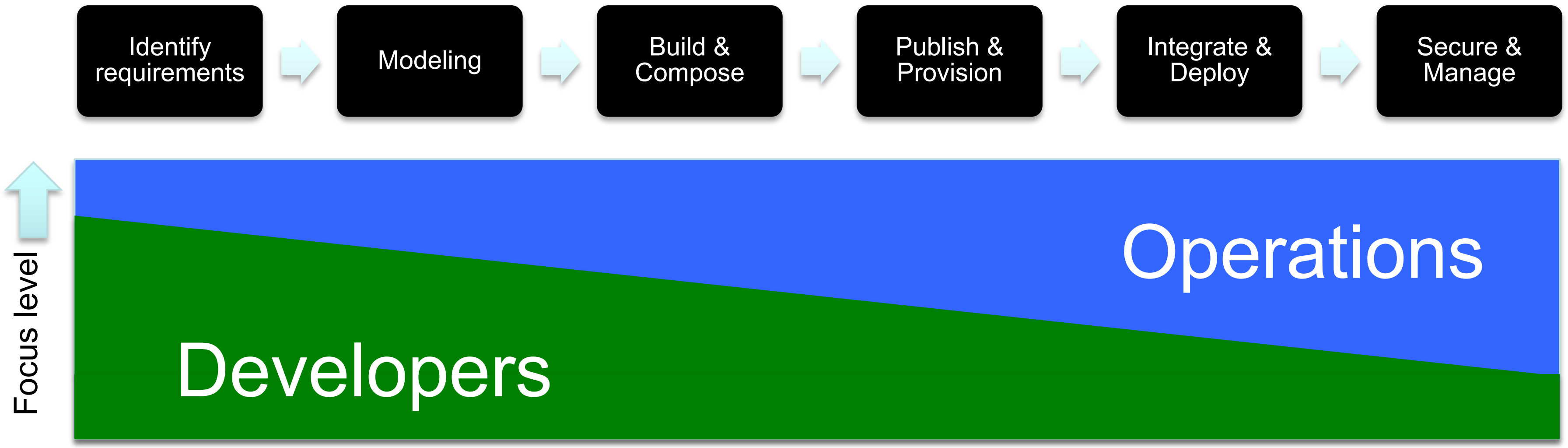
Does the organization have a DevOps Culture?

- Do our developers and operations team members sit near one another and go to lunch together?
- Do operations people regularly attend inception, showcase, and retrospective development meetings?
  - And do developers go to operations standups?
- Do developers rotate through the operations team?

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.
- Ability to remediate issues by pushing out corrective updates.

Remember MacCormack's practice #4: "Modular architecture to support new features". This is a flavor of it.

AUTOMATION

Automation

DevOps mantra: Automate (almost) everything.

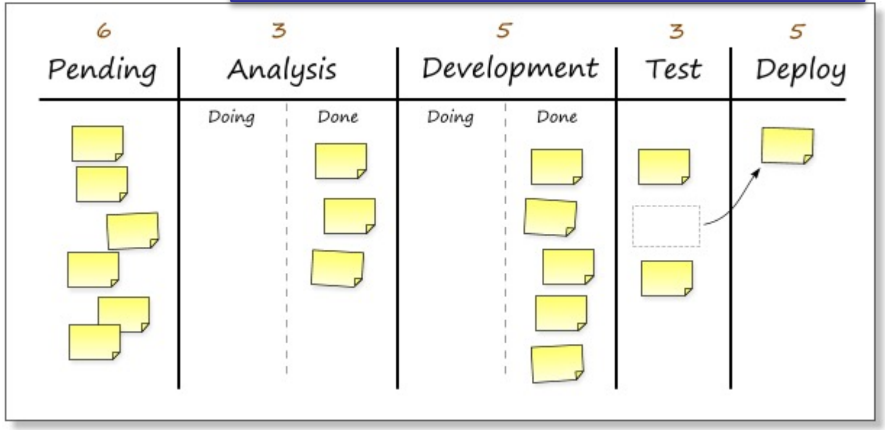
- Humans are brittle.
- Processes must be repeatable.
- Best way to reduce cycle time and improve quality.
- Most cost-effective way to scale work.

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

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

Ops is now on the board!

Kanban board showing queues



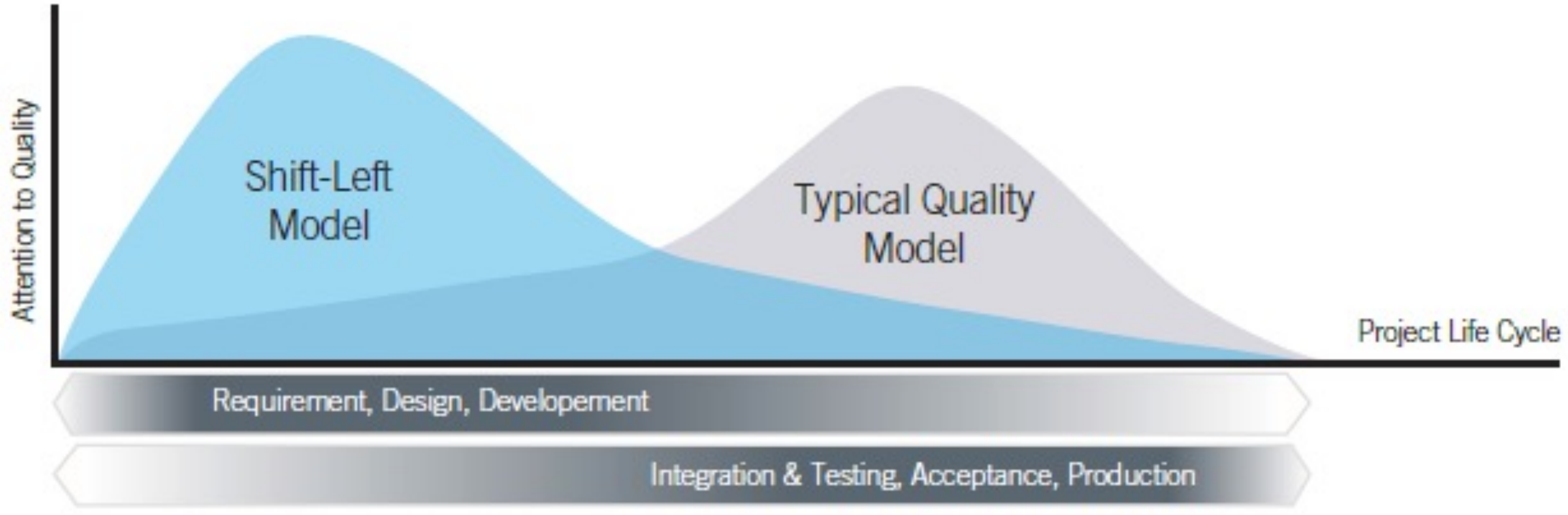
Work moves through the company, queuing at various stations

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

Shift Left

DevOps adopts lessons from Lean

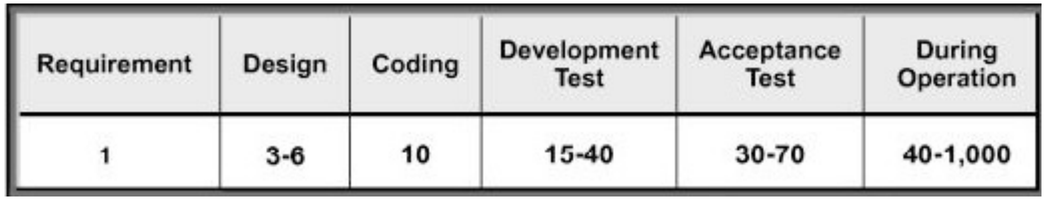
- W. Edwards Deming: “Cease dependence on inspection to achieve quality. Build quality into the product.”
- Oftentimes companies focus their efforts on finding defects versus preventing defects.
  - Manual exploratory testing
  - End-to-end testing
  - User Acceptance testing


Attention to Quality
Shift-Left Model
Typical Quality Model
Project Life Cycle
Requirement, Design, Developement
Integration & Testing, Acceptance, Production

Shift Left

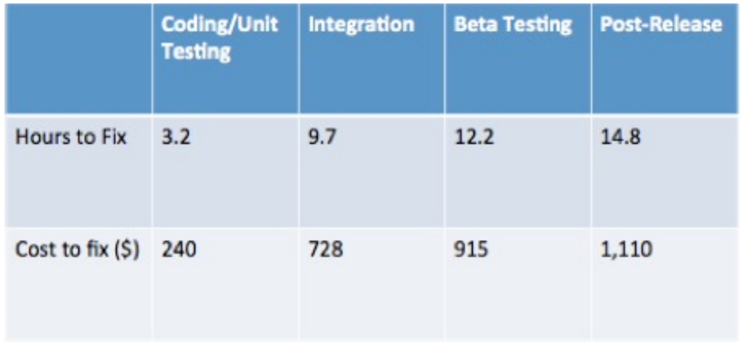
Why does *shift left* improve quality & reduce costs in software dev?

- 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.



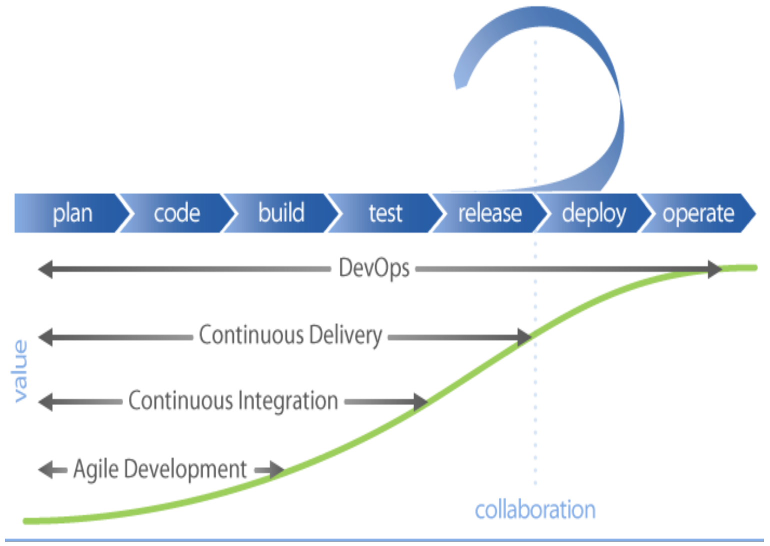
	Coding/Unit Testing	Integration	Beta Testing	Post-Release
Hours to Fix	3.2	9.7	12.2	14.8
Cost to fix ($)	240	728	915	1,110

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.



Done means released!

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.

CODE GOES THROUGH THE SAME WORKFLOW

Applications are code
Infrastructure is code

DEVOPS MEASURES AND SUMMARY

Meaurement and Metrics

This sounds very Lean!

DevOps organizations fanatically collect data.

- Collect data from diverse resources & store in a central repository
  - 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

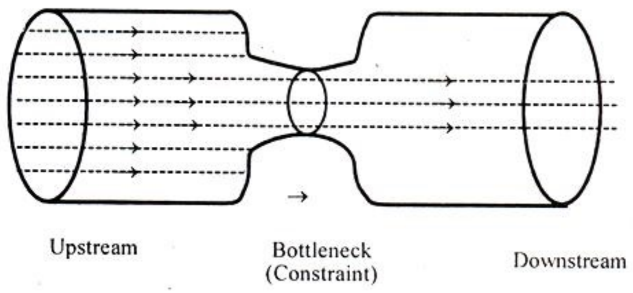
- People perceive the world differently based on experiences & bias
- Oftentimes our technical solutions are educated guesses.
  - "I think if we add more memory app performance will improve."
- DevOps uses metrics to support recommendations.
  - Any recommendation not supported by data is a guess and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

DevOps Culture & Lean

DevOps culture is heavily influenced by **Lean** practices.

Lean addresses muda (*waste*).

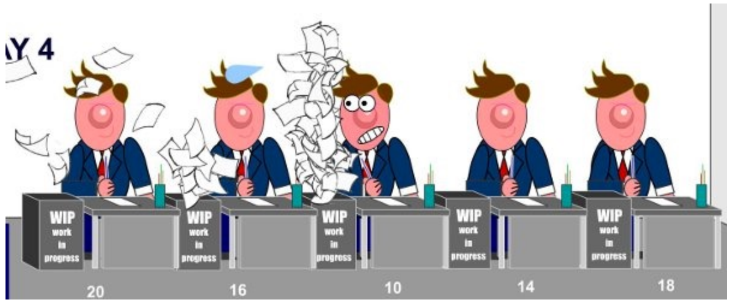
- Relearning – the mental effort to avoid common deployment issues
- Handoffs – DevOps reduces friction of handing off from dev to ops
- Task Switching – Dev, Prod, where am I?
- Delays – Ops batch-oriented cadence



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, which may be the Dev/Ops handoff
- Optimize *flow* from "soup to nuts", not just in Dev!

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+DevOps+Microservices+Containers = ?

What happens when we push work through a complex system?

- **Traditional utilization thinking**: Operate resources at peak capacity.
  - Optimize locally rather than at the system level.
- **Throughput thinking**: focus on identifying the constraints that reduce delivery effectiveness.
  - Lean principle: *Optimize the Whole*

Little's Law: Maximize throughput is by minimizing Work In Progress (WIP) through the system using small batch sizes.

- Ops (including security) doesn't like small batch sizes
- Microservices and containers are software engineering concepts that support "smallness" in deployed software architectures
- They also support the *independence* or *decoupling* we seek

When discussing Lean, we spent time talking about *upstream* activities ([eliminating] grooming & backlog queues), but we really didn't discuss where we go when Dev is "done". The truth is we aren't done, we still have work to do to get it "in the field". DevOps focuses on this *downstream* flow

DevOps Summary

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

THEORY	**PRACTICE**
– Theory of Constraints	- Agile / Scrum / Kanban
– Lean Systems Engineering	- 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 a 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