The simultaneous needs for IT to 1) deploy new features and 2) keep systems up and running creates a core conflict that challenges development, operations to the respond to business needs customers in a timely manner. DevOps represents practices, tools, and a culture that seeks to resolve this core conflict by enabling operations and development engineers to participate together in the entire service life cycle, from design through the development process to production support. This class will explore these practices, tools, and culture using Gene Kim's "Three Ways of DevOps" as guideposts.

**Course Audience**

- **Newcomers to Agile and DevOps** will find this class a welcoming environment to learn the basics on the DevOps mindset, The Three Ways, automation pipelines, common DevOps systems and tools, and continuous integration / continuous delivery (CI/CD)

- **Operations Engineers** will appreciate the class focus on using DevOps to effectively manage the large quantity and frequency of changes demanded in modern IT operations while keeping systems stable

- **Agile Teams** already developing software using agile methods will find this class to be a logical extension toward achieving synchronicity with operations and business using DevOps
Learning Goals

Today

Experience the DevOps way of thinking
Form beliefs about how DevOps can work for you

Tomorrow

Identify actions for your project

Weeks/Months

See improved results
Create DevOps experiences for others

Years

Build a widespread DevOps Culture in our organization

Who are you?
What are you working on?
How do you plan to apply DevOps?

Introductions
Let’s Review Our Progress with Agile So Far…

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

What results have we seen working this way?
Let’s Review Our Progress with Agile So Far…

We applied the agile empirical mindset and agile methods and observed these results:

- Early and continuous delivery of valuable software
- Rapid feedback
- Empirical decision-making
- Satisfied customers
- Business people and technical people working together
- Measurement-based forecasting
- Harnessing change for competitive advantage

- Emergent design
- Technical excellence
- Empowered self-organizing teams
- Personal safety
- Sustainable pace
- High trust environments
- Lean processes
- Continuous improvement

The Agile Manifesto and the agile methods that followed focused on software development – DevOps is a logical evolution of a maturing agile process
DevOps: Key Concepts
The Basics

Leave class able to confidently answer these questions:

Who is Dev? Who is Ops?

What is DevOps?

“The beginning of wisdom is the definition of terms”
- Socrates
Traditional Development

The Inventors

- Create new features and functionality in “dev” environment

- Occasionally deliver new product to operators, along with instructions

- May incorporate feedback from operators in future deliveries

- Rewarded for delivering new features

The inventors are responsible for changing the system
Traditional Operations

The Mechanics

- Receive new product from developers to be installed and operated
- Expected to keep production systems up and running
- Track problems, deployment failures, and system outages
- May provide feedback to the inventors for future consideration
- Penalized for downtime

The mechanics are responsible for keeping the system in operation
Differing Views on Change

Change Orientation

Alienate customers b/c system constantly changes

Stability Orientation

Alienate customers b/c system doesn’t change

Logical extremes
We Have A LOT of Changes

USCIS needs to **update IT capabilities** to support field users

**AND**

USCIS needs to **keep IT capabilities operational** for use by field users

**AND**

USCIS needs to **keep IT capabilities compliant** with security, regulatory, and compatibility requirements

Can we deploy latest version?

Can you deploy this one, small change?

Can we apply this security patch?

Production server is running slow, can you cycle the server?

Can you deploy new patch for the release?

Can you deploy this one, small change?

Production server is down, fix it now!!

Can you deploy this one, small change?

Can you upgrade the operating system?

Can we deploy latest version?

Can you deploy this one, small change?

Can you stage this new environment?

Can you deploy this one, small change?

Can you upgrade the database version?

USCIS applied over 4,000 changes in 2014
As computers became more complex, dev and ops became necessarily specialized:

- Accelerating pace of technology
- Increased demand for turning around new features
- Huge amounts of data and number of calculations
- More and more specialized tools
- Increasingly abstract architectures and design patterns

And these were the problems in 1945!

Nobody can be an expert in everything – your enterprise can’t rely on Brent!
The Reunification of Dev + Ops
DevOps in a Nutshell

DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

Hmm… what would happen if we extend the core drivers of successful agile development to operations?

What if we built a bunch of great tools to help us?
How can dev help system stability? How can ops help accelerate feature delivery?

We can build cross-functional teams around “knowledge overlaps” – people with experience on both sides and “Ops Devs”
Breaking the Silos: Dev and Ops

- Ops can anticipate how new functionality will affect production.
- Dev can respond to bugs and deployment failures quickly.
- Dev and Ops can work together to permanently remove root causes of bugs and failures.
- Ops trusts dev will provide good code.
- Dev trusts ops will put code in prod quickly.
- Visibility enables “trust but verify.”
Dev and Ops Working Together

- Create feedback loops between inventors and mechanics
- Expose real-time metrics from ops enabling dev to learn from the system running under real world conditions
- Expose real-time metrics from dev enabling ops to anticipate production needs and provide early input
- Cross-functional teams collaborate to deliver whole working systems including all infrastructure, software code, and configurations

Feature delivery + stability become shared goals
Matching IT Capacity to Business Demand
Breaking the Silos: Communication, Collaboration, Integration
Dev can better incorporate needs of the business and customers into new development.

Business better understands capability for changes to features and functionality.

Breaking the Silos: Dev, Ops, and Business
Breaking the Silos: Dev, Ops, and Business

Development

Business better understands operational capabilities

Operations

Ops understands better how to support business goals

Business
Business Demand: Continuously Deliver Valuable Software

Modern business is dependent on IT deploying new features

Need very fast time-to-value in the face of change
  - Immigration policy can change rapidly – IT capacity must keep up
  - Immigration Executive Action

Multiyear lead time no longer acceptable

Expectations for delivery times continue to decrease
Software is increasingly customer-facing, rather than internally-facing

Customers expect an interactive, self-service interface

Customers expect deep, direct engagement with their data, not a paper system

Customers expect to be able to get information immediately

Customers can now identify problems in our systems directly – and they expect us to fix them
Business Demand: Rapidly Incorporate Latest Technology

Modern web interfaces
Mobile devices
Social media
Accessibility tools
Live customer interaction tools
Tools for online communities and user-generated content
Amazing new features
Business Demand: “Lean Bureaucracy” Supporting Government Values

“Working in public”

Governance – many, many stakeholders

Transparency in how we work

“Presentability” of what we produce

Mission alignment

Risk aversion

Baked in support of values such as:
  • Contracting preferences
  • Hiring fairness
  • Procurement fairness
System operations increasingly yields insights that must be acted immediately to keep pace with demand.

Availability of ubiquitous automated data collection yields expectations that organizations will rapidly act on key data points to improve efficiency in mission and services.

With so many routes to innovation, organizations are expected to test and identify the best options very quickly.

Well run companies are expected to maintain very low MTTR (mean-time to repair) times – delays in fixing problems can be catastrophic.
Beware!

If you turn back from the journey now...

Pain Ahead!

...we won't judge
Business makes even more audacious commitments to catch up

Developers see more and more urgent projects coming in

All effort is spent on features as opposed to non-functional requirements

More shortcuts, more technical debt, more fragility

Deployments become more difficult – what took a weekend now takes 3 days!

We try to fix this by doing less deployments, increasing batch size

More moving parts, more failures… we are consumed by unplanned work

This approach preordains us to failure

Creates a permanent wedge between making urgent business changes and maintaining stability

Working here is a major drag
Results: Puppet Labs State of DevOps 2014 Report

- Scientific study of relationship between organizational performance, IT performance, and DevOps practices
- 9200 respondents representing 110 countries

Findings

- DevOps adoption is accelerating
- High-performing IT organizations deploy code 30 times more frequently with 50% fewer failures
- Strong IT performance is a competitive advantage
- DevOps improves IT performance
- Organizational culture matters
- Job satisfaction is the No. 1 predictor of organizational performance

High performing companies are good at getting better – nobody starts out high performing
The Three Ways of DevOps by Gene Kim

The Three Ways describe the values that frame the processes of DevOps and they provide prescriptive steps.

1st Way
Emphasize entire system performance versus a specific silo of work.

2nd Way
Creating feedback loops.

3rd Way
Culture of continual experimentation. Understanding that mastery requires practice.
DevOps: The First Way
Use systems thinking to ensure work always flows forward

“Work moving backwards, or standing still, is almost always indicative of problems that need to be solved, and will span people, process and technology.” –Gene Kim
What is a silo, really?

- Disconnection from other people
- No shared context
- Different management

Barriers build up
- Different incentives
- Different objectives

Bad handoffs
- Lack of understanding
- Lack of empathy

“The nature of a large, complex organization is to fall out of alignment without deliberate effort – inertia pulls it apart” – Damon Edwards
The First Way: Understand the Flow of Work

- **Work starts** with a description of features needed by the business

- **Work ends** with the stable, secure and reliable delivery of services to the customer

- Additional sources of work:
  - IT finds defects
  - Help desk fields incident reports
  - Security raises compliance requirements
  - Enterprise architecture initiatives e.g. single sign-on

- Visualize work
- Measure the flow of work (cycle time, lead time, wait times)
- Think about software production as a value stream similar to a manufacturing value stream
Organizations are Complex Systems

Human complex system

- Communication patterns
- Locations
- Work styles
- Personalities
- Roles and responsibilities
- Skill sets

Technology complex system

- Programming Languages
- Tools
- Networks
- Configurations
- Interconnections

One complex system working on another complex system
The First Way: Always Seek to Increase Flow

- Reduce work in progress (WIP)
- Reduce batch size of deliveries
- Reduce variation in size of work items

- Make policies explicit
- Eliminate inventory and other waste
- Maintain a steady, sustainable pace

Deliver often – and get really good at it
The First Way: Optimize Flow Globally, Not Locally

- Focus on interactions between parts of the system
- Build controls into the system
- Local efficiencies are good, but should never jeopardize global goals
- Avoid tribal warfare!
- Know your bottlenecks … and elevate them
- The bottleneck is the lever of control for speed of flow through your process

![Diagram illustrating upstream queues, bottleneck, and downstream starved of full flow]
The First Way: Never Consciously Pass Defects Downstream

- Create quality at the source
- Make rework visible
- Understand the origination point of defects in order to avoid recurrence

“This is legacy code, I’ll just make the change for my story, I don’t have time to fix the rest of this”

“That issue is a doozy… leave it to fix in the hardening sprint”

“Just push this feature over to the testers… it’s their job to find defects, right?”

“Call the story done. We know there are still a few problems so just open up some defects against it”
The First Way: The System of Profound Knowledge

Organizations are systems of interrelated processes and people which form the system’s components.

Components of the system must reinforce, not compete with each other to accomplish the aim of the system.

Workers’ success depends on managing the balance between each component to optimize the system.

Understand business goals – how value is achieved

Understand people, processes, and technologies

Understand risks and risk controls

Understand cause and effect

Make informed decisions based on rich, accurate, and timely information

Teach the organization how to fix and regulate itself
The First Way: Bringing It All Together

Business

End Users

Dev

Ops

What is the minimum viable product? Is it profitable? Do we have the capability to build it and maintain it?
DevOps: The First Way – Practices
Traditionally, dev is responsible for applications while ops is responsible for environments.

In DevOps, we use a single repository for everything – functional code, test code, environment configurations, and tool configurations.
“Programmable infrastructure”

“Fully automated configuration management”

Code to automate provisioning
Code to manage configurations
Code to automate deployments
DevOps Practice: One Step Environment Creation

Provision and configure environments at the touch of a button

Make production-like environments available early in the dev process

Build code & environment at the same time

Create a common dev, testing, and prod environment creation process

Everyone uses a consistent environment
DevOps Practice: The Daily Build

- “Heartbeat of the project” and “clean room every day”
- Rebuild every line of code from scratch – be able to reconstitute the system from “bare metal”
- Run all the tests!
- Check all dependencies
- Verify no defects introduced yesterday
- Build all versions
- Automate with Continuous Integration (CI) server
DevOps Practice: Deploy Early, Often, and Quickly

Deploy very frequently

Practice, Practice, Practice

Small deployments mean Fast deployments mean more deployments mean easier deployments mean lower cycle times mean faster time to market
DevOps Practice: Classify Ops Work by Four Types

Systematically allocating time to the 4 types enables all the work to get done and becomes routine.

Types of work:

- Business Projects
- Internal IT Projects
- Changes
- Unplanned Work

Exercise: Classify real USCIS work according to the four types.
Doing DevOps at USCIS – First Way

- Understand and measure your flow of work using visualizations such as a Kanban board and value stream map
- Use a single USCIS repository for source code, test code, and environment configuration scripts
- Builds done by automated, script-driven retrieval of source code by a Continuous Integration (CI) server
- Frequent deployments – no less than every two weeks
- Consistent record of successful deployments
- Baked in accessibility and security compliance – no compliance work flowing backwards

What is the concept of a “Team-Managed Deployment” at USCIS?
DevOps: The Second Way
The Second Way: Amplify Feedback Loops

- Shorten and amplify “right to left” feedback loops
- Use feedback to create even higher quality at the source
- Create and embed knowledge where it is needed to provide immediate feedback
- Understand needs of all customers, internal and external, and respond to their feedback

The goal of any process improvement is to shorten and amplify feedback loops.
The Second Way: Shorten and Amplify Feedback Loops

10 steps to get feedback & VERY long delay

Product Owner/Value Team → Triage → Issue Tracker → Help Desk → End Users

Operations

Develop → Commit → Test Build

Manual tester overloaded due to end of sprint

Product Backlog
The Second Way: Shorten and Amplify Feedback Loops (cont)

5 steps to get feedback

1. Develop
2. Commit
3. Test Build
4. Manual Test
5. Issue Tracker
Most users won’t call… some may just quit being customers
Many defects remain latent for a long time
By the time defects come back, dev forgets how the code works

The Second Way: Shorten and Amplify Feedback Loops (cont)

4 Steps to get feedback – automated and quick!

- Develop
- Commit
- Failed Automated Test
- Test Build
The Second Way: Use Feedback to Create Quality at the Source

- Development is the source of quality – or problems

- As applications evolve, changes must not negatively impact end user experiences

- Developers need access to deep diagnostics so they can incorporate latest operational concerns and understand impact of their changes

Browser performance metrics
Application response times
Server usage
Performance data by technology component
Runtime code diagnostics including database queries

Traces from slow transactions that suggest performance bottlenecks in distributed applications
Service-oriented architecture issues spanning multiple application tiers
Correlation of application response times on end-user satisfaction levels
The Second Way: Create and Embed Knowledge

• Ops and Security:
  • Become part of the agile process – especially planning and prioritization
  • Provide recommendations and requirements as new code developed
  • Ensure relevant metrics are monitored early in the dev process

• Dev participates in incident handling to acquire knowledge to prevent future problems:
  • Incident escalation
  • Root cause analyses
  • Post-mortems

• Ops receives cross training by dev and security

• Extend agile practices to all teams
  • Visible work
  • Open meetings
  • Working agreements
  • Explicit policies
The Second Way: Respond to Needs of All Customers

- Use a service model for both internal and external customers
- Agile encouraged dev and test to focus on customer collaboration with business stakeholders and end users
- DevOps extends the service model to Dev and Ops treating each other as customers
DevOps: The Second Way - Practices

- Communication
- Collaboration
- Integration
DevOps Practice: Deployment Automation

- Problems in deployment procedure will be found quickly and can be permanently eliminated
- Runs fast “smoke test” to ensure system is running as expected
- Built-in automatic rollback and/or redeploy
- Build confidence through frequent repetition – the prospect of deployments and rollback no longer instill fear
- Create a virtuous cycle of successful deployments, smaller deployments, and lower risk
DevOps Practice: Operations Monitoring

Monitoring gives us continuous, live feedback about how the system is running

“Tell me what is happening before the phone rings”

<table>
<thead>
<tr>
<th>User Feedback Approach</th>
<th>Monitored Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field user calls</td>
<td>Automatic alert about a problem when it happens</td>
</tr>
<tr>
<td>Multiple people call</td>
<td>Monitoring tools show me how widespread the problem is</td>
</tr>
<tr>
<td>Users can’t tell me the real source of the problem</td>
<td>I can see which component of the application is generating errors</td>
</tr>
</tbody>
</table>
Operations Monitoring – Needs and Challenges

Monitoring Challenges

- Operators typically responsible for numerous applications
- Environments can be complex with unique or complicated application stacks
- Visibility into different components can be vague or non-existent
- Quantity of logged data can be overwhelming
- Combining monitoring tools into a single view that provides insight

Monitoring Needs

- Active production monitoring, not just reacting to downtime
- Easily monitor critical areas of application stability with minimal tooling
- Tune dashboard to display key database, network, server, and application performance measurements in a holistic view
- Ability to quickly share potential performance issues with your team
DevOps Practice: Operations Monitoring Dashboard

- Application Response Time
- Application Performance Index (User Satisfaction)
- Application Throughput
- Transaction Timings (drill down capability to code level, transaction level)
- Alerts
DevOps Practice: Operations Monitoring Drives Dev & Ops Priorities
Prioritize fixing defects very fast

- Assume incidents will occur
- Ensure ops feedback will come back rapidly
- Ensure developers will get the info they need to fix the problem
- Add automated tests to ensure problem cannot reoccur
- Get really good at fixing defects very fast
DevOps Practice: Reusable Ops and Security User Stories

User Story
As security I want cross-site scripting attacks prevented so that access controls cannot be bypassed

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 points</td>
<td>1 (High)</td>
</tr>
</tbody>
</table>

Acceptance Criteria
- Verify all input is filtered as potentially malicious
- Verify all output of the page is encoded to the explicitly defined character set
- Verify output is sanitized by escaping dynamic content to properly enforce separation of code and data

User Story
As an ops engineer I want to monitor how many people are listening to audio feeds so I can tune playback quality during spikes in demand

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 points</td>
<td>2 (Med)</td>
</tr>
</tbody>
</table>

Acceptance Criteria
- Verify the count of active audio sessions is displayed in the application’s admin dashboard
- Verify the count is accurate as playback sessions are added or completed
DevOps Practice: Dev & Ops Common Communication System

Remove all barriers to internal communication, collaboration, and integration

- Use common, intuitive dashboards combining information from all groups
  - Key operational metrics
  - Visible dev, ops, and security workflow (e.g. Kanban boards)
  - List of recent and upcoming system changes
  - Stability of the system
  - Security status
  - Schedules, planned release dates, and critical business dates

- Integrated alert policies

- Common internal note system – histories of defects and incidents can be very useful

- Shared wikis, file repositories, chat spaces, specs, and documentation
DevOps Practice: Track Dev & Ops Business Impact

- **MTTR** – Mean Time To Repair – How long is the system down?
- **MTBF** – Mean Time Between Failures – How often is the system down?

**USCIS Example:**

<table>
<thead>
<tr>
<th>Key Performance Parameter (KPP)</th>
<th>Service Agreement Low Threshold</th>
<th>Objective</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reliability</strong> – uninterrupted correct function</td>
<td>641 hours</td>
<td>712 hours</td>
<td>739 hours Exceeded Objective</td>
</tr>
<tr>
<td><strong>Availability</strong> – 24/7 operations</td>
<td>97.63 %</td>
<td>98.88%</td>
<td>99.32% Exceeded Objective</td>
</tr>
<tr>
<td><strong>Maintainability</strong> – prompt restoration of service after outage</td>
<td>No more than 10 hours</td>
<td>No more than 8 hours</td>
<td>5 hours Exceeded Objective</td>
</tr>
</tbody>
</table>
Doing DevOps at USCIS – Second Way

- Demonstrated information integration and collaboration between dev, ops, security, and business

- Partial or completely automated deployment – rapid, reliable, testable, repeatable

- Operational Monitoring Plan – preferably a dashboard

- Defined business impact measurements and thresholds

See Team-Managed Deployment Management Instruction for more information
Automating an Integrated DevOps System
A good method of enabling DevOps is to simply begin connecting and automation the systems you use to make software.

- Start where you are
- Identify possible interconnections
- Research tools to automate
- Create future state roadmap
- Let pipeline emerge
- Continue to improve the sequence of connections as systems change
Version Control

Version Control (CM)

Communication System

Monitoring System

Deploy System

Documentation System

Issue Tracking

Test System

Build System

Requirements

Version Control

Ensures you’re working on the right version of something
Requirements System

Houses project requirements in a prioritized list and allows for item allocation to sprint/team member; allows for traceability of dependencies between stories.
Software tools designed to automate the process of program compilation to create a deployable package.
Test System

orchestrated set of tests, both manual and automated that ensure the functionality and accuracy of code
Ensures that code complies with standards and identifies low level bugs and coding errors; Identifies design and requirements compliance
Issue Tracking System

Collects issues throughout the cycle of the project and tracks them through completion.
Documentation System

- Communication System
- Version Control (CM)
- Requirements
- Build System
- Test System
- Code Review System
- Deploy System
- Monitoring System
- Issue Tracking
- Documentation System

Documentation System
Repository of system information throughout the lifecycle
Deploy System

Installs and configures the package created by the build system into appropriate environments.
Monitoring System

Collects current-state data to determine health of all environments
Communication System

Method for conveying information between systems
Automate All the Connections!

- Communication System
- Monitoring System
- Deploy System
- Documentation System
- Issue Tracking
- Version Control (CM)
- Requirements
- Build System
- Test System
- Code Review System
Orchestrating Integration with a Pipeline
A Pipeline is a chain of tasks that can be automated

- Integration tools use pipelines to perform tasks repetitively and continuously
- The process is called Continuous Integration (CI)
- Pipelines keep work flowing forward in our DevOps system
We Need Something to Integrate the Systems
Development Pipeline Example with Integration System

- Communication System
- Monitoring System
- Deploy System
- Documentation System
- Issue Tracking
- Version Control (CM)
- Requirements
- Build System
- Test System
- Code Review System

Pipeline Orchestration:
- Commit code
- Code is committed and Merged
- Log Issues
- Initiate Code Review
- Initiate Testing
- Initiate Build
- Deploy to Target Environment
- Communicate Build Status

Example with Integration System:
- Commit code
- Code is committed and Merged
- Log Issues
- Initiate Code Review
- Initiate Testing
- Initiate Build
- Deploy to Target Environment
- Communicate Build Status
Pipeline Stages

**Continuous Integration**

- Code Done → Unit Tests → Integrate
  - Auto → Auto

**Continuous Delivery**

- Code Done → Unit Tests → Integrate → Acceptance Testing → Deploy to Production
  - Auto → Auto → Auto → Manual

**Continuous Deployment**

- Code Done → Unit Tests → Integrate → Acceptance Testing → Deploy to Production
  - Auto → Auto → Auto → Auto
CI Pipeline Example

CM Repository

Build gate
- Compile Code
- Code Quality Gates Applied
- Smoke Tests
  - If **Successful**, Merge with Staging

Staging gate
- Compile Code
- Functional Tests
  - If **Successful**, Merge with Integration Branch

Integration gate
- Compile Code
- Merge with Master
- Fortify Scans
- Release is packaged

Master
Team Managed Deployment (TMD) provides the approval step for a CI/CD Pipeline. The CI/CD Pipeline provides the forward link from Development to Operations.
Automation used in a CI/CD pipeline allows data to be collected as true artifacts. In an RRR approach, the information is manually collected into documents.
DevOps: The Third Way
DevOps is Not…

A tool

A role

A team

Something that can be purchased or simply switched on

DevOps requires a culture of operations and development engineers participating together in the entire service lifecycle.
• Continual experimentation, taking risks, and learning from failure

• Understanding that repetition is the prerequisite to master
The Third Way: Experimentation, Risk-Taking, and Learning

Develop a culture that pushes into the danger zone

Develop habits to survive danger

Build experimentation, risk-taking, and learning into our way of doing business

Break things early and often

Intuit ran 165 experiments on their TurboTax product in the 3 main months of tax season – ideas made it to market a year earlier and they increased customer conversion rate by 50%
The Third Way: Repetition for Mastery

- Do the hard parts often
- Work through pain points to make the process easier
- Do painful things MORE frequently to make it less painful
- Reduce anxiety about unexpected outcomes
- Automate!!!
DevOps: The Third Way - Practices

- Communication
- Collaboration
- Integration
DevOps Practice: Inject Failures

- Netflix services are hosted completely in Amazon Web Services cloud
- Design each distributed system to expect and tolerate failure
- Chaos Monkey randomly kills services within architecture in order to learn to tolerate and respond to failure

DevOps Approach
- How does this system react if I do this?
- Can we continue operations without this server?
- Will the users prefer option A or option B?

Traditional Approach
- Not in my system, you don’t
- Not in my system, you don’t
- Not in my system, you don’t
DevOps Practice: Make Your Improvement Work Visible

Along with regular user stories, use colored cards to indicate:
- Technical debt
- Unplanned work
- Experiments
- Learning backlog

Allocate time to improve daily work

Track the work needed to maintain overall health of the system
DevOps Practice: Regularly Improve Technical Debt

Allocate 20% of cycles to Technical Debt Reduction

- Write tests to find misconfigurations – and fix them
- Constantly run automated static code analysis during continuous integration and testing, and raise the bar in your quality gates
- Enforce consistency in code, environments, and configurations
- Repeatedly tackle the hard stuff
DevOps Practice:
Regularly Improve Tools

- Good tools are key to enabling DevOps collaboration, automation, and visibility
- Provide teams the best tools available
- Regularly invest time researching and piloting new tools
- Provide expert support
DevOps Practice: Reward Contributions to a DevOps Culture

- Incentivize DevOps practices and behaviors

- Recognize experimentation and risk-taking that leads to valuable learning

- Model honest self-assessment of organizational strengths and weaknesses and use of improvement techniques such as Toyota Kata

- Quantify and promote the link between DevOps practices and organizational performance
DevOps Practice: Conduct Deliberate Culture Change Experiments

Discussion: What are our biggest cultural challenges? What experiments should we run?
DevOps Team Profiles

DevOps Team Member

- End-to-end viewpoint
- Contributes to and uses visibility
- Automator
- Collaborative, cross-functional, friction reducer
- Participates in collective ownership of code and code delivery
- Personal success = team success
- Enjoys working this way

DevOps Expert Support Team

- Helps introduce DevOps-supportive processes and tools
- Works with teams to automate environment creation and deployment
- Helps teams use operational performance logs and dashboards
- Provides infrastructure support
## Food for Thought – Maturing DevOps Practices

<table>
<thead>
<tr>
<th>Culture &amp; Processes</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frequent commits</td>
<td>One backlog team and a master backlog</td>
<td>Team collaboration</td>
<td>Dedicated tools team for automation</td>
<td>Cross functional teams</td>
</tr>
<tr>
<td></td>
<td>Prioritized work</td>
<td>Adopt basic agile methods</td>
<td>Remove boundary of dev &amp; ops</td>
<td>Deploy disconnected from Release</td>
<td>No rollbacks</td>
</tr>
<tr>
<td></td>
<td>Defined &amp; documented process</td>
<td>Remove boundary of dev &amp; test</td>
<td>Act on metrics</td>
<td>Continuous improvement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defined &amp; documented process</td>
<td>Common processes for all changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>One backlog team and a master backlog</td>
<td>Act on metrics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adopt basic agile methods</td>
<td>Common processes for all changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove boundary of dev &amp; test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Team collaboration</td>
<td>Dedicated tools team for automation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove boundary of dev &amp; ops</td>
<td>Deploy disconnected from Release</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Act on metrics</td>
<td>Continuous improvement</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Common processes for all changes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove boundary of dev &amp; ops</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Define Context View, Logical Composition &amp; Physical Composition</td>
<td>Stage SDD wiki with Views in place</td>
<td>Review process in place</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Define related views</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build / Deploy</td>
<td>Level 1</td>
<td>Level 2</td>
<td>Level 3</td>
<td>Level 4</td>
<td>Level 5</td>
</tr>
<tr>
<td></td>
<td>Versioned code base</td>
<td>Poling builds</td>
<td>Auto triggered builds</td>
<td>Zero downtime deploy</td>
<td>Zero touch continuous deployments</td>
</tr>
<tr>
<td></td>
<td>Scripted builds</td>
<td>Build are stored</td>
<td>Automated tag &amp; versioning</td>
<td>Multiple build machines</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Basic scheduled builds</td>
<td>Manual Tag &amp; Versioning</td>
<td>Automated bulk of DB changes</td>
<td>Full automated DB deploys</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dedicated build server</td>
<td>First step towards standardized deploys</td>
<td>Basic pipeline with deploy to prod</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Documented manual deploy</td>
<td>Standard process for all environments</td>
<td>Scripted configuration changes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test &amp; Verification</td>
<td>Level 1</td>
<td>Level 2</td>
<td>Level 3</td>
<td>Level 4</td>
<td>Level 5</td>
</tr>
<tr>
<td></td>
<td>Automated Unit Tests (Coverage &lt;50%)</td>
<td>Automated Unit Tests (Coverage &gt;50% &amp; &lt; 80%)</td>
<td>Automated Functional tests</td>
<td>Automated acceptance criteria (80%)</td>
<td>All tests automated Coverage 100%</td>
</tr>
<tr>
<td></td>
<td>Separate test environment</td>
<td>Automated Integration Tests (Coverage ??)</td>
<td>Automated acceptance criteria (&lt;40%)</td>
<td>Automated performance tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaboration &amp; Information Sharing</td>
<td>Level 1</td>
<td>Level 2</td>
<td>Level 3</td>
<td>Level 4</td>
<td>Level 5</td>
</tr>
<tr>
<td></td>
<td>Baseline process metrics</td>
<td>Static code analysis</td>
<td>History of reports available</td>
<td>Report trend analysis</td>
<td>Reports accessible via common dashboard</td>
</tr>
<tr>
<td></td>
<td>Manual reporting</td>
<td>Quality reports</td>
<td>Traceability built into pipeline</td>
<td>Graphing as a service</td>
<td>Dynamic graphing</td>
</tr>
</tbody>
</table>
How much more productive, effective, and enjoyable might our work be? How much business value is left on the table due to unmatched demand and capacity?

Can we afford not to do DevOps?

Questions