The DevOps Handbook – Part 1 & 2

1. Preface xi
   1. Myth—DevOps is Only for Startups:
   2. Myth—DevOps Replaces Agile
   3. Myth—DevOps is incompatible with ITIL
   4. Myth—DevOps is Incompatible with Information Security and Compliance:
   5. Myth—DevOps Means Eliminating IT Operations, or “NoOps”
   6. Myth—DevOps is Just “Infrastructure as Code” or Automation:
   7. Myth—DevOps is Only for Open Source Software:
2. Foreword xix
3. Imagine a World Where Dev and Ops Become DevOps:
   1. THE CORE, CHRONIC CONFLICT
      1. Among them are the two following goals, which must be pursued simultaneously:
         1. Respond to the rapidly changing competitive landscape
         2. Provide stable, reliable, and secure service to the customer
   2. THE BUSINESS VALUE OF DEVOPS
      1. Code and change deployments (thirty times more frequent)
      2. Code and change deployment lead time (two hundred times faster)
      3. Production deployments (sixty times higher change success rate)
      4. Mean time to restore service (168 times faster)
4. An Introduction to The DevOps Handbook xxi
5. PART I—THE THREE WAYS 1
   1. Introduction
      1. THE LEAN MOVEMENT
         1. manufacturing lead time required to convert raw materials into finished goods was the best predictor of quality, customer satisfaction, and employee happiness
         2. one of the best predictors of short lead times was small batch sizes of work
   2. Agile, Continuous Delivery, and the Three Ways 7
   3. The First Way: The Principles of Flow 15
      1. MAKE OUR WORK VISIBLE
         1. A significant difference between technology and manufacturing value streams is that our work is invisible. Unlike physical processes, in the technology value stream we cannot easily see where flow is being impeded or when work is piling up in front of constrained work centers.
         2. By putting all work for each work center in queues and making it visible, all stakeholders can more easily prioritize work in the context of global goals.
      2. REDUCE BATCH SIZES
         1. Another key component to creating smooth and fast flow is performing work in small batch sizes.
         2. Large batch sizes result in skyrocketing levels of WIP and high levels of variability in flow that cascade through the entire manufacturing plant. The result is long lead times and poor quality
         3. Consider when we have an annual schedule for software releases, where an entire year’s worth of code that Development has worked on is released to production deployment. Like in manufacturing, this large batch release creates sudden, high levels of WIP and massive disruptions to all downstream work centers, resulting in poor flow and poor quality outcomes. This validates our common experience that the larger the change going into production, the more difficult the production errors are to diagnose and fix, and the longer they take to remediate.
      3. CONTINUALLY IDENTIFY AND ELEVATE OUR CONSTRAINTS
         1. If we improve a work center that is positioned before the constraint, work will merely pile up at the bottleneck even faster, waiting for work to be performed by the bottlenecked work center.
         2. If we improve a work center positioned after the bottleneck, it remains starved, waiting for work to clear the bottleneck.
         3. In typical DevOps transformations, as we progress from deployment lead times measured in months or quarters to lead times measured in minutes, the constraint usually follows this progression:
            1. Environment creation:
            2. Code deployment:
            3. Test setup and run:
            4. Overly tight architecture:
      4. ELIMINATE HARDSHIPS AND WASTE IN THE VALUE STREAM
         1. In the book Implementing Lean Software Development: From Concept to Cash, Mary and Tom Poppendieck describe waste and hardship in the software development stream as anything that causes delay for the customer, such as activities that can be bypassed without affecting the result. The following categories of waste and hardship come from Implementing Lean Software Development unless otherwise noted:
            1. Partially done work:
            2. Extra processes:
            3. Extra features:
            4. Task switching:
            5. Waiting:
            6. Motion:
            7. Defects:
            8. Nonstandard or manual work:
            9. Heroics:
   4. The Second Way: The Principles of Feedback 27
      1. KEEP PUSHING QUALITY CLOSER TO THE SOURCE
         1. In complex systems, adding more inspection steps and approval processes actually increases the likelihood of future failures. The effectiveness of approval processes decreases as we push decision-making further away from where the work is performed.
         2. Examples of ineffective quality controls include:
            1. Requiring another team to complete tedious, error-prone, and manual tasks that could be easily automated and run as needed by the team who needs the work performed
            2. Requiring approvals from busy people who are distant from the work, forcing them to make decisions without an adequate knowledge of the work or the potential implications, or to merely rubber stamp their approvals
            3. Creating large volumes of documentation of questionable detail which become obsolete shortly after they are written
            4. Pushing large batches of work to teams and special committees for approval and processing and then waiting for responses
         3. Instead
            1. Use peer reviews of our proposed changes to gain whatever assurance is needed that our changes will operate as designed.
            2. automate as much of the quality checking typically performed by a QA or Information Security department as possible
            3. Gary Gruver observes, “It’s impossible for a developer to learn anything when someone yells at them for something they broke six months ago—that’s why we need to provide feedback to everyone as quickly as possible, in minutes, not months.”
   5. The Third Way: The Principles of Continual Learning and Experimentation 37
      1. culture of fear and low trust
         1. workers who make mistakes are punished
         2. those who make suggestions or point out problems are viewed as whistle-blowers and troublemakers
         3. leadership is actively suppressing, even punishing, learning and improvement, perpetuating quality and safety problems
      2. high-trust culture
         1. we are all lifelong learners who must take risks in our daily work
         2. we learn from our successes and failures, identifying which ideas don’t work and reinforcing those that do
         3. local learnings are rapidly turned into global improvements, so that new techniques and practices can be used by the entire organization
      3. ENABLING ORGANIZATIONAL LEARNING AND A SAFETY CULTURE
         1. When we work within a complex system, by definition it is impossible for us to perfectly predict all the outcomes for any action we take. This is what contributes to unexpected, or even catastrophic, outcomes and accidents in our daily work, even when we take precautions and work carefully.
         2. Dr. Ron Westrum was one of the first to observe the importance of organizational culture on safety and performance. Dr. Westrum defined three types of culture:
            1. Pathological organizations are characterized by large amounts of fear and threat. People often hoard information, withhold it for political reasons, or distort it to make themselves look better. Failure is often hidden.
            2. Bureaucratic organizations are characterized by rules and processes, often to help individual departments maintain their “turf.” Failure is processed through a system of judgment, resulting in either punishment or justice and mercy.
            3. 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.
      4. INSTITUTIONALIZE THE IMPROVEMENT OF DAILY WORK
         1. Mike Rother observed in Toyota Kata that in the absence of improvements, processes don’t stay the same—due to chaos and entropy, processes actually degrade over time.
         2. We improve daily work by explicitly reserving time to pay down technical debt, fix defects, and refactor and improve problematic areas of our code and environments—we do this by reserving cycles in each development interval, or by scheduling kaizen blitzes, which are periods when engineers self-organize into teams to work on fixing any problem they want.
      5. LEADERS REINFORCE A LEARNING CULTURE
         1. Traditionally, leaders were expected to be responsible for setting objectives, allocating resources for achieving those objectives, and establishing the right combination of incentives. Leaders also establish the emotional tone for the organizations they lead. In other words, leaders lead by “making all the right decisions.”
         2. Greatness is not achieved by leaders making all the right decisions—instead, the leader’s role is to create the conditions so their team can discover greatness in their daily work. In other words, creating greatness requires both leaders and workers, each of whom are mutually dependent upon each other.
         3. Leaders must elevate the value of learning and disciplined problem solving. Mike Rother formalized these methods in what he calls the coaching kata. The leader helps coach the person conducting the experiment with questions that may include:
            1. What was your last step and what happened?
            2. What did you learn?
            3. What is your condition now?
            4. What is your next target condition?
            5. What obstacle are you working on now?
            6. What is your next step?
            7. What is your expected outcome?
            8. When can we check?
6. PART II—WHERE TO START 47
7. Ch. 5 Selecting Which Value Stream to Start With 51
   1. GREENFIELD vs BROWNFIELD SERVICES
      1. DevOps is not just for Greenfield
      2. Important Predictor – Is the application architected (or could be re-architected) for testability and deployability?
      3. Successful Brownfield transformations
         1. CSG – COBOL mainframe and supporting applications
            1. They 2X release frequency
            2. Resulted in increased application reliability
            3. Reduced deployment lead time from 2 weeks to <1 day
         2. Etsy
            1. “Barely survived the holiday retail season”
            2. Transformed every aspect of the organization
            3. Most admired DevOps organizations and successful 2015 IPO.
   2. CONSIDER BOTH SYSTEMS OF RECORD AND SYSTEMS OF ENGAGEMENT
      1. Gartner Bi-modal IT
         1. Type 1 – System of Record – “Doing it right”
         2. Type 2 – System of Engagement – “Doing it fast”
      2. DevOps helps reject the bi-modal IT model and lets you do both
   3. START WITH THE MOST SYMPATHETIC AND INNOVATION GROUPS
      1. Chrossing the Chasm
      2. Find the innovators and the early adopters
   4. EXPANDING DEVOPS ACROSS OUR ORGANIZATION
      1. As we generate success, we earn the right to expand the scope of the DevOps Initiative
         1. Find Innovators & Early Adopters
         2. Build Critical Mass & Silent Majority
         3. Identify the Holdouts
      2. Ron van Kemenade, CIO of ING, “Leading change requires courage, especially in corporate envrionments where people are scared and fight you. But if you start small, you really have nothing to fear. Any Leader needs to be brave enough to allocate teams to do some calcuated risk-taking.”
8. Ch. 6 Understanding the Work in Our Value Stream, Making it Visible, and Expanding it Across the Organization
   1. IDENTIFYING THE TEAMS SUPPORTING OUR VALUE STREAM
      1. Product Owner – the internal voice of the business; defines functionality
      2. Development - the team responsible for developing the application
      3. QA – team responsible for ensuring feedback loop exists to ensure functions as desired
      4. Operations – the team responsible for maintaing the production environment and ensuring service levels are met
      5. Infosec – team responsible for securing systems and data
      6. Release Managers – the people responsible for coordinating the production deployment processes
      7. Technology executives or value stream manager – in Lean, someone who is responsible for ensuring meets or exceeds…requirements…from start to finish
   2. CREATE A VALUE STREAM MAP TO SEE THE WORK
      1. Gain concrete understanding of how work is performed
      2. Focus investigation and scrutiny on
         1. Places where work must wait weeks or longer such as getting prod-like environments, approval processes, or review processes
         2. Places where significant rework is generated or received.
      3. Key metrics in value stream
         1. Lead Time
         2. Process time
         3. Percent Complete & Accurate (%C/A)
   3. CREATE A DEDICATED TRANSFORMATION TEAM
      1. Organizations use processes, specialization, and bureaucracies to perpetuate and protect against variance
      2. Changing how we work requires Disruption and Innovation
      3. Dr. Govindarajan & Dr. Trimble assert the need for dedicated transformation teams that are able to operate outside the rest of the organization
         1. Enables the forming of new institutional learning
         2. Enables experimentation with less impact
   4. AGREE ON A SHARED GOAL
      1. Define a measurable goal with a clearly defined deadline approximately 6 months to 2 years out.
   5. KEEP OUR IMPROVEMENT PLANNING HORIZONS SHORT
      1. Act like a startup, strive to generate measurable improvement or actionable data within weeks
   6. RESERVE 20% OF CYCLES FOR NON-FUNCTIONAL REQUIREMENTS AND REDUCING TECHNICAL DEBT
      1. Don’t just make the interest payments, pay down the principle
      2. If not managed, services become so fragile, delivery grinds to a halt
   7. INCREASE THE VISIBILITY OF WORK
      1. Keep the current state visible and up to date
      2. Use tools to reinforce desired behavior - anthropologists describe tools as a cultural artifact
      3. Create shared backlogs and reporting
9. Ch. 7 How to design Our Organization and Architecture with Conway’s Law in Mind
   1. “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.”
   2. ORGANIZATIONAL ARCHETYPES
      1. Functional-oriented
         1. Optimized for expertise, division of labor, or reducing cost
      2. Matrixed-oriented
         1. Combines functional and market orientation
         2. Often results in complicated structures
      3. Market-oriented
         1. Optimized for responding quickly to customer needs
         2. Often flat, cross-disciplined
         3. Most aligned with DevOps
   3. PROBLEMS OFTEN CAUSED BY OVERLY FUNCTIONAL ORIENTATION (“OPTIMIZING FOR COST”)
      1. Leads to limited outward visibility
      2. Encourages long queues and long lead times
      3. Poor handoffs and large rework
   4. ENABLE MARKET-ORIENTED TEAMS (“OPTIMIZING FOR SPEED”)
      1. Goal of many small teams working safely and independently, quickly delivering value
      2. Aligned with full life-cycle of the application
      3. Used by Amazon and Netflix
   5. MAKING FUNCTIONAL ORIENTATION WORK
      1. Requires everyone in the value stream views the customer and organizational outcomes as a shared goal, regardless of where they reside in the organization.
      2. Requires high-trust cultures, transparent prioritization, and sufficient slack in the system
      3. Toyota Kata, “…one cannot reorganize your way to continuous improvement and adaptiveness. What is decisive is not the form of the organization, but how people act and react. The roots of Toyota’s success lie not in its organization structures, but in developing capability and habits in its people.”
   6. TESTING, OPERATIONS, AND SECURITY AS EVERYONE’S JOB, EVERY DAY
      1. In high-performing organizations – shared common goal that is part of everyone’s job, every day
   7. ENABLE EVERY TEAM MEMBER TO BE A GENERALIST
      1. I-shaped – specialist
      2. T-shaped – generalists
      3. E-shaped
   8. FUND NOT PROJECTS, BUT SERVICES AND PRODUCTS
      1. Create stable service teams
      2. Enable them to execute strategy and roadmap.
   9. DESIGN TEAM BOUNDARIES IN ACCORDANCE WITH CONWAY’S LAW
      1. Separation impedes collaboration
      2. Requires significant collaboration to overcome
   10. CREATE LOOSELY-COUPLED ARCHITECTURES TO ENABLE DEVELOPER PRODUCTIVITY AND SAFETY
       1. Loosely coupled services with bounded contexts
       2. Services can update independently without having to update other services.
       3. Decouple from shared infrastructure – shared databases
       4. Bounded context – compartmentalized and well-defined interfaces
   11. KEEP TEAM SIZES SMALL (THE “2-PIZZA TEAM” RULE)
10. Ch. 8 How to Get Great Outcomes by Integrating Operations into the daily Work of Development
    1. CREATE SHARED SERVICES TO INCREASE DEVELOPER PRODUCTIVITY
       1. Enable Dev teams to spend more time building functionality, as opposed to created infrastructure
       2. Automation – across the board
       3. Create known, good environments that are ready for production
    2. EMBED OPS ENGINEERS INTO OUR SERVICE TEAMS
       1. Enables more self-sufficiency
       2. Create with operational maintainability in mind
    3. ASSIGN AN OPS LIAISON TO EACH SERVICE TEAM
       1. Reduces costs compared to embedded operations
    4. INTEGRATE OPS INTO DEV RITUALS
       1. Help better understand development culture.
       2. Radiate operational concerns into the product or service teams
       3. Makes operations work visible on shared boards