Case Study:
Optimizing Delivery Processes at <Name Not Disclosed>

Introduction:
The department is responsible for the development of a wide range of products in Company software suite for telecom providers. This involves managing large amounts of legacy code tied together by a centrally-managed platform of deployment and configuration tooling.
In the last 3 years they've moved to an agile, scrum-based, development process. All the software builds were migrated to Maven and automated for CI using Jenkins. Still - the release cycles are too long, deadlines aren't getting met. The central infrastructure team is overloaded and burnt out, developers are dissatisfied.

Objective:
Evaluate and map out existing delivery processes. Identify bottlenecks. Define KPIs for software delivery. Create an optimization roadmap. Verify optimization results with regular measurements.

Assessment Methods:
Personal interviews.
Delivery systems review.
Value stream mapping.

Some Initial Findings:

- SCM:
  - Perforce is used for most projects. Some have started looking at Git for more dynamic flows.
- CI:
  - Java builds based on Maven.
    - Maven migration completed a couple of years ago.
    - Low level of Maven competence in dev.
    - Reliance on infra team support causes bottlenecks and burnout.
  - Jenkins-based CI implemented and supported by a central infrastructure team
    - Unstable build infra
    - 30-40% of false negatives due to infra issues
    - Low developer trust in CI results
  - Monolithic build and deployment processes

        - Everything gets rebuilt and redeployed for every change
        - No defined focal persons for specific product regions
        - Engineers feel helpless in the face of failures related to other system parts.
        - Nightly deployment failures close to 90%.
    - Test automation
        - Average unit test coverage around 30%
        - Large long-running integration test suites (2-8 hours)
        - A lot of flaky tests (not measured)
    - Manual Approval
        - Release manager has to complete a checklist before approval
        - Sometimes takes up to 2 weeks
- Project management
    - Unsuccessful Jira adoption in infra teams
    - Dev teams in transition to Jira
    - Overall - reliance on TLs/Scrum masters to update issue statuses and descriptions
- Inter-team communication
    - Most information is transferred by email
    - Often the team leaders become the bottlenecks for information and issue resolution.
    - No transparency regarding what each team is working on and what their backlog is.
    - No central information hub. **Confluence** exists but not maintained.
    - Upstream changes aren't getting properly communicated to downstream teams.
    - Changes are pushed downstream. This creates a load of unplanned work for downstream teams.
- Reporting
    - No centralized build/deployment success-failure stats.
    - No defined flow for infrastructure issue resolution and follow-up.
- Infrastructure Tooling
    - Jenkins in CI with custom plugin for complex flows
    - Custom deployment and configuration tool
        - High dependence on core infra team for support
    - VMWare-based environments with support for dynamic provisioning
        - Constant resource shortage. Deployments queueing up.

Recommendations:

- Ensure cross-the-board Jira adoption
    - Provide Otomato training for Jira project management
- Conduct cross-functional training and knowledge sharing sessions

    - Discover areas of potential improvement
    - Define self-service tooling roadmap
    - Create a shared knowledge domain
    - Reduce day-to-day dev dependency on infra support
- Create shared boards for development and infra teams
- Where needed - restructure to truly cross-functional product teams
- Define focal persons for knowledge base maintenance (Confluence)
    - Provide incentives for creating and sharing useful content
- Integrate Jira with Perforce (future - Git) , Jenkins, Confluence and environment monitoring solutions to create a unified information flow.
- Define a set of flow measurement KPIs. Create a metric database. Start taking measurements.
- Create dashboards and alerts for the defined KPIs.
- Migrate deployment automation to Ansible or another IaC tool.
- Review system architecture to allow modularity and componentization.
- Migrate Jenkins builds to pipeline-as-code and transfer pipeline ownership to development.
- Eliminate manual approvals.
- Evaluate expanding lab infrastructure into public clouds (AWS) for faster resource provisioning.
- Move to a pull-based system for downstream change propagation.

Mid-term results:

- Created a cross-functional mastermind team with members from dev, infra and QA.
- Built a yearly automation roadmap to provide answers for the most acute pains.
- A set of standard flow measurement KPIs were defined : 4 velocity metrics and 4 quality metrics.
    - An ELG system established for semi-automated metric collection
    - A roadmap for fully-automated metric collection created.
    - Gradual Improvement in all the KPIs achieved after 6 months.
- Defined a CI/CD system transition to modular microservice based delivery
    - Completed as a part of cross-company microservice initiative.
    - Provided technology evaluations.
    - Defined best practices for inter-component testing and code reviews.
- Provided training and support in migrating teams to Git
    - Defined branching methodology
- Defined an incident follow-up procedure for CI failures.
- First environments spinned up on AWS for greenfield projects.

Clarification:

The work completed with ... department was a part of a company-wide DevOps initiative. Among other topics this also included assisting with evaluating the migration to microservice-based architectures. Even though Otomato has provided a number of technological evaluations for this migration (e.g : Jenkins pipeline POC, Docker/Kubernetes training and architecture) - we must note that most delivery challenges lie in the fields of :

- Collaboration
- Knowledge sharing
- Organizational structure

The sessions we've conducted and the new practices we've established helped a lot with the first two. Department management started looking into re-organizing the teams but this is understandably harder to do.
Changing large organizations is hard. A lot of politics is involved in achieving consensus regarding technological and organizational decisions.
The achieved results currently enable engineers to do more with existing systems. Information is more easily accessible. There's less friction in day-to-day delivery activities. This is manifested by the improved metrics.
After 6 months we feel that the customer is on the road to further improvement. Still - careful monitoring of the process and iterative implementation of defined roadmaps is required. To ensure there is no fallback to old practices.