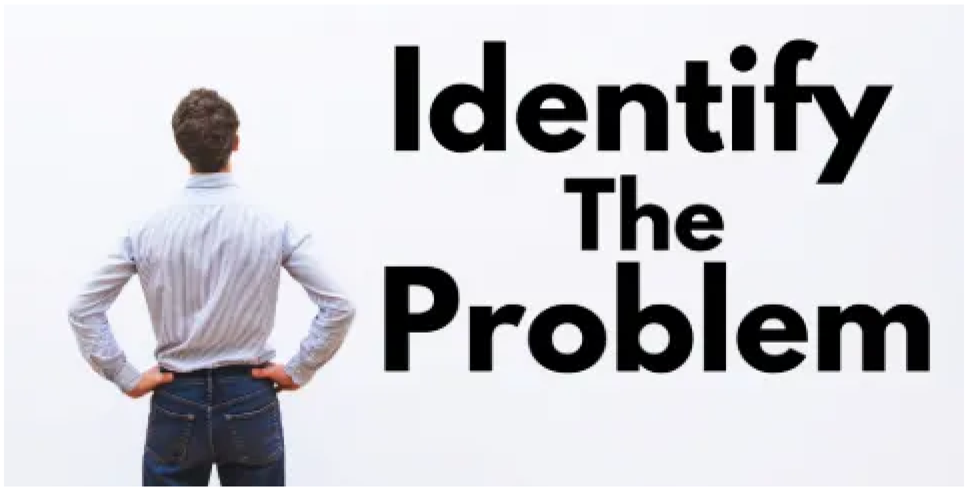
How I Simplified Data Engineering as a Lead to Eliminate Stress and Dependencies

Anurag Sharma 6 min read · Aug 4, 2025

21

As a Data Engineering Lead, my goal was to create a team environment where production issues didn't cause panic, team members could operate independently, and even my absence (or the primary developer's vacation) wouldn't disrupt operations—achieving this required identifying core problems, implementing robust solutions, and fostering a culture of clarity and accountability.

Here's how I did it.



Step 1: Identifying the Problems

The first step was understanding the challenges that led to stress, bottlenecks, and dependency on specific team members. The key issues we faced included:

- **Lack of Clarity in Responsibilities:** Team members were often unsure who should handle specific tasks, leading to delays or duplicated efforts.
- **Dependency on Key Individuals:** Production (P1) issues often required the primary developer or me to step in, creating bottlenecks when we

were unavailable.

- **Poor Documentation:** New team members or those picking up unfamiliar tasks struggled to understand pipelines, data flows, or dependencies due to incomplete or scattered documentation.
- **Ad-Hoc Requests Overload:** Frequent "quick connect" requests for clarifications disrupted workflows and increased stress.
- **Inconsistent Prioritization:** Without clear prioritization, team members sometimes focused on less critical tasks while urgent issues lingered.

By identifying these problems, we can address them systematically and create a framework that enables the team to operate autonomously.

Step 2: Defining Responsibilities and Priorities

To create a stress-free environment, we clearly defined each team member's responsibilities and established a shared understanding of task priorities.

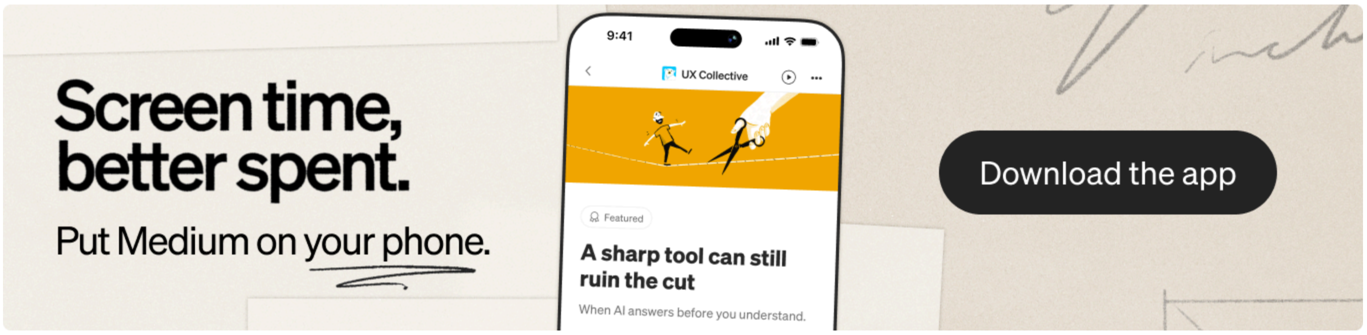
The data engineering team's responsibilities included:

1. **Handling Production Issues (P1):** Critical issues affecting live systems, requiring same-day resolution.
2. **Addressing Product Owner/Data Analyst Queries (P1):** Immediate responses to ensure business continuity and stakeholder satisfaction.
3. **Handling Testing Team Queries (P2):** Supporting QA with timely clarifications to keep testing on track.
4. **Developing New Pipelines (P2):** Building new data pipelines to support business requirements.
5. **Optimizing the Entire Project (P3):** Improving performance, scalability, and efficiency of existing pipelines.
6. **Ad-Hoc Tasks (P4):** Miscellaneous requests, such as one-off analyses or minor updates.

We agreed that P1 tasks must be resolved the same day, regardless of who was available. P2 tasks had a 1–2 day turnaround, while P3 and P4 tasks could be scheduled based on sprint planning. This prioritization ensured everyone knew what to focus on, reducing confusion and stress.

Step 3: The Power of Comprehensive Documentation

The cornerstone of our solution was **clear, comprehensive, and centralized documentation**. Recognizing its importance, we dedicated an entire sprint to documenting all existing pipelines and processes. This wasn't just a "niceto-have" — it became the foundation for our team's independence and efficiency.


Screen time, better spent.
Put Medium on your phone.
9:41
UX Collective
Featured
A sharp tool can still ruin the cut
When AI answers before you understand.
Download the app

For example, consider a reporting pipeline we developed to track IoT device software updates. This pipeline generated a report showing how many devices were updated to the latest software and how many were pending, similar to the "Restart your system for updates" prompt in Windows. Below is how we documented this pipeline (and all others) to ensure clarity and accessibility:

Documentation Components

Team Details:

- Names and roles of everyone involved: Product Owner, Report Owner, Business Analyst, Scrum Master, Lead Developer, Peer Developer, QA, and DevOps (for deployment).
- Contact information and availability notes (e.g., backup contacts for vacations).

Requirement Documentation:

- Link to the requirement document outlining the business need and scope.

- Mock report UI link showing the expected output (e.g., Power BI dashboard mockup).

Technical Artifacts:

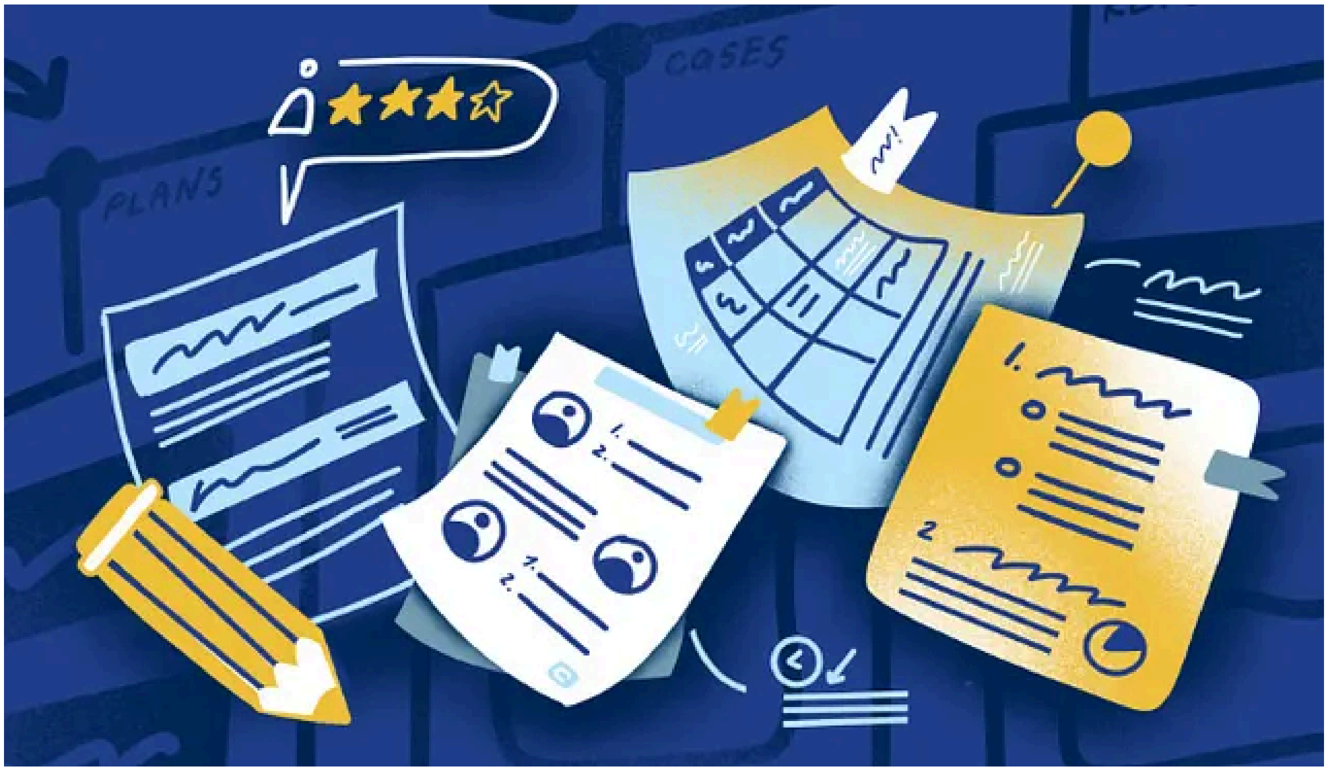
- **GitHub Links:** Code repositories for scripts, configurations, and version history.
- **Airflow DAG Links:** Links to Apache Airflow DAGs for pipeline orchestration.
- **Data Flow Diagram Links:** Visuals showing how data moves from source to target.
- **Data Lineage Diagram Links:** Detailed mappings of source-to-target transformations.
- **Databricks Notebook Links:** Notebooks containing ETL logic, transformations, and queries.

Testing Artifacts:

- **Test Case Links**: Detailed test cases for QA validation.
- **Unit Testing Queries:** Databricks notebooks with unit test scripts to verify individual components.
- **Integration Testing Queries:** Notebooks for end-to-end pipeline testing.
- **Data Analyst Testing Queries:** Power BI or SQL queries used by analysts to validate final outputs.

Change History:

- Links to all subsequent stories, updates, or bug fixes related to the pipeline, ensuring a complete audit trail.



Additional Documentation Practices

To make documentation even more effective, we implemented the following:

- **Standardized Templates**: Created templates for each pipeline's documentation to ensure consistency across projects. This included sections for architecture, dependencies, troubleshooting guides, and FAQs.
- **Centralized Confluence Space**: All documentation was stored in a single, searchable Confluence space, organized by project and pipeline. This eliminated the need to hunt for information across emails, Slack threads, or shared drives.

- **Version Control:** Documentation was versioned alongside code in GitHub, ensuring it stayed up-to-date with pipeline changes.
- **Troubleshooting Guides:** Each pipeline's documentation included a "Runbook" section with step-by-step instructions for common production issues (e.g., "What to do if the Airflow DAG fails").
- **Onboarding Guide:** A dedicated Confluence page for new team members, linking to all pipelines, tools, and processes, with a checklist to get started.

Impact of Documentation

The impact of this one-time documentation effort was transformative:

- **New Developers Empowered:** A new team member could visit the Confluence page, understand the pipeline's purpose, architecture, and dependencies, and start working on issues without needing to interrupt others.
- **Reduced "Quick Connect" Requests:** Clear documentation eliminated the need for constant clarifications, saving hours of back-and-forth on Teams or Slack.
- **Vacation-Proof Operations:** When the primary developer or I were on vacation, the team could handle P1 issues using the runbooks and documentation, ensuring zero disruptions.
- **Faster Onboarding:** New hires could ramp up in days, not weeks, thanks to the centralized onboarding guide and detailed pipeline documentation.

Step 4: Additional Strategies to Remove Dependencies

Beyond documentation, we implemented several practices to further reduce dependencies and stress:

Cross-Training and Knowledge Sharing:

- Regular "knowledge transfer" sessions where developers presented their pipelines to the team, ensuring at least two people were familiar with each pipeline.
- Pair programming for complex stories to spread expertise across the team.

Automated Monitoring and Alerts:

- Configured Airflow and Databricks with automated alerts for pipeline failures, with clear instructions in the runbook for resolving common issues.
- Used monitoring tools (e.g., Datadog or custom scripts) to proactively detect performance bottlenecks or data quality issues.

Standardized Development Practices:

- Enforced consistent coding standards (e.g., SQL style guides, naming conventions) to make codebases easier to understand.
- Used modular, reusable components in Databricks notebooks to simplify maintenance and updates.

Self-Service for Stakeholders:

- Built self-service dashboards for data analysts and product owners to query data directly, reducing P1 query requests.
- Provided read-only access to Airflow and Power BI for stakeholders to track pipeline status themselves.

Continuous Improvement:

- Held quarterly "optimization sprints" to address P3 tasks, such as refactoring inefficient pipelines or improving data quality checks.
- Encouraged team members to propose improvements, fostering a culture of ownership and collaboration.

Step 5: Measuring Success

To quantify the impact of these changes, we tracked key metrics:

- **Mean Time to Resolve (MTTR) P1 Issues**: Reduced from 4 hours to under 1 hour, even during vacations.
- **Onboarding Time**: New developers were productive within 3 days, compared to 2 weeks previously.
- **Stakeholder Query Response Time**: P1 queries dropped by 60% due to self-service tools and documentation.
- **Team Satisfaction**: Surveys showed a 40% increase in team satisfaction, with fewer complaints about stress or unclear responsibilities.

Conclusion

By identifying pain points, clearly defining responsibilities, prioritizing tasks, and investing in comprehensive documentation, I transformed our data engineering team into a self-sufficient, stress-free unit. The one-time effort of perfect documentation paid dividends by empowering team members, eliminating dependencies, and ensuring smooth operations even during vacations or unexpected issues. As a lead, my role shifted from firefighting to coaching, enabling the team to thrive independently. The days of "quick connect?" messages are long gone, replaced by a culture of clarity, collaboration, and confidence.

Big Data Data Analysis Optimization Documentation Data Engineering

Written by Anurag Sharma

83 followers · 3 following

Edit profile

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

No responses yet

Anurag Sharma him/he

What are your thoughts?