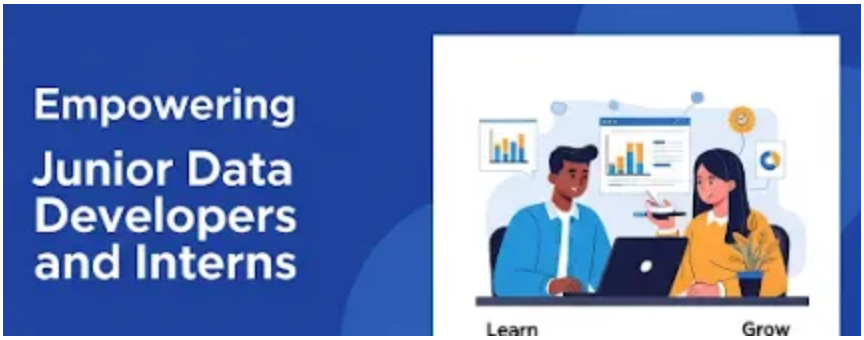


Empowering Junior Developers and Interns: How We Reduced Senior Developer Dependency for Production Failure Handling

Anurag Sharma 6 min read · Jun 13, 2025

At our analytics-focused company, senior developers, data engineers, and data leads juggle a multitude of responsibilities: building new data pipelines, designing frameworks, addressing data discrepancy queries from analysts and product owners, and ensuring team operations run smoothly. However, one recurring drain on their time was handling production (prod) issues –

often low-priority failures caused by environment glitches, Spark errors, missing upstream data, or data quality (DQ) issues. These disruptions pulled senior talent away from strategic work, creating bottlenecks. To address this, we implemented a robust documentation strategy that empowered junior

The Challenge: Senior Developers Overwhelmed by Prod Issues

With over 400 data pipelines running in production and a team of 40 developers building new pipelines daily, our analytics company faced constant prod challenges.

Failures from:

• Environment Issues: Misconfigured clusters or resource shortages.

• Spark Errors: Memory overruns, shuffle failures, or executor crashes.

• Upstream Data Issues: Missing or malformed data from source systems.

• Data Quality Failures: Inconsistent formats, null values, or schema mismatches.

Historically, our DevOps team would debug these issues alongside the job's original developer, often a senior team member, before escalating to an incident. This process was time-intensive, pulling senior developers away from critical tasks like pipeline innovation or framework development. Junior developers and interns, eager to contribute, lacked the context or

confidence to tackle these issues independently. We needed a system to democratize failure resolution without compromising reliability.

The Solution: Comprehensive Job Documentation in Confluence

To shift the responsibility of handling prod failures to junior developers and interns, we prioritized comprehensive, accessible documentation for every pipeline. We dedicated two entire sprints to documenting our 400+ pipelines, with each developer tasked with documenting at least 10 jobs end-to-end. Moving forward, we mandated that new pipelines include documentation as part of their story points, ensuring this practice became a standard part of our workflow.

What's in the Documentation?

The documentation, hosted on Confluence, is designed to be a one-stop resource for anyone – regardless of experience level – to troubleshoot and resolve prod issues. Each pipeline's documentation includes:

1. Pipeline Creation Context:

◦ Links to the original requirement document, initial design draft, and first code commit.

◦ A detailed history of enhancements, including the rationale for each code change (e.g., "Added retry logic to handle intermittent API failures").

◦ A high-level overview of the pipeline's purpose, input sources, output destinations, and source-to-target Mapping.

2. Incident History and Resolutions:

◦ A log of all incidents raised against the pipeline since deployment, including:

▪ The root cause (e.g., "Spark OOM error due to large shuffle").

▪ Detailed steps taken to resolve the issue, including code changes, configuration tweaks, or upstream coordination.

▪ SQL queries or scripts used to validate the fix (e.g., "Ran SELECT COUNT(*) FROM table WHERE date = '2025–06–13' to confirm data completeness").

◦ A summary of recurring issues and their patterns (e.g., "Fails every month-end due to high data volume; increase executor memory to 16GB").

3. Spark-Specific Troubleshooting Guide:

◦ A catalog of Spark-related failures (e.g., memory errors, skewed partitions, or executor timeouts).

◦ Step-by-step mitigation instructions, such as:

▪ “For OOM errors, check Spark UI for memory usage; increase spark.executor.memory by 2GB increments.”

▪ “For skewed data, enable adaptive query execution (spark.sql.adaptive.enabled=true).”

◦ Links to relevant Spark documentation or internal wikis for deeper understanding.

4. Ownership and Collaboration Details:

◦ A list of all developers who have contributed to the pipeline, with their contact information for quick escalation if needed.

◦ The primary owner of the pipeline, responsible for major updates or architectural decisions.

◦ A changelog of who made what changes and when, linked to Git commits for transparency.

5. Upstream and Downstream Dependencies:

◦ A map of upstream data sources, including their schemas, refresh schedules, and points of contact for upstream teams.

◦ Downstream consumers (e.g., dashboards, ML models) and their data requirements to prioritize fixes based on impact.

◦ Common upstream failure scenarios (e.g., “Source API drops[mid|truncate] fails if data is missing; retry after 5 minutes”) and recommended workarounds.

6. Data Quality Checks and Fixes:

◦ Predefined DQ validation queries (e.g., “Check for nulls: SELECT * FROM table WHERE key_column IS NULL”).

◦ Common DQ issues and their resolutions (e.g., “For invalid timestamps, apply TO_TIMESTAMP(column, ‘yyyy-MM-dd’)”).

◦ Instructions for triggering DQ remediation scripts or contacting the DQ team.

7. Environment Setup and Configuration Notes:

◦ Details on the pipeline's environment setup, such as required cluster size, IAM roles, or S3 bucket configurations.

◦ Known environment pitfalls (e.g., "Job fails if S3 bucket versioning is disabled; enable it via AWS console").

◦ Recommended cluster settings for optimal performance (e.g., "Use 10 r5.xlarge executors for datasets > 100GB").

8. Runbook for Common Scenarios:

◦ A checklist for handling frequent failure types (e.g., "If job times out, extend spark.sql.execution.timeout to 3600s").

◦ Prebuilt scripts or commands to restart jobs, clear checkpoints, or rebuild tables.

◦ Guidelines for when to escalate to DevOps or senior developers (e.g., "Escalate if cluster fails to initialize after three retries").

Example Documentation Snippet

Pipeline: iot_temperature_aggregation

• Purpose: Aggregates hourly temperature data from IoT sensors into daily summaries.

• Resources: Requirement Doc, Initial Commit, Enhancement History.

```
Incident Log:
2025-06-15: Failed due to Spark shuffle spill (fixed by increasing spark.memory.
2025-06-16: Missing upstream data (reran job after upstream team fixed API).
```

• **Known Spark Issues:**

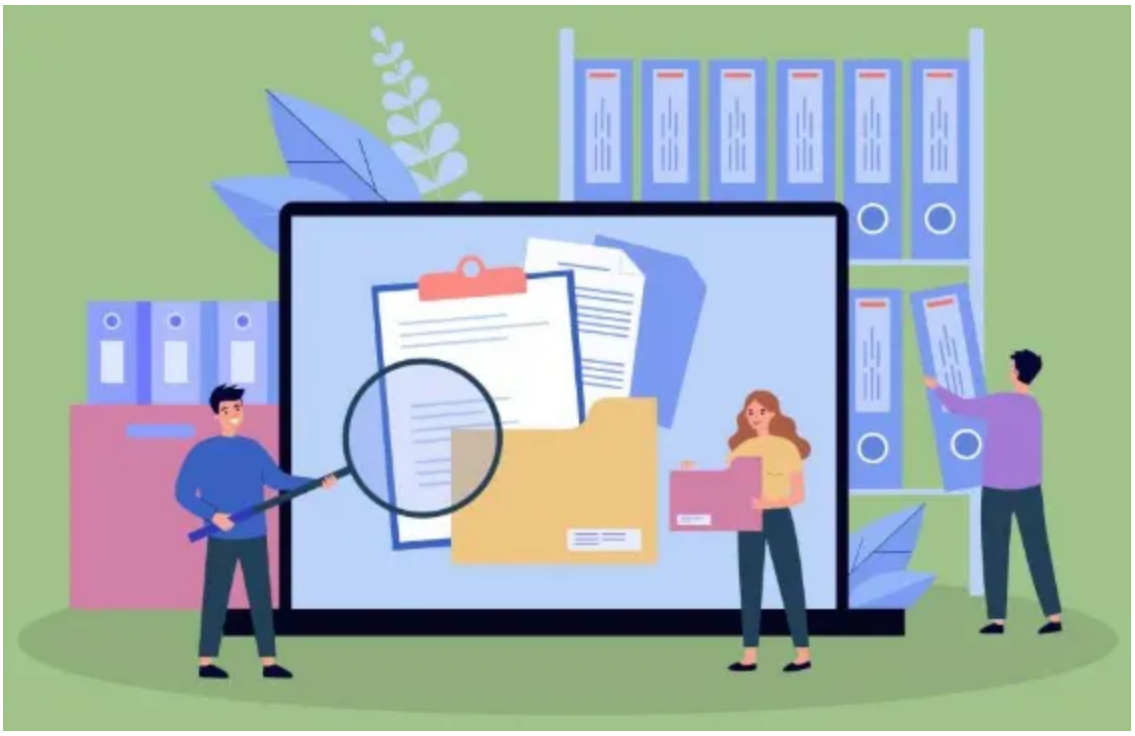
◦ OOM: Increase spark.executor.memory to 16GB; monitor shuffle metrics.

◦ Skew: Repartition data using repartition(col('sensor_id')).

• Owners: Primary: Anurag Sharma (anurag.sharma@company.com); Contributors: Anurag Sharma, Amit Tandon.

• Dependencies: Upstream: raw_sensor_data (refreshes hourly); Downstream: daily_temp_dashboard.

• DQ Checks: Run SELECT sensor_id, COUNT(*) FROM raw_sensor_data GROUP BY sensor_id HAVING COUNT(*) > 24 to detect duplicate hourly records.



Implementation: A Sprint for Documentation

To kickstart this initiative, we allocated a two-week sprint solely for documentation. Each of our 40 developers documented at least 10 pipelines, covering the above details. We created a standardized Confluence template to ensure consistency, with sections for each of the eight components. For new pipelines, we integrated documentation into the development process by allocating a fixed number of story points (e.g., 2 points for documentation per pipeline). A dedicated review team ensured completeness and clarity, mentoring junior developers on effective documentation practices.

How It Empowered Junior Developers and Interns

The detailed documentation transformed how we handle prod failures:

• **Self-Sufficiency:** Interns and junior developers can now resolve 60% of prod issues without senior intervention. For example, an intern fixed a Spark

OOM error by following the documented steps to adjust executor memory, a task previously requiring a senior developer.

• **Faster Resolution:** The runbooks and incident logs provide clear, actionable steps, reducing resolution time by 50%. A junior developer resolved a DQ failure in 30 minutes by running a documented validation query, compared to hours of debugging in the past.

• **Knowledge Transfer:** The documentation serves as a training tool, helping new team members learn pipeline logic, Spark troubleshooting, and DQ best practices.

• **Reduced Escalations:** The escalation guidelines clarify when senior input is needed, cutting unnecessary handoffs to DevOps or senior developers by 60%.

• **Team Confidence:** Junior developers and interns feel empowered to tackle prod issues, boosting morale and productivity.

Additional Benefits:

1. Consistency Across Teams: Standardized documentation ensures all teams follow the same troubleshooting protocols, reducing errors.

2. Auditability: The incident log and changelog provide a clear audit trail for compliance and post-mortem analysis.

3. Scalability: As our pipeline count grows, the documentation framework scales effortlessly, supporting new jobs without additional overhead.

4. Time Savings for Seniors: Senior developers now focus on high-value tasks like framework development, with prod issues handled by junior team members 80% of the time.

Challenges and Lessons Learned:

Initially, some developers viewed documentation as a burden, so we emphasized its long-term time-saving benefits. We also learned to keep documentation concise yet comprehensive, avoiding overwhelming details. Regular audits of Confluence pages ensure they stay up-to-date, with owners assigned to review their pipelines quarterly.

Conclusion:

By investing in detailed, structured documentation for our 400+ data pipelines, we empowered junior developers and interns to handle production failures with confidence, significantly reducing the dependency on senior developers. This initiative not only slashed resolution times and job failures but also fostered a culture of self-reliance and collaboration.

Data Engineering | Data Analytics | Big Data | Automation | Documentation

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!