Michael Leonhard's Resume

michael206@gmail.com 415.361.1808 linkedin.com/in/mleonhard San Francisco, CA

Flexport (flexport.com), Software Engineer, 2022 July - Present

I work on Flexport's web app for customers. My team owns the Reports and Analytics portions of the app. Customers can view standard reports, build custom reports with 5 data grains and >200 columns, and schedule automated emails with CSV attachments. The product is built with React, Flow, Relay, GraphQL, Ruby on Rails, PostgreSQL RDS, Debezium, Kafka, Looker API, Snowflake, dbt, AWS, and Terraform. I joined the team in the middle of a multi-year product rewrite. What I have done:

- Coded features, fixed bugs, and wrote needed automated tests.
- Wrote documentation of our system's technical architecture. Five new team members have used this documentation to quickly learn our system.
- Closed 87 tickets, authored 90 PRs, reviewed 104 PRs, and onboarded a teammate.
- Participate in the oncall rotation (I'm primary oncall as I write this).
- Participated in diagnosing and resolving 7 production incidents. Wrote incident reports. During one large outage, I provided critical knowledge of CORS and how to fix the root problem.
- Identified several important risks in the new system: a security vulnerability, disaster recovery, developing in production, and a few others. We addressed all of them.
- Built Datadog dashboards for the various parts of our system. Cleaned up how the code generates metrics and spans: filled gaps, removed duplicate and obsolete parts, and enabled spans from worker threads. The new dashboards have been crucial sources of information during several production incidents. They have also guided us in our work on improving page load times.
- The new version of our product had a problem that was causing page timeouts for some customers. I refactored a sizable portion of our product's backend and solved the performance issue. The refactoring has unblocked work on some long-standing bugs. The new version follows industry best practices: DRY, YAGNI, KISS, OOP, static types, and testing. My teammates like working with the new code.
- Wrote detailed proposals for new features, improving product quality, and increasing developer velocity. Reviewed proposals from junior teammates and coached them in design doc writing.
- Studied our system's data store (Snowflake), built a query latency dashboard, and investigated ways to reduce query processing times. Consulted our Snowflake account manager and support engineers.
- Identified the gap in our team's software process that was preventing the new version from achieving the same level of data quality as the legacy version: missing tests.
    - Proposed that we build a "CSV Diff" tool to compare the output of old and new code. We built this tool and used it to identify discrepancies between the reports.
    - Wrote a proposal for testing the new system.
    - Created a manual test procedure. It has prevented many production incidents.

- Dug deep into the latency of some slow SQL queries to PostgreSQL. I checked index usage with EXPLAIN ANALYZE, measured cache effectiveness, studied Amazon EBS and how RDS uses it, and analyzed our EBS metrics. I wrote a detailed proposal for specific changes to the Postgres and EBS configs. The infrastructure team has implemented some of them, leading to faster query times.
- Flexport's web apps and API break about once a week. I identified a gap in the company's software engineering process that is the source of these frequent production incidents: CICD without the CI. I did these things to improve the situation:
    - Lobbied management to invest in integration testing
    - Organized a cross-functional group on integration testing, with points of contact on each engineering team in our division.
    - Built our first Datadog synthetic browser checks to monitor our production deployment and quickly alert us to problems. Wrote a doc with best-practices and necessary info that is missing from Datadog's docs. Already, one other team has used this doc and built checks for their customer-facing pages.

Applin www.applin.dev, Founder, April 2022-Present

Applin is a system for making mobile apps with only server-side code. It is a Server-Driven User Interface (SDUI). Airbnb and other large companies use internal SDUI systems for their apps. Applin is open-source commercial software. I am developing it in the open on Github.

- iOS frontend in native Swift and UIKit: github.com/leonhard-llc/applin-ios
- Ruby on Rails backend: github.com/leonhard-llc/applin-rails
- Documentation: www.applin.dev/docs/

FOSS contributions, Rust libraries, GitHub, Personal projects

- safe-regex - A Rust regular expression library. It compiles regular expressions to Rust code, which is then compiled and optimized as part of your program. 350k downloads.
- servlin - A modular HTTP server library in Rust. Uses async internally for excellent performance under load. Unique feature: ergonomic threaded (non-async) request handlers.
- temp-dir and temp-file - Rust libraries to help tests clean up files they create. 950k downloads.
- safina - A safe Rust async runtime
- dns-server - A threaded DNS server library in Rust
- cloudping.info - Measure latency from your browser to various cloud datacenters
- See others on my website: www.tamale.net

Cozy Date App, Founder, 2018-2022 (3.5yr)

- Interviewed hundreds of dating app users and identified key pain points
- Wrote strategies to reduce user pain and enhance positive user experiences
- Studied UX design, wrote user personas, made user flow diagrams, designed the app UI
- Implemented iOS & Android apps in Flutter + Dart
- Implemented the backend with Rust, Rouille, Diesel, PostgreSQL, JSON, and Heroku
- Tested the app with users and iterated on the UI
- Coded some integration tests for tricky codepaths, including end-to-end tests
- Hired a designer to design the website, logo, color scheme, and app icon
- Coded the website in HTML, CSS, JavaScript, Bootstrap, Netlify: www.cozydate.com
- Researched monetization strategies, interviewed cafe owners as potential ad buyers

Google, Software Engineer, 2013-2018 (5yr 2mo)

= Assimilator Team, Technical Infrastructure, 2015-2018 =
Assimilator is an internal system that manages system software on Google's server machines. It installs/updates/rolls-back the Linux kernel, firmware, OS software, machine certificates, and Google system software. Assimilator daemon runs on every Google machine and router. Every datacenter has its own controller instance. Its scheduling functions are extremely complicated because they implement canaries and pipelined releases and comply with machine downtime restrictions for Google Cloud users.

The project had accumulated a lot of technical debt and its maintainers had all left. I was the first member of the new dev team. What I did:

- Refactored the C++ codebase to make it testable. Wrote unit tests. Wrote integration tests for all features in use.
- Added features to unblock other teams
- Removed obsolete features
- Learned Golang and rewrote portions of the system in Golang. This was a multi-year effort with my excellent teammates.
- Mentored a junior teammate
- Got promoted

= Machine Database Team, Technical Infrastructure, 2013-2015 =
Machine Database was the database of record for every piece of hardware costing more than $5 in every Google data center. What I did:

- Helped teammates re-write API server in Java & gRPC
- Ported 100 C++ client programs from other teams to use the new API, and numerous Java & Python clients
- Attention to detail: While porting one client, I identified a bug in how it used our API. The bug would have caused a major outage affecting public apps. I submitted a PR. The other team enthusiastically accepted it and told our Director who gave me a bonus.

- Set up continuous integration tests for schema migrations with API deployments and rollbacks. This saved us a lot of trouble and increased our velocity in changing the database.
- Added features to support new kinds of hardware and new datacenter management software
- Removed obsolete columns, tables, and API features

Rest Backup, Founder, 2011-2012 (2yr)

- Created a REST API for storing application data backup files
- Built a web app for downloading files and for managing backup accounts
- Built the server with Java, EC2, S3, DynamoDB, ELB, SQS, and SES, and CloudFront
- Wrote API tests in Python and browser tests in Java and Selenium.
- Created backup software for MS SQL Server, MS Access and other file-based databases. Python. Inno Setup installer (Pascal). Automated tests on 12 flavors of Windows.
- Sales & support

Amazon Web Services, Software Dev. Engineer, 2007-2010 (2yr 8mo)

SimpleDB is a hosted NoSQL database service. It is the predecessor of Amazon DynamoDB. What I did:

- Owned the subsystem that gathers and reports customer usage for billing purposes. Wrote, tested, and deployed new versions to keep up with SimpleDB architecture changes. Set up the subsystem for new SimpleDB instances in Northern California, Ireland, and Singapore data centers.
- Wrote and maintained operations documentation
- Wrote functional tests for new API features, increased test coverage, added regression tests
- Wrote and maintained the network partition test that stresses the distributed architecture of SimpleDB. With this test, we made SimpleDB tolerate all kinds of network and server problems
- Developed and maintained the team's operations tools
- Wrote and maintained command-line tools, web-based tools, monitors, alarms, automated processes, documentation, and procedures
- Oncall

Education

Bachelor of Science in Computer Science, University of Illinois at Chicago, 2006
National Taiwan Normal University, Mandarin Training Center, Taipei, 2010 – 2011.
Architecting on AWS 3-day course, 2023-04.

Native English proficiency. U.S. citizen.