



Understanding CI/CD in Modern Software Development

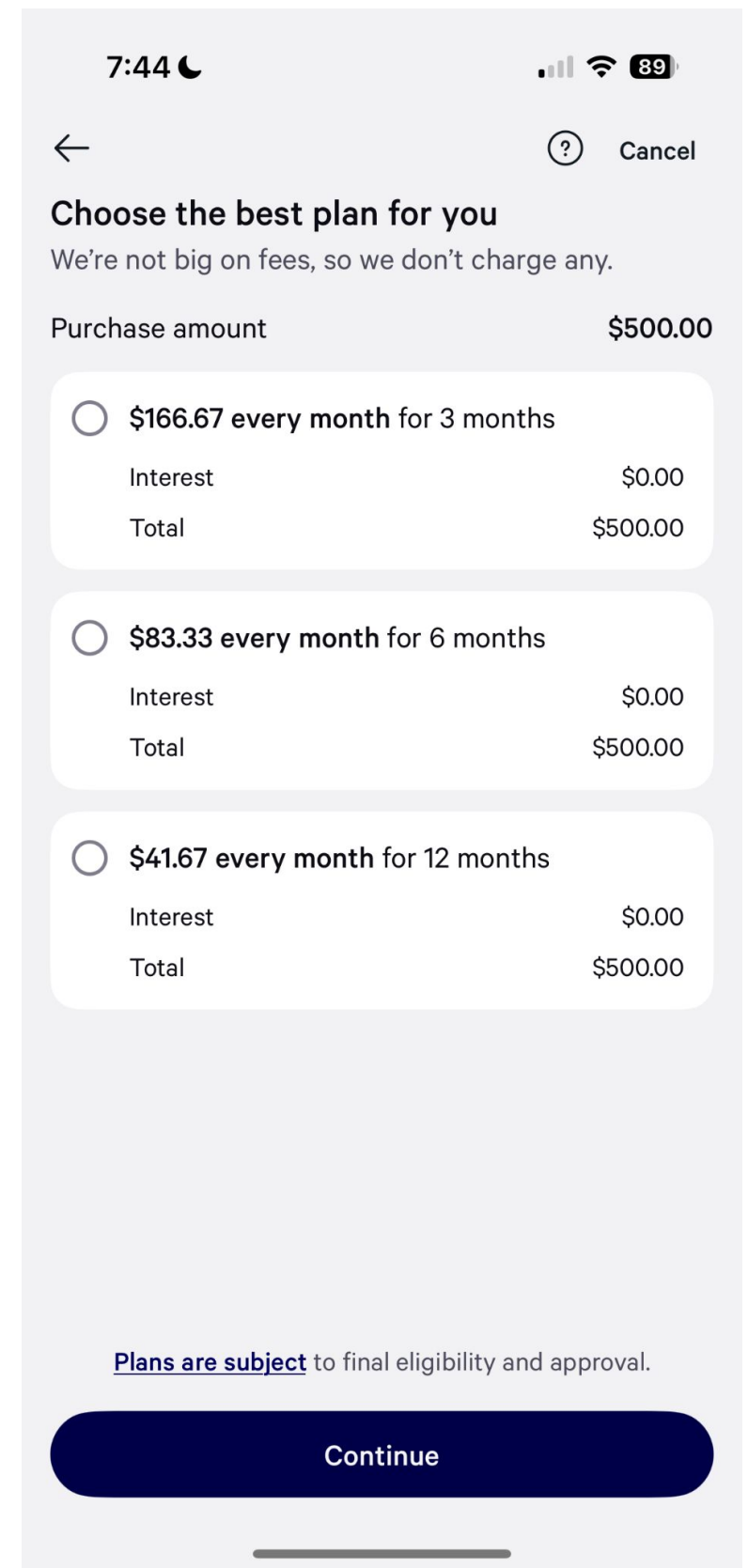
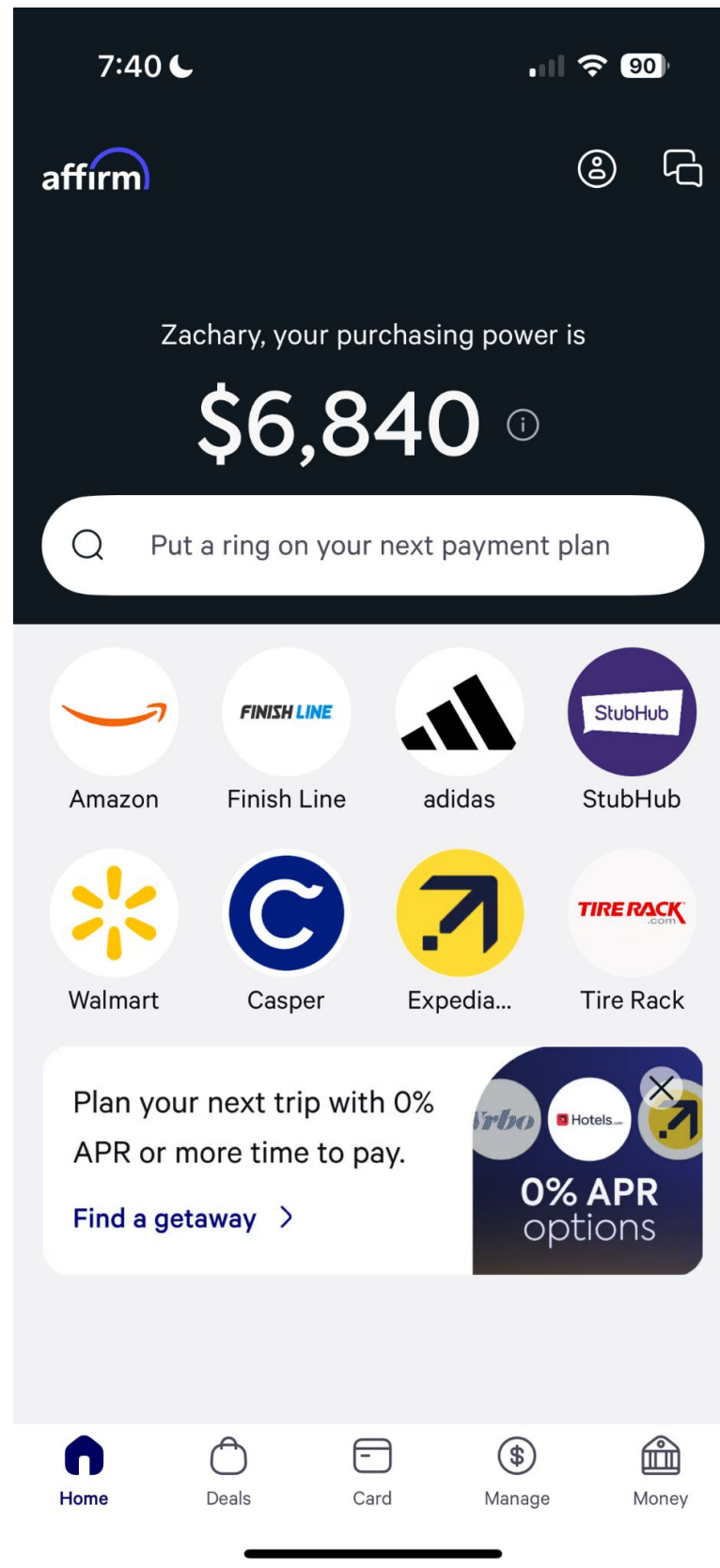
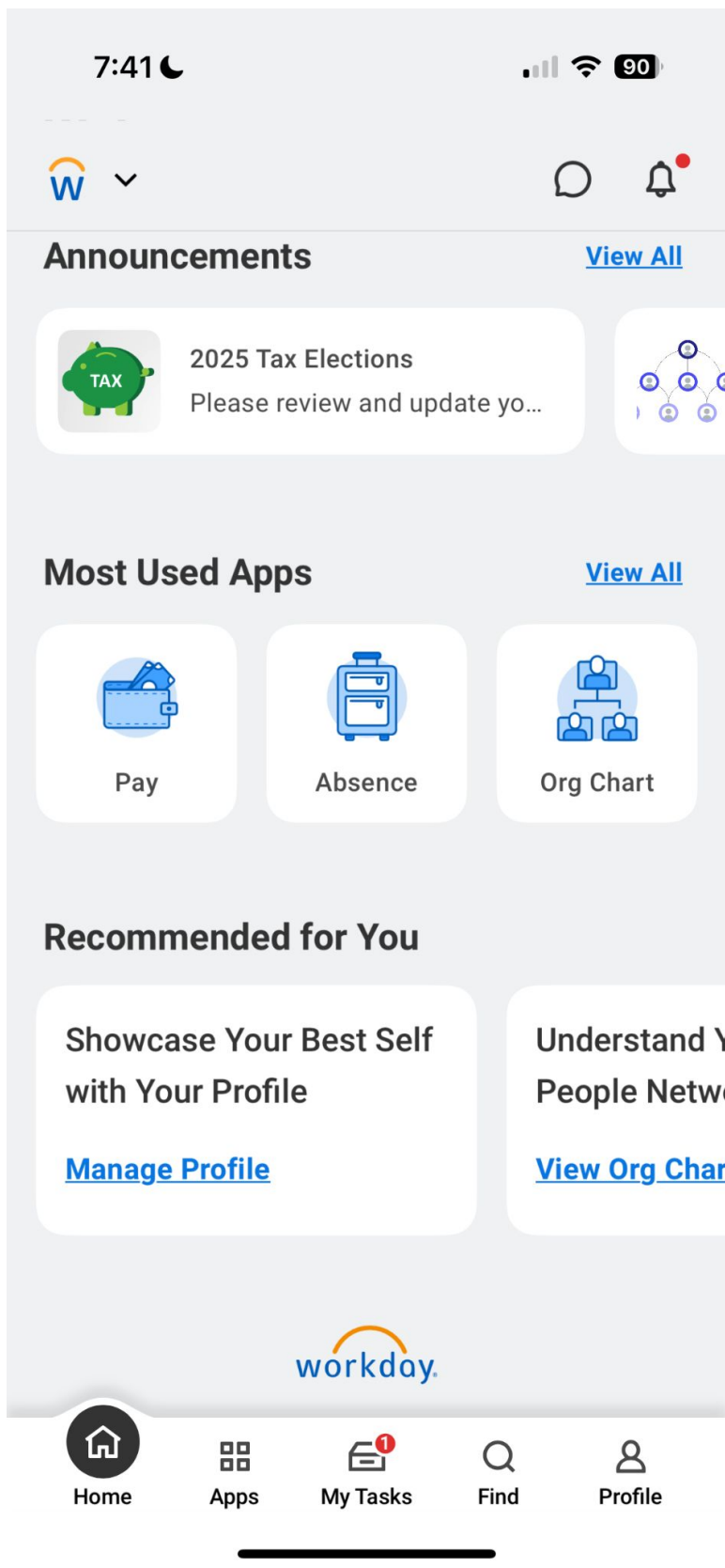
From Code Commits to Automatic Deployments

Zachary Sperske, Staff Software Engineer @ Affirm

About Me



- BS in Computer Science from UCSB
- Mobile engineer (primarily Android)
- 12 years of experience building Android, with some experience building iOS & React Native (as of recently)



Objectives & Agenda

1

Objectives

- Understand CI/CD principles and practices
- Explore benefits and challenges of adopting CI/CD
- See a demo of CI/CD in action
- Learn about popular CI/CD tools
- Discuss real-world examples and best practices

2

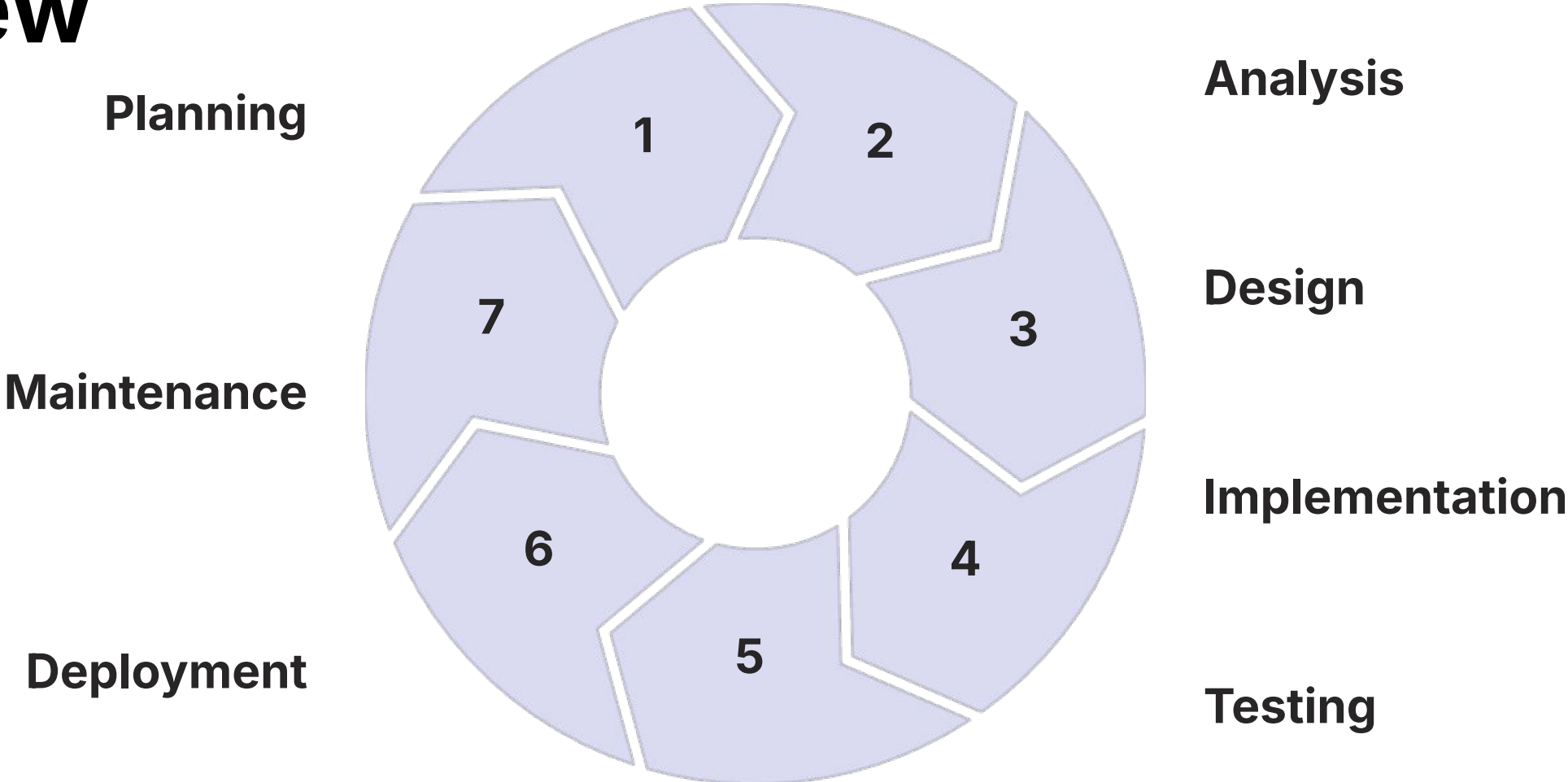
Agenda

1. What is CI/CD?
2. Why is CI/CD important?
3. Core components (Integration, Delivery, Deployment)
4. CI/CD Demo
5. Common CI/CD tools
6. Best practices & challenges
7. Q&A



Software Development Lifecycle (SDLC)

Overview

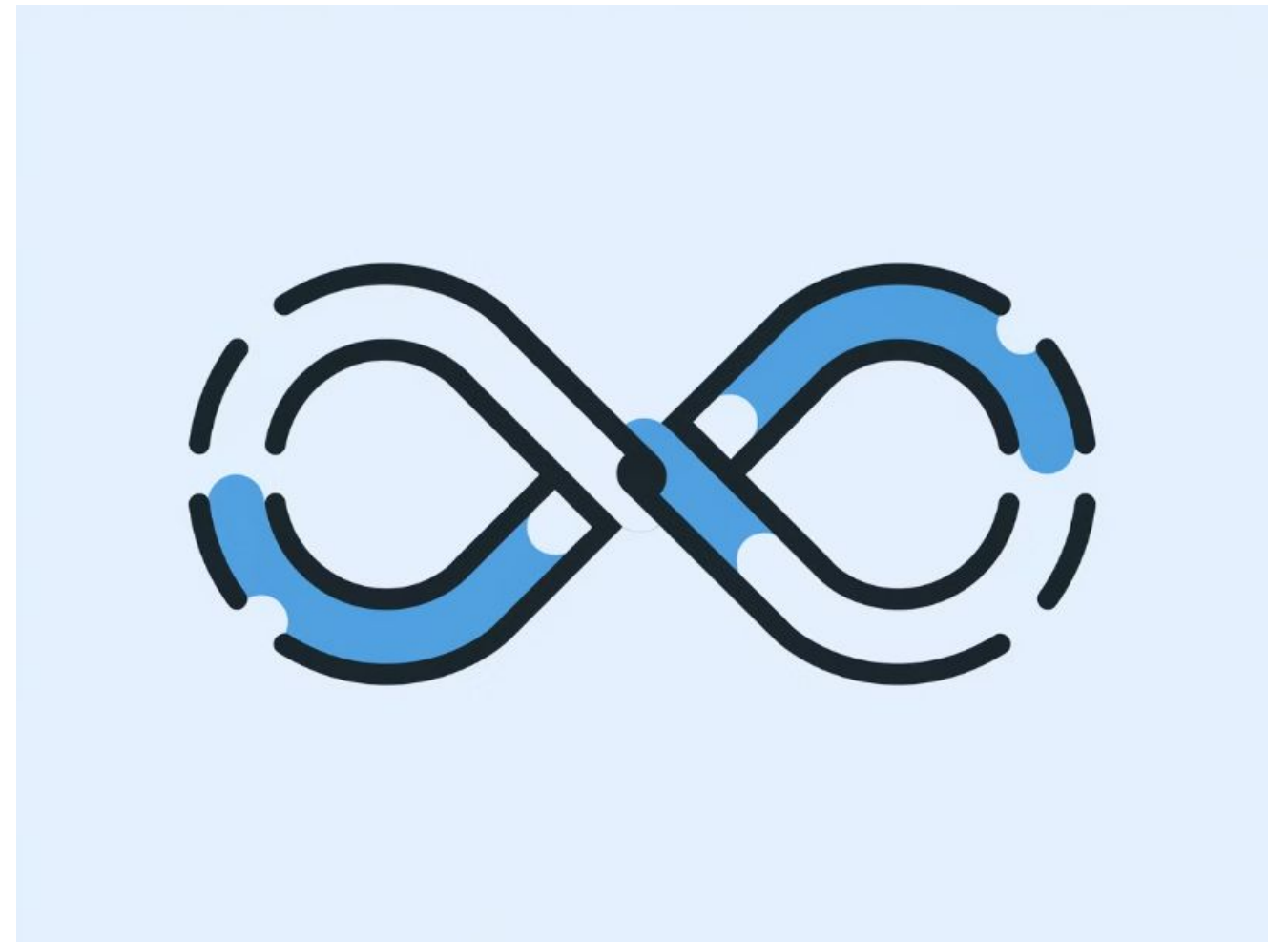


CI/CD fits primarily at the Implementation, Testing, and Deployment phases.

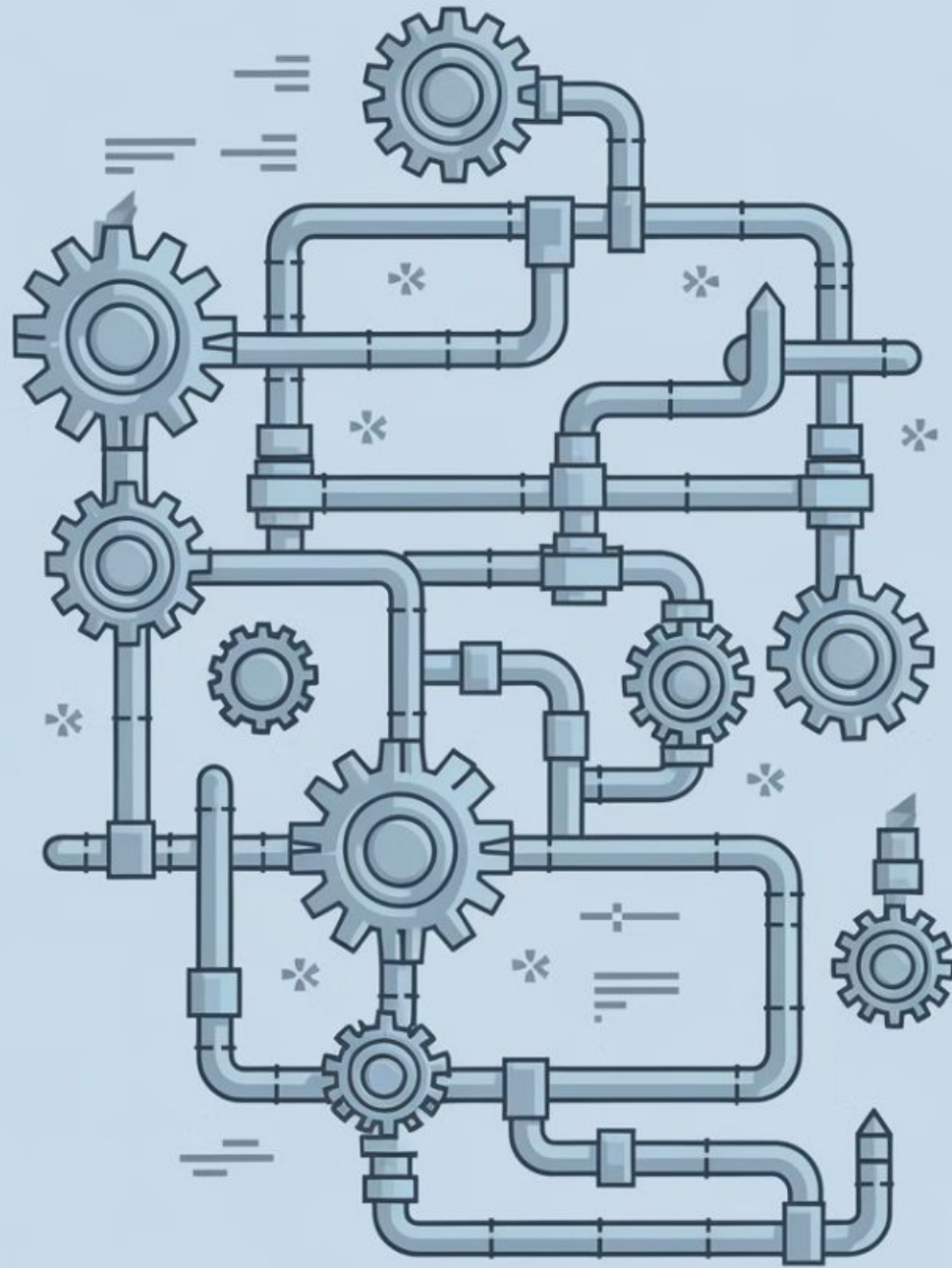
DevOps & Collaboration

DevOps = Development + Operations

Emphasizes automation, continuous feedback, and shared responsibility. Breaks down traditional silos to speed up delivery and improve quality.



Continuous Integration (CI)



Definition

Merging developer code changes into a shared repository multiple times a day, with automated builds and tests.

Core Principles

- Frequent commits (small, incremental changes)
- Automated builds triggered on every commit
- Automated tests for rapid feedback

Benefits

- Early bug detection
- Reduced integration headaches
- Improved team collaboration

CI Workflow Example

1

PR Opened

In a shared repo (e.g., Github)

2

Change Detection

GitHub Actions detects the commit

3

Automated Build

Build process begins

4

Test Suite Runs

Unit, integration tests, etc.

5

Feedback Provided

Pass/fail, code coverage, etc.

Continuous Delivery vs. Continuous Deployment

Continuous Delivery (CD)

Codebase is always in a **deployable state**

May require **manual approval** to push to production

Typical for mobile apps due to app store review process

Continuous Deployment (CD)

Fully **automated release process** to production

No manual steps once tests pass

Common for web sites & backend systems

Common CI/CD Tools



Hosted Services

- GitHub Actions
- GitLab CI/CD
- CircleCI
- Travis CI
- Buildkite



Self-Managed Tools

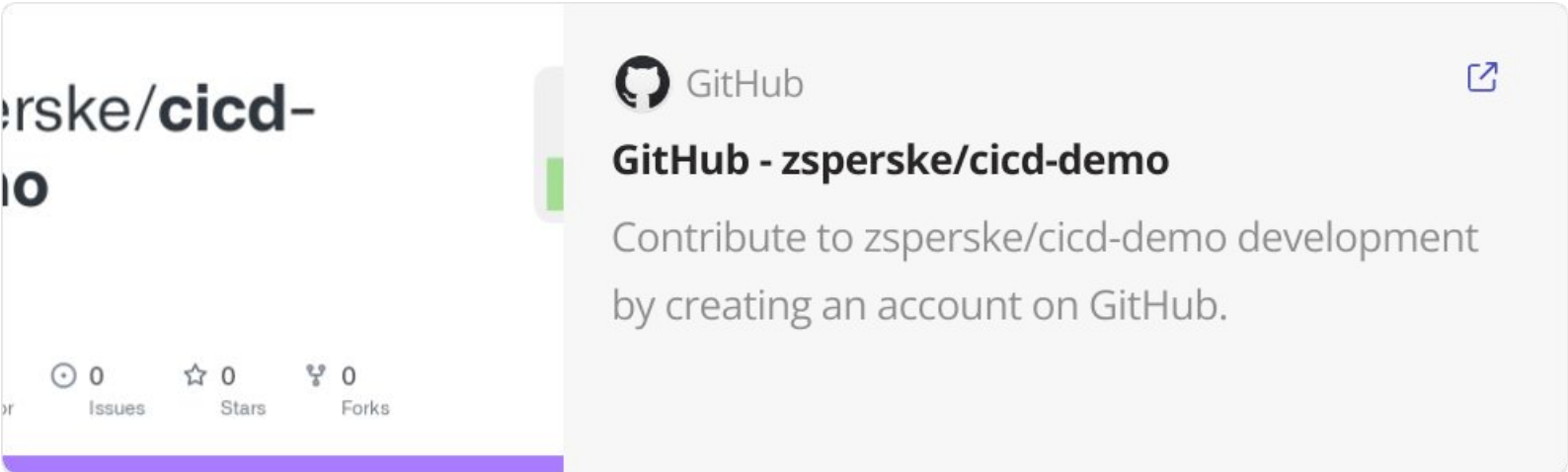
- Jenkins
- TeamCity
- Bamboo



Supporting Technologies

- Docker for containerization
- Kubernetes for container orchestration
- Infrastructure as Code (Terraform, Ansible)

Demo



The screenshot shows a GitHub repository page for 'zsperske/cicd-demo'. The repository name is partially visible as 'erske/cicd-' and 'o'. The GitHub logo and 'GitHub' text are present. Below the repository name, it says 'GitHub - zsperske/cicd-demo' and 'Contribute to zsperske/cicd-demo development by creating an account on GitHub.' At the bottom, there are icons for 'Issues', 'Stars', and 'Forks', each with a '0' next to it.



CI/CD Best Practices

1

Version Control Strategy

Feature branch vs Trunk-Based Development. Pull requests with code review.

2

Automated Testing

Unit tests, integration tests, end-to-end tests.

3

Quality Checks

Linting, static analysis, security scans.

4

Pipeline Structure

Distinct stages (Build → Test → Deploy). Automated gating between stages.

5

Monitoring & Observability

Logging, metrics, alerts. Automated rollbacks on failure.

Challenges

Cultural Resistance

Solution: Provide training & leadership support

Tooling Complexity

Solution: Start small, pick tools that integrate well

Maintaining Quality

Solution: Expand automated tests, maintain test environments

Scalability & Performance

Solution: Parallelize builds, optimize caching, flaky test reporting

Security & Compliance

Solution: Integrate security scanning & compliance checks in the pipeline

Comparing Good vs. Bad (and No) CI/CD

No CI/CD Setup

1 Manual integration & releases

Large, infrequent code merges lead to conflicts discovered late. Error-prone and time-consuming deployment steps.

2 Limited or no automated testing

Bugs often caught in production. High risk of critical downtime.

3 Long feedback loops

Delayed discovery of issues. Slow response to user needs or market changes.

4 Business Impact

High operational costs, more production outages. Negative user experience and potential revenue loss.



Poorly Implemented CI/CD

Incomplete or rarely used pipelines

Build or test stages not automatically triggered, may be skipped or done inconsistently.

Minimal test coverage

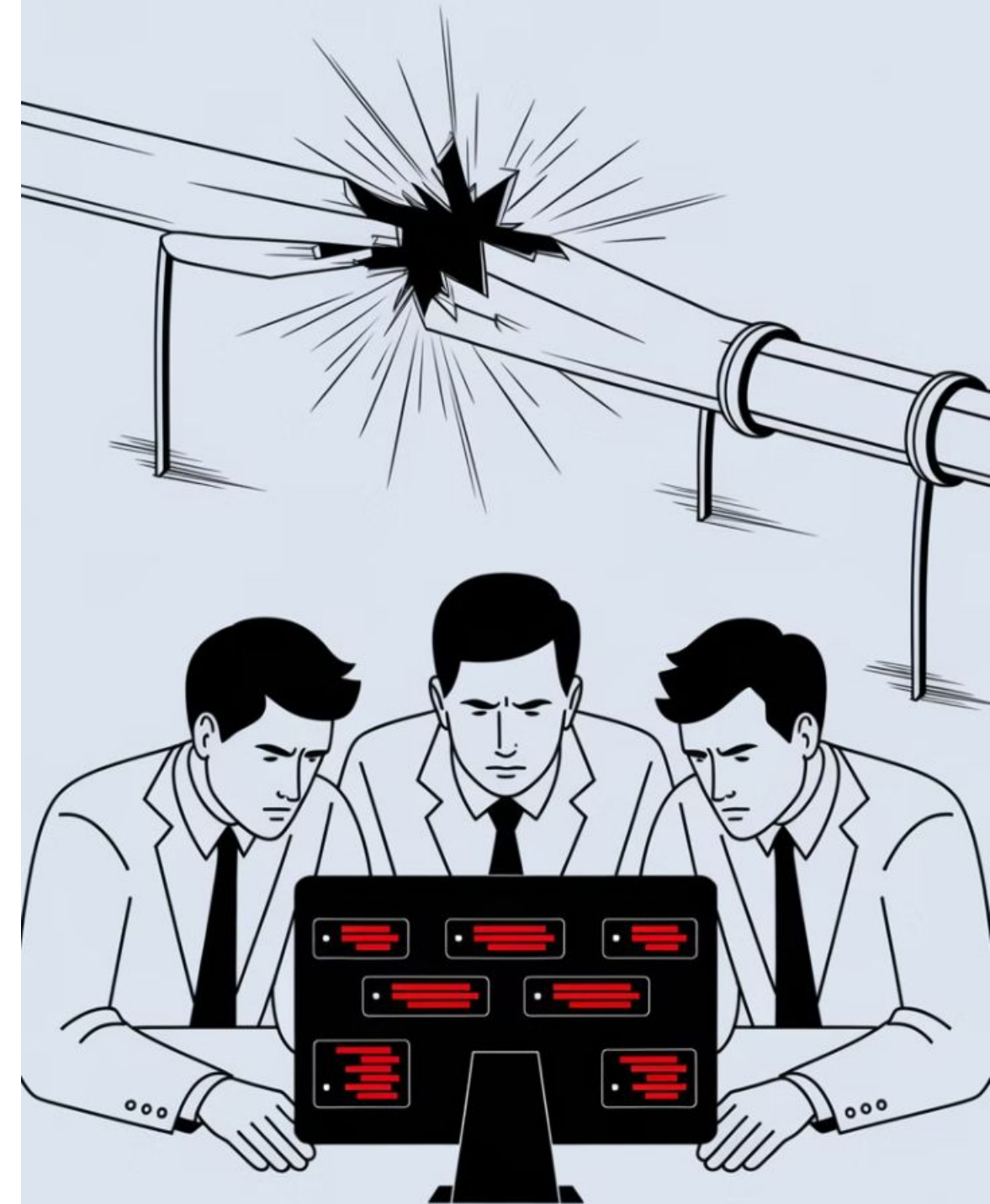
Automated tests exist but don't cover critical functionality. Production bugs still leak through. False sense of security when pipelines pass without catching issues.

Unreliable pipelines

Frequent pipeline failures without clear resolution steps. Tests that pass on one run but fail on the next. Teams lose trust and revert to manual processes.

Business Impact

Reduced benefit from automation; still encountering late-stage errors. Wastes time troubleshooting unclear pipeline failures.





Robust CI/CD Setup

1

Fully automated build & test pipeline

Every commit triggers a build and thorough suite of tests. Faster feedback; issues discovered and fixed early.

2

Frequent, small releases

Easier to deploy, roll back if necessary, and reduce release risk. Users see new features and fixes quickly.

3

High confidence in deployment

Well-defined gating stages ensure only stable code is promoted. Post-deployment monitoring and automatic rollback if critical failures occur.

4

Business Impact

Faster time-to-market, improved quality & reliability, enhanced developer productivity, strong DevOps culture.

How is it really done?



Summary & Key Takeaways

1

CI/CD Principles

Small, frequent changes + automation = faster, more reliable releases

2

Career Impact

DevOps skill sets are extremely valuable

3

Next Steps

- Try a simple CI pipeline on a personal project
- Explore open-source CI/CD tools (Jenkins, GitLab)

- How does Affirm balance the need for rapid feature development while maintaining code quality? Are there specific strategies or processes your team follows?
- What was the biggest adjustment for you when transitioning from college to your first full-time software engineering role?
- In your opinion, what was the most important thing you learned about while in industry that isn't covered at university?
- What's one piece of career advice that has had the biggest impact on your growth as an engineer?
- How do you decide when to specialize in a specific domain (e.g., mobile, backend, infra) versus remaining a generalist?
- How can a new grad best prepare for success in their first software engineering job?
- Advice on trying to get your first cs job? Advice for those who don't have internship experience, but trying to get a full-time job?
- What skills or knowledge did you wish you had developed more in college before starting your career?
- What is your favorite memory as a software engineer?

Links & Info

- [My LinkedIn](#)
- [Cursor IDE](#)
- [Github Repo](#)
- [Firebase](#)



Q&A

All questions welcome!