[updatable repos]

Duolingo's journey to a golden path

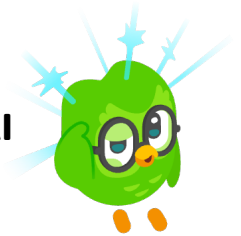
Max Blaze

LDX3 London
June 2026

what i'll cover

- The state and scale of drift
- The dream and how to sell it
- How to establish a golden path
- Repo updates from templates *

* Now with more AI



code @ duolingo

Current scale of the service infrastructure

- ~500 microservices – mostly Python and Java/Kotlin
- ~200 repositories – often more than one service per repo)
- 50/50 AWS ECS and EKS – using infrastructure as code
- Product-driven culture

[the drama]

industry treadmill

Regulatory and business needs

Vulnerabilities and exploits

Software upgrades

New best practices

New tooling

Official software end of life (EOL)

Removal of required resources

Project abandonment

Security compromise

Active attacks

drift

Development resources and practices evolved independently within individual teams

- Hundreds of versions of the same shell scripts
- Thousands of combinations of libraries
- Hundreds of Docker base images
- Differing build tools

ad-hoc service updates

Mass updates coming directly from individual teams and areas

- Individual update campaigns for each update
- Generally happen on irregular schedules
- Changes often need to be tuned for each service
- Difficult to test integration between changes

Pulldozer: CLI tool for batch editing multiple repos

carrots not sticks

Top-down mandates can conflict with culture

- A change committee vetted all engineering-wide changes on a quarterly basis
- Teams had deadlines to apply approved changes or deployments would be disabled
- Product teams couldn't get features out quickly enough (or *were blocked*)
- The quarterly cadence was too long for Infrastructure teams


[dune]

what is a golden path?

Creating paved roads and guardrails

- Clear, opinionated standards that fit the majority of use cases
- Term comes from the book Children of Dune
- Netflix has paved roads, Spotify has a golden path with golden technologies and fleet updates
- Consistency of application matters more than the standards themselves

Software Sprawl, The Golden Path, and Scaling Teams With Agency

primary goals of the golden path

Bring order to the chaos and fight entropy

- Establish **standard anchor points** on which to build services
- Reduce **cognitive load** of managing infrastructure
- Improve **shared collaboration** between teams
- Quickly **apply updates** when necessary

[No Next Next: Fighting Entropy in Your Microservices Architecture]

making engineering efforts easier

- Migrate from AWS ECS to Kubernetes to improve the service deployment and observability experiences
- Fully-integrate pipelines, tools, and IDEas to make onboarding and development easier
- Implement OpenAPI to unify how we define and maintain communication between services
- Reduce the number of vulnerable components to meet compliance requirements

[coffee's for closers]

selling the dream

Facing cultural and political pushback

- You'll break my stuff! – We'll manage the toil for you
- It's going to slow us down! – Development will be faster overall
- What's the business case? – Protect us from supply chain attacks
- Beware of hero culture – Reward preemptive fixes
- Standardization can expose deep technical and organization problems, especially after team transitions

[the matrix]

goldilocks zone

Guiding principles for choosing technologies when the industry is moving so quickly

- Look a year ahead of potential updates by monitoring and aligning internal and external maintenance timelines
- Chose long-term support (LTS) releases whenever possible
- Stay off the bleeding edge to avoid early release bugs
- Chose automatable solutions over manual

endoflife.date

establishing standards

Standard	Upstream EOL *	Decision rule	Owner
Python 3.13	31 October 2026	Latest release - 1	Architecture
Kotlin 2.3	~June 2026	Latest release	
Amazon Corretto 21 JDK	01 October 2029	Latest LTS with sufficient library and tooling support	
Debian Bookworm	30 June 2028	Latest LTS release, but wait 6 months before upgrading	
Valley 8.1	31 March 2030	Latest release - 1	Cloud Operations
PostgreSQL 17.7	28 Feb 2030	Latest AWS LTS release	

* Pay close attention to “standard” vs. “extended” (paid) support and feature vs. security maintenance

endoflife.date

[snakes on a plane]

case study

Python 3.9 to 3.10 upgrades

- Updated **standard base image** from Alpine to Debian
- Unpinned and updated all packages to latest version **except protobuf** (only a *very* specific minor version worked)
- Worked through many **mypy upgrade issues**
- Upgrading all resources **fixed** a deploy-time **memory leak**
- **Debian-based image** allowed **numpy** to be integrated more quickly compared to Alpine

case study

Java and Spring Boot upgrades

- Updated standard base image from Java 11/17 to 21
- Upgraded Gradle to the latest 8.x version
- Upgraded to the latest internal base library
- Upgraded from Spring Boot 2.x to 3.x

[Automating Golden Path upgrades at scale: a journey from manual upgrades to an AI-powered workflow]

case study

Java and Spring Boot upgrades

- Open Rewrite for the initial static passes with AI in a debug loop
- Using both together sped up the process by weeks
- Breaking upgrades into smaller steps saves a lot of time
- **Don’t over-rely on AI – Learn to love determinism**

[Automating Golden Path upgrades at scale: a journey from manual upgrades to an AI-powered workflow]

repo template tooling

Tools to establish and update service baselines

- Cruft – uses the popular Cookiecutter templating system and patch-based merges for repo updates
- Copier – uses Jinja2 and 3-way merges to manage repository
- Nunjucks – templating language for JavaScript that is native to Backstage, but no built-in repo update mechanism

effective templates

Making templates easy to use and maintain

- Have proper linting, build, and integration tests to make sure that every change actually works in a new service
- Layer templates and apply DRY principles
- Support external services and infra resources
- Store template inputs in a single file in each service repo

too much or too little abstraction?

Duolingo

Full control over entire runtime (base image, libraries, etc.), language versions, and upgrades

Heroku-like environment prepopulated with all necessary dependencies

updates from templates

Advantages of using repo templates as the source of truth for service updates

- Infra teams: commit, coordinate, and integrate updates in a single place
- Product teams: choose when to pull updates
- Best practices can be aggregated and seen in context
- Shared functionality can be abstracted away and eventually removed from repos over time, allowing teams to focus on business logic

updates from templates

Viewing and applying a diff with Cruft

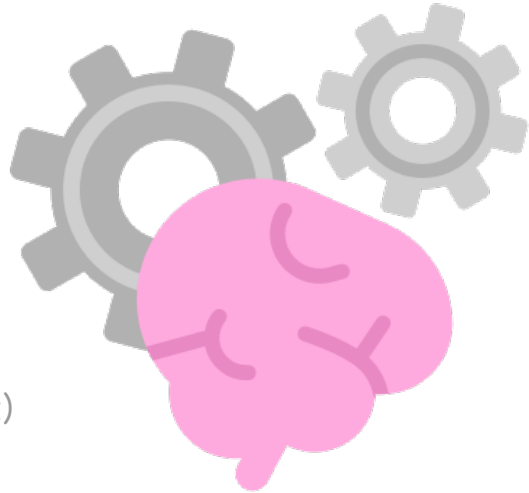
```
$ cruft check
FAILURE: Project's cruft is out of date! Run `cruft update` to
clean this mess up.
$ cruft update
diff --git a/.pre-commit-config.yaml b/.pre-commit-config.yaml
index dcd8d27..3ed70da 100644
--- a/.pre-commit-config.yaml
+++ b/.pre-commit-config.yaml
@@ -1,7 +1,7 @@
 # https://duo.fyi/pre-commit
 repos:
   - repo: https://github.com/duolingo/pre-commit-hooks.git
-          rev: 1.7.3
+          rev: 1.8.0
           hooks:
           - id: duolingo
   - repo: local
Apply diff and update? (y, n, s, v) [y]: y
Applied patch to '.pre-commit-config.yaml' cleanly.
```

Duolingo pre-commit hooks

[back to the future]

agentic workflow

Mass updates using Claude



- Get familiar with the codebase *without* an LLM
- Download all repos that are to be changed
- Keep the human in the workflow (don't attempt a one-shot)
- Create a Claude skill from the *resulting* session context
- Goal: prepare the repo for fully-autonomous changes in the future

Confidently Automating Changes Across a Diverse Fleet

continuing work

Refining repos to be more updatable

- Rebuild repos from templates, drop in code, and then modify
- Remove all boilerplate, leaving just business logic in the repo
- Automatically remove services and projects with little activity
- The more standardized a repo gets, less AI is needed to update it

recap

- **Start early** – standards will start paying dividends as soon as they're in place
- "Carrots" not "sticks" – make the Golden Path the easiest option, don't force it
- Use what you already have – no need to come up with a brand new set of standards
- AI doesn't solve everything – static tooling might work better, especially as repos become more standardized
- Be persistent – mass standardization is *hard*, don't give up!

gracias!