Engineer

Codebasis

als Einheit gepflegter, geprüfter und gelieferter Quellcode

Was es ist

Eine Codebasis, auf Englisch codebase oder auch code base, ist der gesamte Quellcode, den ein Team als eine Einheit bearbeitet, prüft und ausliefert. Sprechen Entwickler von „der Codebasis“, meinen sie den Ort für jede Anwendungsänderung: den Code selbst, seine Tests, die Skripte zum Bauen und Prüfen, Datenbankänderungen und die Dokumente, die das Zusammenspiel erklären.

Die Twelve-Factor App, eine Methodik für Webdienste, liefert die nützlichste Arbeitsdefinition, weil sie die Grenze prüfbar macht. In einer zentralen Versionsverwaltung wie Subversion ist eine Codebasis ein Repository. In einem verteilten System wie Git ist sie eine beliebige Menge von Repositories mit gemeinsamem Wurzelcommit: dem ersten Commit der Historie, der keinen Elterncommit hat. Zwei Repositories, die aus demselben ersten Commit entstanden sind, bleiben eine Codebasis, wie weit sie auch auseinandergedriftet sind. Zwei getrennt gestartete Repositories sind zwei Codebasen, selbst mit nahezu identischen Dateien.

Dasselbe Dokument verbindet diese Definition mit einer zweiten Regel: eine Codebasis, viele Bereitstellungen. Jede Bereitstellung ist eine laufende Anwendungsinstanz. Meist gibt es mehrere zugleich: Produktion, Staging und auch die lokale Kopie jedes Entwicklers. Alle verwenden dieselbe Codebasis, oft mit verschiedenen Commits. Ein Entwickler hat Arbeit, die Staging noch nicht gesehen hat, und Staging Arbeit, die Produktion noch nicht gesehen hat.

Daraus ergibt sich der Inhalt. Der Quellcode und alles, was daraus ein funktionsfähiges Programm macht, gehören hinein. Dazu zählt ein Manifest mit jeder Abhängigkeit und ihrer Version. Die Abhängigkeiten selbst bleiben meist

draußen und werden von einem Paketmanager geholt. Auch von Bereitstellung zu Bereitstellung wechselnde Werte bleiben draußen, vor allem Zugangsdaten: Ein einmal committetes Passwort reist danach mit jeder Kopie der Historie.

Eine Codebasis kann mehrere Programme enthalten. Ein Monorepo hält Frontend, API und Datenbankcode nebeneinander in einem Repository. Ein Commit kann deshalb alle drei ändern. Nach der Wurzelcommit-Regel bleibt das eine Codebasis. Die Gegenentscheidung verteilt eine Anwendung auf Repositories mit getrennten Historien. Jedes ist dann eine eigene Codebasis. Twelve-Factor benennt die Folge klar: Bei mehreren Codebasen handelt es sich um ein verteiltes System, und jede Komponente ist eine eigenständige App.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Codebasis?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Struktur eines Systems wird zuerst auf dem Papier durchdacht, damit die Teile zusammenpassen und die Kosten feststehen, bevor die erste Zeile Code entsteht.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

Warum es wichtig ist

Die Grenze entscheidet, was sich gemeinsam ändern lässt. Innerhalb einer Codebasis können Änderungen an einer API und an der aufrufenden Ansicht als Einheit geschrieben, geprüft und zusammengeführt werden. Über zwei Codebasen hinweg werden daraus zwei Pull Requests, zwei Veröffentlichungen und eine Zeitspanne, in der eine Seite der anderen voraus ist. Beides funktioniert. Der Preis folgt aus der gezogenen Grenze.

Die Codebasis ist auch die Einheit, für die Standards gelten. Testsammlung, Linter, Formatter, Commitnachrichtenregel und Task Runner betrachten „die Codebasis“ als ihren Geltungsbereich. Was außerhalb liegt, prüft keines dieser Werkzeuge.

Frühe Entscheidungen verfestigen sich in einer Codebasis schneller als anderswo. Jede spätere Zeile setzt die Entscheidung voraus. Eine Schichtenregel oder Qualitätsschwelle aus den ersten Wochen formt deshalb alles Folgende. Sie

später umzukehren heißt, den darum gewachsenen Code neu zu schreiben.

Für Menschen ist eine Codebasis das, was ein neues Teammitglied erhält. Enthält sie alles zum Bauen, Testen und Ausführen der Anwendung, reichen ein Clone und ein Befehl für den Einstieg. Liegt das Wissen teils im Code und teils im Gedächtnis anderer, geht die erste Woche dafür drauf, herauszufinden, welcher Teil wo liegt.

Wie es in der Praxis aussieht

Go, TypeScript, SQL, Python und Shell teilen das Repository einer Plattform. Dort ist die Codebasis ein mehrsprachiges Monorepo, und jede Sprache bringt eigene Wege zum Bauen, Testen und Linten mit. Die Antwort war ein Einstiegspunkt: eine go-task-Schicht, die auf 578 benannte Ziele anwuchs, 50 in der Wurzeldatei und 528 in darunterliegenden Dateien mit Namensräumen. Sie deckten native Entwicklung, Docker- und HTTPS-Entwicklungsmodi, Codequalität, Datenbankverwaltung und Bereitstellung ab. Mit `task --list` kann jeder die ganze Werkzeugkette sehen. Die Zahl ist auch eine Warnung: 578 Ziele kann niemand mehr im Kopf behalten. Benennung und Namensräume halten sie nutzbar.

Ein anderes Projekt begann am 25. November 2025 mit einem leeren Repository. Drei Entscheidungen aus seinen ersten Wochen galten noch neun Monate und mehrere Tausend Commits später: ein geschichteter Stack, Datenzugriff hinter gespeicherten Funktionen und eine Vorgabe von null Warnungen, gesetzt, bevor viel Code daran zu messen war. Die Begründung des letzten Punkts trifft den Kern dieses Artikels: Ein bei Commit 5.000 eingeführter Standard ist ein Aufräumprojekt. Derselbe Standard bei Commit 50 ist einfach die Arbeitsweise des Repositorys.

Die Grenzen innerhalb einer Codebasis können der Entwurf selbst sein. Im Architekturbeitrag der Gründerin unter dem Dach von NextMariner zu einer maritimen Plattform mit drei Schichten übernimmt das Next.js-Frontend ausschließlich die Darstellung. Eine bewusst schlanke Go-API mit Huma auf Fiber routet, validiert und orchestriert. Die Geschäftslogik liegt in PostgreSQL-Funktionen. Ändert sich eine Regel, ändert sie sich in einer Schicht, in SQL. Die Grenzen wurden aufgeschrieben und in Reviews durchgesetzt. Eine Konvention, deren Einhaltung niemand prüft, hört auf, eine zu sein.

Die eigene Codebasis dieser Website zieht eine andere Grenze. Ihre Projektseiten werden in einem anderen Repository geschrieben, per Skript in dieses exportiert und committet. Die Website lässt sich deshalb eigenständig bauen. Innerhalb dieser Codebasis sind diese Seiten generierte Dateien. Die Arbeitsanweisungen verbieten ihre manuelle Bearbeitung.

Wo es schiefgeht

Kopierter Code entwickelt sich auseinander

Zwei Anwendungen brauchen dieselbe Hilfsfunktion. Kopieren ist der schnellste Weg. Ein Jahr später gibt es zwei Versionen mit unterschiedlichen Korrekturen. Ein in einer behobener Fehler bleibt in der anderen aktiv. Twelve-Factor nennt gemeinsam verwendeten Code in mehreren Apps einen Verstoß und benennt die Lösung: Den gemeinsamen Teil in eine Bibliothek verschieben, über den Abhängigkeitsmanager einbinden und jede Codebasis die verwendete Version angeben lassen.

Ein Geheimnis landet in der Historie

Eine Verbindungszeichenfolge oder ein API-Schlüssel wird in einer Konfigurationsdatei committet. Von da an steht der Wert in der von jedem Clone mitgeführten Historie, selbst wenn ein späterer Commit die Zeile löscht. Vor jedem Commit lohnt sich der Twelve-Factor-Test: Könnte diese Codebasis heute als Open Source veröffentlicht werden, ohne dabei Zugangsdaten offenzulegen? Alles, was diesen Test nicht besteht, gehört in die Umgebung der jeweiligen Bereitstellung.

Eine generierte Datei wird von Hand geändert

Eine von einem Skript geschriebene Datei sieht wie andere Quelldateien aus. Jemand korrigiert deshalb direkt einen Tippfehler darin. Der nächste Skriptlauf schreibt den alten Text zurück, und die Korrektur verschwindet wortlos. Markieren Sie generierte Pfade deutlich, lassen Sie nur den Generator hineinschreiben und nehmen Sie jede Korrektur an der Eingabe des Generators vor.

Innere Grenzen werden durchlässig

Unter Zeitdruck landet eine Abfrage direkt in einem Anfragehandler, weil das der kürzeste Weg ist. Die anfängliche Schichtung beschreibt den Code nicht mehr. Jede Abkürzung ist klein. Gemeinsam machen sie aus einer Änderung mit zwei

Zeilen eine, die jede Schicht berührt. Eine schriftliche Regel hilft, ein Review mehr. Am meisten hilft eine Prüfung, die den Build fehlschlagen lässt, sobald eine Schicht in eine andere greift.

Ein Standard gilt für die Hälfte des Codes

Ein für das Frontend eingerichteter Linter lässt Shellskripte und SQL unangetastet. Diese Dateien sammeln genau die Probleme an, die der Linter verhindern sollte. Ein für die ganze Codebasis geltender Qualitätsmaßstab muss jede enthaltene Sprache benennen. Das ist der versteckte Preis eines mehrsprachigen Repositorys.

Ein Fork startet ein zweites Produkt

Ein Team kopiert ein Repository für ein verwandtes Produkt. Beide Kopien behalten denselben Wurzelcommit. Nach der Twelve-Factor-Regel bleiben sie deshalb eine Codebasis, nun für zwei Apps. Das beschriebene Auseinanderdriften beginnt auf Ebene eines ganzen Repositorys. Der Vergleich der Wurzelcommits zeigt, welcher Fall vorliegt:

```
git -C app-a rev-list --max-parents=0 HEAD
git -C app-b rev-list --max-parents=0 HEAD
```

Identische Ausgabe bedeutet gemeinsame Wurzel und gemeinsame Codebasis. Unterschiedliche Ausgabe bedeutet zwei getrennte Historien.

Weiterführende Lektüre

- I. Codebasis — die Regel einer Codebasis mit vielen Bereitstellungen und der Test auf den gemeinsamen Wurzelcommit dahinter.
- II. Abhängigkeiten — jede Abhängigkeit in einem Manifest angeben und isolieren, damit die Codebasis ihre eingebundenen Bestandteile genau benennt.
- III. Konfiguration — der Open-Source-Test dafür, was außerhalb einer Codebasis bleiben muss.
- gitsubmodules — wie Git ein Repository in ein anderes einbettet und beide ihre eigene Historie behalten.
- git-rev-list — die Referenz zur Commitauflistung einschließlich der Wurzelcommits einer Historie.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Codebasis?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/codebase/