Engineer

Bibliothek

für Aufrufe durch andere Programme gebündelter Code

Was es ist

Eine Bibliothek ist eine Sammlung fertiger Routinen, die ein Programm übernimmt, statt sie selbst zu schreiben: Jemand hat ein Problem einmal gelöst, etwa Datumsangaben zu parsen, Dateien zu komprimieren oder einen QR-Code zu zeichnen, und die Lösung mit einer Reihe benannter Einstiegspunkte veröffentlicht, die jedes Programm nutzen kann. Der englische Plural lautet libraries, der deutsche Bibliotheken. C-Bibliotheken unter Linux tragen häufig das Präfix lib-, Windows nennt die gemeinsam genutzte Art DLL, und Rust-Programmierer sagen Crate. Das Rust-Buch merkt an, dass dieser Begriff austauschbar mit Bibliothek verwendet wird. Die meisten Sprachen liefern außerdem eine Standardbibliothek mit, also die Module, die in jeder Installation ohne zusätzliche Ergänzungen verfügbar sind.

Eine Bibliothek ist eine Form wiederverwendbaren Codes und lässt sich am einfachsten im Vergleich mit benachbarten Begriffen definieren. Eine Anwendung läuft selbstständig und hat einen Nutzer; eine Bibliothek läuft weder selbstständig noch hat sie einen eigenen Nutzer. Sie läuft nur innerhalb eines Programms, das sie aufruft. Näher liegt der Vergleich mit einem Framework. Bei einer Bibliothek hat Ihr Code die Kontrolle und ruft sie auf, wenn er etwas braucht. Ein Framework hat selbst die Kontrolle und ruft Ihren Code an den von ihm definierten Stellen auf. Ein Paket schließlich ist die Schachtel, in der eine Bibliothek ausgeliefert wird: mit einem Namen, einer Version und einer Liste dessen, was sie benötigt. Die Bibliothek ist der Code darin.

Nutzbar wird eine Bibliothek durch ihre Schnittstelle: die Funktionen, Typen und Konstanten, deren Erhalt sie zusichert. Semantic Versioning baut sein gesamtes Schema auf dieser Zusage auf. Ein Projekt, das ihm folgt, muss eine öffentliche API deklarieren, und seine Hauptversionsnummer steigt bei jeder inkompatiblen Änderung dieser API.

Das Program Library HOWTO ordnet Bibliotheken unter Linux danach, wann sie mit dem Programm verbunden werden. Eine statische Bibliothek wird beim Erstellen in die ausführbare Datei kopiert. Eine gemeinsam genutzte Bibliothek wird beim Programmstart geladen und von jedem Programm verwendet, das sie braucht; sie kann deshalb aktualisiert werden, ohne diese Programme neu zu bauen. Eine dynamisch geladene Bibliothek wird während der Programmausführung geöffnet. So arbeiten gewöhnlich Plugin-Systeme.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Bibliothek?“ 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

Jede Bibliothek, die ein Projekt übernimmt, bedeutet eine Entscheidung mit drei möglichen Antworten: eine vorhandene verwenden, den Code selbst entwickeln oder ein kleines, eng umrissenes Stück in das Repository kopieren und fortan selbst dafür verantwortlich sein. Letzteres heißt Vendoring.

Eine Bibliothek zu übernehmen ist meist richtig. Ihre Autoren kennen die Randfälle, und andere Nutzer haben bereits Fehler gemeldet. Die Kosten kommen später und werden leicht übersehen: Die eigenen Abhängigkeiten der Bibliothek kommen mit, ihre Sicherheitsmeldungen werden zu Ihren, ihr Veröffentlichungstakt bestimmt den Rhythmus Ihrer Aktualisierungen, und ihre Lizenz gelangt in Ihr Produkt. Eine Bibliothek, die ihr Betreuer aufgegeben hat, bleibt in Ihrem Build, bis jemand sie ersetzt.

Eigener Code verschafft Kontrolle und kostet alles, was die Bibliothek eingespart hätte: Randfälle, Tests und Wartung, auf unbestimmte Zeit. Das ist die richtige Entscheidung, wenn das Problem eng umrissen und zentral ist, wenn verfügbare Bibliotheken weit größer als der Bedarf sind oder wenn ein Projekt einen Grund hat, seine Zahl an Abhängigkeiten niedrig zu halten.

Auch die Art der Einbindung ist wichtig. Ein mit einer statischen Bibliothek gebautes Programm enthält eine feste Kopie. Eine Sicherheitskorrektur der Bibliothek erreicht die Nutzer deshalb erst, wenn das Programm neu gebaut und erneut ausgeliefert wird. Eine gemeinsam genutzte Bibliothek lässt sich einmal für das ganze System korrigieren; dafür vertraut das Programm der Version, die das System bereitstellt. Die Lizenzierung macht diesen Unterschied nicht: Laut FAQ zu den GNU-Lizenzen bildet ein GPL-lizenziertes Werk, das statisch oder dynamisch mit anderen Modulen verbunden wird, ein kombiniertes Werk. Die Bedingungen der GPL gelten dann für die gesamte Kombination.

Wer Software in Auftrag gibt, sollte klären, von welchen Bibliotheken das Ergebnis abhängt, unter welchen Lizenzen sie stehen und wem die Bibliotheken gehören, die der Lieferant mitbringt.

Wie es in der Praxis aussieht

Ein Dokumentgenerator brauchte einen QR-Code, der auf die Onlineversion jedes Dokuments verwies. Jede der vorhandenen QR-Bibliotheken hätte diese Aufgabe erfüllt. Das Projekt hatte sich jedoch bewusst auf zwei Laufzeitabhängigkeiten beschränkt, eine Template-Engine und einen PDF-Renderer. Diese Grenze sollte dafür sorgen, dass sich der Generator noch lange nach der Veröffentlichung problemlos installieren lässt. Die Antwort war, einen QR-Encoder mit 522 Zeilen ins Repository zu übernehmen: Kodierung im Byte-Modus, Reed-Solomon-Fehlerkorrektur über GF(256), Positions-, Takt- und Ausrichtungsmuster, Format- und Versionsinformationen sowie alle acht Maskenmuster, bewertet anhand der vier Strafregeln. Ein separat geschriebener Decoder liest bei jedem Durchlauf jeden erzeugten Code zurück und prüft, ob er die Eingabe zurückliefert. Die Seite grenzt ihre Aussage sorgfältig ein: Der Encoder deckt nur die Modi und Versionen ab, die diese Dokumente verwenden. Der Grund dafür ist das Budget für Abhängigkeiten und nicht der Glaube, besser als eine ausgereifte Bibliothek zu sein.

Ein nativer macOS-Client ruft eine Kernbibliothek über deren C-Schnittstelle auf. Anfangs wurde diese Schnittstelle durch einen handgeschriebenen Header repräsentiert. Aufrufe von drei Funktionen, die inzwischen aus der Kernbibliothek entfernt worden waren, ließen sich kompilieren und linken, stürzten aber zur Laufzeit ab. Compiler und Linker hatten nur den handgeschriebenen Stellvertreter gesehen und nie die echten Deklarationen der Kernbibliothek. Die Lösung war ein Parser, der jede Aufrufstelle prüft, und zwar gegen den echten Header mit 7.308 Zeilen und seinen 268 Deklarationen. Er lässt den Build bei jedem Aufruf und jeder Enum-Konstante scheitern, die

der Header nicht deklariert. Er prüft Deklarationen: Eine Funktion, die bei gleichbleibender Signatur ihre Bedeutung ändert, besteht diese Prüfung also weiterhin.

Die Bedingungen jedes Auftrags legen fest, wem was gehört. Für einen Kunden geschriebener Code geht mit vollständiger Bezahlung in dessen Eigentum über. Bibliotheken und allgemein verwendbare Werkzeuge, die das Unternehmen früher oder für sich selbst geschrieben hat, sind die Ausnahme: Das Eigentum bleibt beim Unternehmen. Der Kunde erhält für ihre Nutzung innerhalb des gelieferten Werks eine Lizenz, die nie ausläuft und nicht widerrufen werden kann. Quelloffener Code Dritter bleibt unter der von seinen Autoren gewählten Lizenz. Der Kunde erfährt, welche dieser Bestandteile in das Ergebnis eingeflossen sind.

Wo es schiefgeht

Eine gewöhnliche Aktualisierung lässt den Build scheitern

Eine Abhängigkeit wechselt von 2.3 auf 2.4, und Code, der gestern kompilierte, scheitert oder verhält sich anders. Entweder hat die Bibliothek ihre öffentliche Schnittstelle ohne Erhöhung der Hauptversion geändert, oder das Projekt hat sich auf etwas verlassen, das die Bibliothek nie zugesichert hatte, etwa eine interne Funktion oder ein undokumentiertes Verhalten. Fixieren Sie genaue Versionen in einer Lockdatei, aktualisieren Sie bewusst mit geöffnetem Änderungsprotokoll und rufen Sie nur auf, was die Bibliothek als öffentlich dokumentiert.

Das Programm vertraut einer Beschreibung der Bibliothek

Ein fehlerfrei gebautes Programm stürzt ab, sobald eine Funktion zum ersten Mal verwendet wird. Der Code wurde gegen handgeschriebene Header, Typ-Stubs oder Bindings kompiliert, die sich von der tatsächlich eingebundenen Bibliothek entfernt hatten. Erzeugen Sie Bindings aus der veröffentlichten Schnittstelle der Bibliothek selbst oder prüfen Sie jeden Aufruf im Rahmen des Builds dagegen. Eine Abweichung lässt dann den Build scheitern, statt erst bei einem Nutzer aufzufallen.

Eine kleine Funktion zieht einen großen Baum nach sich

Benötigt wurde eine einzige Funktion. Die Bibliothek, die sie bereitstellt, brachte jedoch Dutzende eigener Abhängigkeiten mit, jede mit einem weiteren Satz an Sicherheitsmeldungen, Lizenzen und Aktualisierungen. Prüfen Sie vor dem Hinzufügen einer Bibliothek, was sie nach sich zieht, nicht nur ihre eigene Größe.

Ist der Bedarf eng umrissen und klar spezifiziert, kann es günstiger sein, ein kleines, ins Repository übernommenes Stück Code mit eigenen Tests selbst zu verantworten.

Die Lizenz kommt mit dem Code

Spät im Projekt bemerkt jemand, dass eine Bibliothek in einem nicht quelloffenen Produkt unter einer Lizenz steht, deren Bedingungen sich auf das gesamte kombinierte Werk erstrecken. Beim Hinzufügen hatte niemand die Lizenzen geprüft, und die dynamische Einbindung änderte daran nichts. Prüfen Sie die Lizenz jeder Bibliothek zum Zeitpunkt ihrer Aufnahme, bewahren Sie die Liste mit der Veröffentlichung auf und verlangen Sie diese Listen auch von Lieferanten.

Weiterführende Lektüre

- Program Library HOWTO: Einführung — was eine Programmbibliothek unter Linux ist und wie sich statische, gemeinsam genutzte und dynamisch geladene Bibliotheken unterscheiden.
- The Rust Programming Language: Pakete und Crates — Bibliotheks-Crates gegenüber Binär-Crates und wie ein Paket sie aufnimmt.
- Semantic Versioning 2.0.0 — die Regeln für Versionsnummern, die die öffentliche API einer Bibliothek zu einer Zusage machen.
- Die Python-Standardbibliothek — eine Standardbibliothek in voller Größe: alles, was jede Python-Installation mitliefert.
- Häufig gestellte Fragen zu den GNU-Lizenzen — was die GPL für Programme bedeutet, die eine von ihr erfasste Bibliothek statisch oder dynamisch einbinden.

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 Bibliothek?“ 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/library/