Dependency Management

Formalisierte Pflege führt zu sichereren, stabileren Softwareprodukten

Die Integration von Open-Source-Komponenten beschleunigt die Entwicklungszeit enorm. Sicherheitsprobleme, Bugs und rechtliche Herausforderungen können jedoch dazu führen, dass die zuvor gewonnene Zeit wieder verloren geht. Durch sorgfältiges Management der Abhängigkeiten lässt sich dies weitgehend vermeiden.

Definition

Dependency Management ist eine Methode zur formalen Festlegung, Auflösung, Nachverfolgung und Aktualisierung von Softwarekomponenten innerhalb eines Projekts.

Die formale Festlegung deklariert Informationen zu den abhängigen Softwarekomponenten, etwa Namen und Versionsnummer, in einem einheitlichen und strukturierten Format.

Dies bildet die Grundlage zum Auflösen der Abhängigkeiten. Auflösen bedeutet, alle Abhängigkeiten zu identifizieren und zur Verfügung zu stellen. Das umfasst nicht nur die unmittelbar abhängigen, deklarierten Softwarekomponenten, sondern auch die transitiven Abhängigkeiten, wie die Abhängigkeiten der Abhängigkeiten.

Das Nachverfolgen stellt sicher, dass alle Softwarekomponenten jederzeit den Anforderungen entsprechen. In der Regel liegt der Fokus auf technischer und fachlicher Aktualität sowie Fehlerfreiheit und der Einhaltung von Lizenzen.

Beim Aktualisieren ändert sich die Version von mindestens einer direkt abhängigen Softwarekomponente. Das führt zu einer neuen Deklaration der Abhängigkeiten; Auflösen und Nachverfolgen werden erneut notwendig. Das setzt einen Kreislauf in Gang.

Compliance

- Informationssicherheit
- Qualitätssicherung
- gesetzliche Vorschriften
- Corporate Governance
- Transparenz

Standards

- SPDX
- CycloneDX
- SWID
- CPE
- Semantic Versioning
- CVE

DEP

Softwareentwicklung

- Frameworks und Bibliotheken
- Programmiersprachen
- Tools und Werkzeuge
- Software-Lebenszyklus

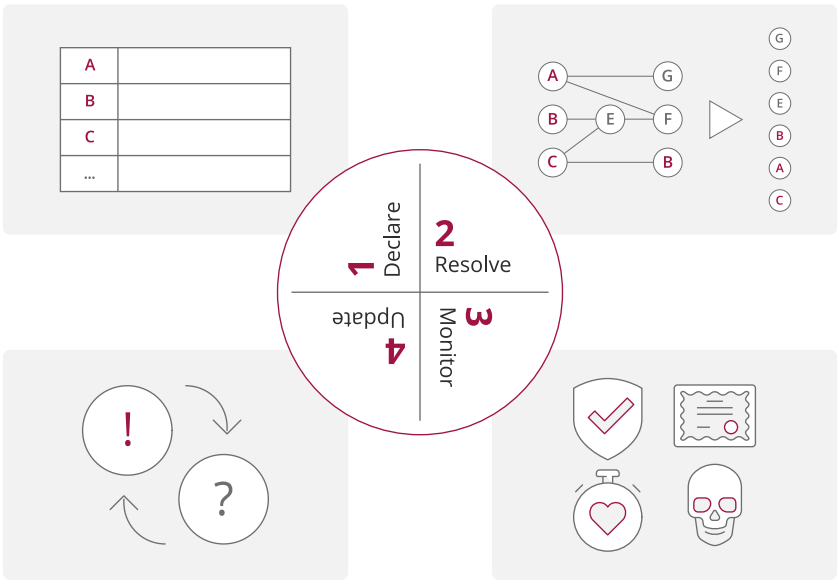
Prozesse

- Product Management
- Release Management
- Change Management
- Incident Management

Referenzszenario

Grundsätzlich kommt Dependency Management im Projekt bereits ab der ersten verbauten Softwarekomponente zum Einsatz.

Sobald die Anforderungen definiert sind, wird geprüft, welche Softwarekomponenten Teillösungen bereitstellen können. Diese werden dann evaluiert und gegebenenfalls in das Projekt integriert. Nach der Implementierung findet in der



Wartungsphase ein ständiges Monitoring statt. Damit begleitet Dependency Management ein Produkt während seines gesamten Software-Lebenszyklus, bis es außer Betrieb genommen wird.

Potenzial

Aus Unternehmenssicht ist Dependency Management notwendig, um die potenziell große Masse an verbauten Softwarekomponenten verwalten zu können. Moderne Webapplikationen referenzieren mitunter bis zu tausend externe Komponenten. Jede Komponente hat ihren eigenen Vertrag und verursacht Wartungsaufwand für Fehlerkorrekturen.

Monitoring gewährleistet die Einhaltung rechtlicher Vorgaben und minimiert das Risiko von Klagen. Häufige Updates führen zu einem stabileren System, verringern den Erfolg von Hackerangriffen und senken das Risiko eines Datenverlustes.

Reifegrad

Dependency Management wird schon lange in Projekten eingesetzt. Die Art und Weise, wie es betrieben wird, verändert sich stetig. In älteren Projekten wurden Abhängigkeiten noch unstrukturiert aufgeschrieben. Die Entwickler mussten sie dann einzeln von den Herstellerseiten herunterladen und einbinden.

Moderne Projekte sind heutzutage von bis zu tausend Softwarekomponenten abhängig, weshalb ein hoher Automatisierungsgrad unerlässlich ist. Für viele Bereiche, etwa Sicherheit und Lizenzen, existieren mittlerweile Automatisierungstools. Sie durchlaufen aktuell einen stetigen Verbesserungsprozess.

Marktübersicht

Es gibt keine Einzellösung, die alle Aspekte des Dependency Managements abbildet. Anbieter fokussieren sich auf einzelne Teilbereiche. Daher ist stets eine individuelle Kombination von Produkten notwendig.

Repositories bieten eine Möglichkeit, Softwarekomponenten zu speichern. Eines davon ist das JFrog Artifactory. Für jede Programmiersprache existiert ebenso ein kostenloses Repository, wie zum Beispiel das Maven Central Repository für Java-Komponenten.

Für das Auflösen von Abhängigkeiten bieten Tools in den einzelnen Programmiersprachen unterschiedliche Lösungsansätze. JavaScript hat den separaten Paketmanager npm. Währenddessen ist in Java der Paketmanager in den Build Tools Maven, Gradle und Buildr integriert. Paketmanager lösen die Abhängigkeiten auf und laden die deklarierten Softwarekomponenten von den Repositories herunter.

Tools wie Snyk und OWASP Dependency-Check legen den Fokus auf Sicherheit. Sie prüfen die verbauten Komponenten auf bekannte Sicherheitslücken. Wohingegen FOSSA und Black Duck Projekte scannen, um alle verwendeten Lizenzen zu identifizieren und einen Überblick zur Einhaltung der Lizenzpflicht zu geben. In der Regel landen die Ergebnisse auf Dashboards. Bei schwerwiegenden Problemen wird per E-Mail informiert.

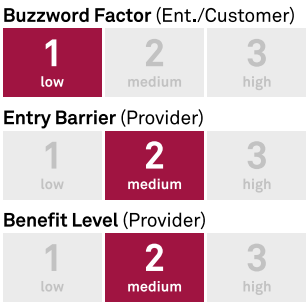
Dependabot und Renovate warten darauf, dass neue Versionen der Abhängigkeiten zur Verfügung stehen. Sobald eine neue Version veröffentlicht wird, versuchen die Programme diese in das Projekt zu integrieren.

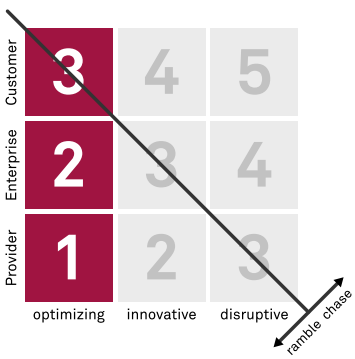
Alternativen

In der Regel werden immer Softwarekomponenten eingebunden und deshalb gibt es keine Alternative zum Dependency Management.

Fazit

- \+ vereinfachte Abhängigkeitspflege
- \+ Transparenz
- \+ vereinfachte Lizenzprüfung
- \+ Vereinfachte Sicherheitsprüfung
- \+ Eindeutigkeit
- \- verleitet zu womöglich überflüssigen Abhängigkeitsaufbau
- \- organisatorisch schwierig nachzuvollziehen
- \- nur mit fixierten Versionen reproduzierbar





https://msg.direct/techrefresh

Stand: Dezember 2023

msg systems ag

Robert-Bürkle-Straße 1 | 85737 Ismaning/München | Telefon: +49 89 96101-0 | Fax: +49 89 96101-1113 | www.msg.group | info@msg.group