Beschleunigung der digitalen Produktentwicklung durch den Einsatz von Open Source Bausteinen

Christoph Thelen · Susanne Apel · Christian Fritz

Eingegangen: 5. November 2020 / Angenommen: 4. Dezember 2020 / Online publiziert: 18. Dezember 2020
© Der/die Autor(en) 2020

Zusammenfassung Open Source Software ist aus der heutigen Software-Entwicklung nicht mehr wegzudenken. Sie ergänzt die kommerziellen Produkte oder geht sogar mit innovativen Lösungen voran. Die Eigenschaften von Open Source Software erfordern dabei besondere Sorgfalt bei der Auswahl. Dieser Artikel beschreibt die verschiedenen Einsatzfelder für Open Source Bausteine aus der Sicht eines Software-Dienstleisters und die Motivation, sie nicht nur zu verwenden, sondern auch daran mitzuwirken.

Schlüsselwörter Open Source · Industrialisierung der Software-Entwicklung · Open Source Ökosysteme · Bewertung und Auswahl von Open Source Bausteinen

Accelerating Digital Product Development Through Usage of Open Source Components

Abstract Open Source Software is an indispensable part of today's software development. It complements commercial products or even leads the way with innovative solutions. The characteristics of open source software require special care in the selection process. This article describes the different fields of application for open source components from the perspective of a software service provider and the motivation to not only use them but also to contribute to them.

Keywords Open source · Industrialisation of software development · Open source ecosystems · Evaluation and selection of open source components

C. Thelen (✉)
QAware GmbH, Rheinstraße 4C, 55116 Mainz, Deutschland
E-Mail: christoph.thelen@qaware.de

S. Apel · C. Fritz
QAware GmbH, Aschauer Straße 32, 81549 München, Deutschland

1 Industrialisierung der Software-Entwicklung

Konkurrenzfähig bleiben in einer Zeit, die von immer kürzeren Produktionszyklen bei gleichzeitig höherer Kundenorientierung geprägt ist, funktioniert nur auf der Basis von qualitativ hochwertiger, an die konkreten Anforderungen angepasster Software. Dies gilt sowohl, wenn die Software selber Teil des Produkts ist, als auch, wenn sie für den Entwurf oder die Produktion eingesetzt wird oder – als Datenanalyse oder bei Anwendungen der KI in verschiedenen Varianten – für die Analyse und Verbesserung existierender Produkte. Die PWC Studie „Digital Product Design 2025“ (Geissbauer 2019) listet unter anderem Digital Prototyping, Digital Twins und Product Lifecycle Management als Unterstützer der modernen Produktentwicklung auf. Gleichzeitig bedingt eine steigende Digitalisierung auch, die Cybersicherheit ernst zu nehmen, um die digitalen Bausteine des Entwicklungsprozesses vor Zugriff und Beschädigung durch Dritte zu schützen.

In Sparten mit direktem Software-Bezug gilt schon lange, dass Software qualitativ hochwertig, zeitzah und auf den konkreten Kundennutzen fokussiert entwickelt werden muss. Mit der wachsenden Abhängigkeit von Software wird diese Erkenntnis zukünftig in allen Industriezweigen immer relevanter.

Abb. 1 Systematik der Bausteine

Wie kann die Software-Entwicklung mit diesen stetig wachsenden Anforderungen Schritt halten? Hier können Ansätze der Industrialisierung der Software-Entwicklung (Adersberger 2015) weiterhelfen. Dabei zählt es, neben der Produktivität auch die Qualität der Software-Entwicklung zu verbessern. Nur so lässt sich Software nachhaltig bedarfsgerecht und wirtschaftlich produzieren.

Eine Option zur Steigerung der Produktivität ist die Verringerung der Fertigungstiefe. Das bedeutet, nur so viel neue Software zu schreiben, wie für den jeweiligen Anwendungsfall benötigt wird und für bereits gelöste Aufgaben Bausteine von existierender Software wiederzuverwenden. Hier kommt insbesondere Open Source Software ins Spiel, die Lösungen für viele der üblichen Problemstellungen anbietet. Dabei kann man (mindestens) drei Ebenen betrachten (Siehe auch Abb. 1):

- Bestandteile der zu entwickelnden Software
- Entwicklungsbegleitende Werkzeuge (Tools)
- Infrastruktur-Komponenten

Zu allen Punkten gibt es mittlerweile eine Vielzahl an Lösungen, sowohl Closed als auch Open Source. Letztere werden im Folgenden näher betrachtet.

2 Open Source Bausteine innerhalb des Produkts

Bei der Entwicklung eines Software-Produkts gibt es viele Gelegenheiten, von Wiederverwertung Gebrauch zu machen. Neben dem simplen Einsparen von Aufwand für eine Eigenimplementierung profitiert das Produkt bei guter Auswahl der Bausteine auch von dem Wissen, das in deren Implementierung geflossen ist. So müssen sich die eigenen Software-Entwickler*innen nicht tief in alle nötigen technischen Domänen einarbeiten, wodurch mehr Kapazität für das Verstehen der fachlichen Domäne bleibt, in der die eigentliche Wertschöpfung passieren soll. Eine Stichprobe in unseren Projekten ergab, dass die Fertigungstiefe (individuell entwickelter Code zum gesamten Code des fertigen Produkts) darin zwischen 1 und 7 % liegt. Umso wichtiger wird also die richtige Auswahl der Bausteine, damit die Qualität des Produkts nicht darunter leidet.

2.1 Programmiersprachen

Auf der untersten Ebene eines Software-Projekts wird in der Regel mindestens eine der höheren Programmiersprachen verwendet. Eine Ausnahme ist der direkte Umgang mit Hardwarekomponenten, den wir hier ausklammern.

Viele der momentan gebräuchlichen Sprachen sind entweder vollständig Open Source oder zumindest auch als Open Source Variante verfügbar. Prominente Beispiele sind sowohl Python, das unter einer eigenen OS-Lizenz angeboten wird, als auch Go unter einer „BSD-Style“-Lizenz. Beide sind damit auch für kommerzielle Software-Produkte geeignet.¹ Java wird in einem offenen Prozess (Java Community

¹ Zur Verwendung der unterschiedlichen Open Source Lizenzen siehe auch (Goldstein 2019).

Process) weiterentwickelt und hat mit OpenJDK auch eine quelloffene Implementierung, für die das Gleiche gilt.

2.2 Frameworks

Auf der nächsthöheren Ebene oberhalb der Programmiersprache helfen Frameworks dabei, häufig wiederkehrende Probleme mit geringem Aufwand zu lösen und typische Anwendungstypen einheitlich abzubilden. Sie unterstützen dabei, den Code in einer standardisierten Struktur zu organisieren und erleichtern den Entwickler*innen damit sowohl beim initialen Schreiben als auch beim späteren Verstehen und Ändern der Software die Arbeit.

Frameworks gibt es in verschiedenen Ausprägungen zu vielen Problemstellungen. Seien es Anwendungsframeworks wie Spring² oder JEE³ im Java-Umfeld, Lösungen für spezifische technische Domänen wie Hibernate⁴ für Datenbankanbindung und Objektrelationales Mapping, (Web) Frontend Frameworks mit den bekannten Vertretern Angular⁵ und React⁶ oder eines der vielen anderen Projekte.

Gemein ist fast allen Frameworks, dass sie sich das Ziel gesetzt haben, eine bestimmte Aufgabe in der Software-Entwicklung zu vereinfachen, zu generalisieren und – in gewissem Rahmen – zu standardisieren. Dazu wird in der Regel eine API für den Aufruf der Framework-Funktionalitäten bereitgestellt. Je nach Projekt und Anwendungszweck gibt es mehr oder weniger fest vorgegebene Code-Strukturen, Verwendungsweisen und/oder Abläufe. Im besten Fall gibt dies dem eigenen Code einen Rahmen und erhöht die Verständlichkeit. Wenn das Framework den gewünschten Use Case allerdings nicht gut unterstützt, können diese Vorgaben auch einengen und die Implementierung behindern.

Hier zeigt sich ein Vorteil der Open Source Varianten, denn in einem solchen Fall kann der Framework-Code leichter analysiert und bei Bedarf angepasst werden. Durch die Rückgabe dieser Änderung an das originäre Projekt kann diese Erweiterung nachhaltig im Framework ergänzt werden.

2.3 Libraries

Von einfachen, aber häufig wiederkehrenden, bis zu komplexen Problemen zu einer spezifischen Domäne gibt es in der Software-Entwicklung viele Aufgaben, bei denen Libraries unterstützen. Im Vergleich zu Frameworks lösen Library-Funktionen klar abgegrenzte Problemstellungen, ohne Bedingungen an den aufrufenden Code zu stellen.

Häufig entstehen Library-Projekte daraus, dass ein*e Software-Entwickler*in oder ein Team ein konkretes Problem in einer wiederverwendbaren Form abbildet und diese Lösung öffentlich bereitstellt. Für Java gibt es solche Bibliotheken

² <https://spring.io/>.

³ <https://www.oracle.com/java/technologies/java-ee-glance.html>.

⁴ <https://hibernate.org/>.

⁵ <https://angular.io/>.

⁶ <https://reactjs.org/>.

unter anderem von Apache⁷ (commons-lang usw.), aber auch spezielle Themen wie die Krypto-Library Bouncy Castle⁸ sind vertreten.

3 Entwicklungsbegleitende Werkzeuge (Tools)

In der Software-Entwicklung werden viele Werkzeuge verwendet, die den Entwicklungsprozess unterstützen, ohne in das fertige Produkt einzufließen. Dazu gehören Integrierte Entwicklungsumgebungen (IDE), Versionskontrollsysteme, Build-Tools, Testausführung und -Auswertung und viele weitere Hilfsmittel für die vielfältigen Schritte, die von den Anforderungen über Programmcode schließlich zum fertigen Software-Produkt führen.

Für alle zugehörigen Aufgaben gibt es Closed und Open Source Lösungen mit unterschiedlichen Stärken und Schwächen. Im Java-Umfeld setzt sich die kommerzielle IDE IntelliJ⁹ langsam gegen die Open Source Varianten Eclipse¹⁰ und Netbeans¹¹ durch (Schlosser 2018). In der Source Code Verwaltung findet das quelloffene Git¹² breite Verwendung und fließt auch in kommerzielle Produkte mit erweiterter Funktionalität ein (siehe z.B. Bitbucket von Atlassian), die Git in bezahlte Services einbetten. Aber auch Closed Source Produkte wie Microsofts Azure DevOps Server¹³ (vormals Team Foundation Server) oder Helix Core¹⁴ von Perforce sind im Einsatz.

Im Vergleich zu Frameworks oder Libraries ist die Lizenzfrage bei Tools meist weniger relevant, da die Software nicht in das fertige Produkt eingeht. Dies erlaubt zum Beispiel die Verwendung von Projekten mit GPL-Lizenz, die nicht direkt in den Quellcode einer Closed Source Anwendung einfließen dürfen.

4 Fallbeispiel: Lasttests mit Selenium und Gatling

4.1 Die Situation

Im Projekt eines Kunden standen wir vor folgender Situation: Für etwa 10 JSF-Bestands-Anwendungen waren Lasttests aufzubauen. Die Anwendungen waren zum Teil historisch gewachsen und hatten einen ähnlichen technischen Kern. Trotzdem hatte jede Anwendung ihre Eigenheiten, die zum Teil bekannt, zum Teil unbekannt waren. Für jede Anwendung gab es einen Fachbereich, der die Anwendung kannte und im Idealfall eigenständig Testszenarien erstellen sollte. Die Umsetzung der Lasttests wurde zeitnah benötigt, sollte aber dennoch soweit wie möglich reproduzierbar

⁷ <https://commons.apache.org/>.

⁸ <https://www.bouncycastle.org/>.

⁹ <https://www.jetbrains.com/de-de/idea/>.

¹⁰ <https://www.eclipse.org/ide/>.

¹¹ <https://netbeans.org/>.

¹² <https://git-scm.com/>.

¹³ <https://azure.microsoft.com/de-de/services/devops/server/>.

¹⁴ <https://www.perforce.com/products/helix-core>.

sein. Und es gab noch ein technisches Detail: Für die einzelnen Anwendungen war nicht bekannt, wie viel der jeweiligen Anwendungslogik im Browser ausgeführt wurde und welche UI-Anteile sich zwischen verschiedenen Sitzungen unterscheiden konnten (z.B. html-ids).

Damit war der folgende, oft verwendete Ansatz fehleranfällig: Viele Tools zeichnen den Netzwerkverkehr auf, während die Anwendung bedient wird, und wiederholen die Anfragen bei der Ausführung des Tests. Oft sind danach manuelle Anpassungen notwendig. In komplizierten Fällen müsste man JavaScript-Code aus der Anwendung duplizieren. Das funktioniert, wenn man die Anwendung im Quellcode analysieren kann und wenn man nur für eine oder wenige Anwendungen Lasttests erstellen möchte. Wir wollten einen Ansatz entwickeln, der diese Probleme möglichst umgeht und bei dem man Lastszenarien möglichst einfach aufnehmen und danach im Lasttest abspielen kann – egal was zwischendurch im Browser passiert.

4.2 Der Ansatz

Der folgende Aufbau beschreibt eine auf Open Source Bausteinen basierende Lösung. Der erste Teil davon bestand darin, eben genau einen Browser für den Anwendungsaufruf zu verwenden. Hier bietet sich das Framework Selenium¹⁵ an, das entwickelt wurde, um Browser automatisiert zu steuern, z.B. in Unit-Tests. Selenium bringt auch eine Selenium-IDE mit: Ein Browser-Plugin, mit dem Testfälle aufgezeichnet werden und danach in eine Datei exportiert werden können.

Für einen Lasttest fehlt noch die Lastgenerierung und -messung: Diese Szenarien müssen schnell und oft hintereinander ausgeführt werden. Dabei möchte man steuern, wie viel Last wann erzeugt wird. Optimalerweise ist anschließend bekannt, wie lange die einzelnen Schritte sowie das ganze Testszenario gedauert haben. Wir haben uns für Gatling¹⁶ entschieden, da Gatling Last zuverlässig erzeugen kann und nicht durch langsame Antwortzeiten selbst verlangsamt wird.

4.3 Das Setup

Wir haben dazu ein eigenes Gatling-Modul gebaut, das im Wesentlichen die mit Hilfe der Selenium-IDE vom Fachbereich erstellten Szenarien liest und sie von einer Menge Browser-Instanzen ausführen lässt. Durch Gatling Bordmittel erhalten wir dann einen interaktiven Report (siehe Abb. 2). Darin kann man z.B. ablesen, welche Schritte bei welcher angelegten Last wie lange gedauert haben oder wie in einem bestimmten Zeitraum die Antwortzeiten waren.

Uns war klar, dass das Ausführen der Tests mit echten Browsern viele Ressourcen benötigen wird. Daher haben wir vorgesehen, viele Instanzen in der Cloud hochzufahren. Da der Lasttest selten ausgeführt wird, sind die Kosten an dieser Stelle kein Problem. Wichtig und nicht zu unterschätzen ist bei diesem Vorgehen: man muss sicherstellen, dass eine Compliance-kompatible Cloud mit genügend Rechenpower zur Verfügung steht und dass die Anwendung von dieser Cloud aus aufrufbar ist.

¹⁵ <https://www.selenium.dev/>.

¹⁶ <https://gatling.io/>.

Abb. 2 Gatling Test-Report

Gerade weil die Lasttests selten ausgeführt werden müssen, bietet das Setup noch einen weiteren Vorteil: Alle zur Ausführung notwendigen Bestandteile sind in Form von Code abgelegt und versioniert. So kann ein*e Entwickler*in die Lasttests ausführen, ohne jede einzelne Komponente genau im Detail zu kennen. Java- und Kubernetes-Kenntnisse reichen vollkommen aus.

4.4 Fazit

Wir haben als OSS Komponenten Selenium, Gatling (inklusive Java und Scala) sowie ein Kubernetes-Cluster verwendet. Jede einzelne Komponente bringt viele Features mit, die wir mit geringem Aufwand nutzen konnten. Wir mussten nur noch den Aufwand investieren, diese zu kombinieren.

Selenium bringt uns die Browser-Fernsteuerung und die Szenario-Aufzeichnung. Gatling liefert die zuverlässige Last-Erzeugung und gut lesbare und informative Reports. Durch das Deployment des Setups in Kubernetes kann die Last unabhängig vom Laptop des Entwicklers ausreichend hoch skaliert werden.

Bei einer Umsetzung mit einem kommerziellen Produkt hätten wir vermutlich in der zur Verfügung stehenden Zeit nicht einmal die Lizenzen besorgen können, insbesondere durch Einkaufsprozesse des Kunden. Eine Erweiterung wie das hier erzeugte Gatling-Modul wäre mindestens schwieriger, wenn nicht unmöglich, gewesen.

5 Infrastruktur-Komponenten

Virtualisierung von Infrastruktur ermöglicht den Einsatz von Software-Bausteinen in diesem Bereich und die damit einhergehende stärkere Verteilung macht den Einsatz von unterstützender Software geradezu notwendig. Dadurch wird die Schnittstelle von Software-Entwicklung und „reinem“ Betrieb weiter in Richtung der Hardware verschoben. Unterstützt und gegenseitig verstärkt wird dieser Effekt beispielsweise vom DevOps-Gedanken, nach dem Entwicklungsteams unter anderem auch die Entscheidung über die eingesetzte Infrastruktur treffen. Im Folgenden betrachten wir eine Auswahl der zugehörigen Bausteine.

5.1 Betriebssysteme

Auf dem Betriebssystem-Markt für Server hat sich das quelloffene Linux bereits seit einiger Zeit gegen die Closed Source Konkurrenz wie das ehemals dominierende Unix in seinen verschiedenen Varianten durchgesetzt (W3Techs 2020; Datanyze 2020). Einerseits sicher aus Kostengründen und der wesentlich einfacheren Lizenzierung, andererseits aber nicht zuletzt auch weil es die offene Architektur von Linux ermöglicht, das Betriebssystem zielgenau auf die eigenen Anforderungen zu konfektionieren und bei Bedarf sogar im Code anzupassen. Ein Beispiel dafür ist „SELinux“, ursprünglich von der National Security Agency in den USA zur Verbesserung der Betriebssicherheit von Linux entwickelt und 2001 als quelloffene Erweiterung für Linux-Systeme veröffentlicht. Mittlerweile ist diese Erweiterung in den Linux Kernel integriert, sodass alle Linux-Anwender*innen davon profitieren (National Security Agency 2001).

5.2 Infrastruktur-Management

Wichtige Aufgaben des Betriebs von Server-Systemen sind die Provisionierung von Hardware und Systemen sowie das Konfigurationsmanagement der ausgeführten Software. Auch in diesen Bereichen setzen die meisten Rechenzentren auf Open Source Bausteine wie Chef¹⁷, Terraform¹⁸, Ansible¹⁹ und viele weitere Projekte. Im Vergleich zu den auch hier vorhandenen Closed Source Varianten punkten sie durch ihre gute Erweiterbarkeit. Ist für eine spezifische Aufgabe kein fertiges Modul verfügbar ist es meist einfach, eine entsprechende Erweiterung zu schreiben. Offener Beispiel-Code aus den existierenden Modulen erleichtert dies erheblich.

5.3 Containerisierung

Auch das Stichwort Containerisierung wird hauptsächlich von Open Source Projekten getrieben. Das Versprechen von Docker²⁰ und ähnlichen Laufzeitumgebungen

¹⁷ <https://www.chef.io/>.

¹⁸ <https://www.terraform.io/>.

¹⁹ <https://www.ansible.com/>.

²⁰ <https://www.docker.com/>.

ist dabei, den ausgeführten Anwendungen ein genau definiertes Umfeld zu bieten. Sei es die genaue Version des Betriebssystems, seien es installierte Treiber oder Tools – das Container-Image erlaubt es, diese Parameter zu fixieren. Ein einmal erzeugtes Image kann auf beliebigen Umgebungen ausgeführt werden und ist dabei in seinem Inneren stabil. Ein auf Linux basierendes Image kann auch unter Unix oder Windows Hosts ausgeführt werden, ohne dass die enthaltenen Anwendungen einen Unterschied bemerken.

Für Software-Entwickler*innen bedeutet dies, dass sie ihre Anwendungen auf dem eigenen Rechner bauen, ausführen und testen können, unter weitgehend identischen Bedingungen wie zur Laufzeit auf dem Produktivsystem.

Selbstverständlich existieren auch kommerzielle Container- und Virtualisierungslösungen wie VMware Tanzu²¹. Sie erfordern aber eine deutlich stärkere Bindung an einen Hersteller oder ein Vertriebsmodell, womit die Hürde für die Verwendung deutlich erhöht wird. Und nicht zuletzt das aus der Open Source Community entstandene Ökosystem an erweiternden Tools macht den großen Erfolg von Docker aus. Die Vermutung liegt nahe, dass keine so große Zahl an Projekten und Ergänzungen für Docker als Closed Source Software entstanden wäre.

5.4 Orchestrierung

Die konsequente Weiterführung dieses Trends findet sich in den modernen Cloud-Technologien. OpenStack²², Kubernetes²³ und verwandte Systeme bauen insbesondere auf der Containerisierung auf und ermöglichen damit automatisches Deployment und Management von Anwendungen in bisher ungeahnter Einfachheit. Die Verbindung aller genannten Bausteine erlaubt die vollständige Beschreibung der Infrastruktur und des Deployments einer Anwendung („Infrastructure as code“, vgl. (Özel et al. 2020)) und macht das Ziel der DevOps-Bewegung, dass Software-Entwicklungsteams vollständig und umfänglich verantwortlich für ihr Produkt sind, greifbar.

Auch hier spielt wieder die Community, spielt das Ökosystem um das eigentlich Projekt eine entscheidende Rolle. Kubernetes, entstanden als internes Google-Projekt zur Orchestrierung von Applikationen auf der eigenen Infrastruktur, begann seinen rasanten Aufstieg zum quasi-Standard des Cloud Betriebsmodells (einziges Orchestration Projekt im Portfolio der Cloud Native Computing Foundation (CNCF)) mit der Veröffentlichung als Open Source Projekt in 2014 (Burns 2018). Eine Vielzahl an Entwickler*innen arbeitet an neuen Features und Bugfixes, zusätzliche Funktionalität wird in Form von – in aller Regel – quelloffenen Erweiterungen implementiert. Trotzdem finden sich auf der Cloud Native Landscape der CNCF (siehe auch Abb. 3) 200 Closed Source Produkte (vor allem in den Bereichen *Platform*, *Storage* sowie *Observability and Analysis*) und zeigen, dass auch kommerzielle Produkte im Rahmen eines Open Source Ökosystems erfolgreich sein können.

²¹ <https://tanzu.vmware.com/de/>.

²² <https://www.openstack.org/>.

²³ <https://kubernetes.io/>.

Abb. 3 CNCF Landscape im Oktober 2020 © (CNCF 2020)

6 Erfolgsfaktoren und Probleme

Erfolgreiche Open Source Bausteine werden von vielen Software-Projekten verwendet und sind damit vielfach getestet und bewertet. Das Gleiche gilt selbstverständlich auch für Closed Source Bausteine, hier ist allerdings eine Beurteilung nur „von außen“ möglich. Weder die Architektur noch der Code-Stil oder die automatisierten Tests lassen sich bewerten. Dagegen kann auch der Erfolg eines Open Source Projekts relativ gut anhand der Aktivität innerhalb des Repositorys oder auf der Webseite abgeschätzt werden. Gibt es viele Änderungen von Team-Mitgliedern oder Pull-Requests von anderen Entwickler*innen? Wie viele Bugtracking-Tickets sind offen? Wann war das letzte Release?

Vorsicht walten lassen muss man, wenn die Antworten auf diese Fragen darauf hindeuten, dass es sich um ein weniger erfolgreiches oder aufgegebenes Projekt handelt. Hier kehren sich die Vorteile nämlich zum Teil ins Gegenteil: Fehler-Tickets werden nicht beantwortet, Anpassungen an neue Rahmenbedingungen finden nicht mehr statt. Entwickler*innen, die solche Bausteine einsetzen, sind gezwungen, die notwendigen Änderungen selbst durchzuführen. Durch das Open Source Modell ist dies immerhin möglich, kostet aber Ressourcen, die in der Regel im konkreten Projekt nicht eingeplant sind. Als Alternative bleibt sonst nur, neue Bausteine auszuwählen und zu integrieren.

Im Gutfall allerdings werden Fehler behoben bzw. können mit vertretbarem Aufwand von den Anwender*innen selbst behoben werden. Durch den Rückfluss der Fixes an das Projekt profitieren auch alle anderen Anwender*innen, was insbesondere bei Security Fixes relevant ist.

Bei Closed Source Produkten muss hierfür ein Ticket eröffnet werden, das ggf. nicht oder erst nach längerer Wartezeit umgesetzt wird. Die Erfüllung von Feature-Requests ist abhängig von den Interessen des Betreibers.

Allgemein liegt der Vorteil beim Support des Bausteins aber bei den kommerziellen Produkten. Mit dem Kauf der Lizenz und ggf. einem separaten Supportvertrag haben die Anwender*innen Anspruch auf Unterstützung. Die meisten Hersteller bemühen sich, dieses Argument durch kurze Antwortzeiten zu bekräftigen. Auf Seiten der Open Source Projekte ist man oft abhängig von der Community des Projekts. Hier gibt es keine Garantie, dass Fragen beantwortet und Probleme gelöst werden. Aktive Communitys fangen dieses Problem in der Regel auf, bei unbekannten Projekten ist man auf sich allein gestellt. Dann hängt es stark von der Architektur- und Code-Qualität ab, ob sich Bugfixes oder neue Features gut im Alleingang einarbeiten lassen. Zu einigen der größeren Projekten kann Support von beteiligten Unternehmen oder unabhängigen Dienstleistern bezogen werden, womit sich sowohl das Kosten- als auch das Unterstützungsargument den Closed Source Varianten angleichen.

Ein weiterer Aspekt beim Einsatz von Open Source Software betrifft die rechtliche Seite der Verwendung von fremdem Code. Jedes Open Source Projekt kann über die Bedingungen für die Verwendung seines Codes entscheiden, darum müssen die jeweils geltenden Lizenzen sorgfältig geprüft werden. Insbesondere beim Einsatz in Closed Source Produkten dürfen Bausteine unter bestimmten Lizenztypen wie z.B. der GPL nicht verwendet werden (Goldstein 2019). Um der großen Vielfalt der Lizenzen von verwendeten Bausteinen und deren Abhängigkeiten (inklusive aller transitiven Abhängigkeiten) Herr zu werden, braucht es einer bestimmten Grundkenntnis und Sensibilität gegenüber diesen Themen. Zum Glück gibt es auch hier Tools, die z.B. dabei unterstützen, die verwendeten Lizenzen übersichtlich zu reporten.

Den größten Vorteil spielen Open Source Projekte dann aus, wenn sie ein florierendes Ökosystem begründen, in dem sich Anwender gegenseitig unterstützen, an der Weiterentwicklung beteiligen oder komplementäre Bausteine schaffen, die den Mehrwert des Projekts erhöhen. Gute Beispiele dafür sind das schon genannte Kubernetes, das Java Framework Spring und natürlich Linux.

7 Kriterien für Auswahl und Einsatz von Open Source

Vor der Entscheidung, ob ein bestimmter Open Source Baustein eingesetzt wird, sollten – wie bei jeder Investitionsentscheidung – Vor- und Nachteile abgewogen werden. Neben den angesprochenen rechtlichen Aspekten gibt es noch weitere Punkte, die in eine Bewertung einfließen sollten. Für unsere Software-Projekte haben wir eine entsprechende Open Source-Checkliste (Tab. 1) erarbeitet, mit der die wichtigsten Fragen überprüft werden.

Bei jedem Aufläuten einer „Alarmglocke“ sollte genau untersucht und bewertet werden, wie sehr dieser Fall eine Verwendung im eigenen Produkt beeinträchtigt. Sollten sich daraus große Bedenken ergeben, so sucht man lieber nach Alternativen. Bei kleinen oder mittleren Bedenken sollte der Baustein zumindest so eingebunden werden, dass er später leicht ausgebaut oder ersetzt werden kann.

Tab. 1 Open Source Checkliste

Frage	Alarmglocke	Quelle
Unter welcher Lizenz steht der Baustein?	Lizenz mit starkem Copyleft (z. B.: GPL). Keine OSI-konforme Lizenz	Projekt-Website, openhub.net, tldrlegal.com, opensource.org
Wie reif ist die Software?	Es gibt kein Final-Release. Das aktuelle Release hat eine Versionsnummer unterhalb 1.0. Das Codevolumen im Projekt steigt stark an	Projekt-Website, mvnrepository.com, antepedia.com
Wie gut ist der Support?	Kein Community-Bereich oder durchschnittliche Antwortzeit in der Community >24h. Kein Issue Tracker zugreifbar	Projekt-Website, stackoverflow.com
Steht ein Maven-Artefakt zur Verfügung? (Java-spezifisch)	Es ist kein Artefakt bei mvnrepository.com verzeichnet. Es steht ein Maven-Artefakt in einem nicht vertrauenswürdigen Repository zur Verfügung	Projekt-Website, mvnrepository.com
Wie gut ist die Dokumentation?	Keine API-Dokumentation. Kein Getting Started Guide/keine Nutzungsbeispiele. Keine Nutzer-Dokumentation. Keine Dokumentation zur aktuellen Version vorhanden	Projekt-Website
Gibt es kritische Mängel?	Kritische Bugs oder kritische Sicherheitslöcher sind bekannt, für deren Target-Release es noch keine Terminaussage gibt bzw. der Release-Termin länger als 2 Monate entfernt liegt bzw. die schon länger als 4 Monate offen sind	Projekt-Issue-Tracker, Projekt-Website, antepedia.com, cve.mitre.org, osvdb.org
Wie gut ist die innere Qualität?	Dokumentationsgrad <15 %. Kein Build-Skript vorhanden oder Build-Skript baut nicht. Schwerwiegende Code-Anomalien. Testüberdeckung <30 %. Hohe Anzahl an Zyklen. Undurchsichtige Abhängigkeitsstruktur	Openhub.net, Projekt-Repository, FindBugs, Sonar, Structure 101
Ist das Projekt tot oder irrelevant?	Es existiert kein Eintrag für das Projekt auf openhub.net. Es gab kein Release in den vergangenen 12 Monaten. Es gab in den letzten 12 Monaten mehr als einen Monat ohne Commit	Openhub.net, Projekt-Website, mvnrepository.com
Wie stabil ist die Community?	Das Projekt hat weniger als 3 aktive Committer. Das Versionskontrollsystem ist nicht öffentlich zugänglich	Openhub.net, Projekt-Website
Wie gut ist die Akzeptanz?	Es gibt im Vergleich zu anderen Bausteinen mit vergleichbarer Funktionalität kaum öffentliches Interesse in Form von Blog-Einträgen, stackoverflow-Fragen oder Google-Suchen	Google Trends, stackoverflow, Google

OSI Open Source Initiative: <https://opensource.org/licenses>

Die angegebenen Kriterien und Werte sind aus unserem „Bauchgefühl“ für den Einsatz in unseren Projekten entstanden und erheben keinen Anspruch auf Allgemeingültigkeit – je nach Rahmenbedingungen können sich Relevanz und Grenzwerte verschieben.

8 Motivation für Verwendung und Mitarbeit an Open Source Projekten

Aus der modernen Software-Entwicklung ist Open Source nicht mehr wegzudenken. Open Source Bausteine werden erfolgreich auf allen Ebenen von Entwicklung und Betrieb eingesetzt und tragen erheblich zur Produktivität bei. Gibt es einen allgemeinen Trend, Closed Source Bausteine durch Open Source zu ersetzen? Aus der Sicht eines einzigen Software-Dienstleisters ist dies nicht umfassend bewertbar. Wir nehmen allerdings durchaus wahr, dass unsere Kunden offener gegenüber Open Source werden. Auch Kunden, die bisher strikt auf Closed Source Software gesetzt haben, sind nun aufgeschlossen gegenüber der Verwendung entsprechender Open Source Pendants. Das liegt sicher auch daran, dass sich der Markt für Open Source Bausteine in den letzten Jahren weiter professionalisiert hat. So ist für die relevanten Projekte meist „echter“ Support verfügbar, entweder direkt vom Hersteller oder von unabhängigen Dienstleistern. Auch die vielleicht befürchteten rechtlichen Probleme sind managebar, wenn man die Bausteine sorgfältig auswählt. Dem entgegen stehen weiterhin Punkte wie fehlende Gewährleistung und die eben doch noch latent vorhandene Rechtsunsicherheit beim Einsatz von Open Source Software. Hier sind Anwender von Closed Source auf der sicheren Seite, wenn sie entsprechende Verträge abschließen.

Oft werden geringere Einarbeitungszeit und -Kosten als Vorteil von kommerziellen Produkten angeführt, weil die Hersteller meist Schulungen anbieten. Erfahrungsgemäß ist dies aber stark abhängig von der Qualität der Schulungsleistung. Spätestens wenn bei der Implementierung des eigenen Produkts mehr als „Hallo Welt“-Anforderungen an den Baustein gestellt werden, muss man doch wieder die (hoffentlich gute) Dokumentation wälzen. Dies gilt im gleichen Maße für Open und Closed Source Produkte.

Eine wichtige Informationsquelle ist in beiden Fällen der Austausch mit anderen Anwendern des Bausteins. Erfahrungen, Ideen und Problemlösungen Anderer helfen sowohl bei der Auswahl als auch beim Einsatz in der eigenen Software. Open Source Projekte haben es leichter, eine aktive Community zu entwickeln, da die Motivation zum freien Teilen von Informationen höher ist, wenn dies in beiden Richtungen passiert. Kommerzielle Produkte müssen mehr Aufwand investieren, um dies zu fördern.

Für uns bietet die aktive Mitwirkung an Bausteinen, die wir oft und/oder an zentralen Stellen unserer Projekte einsetzen, die Möglichkeit, unsere Anforderungen direkt einzubringen und Problemstellungen unserer Kunden optimal zu unterstützen. Bugs, die wir in unseren Projekten aufgedeckt haben, konnten wir schnell und ohne bürokratischen Umweg über den Hersteller beheben und so schnell zu einer guten Lösung kommen. Ebenso hilft uns der offene Austausch mit den Entwickler*innen und anderen Anwender*innen der Bausteine dabei, Nutzungsoptionen gut zu bewerten und auszuwählen. Für relevante Projekte wirken wir zudem gerne als „Botschafter“ und sprechen bei Konferenzen und Meetups über deren Einsatz. Dies hilft nicht nur den Projekten, sondern erhöht auch unsere Sichtbarkeit als Experten.

9 Fazit

Als Software-Dienstleister sind wir darauf angewiesen, uns mit Qualität, Innovationskraft und Methoden der Software-Industrialisierung gegen vermeintlich günstigere Konkurrenz aus In- und Ausland durchzusetzen. Dies erfordert den Einsatz von bewährten Lösungen für Standardprobleme, damit sich das konkrete Projekt auf die spezifischen Anforderungen konzentrieren kann. Nur so lassen sich Innovation, Qualität und Entwicklungsgeschwindigkeit auf dem gewünschten Niveau halten. Open Source Bausteine sind dabei kommerziellen Varianten in der Funktion ebenbürtig und bieten bei umsichtigem Einsatz viele Vorteile. Insbesondere die Möglichkeit der Mitgestaltung eines Bausteins sowie die oft sehr aktiven Communities helfen bei der zielgerichteten Verwendung in unseren Projekten.

Wie bei allen Entscheidungen gilt es auch hier, die Stärken und Schwächen der vorhandenen Optionen abzuwägen und auf die Eignung für den jeweiligen Use Case zu untersuchen. Letztendlich sollte man sich dazu weder auf Marketing-Versprechen noch auf Community-Hypes verlassen, sondern auf den eigenen Sachverstand.

Open Access Dieser Artikel wird unter der Creative Commons Namensnennung 4.0 International Lizenz veröffentlicht, welche die Nutzung, Vervielfältigung, Bearbeitung, Verbreitung und Wiedergabe in jeglichem Medium und Format erlaubt, sofern Sie den/die ursprünglichen Autor(en) und die Quelle ordnungsgemäß nennen, einen Link zur Creative Commons Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden.

Die in diesem Artikel enthaltenen Bilder und sonstiges Drittmaterial unterliegen ebenfalls der genannten Creative Commons Lizenz, sofern sich aus der Abbildungslegende nichts anderes ergibt. Sofern das betreffende Material nicht unter der genannten Creative Commons Lizenz steht und die betreffende Handlung nicht nach gesetzlichen Vorschriften erlaubt ist, ist für die oben aufgeführten Weiterverwendungen des Materials die Einwilligung des jeweiligen Rechteinhabers einzuholen.

Weitere Details zur Lizenz entnehmen Sie bitte der Lizenzinformation auf <http://creativecommons.org/licenses/by/4.0/deed.de>.

Literatur

- Adersberger J (2015) Software-Industrialisierung – Wie industrialisiert man Wissensarbeit? In: Linnhoff-Popien C, Zaddach M, Grahl A (Hrsg) Marktplätze im Umbruch. Springer Vieweg, Berlin, Heidelberg, S 643–653
- Burns B (2018) The history of Kubernetes & the community behind it. <https://kubernetes.io/blog/2018/07/20/the-history-of-kubernetes-the-community-behind-it/>. Zugegriffen: 20.10.2020
- Datanyze (2020) Windows Server market share. <https://www.datanyze.com/market-share/operating-systems--443/windows-server-market-share>. Zugegriffen: 09.10.2020
- Geissbauer R (2019) Digital product development 2025. <https://www.pwc.de/de/digitale-transformation/pwc-studie-digital-product-development-2025.pdf>. Zugegriffen: 02.10.2020
- Goldstein A (2019) Open source licenses explained. <https://resources.whitesourcesoftware.com/blog-whitesource/open-source-licenses-explained>. Zugegriffen: 07.10.2020
- National Security Agency (2001) National Security Agency shares security enhancements to LINUX. <https://www.nsa.gov/news-features/press-room/Article/1637367/national-security-agency-shares-security-enhancements-to-linux/>. Zugegriffen: 09.10.2020
- Özel A, Pautz T, Schmidt N (2020) Infrastructure as Code als Maßnahme zur Cloud Automatisierung – Hilfestellung zur Auswahl des richtigen Werkzeugs. HMD 57:936–948. <https://doi.org/10.1365/s40702-020-00657-0>
- Schlösser H (2018) Eclipse vs. IntelliJ vs. NetBeans: Das ist die populärste Java-IDE 2018. <https://jaxenter.de/eclipse-intellij-netbeans-68567>. Zugegriffen: 07.10.2020

- The Linux Foundation (2020) Cloud native computing foundation landscape. <https://landscape.cncf.io/>. Zugegriffen: 28.10.2020
- W3 Techs (2020) Usage statistics of operating systems for websites. https://w3techs.com/technologies/overview/operating_system. Zugegriffen: 09.10.2020