Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Bachelorarbeit

Titel der Arbeit // Title of Thesis

Konzeption und Entwicklung einer Continuous-Integration-Strategie für Kundenprojekte auf Basis der Shopware-Platform

---

Akademischer Abschlussgrad: Grad, Fachrichtung (Abkürzung) // Degree

Bachelor of Science (B.Sc.)

Autorenname, Geburtsort // Name, Place of Birth

Frederik Bußmann, Coesfeld

Studiengang // Course of Study

Informatik.Softwaresysteme

Fachbereich // Department

Wirtschaft und Informationstechnik

Erstprüferin/Erstprüfer // First Examiner

Prof. Dr.-Ing. Martin Schulten

Zweitprüferin/Zweitprüfer // Second Examiner

Martin Knoop

Abgabedatum // Date of Submission

xx.xx.2023

Eidesstattliche Versicherung

Bußmann, Frederik

Name, Vorname // Name, First Name

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel

Konzeption und Entwicklung einer Continuous-Integration-Strategie für Kundenprojekte auf Basis der Shopware-Platform

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Stadtlohn, den

Ort, Datum, Unterschrift // Place, Date, Signature

Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis

API . . . . . Application Programming Interface

CD . . . . . Continuous Deployment

CDE . . . . . Continuous Delivery

CI . . . . . Continuous Integration

CLI . . . . . Command-Line Interface

CMS . . . . . Content Management System

CSS . . . . . Cascading Style Sheets

DI . . . . . Dependency Injection

IDE . . . . . Integrated Development Environment

MS . . . . . Mutation Score

MSI . . . . . Mutation Score Indicator

NPM . . . . . Node Package Manager

ORM . . . . . Object-Relational Mapping

PHPCS . . . . . PHP Code Sniffer

PHPMD . . . . . PHP Mess Detector

PR . . . . . Pull Request

QA . . . . . Quality Assurance

TTY . . . . . Teletypewriter

VCS . . . . . Version Control System

VM . . . . . Virtual Machine

YAML . . . . . YAML Ain't Markup Language

Abbildungsverzeichnis

1 Einleitung

Im Rahmen dieser Arbeit werden verschiedene Aspekte betrachtet, um ein Konzept für das Einbinden von Continuous Integration (CI) in Shopware-basierten Projekten zu erarbeiten. Shopware ist eine E-Commerce-Plattform und eine der meistgenutzten Online-Shop-Lösungen in Deutschland[1], welche es Unternehmen ermöglicht, ihre Präsenz im digitalen Markt auszubauen. Die Implementierung von CI-Praktiken in solchen Projekten kann den Entwicklungszyklus beschleunigen und die Qualität des Endprodukts erhöhen. Dies trägt zur Stärkung der Wettbewerbsfähigkeit bei und fördert eine agile und reaktionsschnelle Entwicklungsumgebung.

1.1 Motivation

In der heutigen schnelllebigen digitalen Welt ist die Fähigkeit, qualitativ hochwertige Softwareprodukte schnell auf den Markt zu bringen, von entscheidender Bedeutung. Die E-Commerce-Branche ist sehr wettbewerbsintensiv und Unternehmen müssen ständig innovieren, um relevant zu bleiben. Dabei spielt die Effizienz der genutzten Softwareentwicklungsmethoden eine wichtige Rolle. Um eine möglichst reaktionsschnelle und effektive Umgebung für Entwicklerteams in Shopware-basierten Projekten zu schaffen, können Methodiken des Continuous-Software-Engineering verwendet werden. Die Entwicklung einer robusten, jedoch flexiblen CI-Strategie ist im Hinblick auf sich ständig ändernde Technologien und Anforderungen besonders wichtig für die Qualität von Software und wird in Zukunft immer relevanter.

1.2 Zielsetzung

Die Entwicklung einer Continuous-Integration-Strategie für auf Shopware basierende Kundenprojekte ist das primäre Ziel dieser Arbeit. Die Strategie soll dazu beitragen, die Qualität der Software zu verbessern, die Effizienz des Entwicklungsprozesses zu steigern und letztendlich die Kundenzufriedenheit zu erhöhen. Die nachfolgend definierten Ziele Z_n dienen als Leitfaden für die Entwicklung der CI-Strategie:

- Z_1 **Höhere Entwicklungsgeschwindigkeit:**

  Die Einführung einer umfangreichen CI-Strategie soll durch die Automatisierung von Prozessen die Effizienz der Softwareentwicklungsteams verbessern und die Zeit bis zum Produkt-Release senken.

- Z_2 **Steigerung der Softwarequalität:**

  CI soll aufgrund automatischer Tests und Analysen dazu beitragen, Fehler frühzeitig im Entwicklungsprozess erkennen und beheben zu können, was die Qualität des Endprodukts verbessert.

- Z_3 **Bessere Kommunikation:**

  Die CI-Strategie und die damit verbundenen Prozesse die sich für Entwicklerteams ergeben, sollen zu einer Steigerung der Transparenz innerhalb des Teams und somit zu einer verbesserten Kollaboration führen.

[1] Vgl. eCommerceDB. *The Most Commonly Used Shop Software Among Online Shops in Germany - Shopware and Salesforce Share Rank No. 1.* 2023.

Bei der Konzeption sollen diese Ziele verfolgt und die Maßnahmen der zu erarbeiteten CI-Strategie dementsprechend ausgerichtet werden. Die Strategie soll dabei nicht nur die technischen Aspekte von Continuous Integration berücksichtigen, sondern auch die organisatorischen und kulturellen Veränderungen, die mit der Einführung von CI einhergehen. Darüber hinaus soll die Strategie flexibel genug sein, um sich an zukünftige Veränderungen und Entwicklungen anpassen zu können.

1.3 Struktur der Arbeit

Die Arbeit ist strukturell in die folgenden fünf Hauptabschnitte unterteilt:

- **Fachlicher Hintergrund**

  Zunächst wird der fachliche Hintergrund für die Arbeit festgelegt, der Abschnitt umfasst eine Einführung in das Continuous-Software-Engineering und die Prinzipien und Praktiken von Continuous Integration, sowie eine Übersicht über die Shopware-Plattform. Dieser Abschnitt dient dazu, ein grundlegendes Verständnis für die Themen und Technologien zu schaffen, die in der Arbeit behandelt werden.

- **Analyse und Konzept**

  Dieser Abschnitt befasst sich mit der Analyse der aktuellen Situation und der Entwicklung eines Konzepts für die CI-Strategie. Dies beinhaltet die Identifizierung von Herausforderungen und Anforderungen, die Berücksichtigung von Best Practices und die Ausarbeitung eines Plans für die Implementierung der Strategie. Die Konzeptionierung stützt sich dabei auf die im vorherigen Abschnitt aufgezeigte Fachliteratur.

- **Entwicklung der CI-Strategie**

  In diesem Abschnitt wird die Entwicklung der CI-Strategie beschrieben. Dies umfasst die Auswahl und Konfiguration der benötigten Tools, die Definition von Prozessen und Workflows und die Implementierung von Automatisierungen und Tests. Der Fokus liegt dabei auf der praktischen Anwendbarkeit des im vorherigen Abschnitt entwickelten Konzepts.

- **Evaluierung**

  Die Auswertung der implementierten CI-Strategie wird im folgenden Abschnitt behandelt. Dies beinhaltet die Durchführung von Tests und Messungen, um die Wirksamkeit und Effizienz der Strategie zu bewerten. Dabei wird die umgesetzte Strategie im Hinblick auf die im fachlichen Hintergrund aufgezeigten Prinzipien geprüft. Die Ergebnisse dieser Evaluierung werden analysiert und interpretiert, um Rückschlüsse auf den Erfolg der Implementierung zu ziehen.

- **Schlussfolgerung und Ausblick**

  Der letzte Abschnitt fasst die Ergebnisse der Arbeit zusammen und es werden Schlussfolgerungen über die CI-Strategie und dessen Nutzung in Shopware-Projekten gezogen. Darüber hinaus wird ein Ausblick auf mögliche zukünftige Entwicklungen und Verbesserungen gegeben. Dieser Abschnitt dient dazu, die Arbeit abzurunden und einen Ausblick auf weitere Forschungs- und Entwicklungsarbeiten in diesem Bereich zu geben.

2 Fachlicher Hintergrund

Für die Erarbeitung einer geeigneten CI-Strategie wird zunächst ein Einblick in die Disziplin des Continuous-Software-Engineering gegeben. Anschließend werden die Begrifflichkeiten und Prinzipien von Continuous Integration definiert und weitere relevante Technologien und Bereiche für die Nutzung von CI erläutert. Darüber hinaus wird eine Übersicht über die Funktionen und Mechanismen der Shopware-Plattform gegeben.

2.1 Continuous Software Engineering

Continuous-Software-Engineering fasst die Prinzipien der Continuous Integration (CI), Continuous Delivery (CDE), und Continuous Deployment (CD) zusammen. Shahin et al. definieren den Begriff als einen Bereich der Softwareentwicklung, bei dem es um die Entwicklung, Auslieferung und das schnelle Feedback von Software und Kunde geht. Die Disziplin umfasst Geschäftsstrategie und Planung, sowie Entwicklung und den Betrieb der Software.[2] Diese kontinuierliche Integrierung von Software ist sehr kompatibel mit den häufigen iterationen in der agilen Softwareentwicklung und wurde unter anderem durch die agile Methodik des Extreme Programming bekannt.[3] Nachfolgend werden die Bereiche der agilen Softwareentwicklung und der CI, CDE und CD kurz erläutert.

Agile Software Development

Agile Softwareentwicklung ist ein Ansatz zur Softwareentwicklung, der auf Flexibilität und Kundeninteraktion setzt. Im Gegensatz zu traditionellen, plangetriebenen Methoden, die die Anforderungen und Lösungen am Anfang des Projekts festlegen, erlaubt die agile Methodik Änderungen und Anpassungen während des gesamten Entwicklungsprozesses. Dies wird durch iterative Entwicklung und regelmäßiges Feedback erreicht. Zu den wichtigsten Prinzipien der agilen Softwareentwicklung gehören die kontinuierliche Auslieferung von Software, Offenheit für sich ändernde Anforderungen und enge Zusammenarbeit zwischen Teams und Entwicklern.[4] Gren und Lenberg fassen Agile als die Reaktionsfähigkeit im Hinblick auf sich ständig ändernde Anforderungen und Umgebungen zusammen.[5]

Continuous Integration

Continuous Integration (CI) ist ein Softwareentwicklungsprozess, bei dem Entwickler ihre Änderungen regelmäßig, oft mehrmals täglich, in ein gemeinsames Repository integrieren. Jede dieser Integrationen wird dann von einem automatisierten Build-System überprüft, um sicherzustellen, dass die Änderungen mit der bestehenden Codebase kompatibel sind und keine Fehler verursachen. Dieser Prozess ermöglicht es Teams, Probleme frühzeitig zu erkennen und zu beheben, was die Qualität der Software verbessert und die Zeit bis zur Auslieferung der Software reduziert.[6]

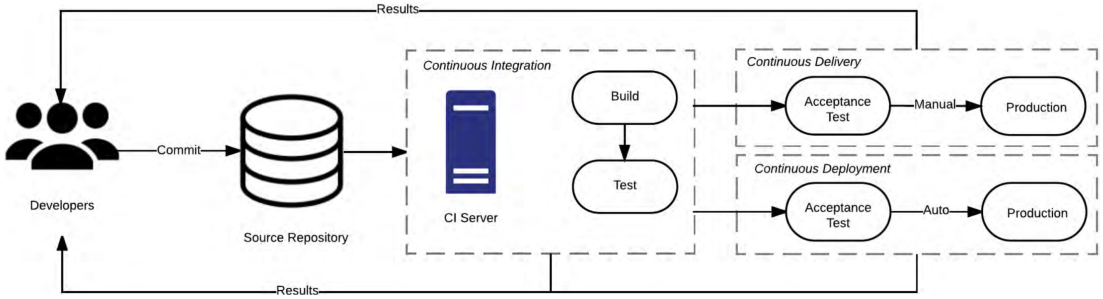
[2] Vgl. Shahin, Ali Babar und Zhu. „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices“. 2017, S. 3910–3911.
[3] Vgl. Fitzgerald und Stol. „Continuous software engineering: A roadmap and agenda“. 2017.
[4] Vgl. Beck, Beedle, Bennekum et al. *Manifesto for Agile Software Development.* 2001.
[5] Vgl. Gren und Lenberg. „Agility is responsiveness to change“. 2020.
[6] Vgl. Fowler. *Continuous Integration.* 2006.

Continuous Delivery

Continuous Delivery (CDE) erweitert das Konzept der Continuous Integration, indem es sicherstellt, dass die Software stetig in einem Zustand ist, der sicher in die Produktionsumgebung ausgerollt werden kann. Dies wird durch das Einführen von Integrationstests, die die Funktionalität der vollständigen Software inklusive aller Module testen, erreicht. Das Ziel von CDE ist es, den Prozess der Softwareauslieferung zu beschleunigen und zuverlässiger zu machen, indem menschliche Fehler minimiert und schnelles Feedback über Probleme in der Produktionsumgebung ermöglicht wird.[7]

Continuous Deployment

Continuous Deployment (CD) ist der nächste Schritt nach Continuous Delivery. Bei CD wird jede Änderung, die den automatisierten Testprozess besteht, automatisch in die Produktionsumgebung eingespielt.[8] Dies bedeutet, dass neue Features und Updates mehrmals täglich an die Endbenutzer ausgeliefert werden können, was eine schnelle Reaktion auf Marktbedingungen und Kundenfeedback ermöglicht. Es ist jedoch zu beachten, dass CD eine hohe Reife der Entwicklungsprozesse und Testautomatisierung erfordert, um die Anzahl der in der Produktionsumgebung aufkommenden Fehler zu minimieren. Der Zusammenhang zwischen Continuous Integration, Delivery und Deployment wird in Abbildung 1 aufgezeigt.



Quelle: Übernommen von Shahin, Ali Babar und Zhu (2017)

Abbildung 1: Zusammenhang zwischen CI, CDE und CD

2.2 Begrifflichkeiten und Prinzipien von Continuous Integration

Um die Bedeutung und den Nutzen von Continuous Integration und des Continuous-Software-Engineering für die Softwareentwicklung nachvollziehen zu können, ist es hilfreich, einen Blick auf die historische Entwicklung und die grundlegenden Prinzipien der CI zu werfen. Ein Kernprinzip hinter Continuous Integration wurde bereits im Jahr 1991 von Grady Booch definiert. Hierbei werden Software-Releases nicht als ein großes Ereignis betrachtet, sondern regelmäßig durchgeführt, wobei die vollständige Software stetig größer wird.[9] Kent Beck popularisierte im Jahr 1998 die Disziplin des „Extreme Programming“, wobei großer Wert auf das frühe und regel-

[7] Vgl. Humble und Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.
[8] Vgl. Shahin, Ali Babar und Zhu. „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices“. 2017, S. 3911.
[9] Vgl. Booch. *Object oriented design with applications*. 1991.

mäßige Testen und Integrieren der entwickelten Komponenten einer Software gelegt wird. Beck behauptet hierbei, dass ein Feature, für welches es keine automatisierten Tests gibt, auch nicht funktioniert.[10] Im Jahr 2006 fasste Software-Entwickler Martin Fowler einige Bereiche dieser Methodiken in dem Artikel „Continuous Integration“ unter dem gleichnamigen Begriff zusammen. Fowler beschreibt CI als einen Prozess, bei dem Teammitglieder ihre Arbeit regelmäßig Integrieren, wobei Integration als der Build-Prozess, inklusive automatisierter Tests, für die vollständige Software mitsamt der erarbeiteten Änderungen zu verstehen ist.[11] Ein grundlegendes Ziel dieser Vorgehensweise ist, neben der frühzeitigen Erkennung von Fehlern im Quellcode durch automatisierte Tests und QA-Prüfungen, die Reduzierung der „Cycle Time“, welche die Zeitspanne von der Entwicklung eines Features bis zum Erhalten des Kundenfeedbacks nach dessen Auslieferung beschreibt.[12] In 2010 betonen Humble und Farley die Wichtigkeit von CI und behaupten, dass Software ohne Continuous Integration als defekt gilt, bis sie als funktionierend nachgewiesen wird, während mit CI Software mit jeder erfolgreich integrierten Änderung als funktionierend bewiesen wird.[13] Über Zeit hat sich CI als essenzielle Praktik in der Software-Welt etabliert. Im Jahr 2016 zeigte eine Analyse von 34.544 auf der Plattform GitHub verwalteten Open-Source-Projekten, dass 40% der Projekte CI einsetzen. Bei den populärsten 500 analysierten Projekten lag der Anteil bei 70%.[14] Da die erfolgreiche Implementierung von Continuous Integration auf der Kombination von verschiedenen Methoden zur Verwaltung von Softwareprojekten fußt, werden im Folgenden einige dieser Schlüsselaspekte aufgezeigt:

- **Regelmäßige Integration**

  Die namensgebende Methodik der CI ist das regelmäßige Integrieren von Software und damit Verbunden ist der Software-Release. In der traditionellen Softwareentwicklung wird der Release als ein einmaliges, großes Ereignis betrachtet. Als Integration wird der Prozess des Einbindens einer einzeln entwickelten Komponente in die bisher bestehende Gesamtheit einer Software bezeichnet, wobei der Release das Zusammenfinden aller Komponenten und das Ausführen des Build-Prozesses bis hin zur fertigen, ausführbaren und auslieferbaren Software beschreibt. In der Vergangenheit wurde bei einem Release jede Einzelkomponente der Software manuell integriert und getestet, wobei dies als eigene Phase in der Entwicklung einer Applikation galt. In einem CI-gestützten Projekt wird der Prozess des Software-Releases vollständig automatisiert, sodass jedes Teammitglied eine entwickelte Komponente schnell Integrieren und einen Software-Build erzeugen kann, was sich positiv auf die Cycle Time auswirkt. Fowler empfiehlt hierbei, dass ein entwickeltes Feature erst nach der vollständigen Integration durch einen erfolgreichen Build als fertig angesehen werden soll.[15] In der Regel finden das Bauen und Testen der Software in einer „Pipeline“ statt, welche die genutzten CI-Tools wie Test-Suites und Static-Code-Analysis ausführt.[16]

[10] Vgl. Beck. „Extreme programming: A humanistic discipline of software development“. 1998, S. 2–4.
[11] Vgl. Fowler. *Continuous Integration*. 2006.
[12] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2580.
[13] Vgl. Humble und Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010, S. 56.
[14] Vgl. Hilton, Tunnell, Huang et al. „Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects“. 2016, S. 428–429.
[15] Vgl. Fowler. *Continuous Integration*. 2006.
[16] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2571.

- **Automatisierte Tests**

  Neben der regelmäßigen Integrierung von Code gelten automatisierte Software-Tests und Quality-Checks als ein wichtiger Aspekt von CI. Hierbei werden oftmals Unit-Tests verwendet, welche eine einzelne Softwarekomponente auf ihre Funktion prüfen, abgekapselt von anderen Komponenten.[17] Wenn ein Test fehlschlägt, wird in der Regel die Pipeline unterbroche. Neben den typischerweise schnell durchlaufenden Unit-Tests, die aufgrund ihrer isolierten Natur spezifische Komponenten prüfen, können nach dem Build-Prozess umfassende End-To-End-Tests für die gesamte Software durchgeführt werden. Diese Systemübergreifenden Tests, welche das Zusammenspiel verschiedener Komponenten testen, erhöhen zwar die Dauer des Test-Prozesses, können aber wichtige Einsicht in die Qualität der Software gewähren.[18] Tests in einem automatisierten CI-Build können insgesamt dabei helfen, Fehler in eingeführtem Quellcode zu finden.[19] Oftmals kommen auch Static-Code-Analysis Tools zum Einsatz, um eine Pipeline bei Unstimmigkeiten in der Syntax des eingeführten Codes abzubrechen. Diese Programme können zusätzliche Qualitätsprüfungen und Coding-Standards in ein Projekt einführen.[20]

- **Reproduzierbarkeit**

  Ein weiterer zentraler Aspekt von Continuous Integration ist die Reproduzierbarkeit der CI-Pipeline. In einem CI-Umfeld werden der Build- und Testing-Prozess vollständig automatisiert und standardisiert, was bedeutet, dass diese unter den gleichen Bedingungen und mit den gleichen Ergebnissen wiederholt werden können. Dies ist von entscheidender Bedeutung, um sicherzustellen, dass die Software in verschiedenen Umgebungen (z.B. Entwicklung, Test, Produktion) konsistent funktioniert. Durch das Zentralisieren der Software und der CI-Tools in einer Versionierungs-Software, in Verbindung mit dem Automatisieren des Build- und Testing-Prozesses, wird das Wiederherstellen von früheren Ständen in der Entwicklung und das Nachvollziehen von Fehlern im Entwicklungsprozess vereinfacht.[21]

- **Transparenz**

  Transparenz in einem CI-Prozess bedeutet, dass alle Aspekte des Entwicklungsprozesses sichtbar und verständlich sind, sowohl innerhalb des Entwicklungsteams als auch für andere Stakeholder. Sie ermöglicht eine klare Sicht auf den aktuellen Stand des Projekts, einschließlich der Qualität des Codes, der Fortschritte und möglicher Probleme. Durch den Einsatz von CI und der Automatisierung des Build- und Test-Prozesses können Teammitglieder schnell Feedback über den Status von Änderungen an der Codebase erhalten. Wenn ein Problem auftritt, z.B. ein Test fehlschlägt oder ein Build abbricht, wird das Team sofort benachrichtigt, sodass das Problem schnell behoben werden kann. Dies reduziert die Zeit und den Aufwand, die benötigt werden, um Fehler zu finden und zu beheben, und verbessert die Qualität der Software. Darüber hinaus fördert das schnelle Feedback

[17] Vgl. Ahmad, Haleem und Beg. „Test Driven Development with Continuous Integration: A Literature Review“. 2013, S. 280.
[18] Vgl. Fowler. *Continuous Integration*. 2006.
[19] **benefits-and-challenges**.
[20] Vgl. Zampetti, Scalabrino, Oliveto et al. „How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines“. 2017, S. 338.
[21] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 74–75.

die Kommunikation und Zusammenarbeit im Team, da alle Mitglieder ständig über den Status des Projekts informiert sind.[22,23]

Die vorgestellten CI-Praktiken bauen auf unterschiedlichen Technologien und Konzepten auf, die für das Betreiben von Continuous Integration eine Rolle spielen. Um die Praktiken der Continuous Integration vollständig zu verstehen und effektiv umzusetzen, ist es unerlässlich, die zugrunde liegenden Technologien und Konzepte zu betrachten, die die Basis für die Implementierung von CI in einem Softwareprojekt bilden. Dazu gehören Themen wie Version-Control-Systems, Pipelines, Containerization, Software-Testing und Static-Code-Analysis. Im Folgenden werden einige dieser Themen untersucht, um ein umfassendes Verständnis der Mechanismen und Werkzeuge zu vermitteln, die für die erfolgreiche Anwendung von Continuous Integration erforderlich sind.

Version-Control-Systems

Das Verwalten verschiedener Versionsstände von Software wird durch sogenannte Version-Control-Systems (VCS) ermöglicht. Es gibt verschiedene Implementierungen von VCS, namentlich unter anderem Git, Mercurial, Subversion und weitere. Ein VCS speichert den gesamten Quellcode eines Projekts in einem eigenen „Repository“.[24] Innerhalb desselben Repositories können verschiedene Versions-Historien in sogenannten „Branches“ gespeichert werden. Der aktuelle Stand eines Branches kann dabei kopiert werden und daran vorgenommene Revisionen können anschließend wieder in einen Branch des Repositories integriert werden. Versionskontroll-Software ermöglicht es außerdem, verschiedene Versionen in Form von Branches zusammenzuführen. Der Prozess des Zusammenführens verschiedener Branches wird hierbei als „Merge“ bezeichnet. Die Nutzung eines VCS in welchem verschiedene Branches angelegt werden können, ermöglicht Teammitgliedern die gleichzeitige Arbeit an einer einzelnen Codebase ohne sich dabei gegenseitig zu beeinflussen.[25] Fowler setzt für die Nutzung von CI das Führen eines Source-Repositories voraus, welches die Projekt-Software, inklusive aller dazugehörigen Build- und Testing-Konfigurationen, verwaltet.[26]

Pipelines

Um die in einer Versionskontroll-Software eingeführten Code-Änderungen automatisiert bauen, testen und ausliefern zu können, wird eine ausführende Umgebung für diese Prozesse benötigt. In einem CI-Prozess werden hierfür Pipelines eingesetzt, welche zum Ausführen von Befehlen und Prozessen zur automatischen Integration und dem Testen von Code genutzt werden.[27] Pipelines spielen durch das kontinuierliche Integrieren von entwickeltem Code eine integrale Rolle bei der Reduzierung der Cycle Time[28] und sind somit wichtig zum Erreichen des Ziels Z_1 der Arbeit.

[22] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2573.
[23] Vgl. Elazhary, Storey, Ernst et al. „ADEPT: A Socio-Technical Theory of Continuous Integration“. 2021, S. 26–27.
[24] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 381–385.
[25] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 388–390.
[26] Vgl. Fowler. *Continuous Integration*. 2006.
[27] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2571.
[28] Vgl. Fowler. *Continuous Integration*. 2006.

Eine Pipeline in einem CI-Prozess wird in der Regel bei jeder Änderung am Quellcode des Source-Repositories gestartet.[29]

Containerization

Um die zum Bauen, Testen und Ausliefern von Software in einem CI-Projekt benötigten Pipelines nutzen zu können, wird eine isolierte und reproduzierbare ausführende Umgebung benötigt. In der Vergangenheit wurden für solche isolierten Umgebungen hauptsächlich „Virtual Machines“ (VMs) verwendet. VMs nutzen hierbei einen „Hypervisor“, um die Hardware eines physischen Computers emulieren zu können, sodass mehrere Betriebssysteminstanzen und die darin verwalteten Applikationen gleichzeitig auf einem einzigen System ausführbar sind.[30] Containerization ist eine modernere Technologie, die einen Performance-Zuwachs gegenüber VMs bieten kann.[31] Die hierbei genutzten „Container“ sind eigenständige, ausführbare Pakete, die alles enthalten, was für den Betrieb einer Anwendung erforderlich ist, einschließlich Code, Laufzeitumgebung, Bibliotheken und Systemtools. Während VMs zur Ausführung einen Hypervisor und vollständige Betriebssysteminstanzen benötigen, teilen sich Container die Ressourcen eines einzelnen Betriebssystems und isolieren so die Anwendung von der zugrunde liegenden Infrastruktur.[32] Containerization erleichtert also das Vereinheitlichen verschiedener Umgebungen in CI-Projekten, indem es Abhängigkeiten und Konfigurationen in einem Container kapselt.[33] Dies bietet Entwicklern und Testern eine erhöhte Effizienz und Flexibilität, da sie sich darauf verlassen können, dass die Software in jeder Umgebung identisch funktioniert.

Software-Testing

Wenn eine ausführende Umgebung für das Bauen und Testen von Software besteht, können Software-Tests in den CI-Prozess eingeführt werden. Durch manuelle und automatisierte Tests kann ein Software-Produkt auf verschiedene Funktionsbereiche überprüft werden. Da manuelle Tests einen hohen Zeitaufwand darstellen können, wird Testautomatisierung in der agilen Softwareentwicklung als wichtige Aktivität angesehen. Besonders bei repetitiven Aufgaben zum Reproduzieren vordefinierter Systemfunktionalitäten wirken automatisierte Tests effizienzsteigernd.[34] Sie müssen hierbei nur einmal angelegt werden und können anschließend beliebig oft und ohne weitere Aufwände erneut ausgeführt werden. In CI-Projekten werden automatisierte Tests oftmals eingesetzt, um Probleme bei der Einführung von Features mit vorher entwickelter Logik zu vermeiden und um Fehler im Entwicklungsprozess früher entdecken zu können. Häufiges, automatisiertes Testing in der CI kann sich positiv auf die Qualität der entwickelten Software auswirken.[35, 36]

[29] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 3–4.
[30] Vgl. Combe, Martin und Di Pietro. „To Docker or Not to Docker: A Security Perspective“. 2016, S. 54–55.
[31] Vgl. Spoiala, Calinciuc, Turcu et al. „Performance comparison of a WebRTC server on Docker versus virtual machine“. 2016.
[32] Vgl. Combe, Martin und Di Pietro. „To Docker or Not to Docker: A Security Perspective“. 2016, S. 55–57.
[33] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2576.
[34] Vgl. Collins, Dias-Neto und Lucena Jr. „Strategies for Agile Software Testing Automation: An Industrial Experience“. 2012, S. 440.
[35] Vgl. Ahmad, Haleem und Beg. „Test Driven Development with Continuous Integration: A Literature Review“. 2013, S. 281.
[36] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2570.

Nachfolgend werden verschiedene Arten von Tests in der Softwareentwicklung vorgestellt:[37]

- **Unit-Tests**

  Unit-Tests prüfen das verhalten von einzelnen Programm-Einheiten. Eine Programm-Einheit oder Prozedur stellt eine oder mehrere zusammenhängende Anweisungen in Form von Code dar, welche durch andere Teile der Software namentlich aufgerufen werden können. Durch Unit-Tests werden also die Implementierungsschritte eines Software-Programms einzeln geprüft.

- **Module-Tests**

  Ein Modul besteht aus einer Sammlung zusammengehöriger Programm-Einheiten die in einer Datei, einem Paket oder einer Klasse zusammengefasst sind. Module-Tests, auch Component-Tests genannt, zielen darauf ab, diese Module isoliert zu bewerten, im Hinblick auf die Interaktionen zwischen den Einheiten und ihren dazugehörigen Datenstrukturen. In der Praxis können Unit- und Module-Tests zusammengefasst als Prüfung der eigentlichen Implementierung eines Programms betrachtet werden.[38]

- **Integration-Tests**

  Integration-Tests fokussieren sich auf die Integration verschiedener Module einer Software. Sie prüfen das Zusammenspiel von unterschiedlichen Komponenten des Programms und stellen so sicher, dass die Kommunikation zwischen verschiedenen Bereichen der Applikation korrekt verläuft. Integrations-Tests stützen sich dabei auf die Annahme, dass die Module selbst bereits vollständig funktionieren.

- **System-Tests**

  Ähnlich wie Integration-Tests, prüfen System-Tests mehrere Komponenten eines einzelnen Systems auf ihre Zusammenarbeit. System-Tests grenzen sich hierbei allerdings durch ihren Bezug auf vorher definierte Produktanforderungen ab. Sie prüfen ob ein Programm in gänze funktioniert, wobei der Erfolg der Tests aus der Erfüllung von vordefinierten, geschäftsseitigen Anforderungen an das Programm besteht. Diese Art von Test wird auch als „Functional Test“ bezeichnet.[39]

- **Acceptance-Tests**

  Acceptance-Tests werden manuell ausgeführt. Sie prüfen, ob die Bedürfnisse des Projekt-Kunden durch die fertiggestellte Software abgedeckt werden. Hierbei werden also Tests aus der Sicht der Kunden oder vom Kunden selbst durchgeführt, welche fehlschlagen wenn dessen Anforderungen an das Programm nicht erfüllt werden.

Um etwaige Abhängigkeiten die zur Ausführung von Code in einem Test benötigt werden zu umgehen, werden „Mocks“ eingesetzt. Ein Mock stellt eine Nachahmung der Vorgehensweise einer Abhängigkeit dar, so kann zum Beispiel das Verhalten einer Datenbank oder eines anderen System-Moduls simuliert werden, ohne diese als Service im Testing-Prozess zu benötigen.[40]

[37] Vgl. Ammann und Offutt. *Introduction to Software Testing.* 2016, S. 23–24.
[38] Vgl. Collins, Dias-Neto und Lucena Jr. „Strategies for Agile Software Testing Automation: An Industrial Experience“. 2012, S. 441.
[39] Vgl. Collins, Dias-Neto und Lucena Jr. „Strategies for Agile Software Testing Automation: An Industrial Experience“. 2012, S. 441.
[40] Vgl. Ammann und Offutt. *Introduction to Software Testing.* 2016, S. 299–300.

Zur Bewertung der Effektivität und Qualität von erstellten Software-Tests gibt es verschiedene Ansätze, ein wichtiger Indikator ist jedoch die Test-Abdeckung. Der Anteil von durch Tests abgedeckten Klassen und Funktionen im Kontrast zu der Anzahl an ungetesteten Komponenten der Software wird als Abdeckung oder „Coverage“ bezeichnet.[41] Neben Test-Coverage können noch weitere Metriken, wie Mutation-Tests zur Bestimmung der Qualität von Tests genutzt werden. Hierbei wird der zu testende Source-Code verändert, oder mutiert, und dieser abgeänderte Code mit einem bestehenden Test-Set geprüft. Wenn die mutierte Variante des Codes durch das Test-Set aufgedeckt wurde, indem der Test fehlschlägt, wird der mutierte Code, auch Mutant genannt, als eliminiert betrachtet. Durch Mutation-Testing können Schwachstellen in Software-Tests gefunden werden und die Qualität der bestehenden Tests eines Projekts gemessen werden. Die Test-Software erzeugt dazu eine Metrik namens „Mutation Adequacy Score“.[42]

Static-Code-Analysis

Neben automatisiertem Testing ist das statische Analysieren des Codes ein weiterer wichtiger Aspekt für die Qualitätskontrolle von Software in einer CI-Umgebung. Wo Tests nur die Ergebnisse des Ausführens von Komponenten und Funktionen messen, können statische Code-Analysis-Tools die Struktur des Codes und die Einhaltung von vorgegebenen Standards bewerten. Eine laufende CI-Pipeline kann so bei der Einführung von Code, welcher nicht dem vorgegebenen Coding-Standard entspricht, abgebrochen werden. Dies zwingt die Entwickler eines Projekts dazu, eine einheitliche Code-Struktur zu verwenden und kann das Einführen von Fehlern verringern. Außerdem können Static-Code-Analysis-Tools durch Warnungen verhindern, dass unoptimierter Code in die Produktionsumgebung gerät.[43]

2.3 Übersicht über die Shopware-Platform

Shopware wurde als Online-Shop-Software im Jahr 2000 durch Stefan Hamann ins Leben gerufen[44] und bietet heute in ihrer aktuellen Major-Version 6 eine moderne E-Commerce-Plattform auf Basis des PHP-Frameworks „Symfony“. Das Symfony-Framework wird neben Shopware noch von anderen PHP-Basierten Projekten wie dem CMS „Drupal“, dem Shop-System „Magento“ und einigen weiteren Programmen[45] als Grundlage genutzt und bildet somit ein erprobtes Fundament für die Shopware-Plattform. Shopware selbst ist nach der Installation bereits voll funktionsfähig und kann mit einem Backend und optional mit einem Frontend oder für das Konsumieren der mitgelieferten API eingerichtet werden. Ein Application Programming Interface (API) ist hierbei eine Schnittstelle, welche die standardisierte Kommunikation zwischen Programmen ermöglicht. Die Software kann auf verschiedenen Plattformen gehostet werden, darunter Linux-Server und containerisierte Umgebungen. Darüber hinaus bietet Shopware als Unternehmen auch eine eigene Hosting-Lösung an, die speziell auf die Anforderungen der Software zugeschnitten ist. Die Plattform kann sowohl im Einzelbetrieb als auch als Cluster genutzt werden, um eine hohe Verfügbar-

[41] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 132–133.
[42] Vgl. Jia und Harman. „An Analysis and Survey of the Development of Mutation Testing“. 2011, S. 649–652.
[43] Vgl. Zampetti, Scalabrino, Oliveto et al. „How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines“. 2017.
[44] Vgl. Shopware. *The story behind Shopware AG*. 2023.
[45] Vgl. Symfony SAS. *Projects using Symfony - Popular PHP projects using Symfony components or based on the Symfony framework*. 2023.

keit und Skalierbarkeit zu gewährleisten. Nachfolgend wird der Aufbau des Symfony-Frameworks dargestellt und die Architektur der darauf basierenden Shopware-Platform aufgezeigt.

Symfony-Framework

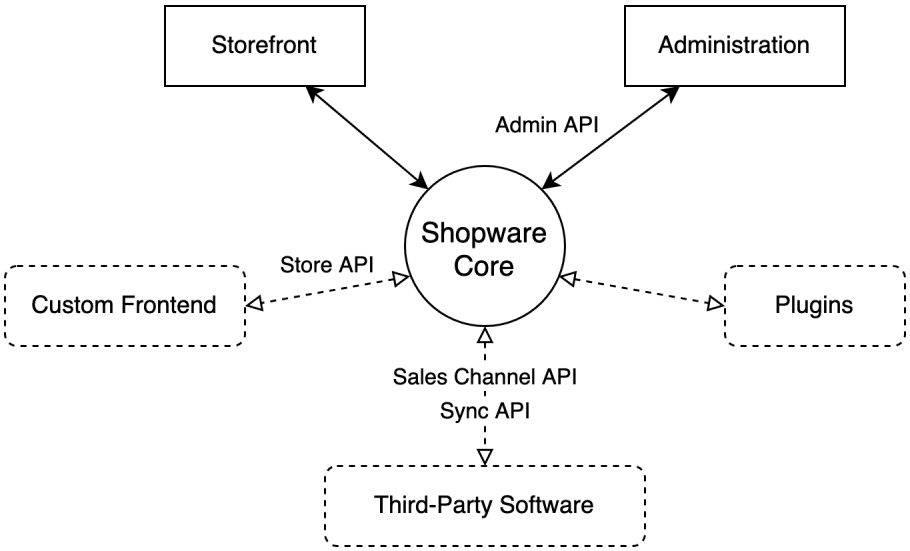
Symfony ist ein im Jahr 2005 von Fabien Potencier entwickeltes Full-Stack-Framework. Das PHP-basierte Projekt besteht aus verschiedenen Einzelkomponenten, welche unabhängig voneinander verwendet werden können, somit ist es sehr flexibel. Gleichzeitig bietet Symfony eine Reihe von Konventionen und Best Practises für das Erstellen und Nutzen von Komponenten, welche die Entwicklung von Anwendungen erleichtern und beschleunigen. Das Framework ist weit verbreitet und stellt eine robuste Grundlage für die Entwicklung umfangreicher Softwareprojekte dar. Es beinhaltet essenzielle Funktionen, wie Dependency Injection (DI), Profiling, Object-Relational Mapping (ORM), Session Handling, Routing, Formularverwaltung und eine Template-Engine. Durch den Einsatz des Paket-Managers „Composer“ können diese Grundfunktionen erweitert und zusätzliche Features in ein Projekt importiert werden. Symfony bietet eine Reihe von Konsolenbefehlen, die das Erstellen von eigenen Komponenten, die Durchführung von Datenbankmigrationen, das Ausführen von Tests und vieles mehr erleichtern. Die umfangreiche Dokumentation und die aktive Entwicklercommunity des Frameworks bieten einen hervorragenden Support für Entwickler. Zudem gewährleistet Symfony eine Langzeitunterstützung für jede Hauptversion mit einem Support-Zeitraum von drei Jahren, was die Zuverlässigkeit und Stabilität des Frameworks unterstreicht.[46]

Architektur der Shopware-Platform

Shopware 6 bietet eine modulare Architektur. Die Plattform besteht aus drei Kernkomponenten, dem Shopware-Core, der Administrations-Oberfläche und der Storefront. Der Core bildet hierbei die grundlegenden Shop-Funktionen und Ressourcen, und stellt für diese verschiedene Schnittstellen zur Nutzung bereit. Die Verwaltung der die Shop-Daten, Produkte und weiterer Betriebsfunktionen wird mit der Administrations-Oberfläche vorgenommen. Diese bildet eine eigene Komponente und kommuniziert mit dem Shopware-Core über die Admin-API. Letztlich wird das Frontend durch die Shopware-Storefront dargestellt, welche mithilfe von Symfonys Template-Engine „Twig“ und der direkten Anbindung an den Shopware-Core das Aufrufen von Produkten, Authentifizieren von Usern und Durchführen von Zahlungen ermöglicht. Die Storefront ist eine Komponente, welche optional auch deaktiviert und auf Wunsch durch die Nutzung der Store-API mit einer eigenen Applikation ersetzt werden kann. Im Standardumfang liefert die Shopware-Platform alle drei Hauptkomponenten aus. Diese stehen als Mono-Repository mit dem Namen `shopware/platform` auf GitHub zur Verfügung.[47] Neben den drei Hauptkomponenten bietet die Shopware-Platform verschiedene Anbindungsmöglichkeiten zur Anpassung des Shop-Systems. Plugins können direkt an den Core angebunden werden und ermöglichen die Erweiterung der Shopware-Logik, das Templating der Storefront oder Anpassungen an der Administration. Zusätzlich können die Sync-API und die Sales-Channel-API für die Anbindung externer Anwendungen verwendet werden, wie zum Beispiel Zahlungsanbieter oder E-Mail-Dienste.[48]

[46] Vgl. Engebreth und Sahu. „Introduction to Symfony“. 2023, S. 273–278.
[47] Vgl. Shopware. *Einblicke in die Core Architektur von Shopware 6*. 2019.
[48] Vgl. Pickware GmbH. *Shopware 6 – Ein Blick auf die neue Architektur*. 2019.

Eine grobe Zusammenfassung der Architektur kann aus Abbildung 2 entnommen werden. Hierbei werden die Kernkomponenten von Shopware aufgezeigt und optionale Services in gestrichelt dargestellt.



Quelle: Eigene Darstellung nach Pickware GmbH (2019)

Abbildung 2: Visalisierung der Shopware-Architektur

Für die Entwicklung der CI-Strategie wird der Fokus auf Shopware-Plugins gesetzt. Durch Plugins können der Shopware-Core, die Administrations-Oberfläche und die Storefront angepasst werden, wobei diese als abgekapselte Module entwickelt oder als Monolith zusammen mit dem Shopware-Projekt in einem gemeinsamen Repository verwaltet werden können. Shopwares Plugin-System baut auf dem Bundle-System von Symfony auf, um standardisierte, modulare Erweiterungen der Software zu ermöglichen, wobei das Bundle-System Funktionen wie Plugin-Lifecycle-Verwaltung bereitstellt. Symfony nutzt teils für eigene Core-Features das Bundle-System, um die einzelnen Teilbereiche des Frameworks modular zu gestalten.[49] Durch die solide Basis der Symfony-Bundles bieten Shopware-Plugins eine klare Struktur und eine hohe Erweiterbarkeit und fördern die Wiederverwendbarkeit von entwickelter Software für das Shop-System.

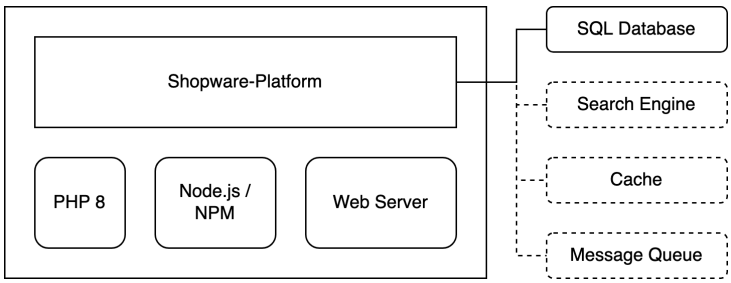
[49] Vgl. Shopware. *Plugins for Symfony developers*. 2023.

3 Analyse und Konzept

Im folgenden Kapitel werden verschiedene Techniken zur kontinuierlichen Integrierung von Code analysiert und eine Konzeption für Shopware-basierte Projekte erarbeitet. Zunächst wird die aktuelle Situation untersucht und eine Grundlage für die Einführung von Qualitätsanalyse-Tools und automatisiertem Testing gelegt. Anschließend werden verschiedene CI-Tools analysiert und für den Einsatz in der zu konzipierenden Pipeline bewertet. Zuletzt wird eine Konzeption durchgeführt, diese stützt sich dabei auf die zuvor festgelegte Ausgangssituation und die untersuchten Tools.

3.1 Analyse der Ausgangssituation

Die aktuelle Situation stellt sich wie folgt dar: In E-Commerce-Unternehmen gibt es in der Regel verschiedene Kunden, die mit unterschiedlichen Versionen von Shopware arbeiten. Diese Kunden sind oftmals bei verschiedenen Hosting-Anbietern untergebracht, was zu einer Vielfalt an technischen Umgebungen führt, in denen die Software betrieben wird. Darüber hinaus verwendet jeder Kunde eine individuelle Kombination aus Plugins und Eigenentwicklungen, die auf seine spezifischen Bedürfnisse zugeschnitten sind. Diese Diversität stellt eine Herausforderung dar, da sie eine Vielzahl von Variablen in die Entwicklung und Wartung der Software einbringt. Um eine generalisierte Strategie erstellen zu können, wird sich also zunächst auf die Gemeinsamkeiten von Shopware-Projekten konzentriert.



Quelle: Eigene Darstellung nach Shopware (2023)

Abbildung 3: Umgebung und Abhängigkeiten der Shopware-Platform

Shopware besteht aus einer Vielzahl von Verzeichnissen und Dateien, jede Version der Software und jedes Shopware-basierte Projekt unterscheiden sich voneinander. Ungeachtet von Projekt und Version gibt es jedoch einige Grundvoraussetzungen für das Ausführen von Shopware. Die Programmiersprache PHP mit einigen Extensions und der Paketmanager Composer müssen in der ausführenden Umgebung installiert sein. Zusätzlich wird die Javascript-Runtime „Node“ mit Paketmanager NPM benötigt, um das Frontend zu bauen. Die Software setzt außerdem eine Datenbank für den Betrieb des Shops und das Durchführen von End-to-End-Tests voraus, und benötigt einen Webserver um aufgerufen werden zu können. Im Live-Betrieb unterstützt die Plattform das optionale Anbinden einer Message-Queue und einer Search Engine, um viele gleichzeitige Zugriffe von Usern verwalten zu können und das Suchen von Produkten und Herstellern für Shop-Kunden zu ermöglichen. Zur weiteren Optimierung der Produktions-Umgebung

kann zudem ein Cache für Warenkorb-Inhalte und Nutzer-Sessions eingeführt werden.[50] In Abbildung 3 werden die Abhängigkeiten der Shopware-Platform und die verschiedenen Services für den Betrieb der Software aufgezeigt. Hierbei werden optionale Services in gestrichelt dargestellt. Um ein Projekt installieren und ausführen zu können, oder um Tests auf der Code-Base durchzuführen, wird also eine Umgebung vorausgesetzt, in der die benötigten Services und Tools installiert sind.

3.2 Auswertung von CI/CD-Tools

Um die Konzeption einer CI-Pipeline durchführen zu können, müssen zunächst die zur Umsetzung verfügbaren Tools erfasst und für den Einsatz in Kundenprojekten bewertet werden. Dabei spielen verschiedene Aspekte eine Rolle, wie die Kompatibilität mit dem bestehenden Technologie-Stack, die Skalierbarkeit und die Benutzerfreundlichkeit. Nachfolgend werden verschiedene Kategorien von Tools und Services, die für eine CI-Pipeline genutzt werden können, vorgestellt.

Version Control Systems

Um eine vollumfängliche CI-Strategie erstellen zu können, muss die Versionskontroll-Software inklusive dazugehöriger Services berücksichtigt werden. Für die Konzeption wurden einige Bekannte und erprobte Anbieter von Enterprise-VCS untersucht:

- GitHub
- GitLab
- Bitbucket
- SourceForge

Pipeline- und Test-Runners

Große Enterprise-VCS-Anbieter wie GitHub, GitLab und Bitbucket haben bereits eine Pipeline-Lösung in ihrem System integriert und bieten Hardware-Ressourcen zum Ausführen der Pipelines an. Neben den eingebauten Pipeline-Services der Enterprise-Anbieter gibt es einige eigenständige Tools, welche Teils selbst gehostet werden müssen. Nachfolgend werden einige dieser CI/CD-Services im Hinblick auf den Einsatz in der zu entwickelnden Strategie untersucht:

- **GitHub Actions:**

  GitHub Actions ist die eingebaute CI/CD-Lösung von GitHub. Actions können in jedem Repository in einem bestimmten Verzeichnis in Form von Konfigurations-Dateien im YAML-Format angelegt werden. Durch Actions kann an verschiedenen Stellen im Nutzungszyklus des VCS angeknüpft werden, um zum Beispiel nach einem Push oder bei Merge-Prozessen verschiedene Aktionen mit dem eingeführten Code durchzuführen. An dieser Stelle können dann je Branch verschiedene Aktionen definiert werden, wie zum Beispiel das Durchführen von Build-Prozessen, Tests und des Software-Deployments. Beim

[50] Vgl. Shopware. *Requirements*. 2023.

Merging-Prozess von verschiedenen Branches kann so zum Beispiel das erfolgreiche Durchlaufen der Pipeline vorausgesetzt werden, um fehlerhaften oder ungetesteten Code im Ziel-Branch zu vermeiden. Die für das Ausführen der Pipeline benötigten Umgebungsvariablen, Passwörter und andere vertrauliche Daten können dabei in einer von GitHub zur Verfügung gestellten, sicheren Datenbank verwaltet werden.[51]

- **GitLab CI/CD:**

  Ähnlich wie GitHub bietet auch GitLab eine eingebaute Pipeline-Lösung für das Ausführen von CI- und CD-Prozessen. GitLab CI/CD ist vollständig in GitLab integriert und verwendet eine Konfigurations-Datei innerhalb des Repositories, um die Phasen des CI/CD-Prozesses zu definieren. Es bietet eine breite Palette von Funktionen, einschließlich der Parallelisierung von Tests, Pipelines für Merge Requests und das Verwalten von Umgebungsvariablen. GitLab CI/CD kann auf eigenen Servern oder in der Cloud ausgeführt werden, und sowohl für bei GitLab, als auch für extern gehostete Repositorys verwendet werden, was es zu einer flexiblen Lösung macht.[52]

- **Bitbucket Pipelines:**

  Bitbucket bietet, ähnlich wie GitHub und GitLab, eine integrierte CI/CD-Lösung. Bitbucket Pipelines ermöglicht die Automatisierung von Build-, Test-/QA- und Deployment-Prozessen und ist dabei direkt an das Repository angebunden. Der Service verwendet ebenfalls eine YAML-Datei zur Konfiguration der Pipelines und bietet eine Vielzahl von vordefinierten Umgebungen und Integrationen mit anderen Tools. Wie GitLab, unterstützt auch Bitbucket Pipelines das parallele Durchführen von Tests. Pipelines können sowohl innerhalb von Bitbucket, als auch in der selbst-gehosteten Version genutzt werden.[53]

- **Jenkins:**

  Jenkins ist ein Open-Source-Tool, das für Continuous Integration entwickelt wurde, aber auch Continuous Delivery unterstützt. Es bietet eine große Anzahl von Plugins und Erweiterungen, die es zu einer flexiblen Lösung für CI/CD-Pipelines machen. Jenkins unterstützt das Verteilen von Arbeitslast auf verschiedene Systeme und kann auf einer Vielzahl von Plattformen und Umgebungen ausgeführt werden. Der Service kann durch Plugins angepasst und erweitert werden, um eigene Build-, Testing- und QA-Tools einführen zu können. Jenkins muss selbst gehostet werden, was eine größere Kontrolle über die Ausführungsumgebung ermöglicht, aber auch einen höheren Wartungsaufwand mit sich bringt.[54]

- **Travis CI:**

  Travis CI ist ein Cloud-basierter CI/CD-Service, der sich besonders gut für Projekte eignet, die auf GitHub gehostet werden. Es bietet eine einfache Einrichtung und Konfiguration durch eine YAML-Datei im Repository und unterstützt eine Vielzahl von Programmiersprachen und Tools. Der Service ermöglicht das gleichzeitige Testen in verschiedenen Umgebungen und die Verwaltung von Umgebungsvariablen und Secrets, sowie das Erweitern

[51] Vgl. GitHub, Inc. *Features · GitHub Actions · GitHub*. 2023.
[52] Vgl. GitLab B.V. *GitLab CI/CD | GitLab*. 2023.
[53] Vgl. Atlassian Corporation. *Bitbucket Pipelines - Continuous Delivery | Bitbucket*. 2023.
[54] Vgl. Jenkins. *Jenkins - Build great things at any scale*. 2023.

der Plattform für die Integration externer Applikationen. Travis CI bietet sowohl kostenlose Pläne für Open-Source-Projekte als auch kostenpflichtige Pläne für private Projekte.[55]

Testing-Tools

Shopware-Projekte gewinnen schnell an Umfang, der Aufwand für das manuelle Testen der einzelnen Entwicklungen steigt mit jedem eingeführten Feature. Software-Tests sind ein nützliches Tool für CI-Pipelines und helfen dabei, das aufwändige manuelle Testen der einzelkomponenten bei einem Software-Release zu minimieren. Nachfolgend werden verschiedene Arten von Software-Tests beschrieben und Tools für das Einbinden dieser Tests in der bestehenden Systemlandschaft untersucht:

- **Unit-Tests:**

  Unit-Tests verifizieren das Verhalten von kleinen Elementen eines Software-Systems, meist einzelner Klassen. Beim Durchführen von Unit-Tests wird das gewünschte Verhalten von Software-Komponenten definiert und diese anschließend ausgeführt um das Ergebnis mit dem zuvor definierten, erwarteten Verhalten zu vergleichen. Diese Tests haben keine äußerlichen Abhängigkeiten wie Datenbanken, da der jeweils getestete Code isoliert vom Rest des Software-Systems getestet wird. Um etwaige Abhängigkeiten die zur Ausführung des Codes benötigt werden zu umgehen, werden Mocks eingesetzt. Ein Mock stellt eine Nachahmung der Vorgehensweise einer Abhängigkeit dar, so kann zum Beispiel das Verhalten einer Datenbank simuliert werden, ohne diese als Service im Testing-Prozess zu benötigen. Oftmals wird der Anteil von durch Tests abgedeckten Klassen und Funktionen im Kontrast zu der Anzahl an ungetesteten Komponenten der Software gemessen, dieser Wert wird als Coverage bezeichnet.[56] Nachfolgend werden Tools für das Auführen von Unit-Tests in Shopware-basierten Projekten untersucht:

  - **PHPUnit:**

    PHPUnit ist ein weit verbreitetes Open-Source Unit-Testing-Framework, welches zum Erstellen von Unit-Tests in PHP-basierten Applikationen verwendet wird. Das Framework ermöglicht das Erstellen von Mocks und das Auswerten der Test-Coverage für das zu testende Projekt.[57]

  - **Jest:**

    Jest ist ein Javascript Testing-Framework und ermöglicht das Einbringen von Unit-Tests in JavaScript-basierten Applikationen. Ähnlich wie PHPUnit, unterstützt Jest das Erstellen von Mocks und bietet Funktionen zur Coverage an. Das Framework parallelisiert laufende Tests, um eine möglichst hohe Performance zu bieten.[58]

- **Functional Tests:**

[55] Vgl. Idera, Inc. *Travis CI - Build Faster.* 2023.
[56] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk.* 2007, S. 132–133.
[57] Vgl. Bergmann. *PHPUnit – The PHP Testing Framework.* 2023.
[58] Vgl. Meta Platforms, Inc. *Jest · Delightful JavaScript Testing.* 2023.

Funktionstests stellen eine Umgebung dar, in der das Software-System auf Interoperatibilität der einzelnen Komponenten und Services geprüft wird. Hierbei stehen alle Abhängigkeiten und Services für die Software zur Verfügung, um das Zusammespiel von verschiedenen Komponenten des Systems aus Sicht des Kunden zu testen. Diese Tests erfordern oft ein längeres Test-Setup und eine höhere Laufzeit.[59] Im Folgenden werden einige Testing-Tools für systemweite Tests untersucht:

- **Cypress:**

  Cypress ist ein Testing Framework für Webapplikationen. Es ermöglicht das automatisierte Testen von Benutzerinteraktionen innerhalb einer Webapplikation in einem realen Browser. Cypress ist besonders nützlich für das Testen von komplexen Webapplikationen und unterstützt sowohl Unit- als auch Integrations- und End-to-End-Tests. Cypress ermöglicht das Testen des Zusammenspiels des gesamten Projekts und kann zur automatisierung von Testfällen genutzt werden, bei denen normalerweise Nutzer-Interaktion erforderlich ist.[60]

- **Selenium:**

  Selenium ist ein weit verbreitetes Open-Source-Tool für automatisierte Tests. Ähnlich wie Cypress, ermöglicht es das Testen von Webanwendungen in verschiedenen Browsern und auf verschiedenen Plattformen. Selenium unterstützt eine Vielzahl von Programmiersprachen, darunter JavaScript, Java, C#, Python und Ruby. Es kann als Testing-Tool in einem Web-basierten Projekt oder als automatisierungs-Tool für andere Aufgaben verwendet werden.[61]

- **Mutation-Tests:**

  Mutation Testing ist eine Testtechnik, bei der die Effektivität eines existierenden Testsets bestimmt werden kann. Hierbei wird der zu testende Source-Code verändert, oder mutiert, und dieser abgeänderte Code mit einem bestehenden Test-Set geprüft. Wenn die mutierte Variante des Codes durch das Test-Set aufgedeckt wurde, indem der Test fehlschlägt, wird der mutierte Code, auch Mutant genannt, als eliminiert betrachtet. Durch Mutation Tests können Schwachstellen in Software-Tests gefunden werden und die Qualität der bestehenden Tests eines Tests gemessen werden. Zur Messung der Effektivität eines Test-Sets wird der „Mutation Adequacy Score“ oder „Mutation Score“ (MS) genutzt. Schlägt der Test eines mutierten Quellcodes nicht fehl, wirkt sicht das negativ auf den MS aus:

  $$MS = \frac{Anzahl\ eliminierter\ Mutanten}{Anzahl\ Mutanten}$$

  Das Ziel ist hierbei das Abdecken möglichst aller Mutanten durch ein Test-Set T, sodass dessen Mutation Score $MS = 1$ entspricht.[62]

  - **Infection:**

[59] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 136–138.
[60] Vgl. Cypress.io, Inc. *JavaScript Component Testing and E2E Testing Framework | Cypress*. 2023.
[61] Vgl. Software Freedom Conservancy. *Selenium*. 2023.
[62] Vgl. Jia und Harman. „An Analysis and Survey of the Development of Mutation Testing“. 2011, S. 649–652.

Infection ist eine Mutation-Testing-Library für PHP. Sie ändert den Quellcode automatisch und führt dann bestehende Tests erneut aus, um zu sehen, ob diese fehlschlagen. Wenn die Tests bestehen, obwohl der Code mutiert wurde, zeigt Infection diese Stellen im Code an, damit diese verbessert werden können. Infection nutzt einen eigenen „Mutation Score Indicator“ (MSI), die Formel zur Berechnung des MSI ist wie folgt definiert:

$$TotalDefeatedMutants = KilledCount + TimedOutCount + ErrorCount$$

$$MSI = \left(\frac{TotalDefeatedMutants}{TotalMutantsCount}\right) * 100$$

Neben dem MSI stellt Infection noch weitere Faktoren zum Bestimmen der Effektivität von Test-Sets bereit. Die Library unterstützt die Integration mit PHPUnit, Codeception und weiteren Testing-Tools.[63]

Static-Code-Analysis-Tools

Da statische Code-Analyse die Qualität der entwickelten Software verbessern können, ist diese hilfreich für das Erreichen des Ziels Z_2 der Arbeit. Im Folgenden werden einige dieser Tools für die in Shopware-Projekten verwendeten Programmiersprachen PHP und JavaScript gesammelt:

- **ESLint:**

  ESLint ist ein statisches Tool zur Analyse von JavaScript-Code. Das Tool ist sehr anpassbar und kann für das Entwickeln mit verschiedenen Frameworks und Libraries genutzt werden. Viele IDEs unterstützen ESLint und zeigen Analyse-Resultate bereits bei der Entwicklung im Editor an. Das Tool unterstützt außerdem das automatische Beheben von bestimmten Syntax-Fehlern mit einem eingebauten Kommandozeilenbefehl.[64]

- **Danger:**

  Danger ist ein statisches Analyse-Tool, welches direkt in VCS angebunden werden kann. Das Tool läuft während des CI-Prozesses und kann zur automatisierung von Code-Review-Aufgaben genutzt werden. Mit Danger können Warnungen und Fehler in der Pipeline geworfen werden, wenn Pull-Requests (PRs) zu groß werden, kein Changelog-Eintrag für Änderungen angelegt wurde, Lockfiles nicht up-to-date sind, der PR unzugewiesen ist und vieles mehr. Die offizielle Ausführung der Software umfasst die Programmiersprachen JavaScript, Ruby, Kotlin, Python und Swift, wobei es auch eine unoffizielle PHP-Implementierung gibt.[65, 66]

- **PHP_CodeSniffer:**

  PHP_CodeSniffer (PHPCS) ist ein Set aus zwei Kommandozeilen-Befehlen für das Analysieren von PHP- und JavaScript-Code und von Cascading Style Sheets (CSS) nach einem

[63] Vgl. Rafalko. *Introduction - Infection PHP*. 2023.
[64] Vgl. OpenJS Foundation. *Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter*. 2023.
[65] Vgl. Therox und Danger JS contributors. *Danger JS*. 2023.
[66] Vgl. Sayakci. *Danger PHP*. 2023.

gegebenen Coding-Standard und für das automatische Korrigieren von Abweichungen dieses Standards. Das Tool kann mit vorgegebenen und eigens erstellten oder angepassten Regelsätzen betrieben werden.[67]

- **PHP Mess Detector:**

  PHP Mess Detector (PHPMD) ist eine Software die nach vorgegebenen Problemen und Unstimmigkeiten in PHP-Code sucht. Das Tool kann zum verhindern des Einführens von überflüssigen Variablen und Methoden, unoptimiertem Code und möglichen Fehlern in die Zielumgebung genutzt werden. PHPMD kann ergänzend zu Danger und PHPCS zur statischen Analyse von PHP-Code verwendet werden.[68]

- **PHPStan:**

  PHPStan ist ein weiteres QA-Tool für das statische Analysieren von PHP-Code. Es konzentriert sich auf die Erkennung von Fehlern, die im laufenden Betrieb zu Problemen führen können, wie z.B. Aufrufe von nicht existierenden Methoden, ungenutzte Variablen, usw. PHPStan kann in den Entwicklungsprozess integriert werden, um die Codequalität kontinuierlich zu verbessern.[69]

- **Deptrac:**

  Deptrac ist ein statisches Code-Analyse-Tool, das dabei hilft, die Architektur eines PHP-Projekts zu verstehen und zu überprüfen. Es stellt sicher, dass die Abhängigkeiten zwischen den Modulen eines Projekts den definierten Architekturregeln entsprechen. Deptrac kann in den Entwicklungsprozess integriert werden, um die Architektur des Projekts kontinuierlich zu überwachen und zu verbessern.[70]

- **License checker:**

  License checker ist ein Tool, das die Lizenzen von Abhängigkeiten in einem Projekt überprüft. Es kann dabei helfen, Lizenzprobleme bei Plugins und Erweiterungen von Drittanbietern zu identifizieren und zu vermeiden. Das Tool ist in PHP geschrieben und konzentriert sich auf das Überprüfen von Composer-Abhängigkeiten.[71]

- **Symfony security checker:**

  Der Symfony Security Checker ist ein Befehlszeilentool, das überprüft, ob die in einem Symfony-Projekt verwendeten Abhängigkeiten bekannte Sicherheitslücken aufweisen. Das Tool kann zusammen mit der Symfony-CLI oder als eigenständiges Software installiert werden und hilft dabei, PHP-Projekte kontinuierlich zu überwachen und sicher zu halten.[72]

3.3 Konzeption der CI-Strategie

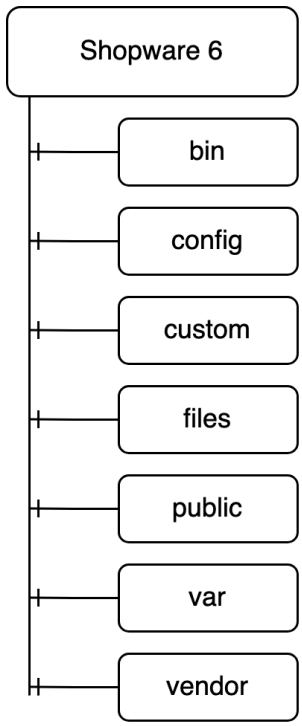
Im Folgenden wird die Konzeption der CI-Strategie für Kundenprojekte auf Basis der Shopware-Platform durchgeführt. Zunächst wird eine generelle Projekt-Struktur für die Strategie definiert,

[67] Vgl. Squiz Labs. *PHP_ CodeSniffer*. 2023.
[68] Vgl. Pichler. *PHPMD - PHP Mess Detector*. 2023.
[69] Vgl. Mirtes. *Find Bugs Without Writing Tests | PHPStan*. 2023.
[70] Vgl. QOSSMIC GmbH. *Deptrac*. 2023.
[71] Vgl. madewithlove BV. *CLI Licence checker for composer dependencies*. 2023.
[72] Vgl. Symfony SAS. *Installing & Setting up the Symfony Framework - Checking Security Vulnerabilities*. 2023.

wobei besonderer Wert auf die Kompatibilität mit unterschiedlichen Shopware-Projekten in verschiedenen Umgebungen gesetzt wird. Anschließend werden, anhand der im vorherigen Abschnitt untersuchten CI/CD-Tools, Pipelines für verschiedene Branching-Strukturen und Projekt-Ansätze erarbeitet. Zuletzt wird der Einsatz von Continuous Delivery und Deployment in der Strategie im Hinblick auf Shopware-Projekte bewertet.

Projekt-Struktur und Umgebung

Die Verzeichnis-Struktur von Projekten auf der Basis von Shopware 6 ist durch die Software vorgegeben. Führt man Shopware zum ersten Mal aus, werden 7 Verzeichnisse angelegt:



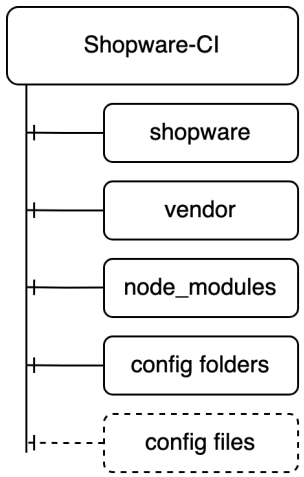
Quelle: Eigene Darstellung

Abbildung 4: Verzeichnis-Struktur eines Shopware-Projekts

- `bin`: Das `bin`-Verzeichnis enthält Skripte und andere ausführbare Dateien, die Aktionen in der Shopware-Platform auslösen. So kann zum Beispiel die Administration oder die Storefront neu gebaut, der Cache geleert oder ein neuer Backend-Benutzer angelegt werden.
- `config`: Im `config`-Verzeichnis, werden Konfigurationsdateien abgelegt, die das Verhalten von Shopware steuern. Hier können zum Beispiel Datenbank-, Cache-, Search-Engine- und Message-Queue-Konfigurationen gespeichert werden.
- `custom`: Benutzerdefinierte Plugins und Themes können im `custom`-Verzeichnis angelegt werden. Hierdurch kann die Funktionalität und das Aussehen von Shopware an die spezifischen Anforderungen eines Projekts angepasst werden.
- `files`: Hochgeladene Dateien und Medien, wie zum Beispiel Produktbilder, Downloads und andere Dateien, die über das Backend hochgeladen werden, werden im `files`-Verzeichnis gespeichert.
- `public`: Alle öffentlich zugänglichen Dateien der Anwendung, einschließlich statischer Ressourcen wie CSS, JavaScript, Schriftarten und generierte Bilder und Thumbnails, werden im `public`-Verzeichnis abgelegt.

- `var`: Temporäre Dateien, die während der Ausführung von Shopware generiert werden, wie Cache-Dateien, Log-Dateien und Sitzungsdaten, werden im `var`-Verzeichnis abgelegt.
- `vendor`: Das `vendor`-Verzeichnis beherbergt alle Abhängigkeiten der Shopware-Instanz und dessen Plugins. Diese Abhängigkeiten werden über den Paketmanager Composer verwaltet.

In Abbildung 4 wird eine Übersicht über die Verzeichnis-Struktur eines Shopware-Projekts gegeben. Um die zu konzipierende Strategie möglichst flexibel zu gestalten, macht es Sinn die Shopware-Installation mit all diesen Verzeichnissen in einen eigenen Ordner auszulagern. Dies hat zur Folge, dass die Dateien, die zum Ausführen und Testen des Shops benötigt werden, getrennt von den Dateien gelagert sind, die den Shop ausmachen. Dadurch kann ein beliebiges Shop-Projekt in den `shopware`-Ordner gelegt werden, wobei weiterhin die Möglichkeit zur Anpassung der ausführenden Umgebung und der Testing-Tools besteht. In der CI-Strategie soll neben dem `shopware`-Verzeichnis ein eigenes Composer-Projekt mit `vendor`-Ordner und verschiedene Verzeichnisse für die Tool-Konfigurationen und Auswertungen und Caches der Tests existieren. JavaScript-Tools und Abhängigkeiten werden über NPM verwaltet und befinden sich im `node_modules`-Verzeichnis. Die restlichen Konfigurations-Dateien für etwaige Tools und Services liegen dabei direkt im Hauptverzeichnis des Projekts.



Quelle: Eigene Darstellung

Abbildung 5: Geplante Verzeichnis-Struktur der CI-Strategie

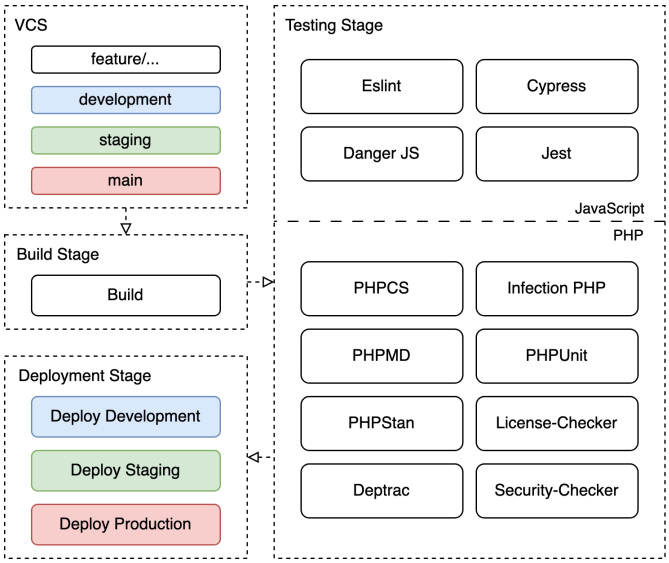
In Abbildung 5 kann die geplante Verzeichnis-Struktur für die CI-Strategie eingesehen werden, hierbei werden Ordner in durchgehenden und Dateien mit gestrichelten Linien dargestellt. Neben diesen Verzeichnissen und Dateien besteht die Struktur des Projekts auch aus einer Umgebung, in der die Shopware-Platform ausgeführt, getestet und analysiert werden kann. Eine Umgebung muss in diesem Fall in einer CI-Pipeline ausführbar sein und die Grundbedürfnisse der Shopware-Instanz bereitstellen. Dies beinhaltet die von Shopware genutzte PHP-Version mit PHP-Erweiterungen, Composer, Node, eine Webserver-Software und weitere Voraussetzungen. Shopware stellt für diese Anforderung ein Docker-Image bereit, welches für verschiedene Shopware-Versionen die unterschiedlichen PHP- und Composer-Versionen und Node, den Apache Webserver und alle anderen Abhängigkeiten der Plattform bereithält.[73] Docker und das darunterliegende Prinzip der Containerization sind Technologien, die Linux-Kernel-Features nutzen, um Anwendungen und ihre Abhängigkeiten in einem leichtgewichtigen, eigenständigen und isolierten Paket zu bündeln, das auf jeder Plattform ausgeführt werden kann, die Container unterstützt. Dies ermöglicht eine hohe Portabilität und Konsistenz über verschiedene Umgebungen hinweg und führt zu einem Performance-Zuwachs gegenüber der traditionellen Virtualisierung. Ein Docker-Image ist eine Vorlage für einen Docker-Container, der eine laufende Instanz eines Docker-Images ist. Ein wesentlicher Vorteil eines Docker-Images besteht darin, dass es sowohl in einer CI-Pipeline, als

[73] Vgl. Shopware. *shopware/development - Docker Image*. 2023.

auch in einer lokalen Entwicklungsumgebung oder in Produktion zum Ausführen der Software genutzt werden kann.[74]
Da die CI-Strategie auf dem Betreiben von Shopware in einer isolierten und reproduzierbaren Umgebung zum Ausführen von Testing- und QA-Tools fußt, wurde sich für die Nutzung des von Shopware gestellten Docker-Images in den Pipelines entschieden. Zunächst wird allerdings eine ausführende Umgebung für die CI-Pipelines benötigt. GitLab CI/CD kann zum Verwalten von Pipelines genutzt werden, um auf Änderungen in einem VCS zu reagieren. Der erhoffte Vorteil in der Nutzung des Services liegt in der Breite der Unterstützung, da es sich um eine große und populäre CI-Software handelt, und in der Möglichkeit, die Software selbst zu hosten und verschiedene VCS Anzubinden. Somit ist die Strategie nicht an eine bestimmte VCS-Software gebunden und kann ungeachtet der Plattform in Projekten integriert werden.

Pipelines

Da GitLab CI/CD mit verschiedenen VCS genutzt werden kann, wird sich in der weiteren Konzeption auf die Gestaltung anpassbarer Pipelines im Hinblick auf unterschiedliche Branching-Strukturen für verschiedene Shopware-Projekte konzentriert. Pipelines reagieren auf Status-Veränderungen in einer VCS-Software und können so an verschiedene Aktionen geknüpft werden. Zunächst soll eine grundlegende Pipeline definiert werden, die bei jedem Push in einen Branch ohne eine definierte Deployment-Umgebung läuft. Diese Pipeline durchläuft eine Build- und eine Testing-Stage. Im Build-Prozess werden die CI-Tools und die Abhängigkeiten der Shopware-Platform installiert. Der Testing-Prozess umfasst zunächst alle für die Strategie ausgewählten Test- und QA-Tools, inklusive lang-andauernder Tests durch End-to-End-Testing. Im späteren Verlauf der Entwicklung soll die Möglichkeit bestehen, diese Tests nur in Branches mit definierter Deployment-Umgebung durchführen zu lassen, falls dessen Laufzeit zu groß wird. Eine grobe Richtlinie für die maximale Dauer einer Pipeline ist die Zehn-Minuten-Marke, da viele Entwickler auf das erfolgreiche Durchlaufen der Pipeline warten, bevor sie mit der nächsten Aufgabe beginnen.[75] Eine möglichst kurze Durchlaufzeit von CI-Pipelines ist somit im Hinblick auf Ziel Z_1 der Arbeit für die Effizienz von Softwareentwicklungsteams relevant.
In der Regel gibt es in einem Shopware-Projekt der Firma best it bis zu drei Branches, für die ei-



Quelle: Eigene Darstellung

Abbildung 6: Visualisierung der konzipierten CI-Pipeline

[74] Vgl. Combe, Martin und Di Pietro. „To Docker or Not to Docker: A Security Perspective“. 2016.
[75] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 87–88.

ne Umgebung vorgesehen ist: `development`, `staging` und `main`. Die Development-Umgebung repräsentiert den Stand des `development`-Branches und ist die erste Server-Instanz die Entwickler zum Testen eines Features verwenden können. Diese Umgebung ist in der Regel stabil, kann aber Ausfälle verkraften, da weder der Endkunde, noch der Auftraggeber Zugriff darauf haben. Der `staging`-Branch stellt die Basis für die Staging-Umgebung bereit. Hierbei handelt es sich um eine stabile Umgebung, in die die erfolgreich getesteten Änderungen des `development`-Branches eingeführt werden, sodass der Auftraggeber des Shops diesen zustimmen kann, bevor sie den Endkunden im Produktions-System erreichen. Letztlich werden die vom Auftraggeber freigegebenen Änderungen gesammelt vom `staging`-Branch in den `main`-Branch überführt. Dieser Branch repräsentiert die Produktions-Umgebung, auf der die Endkunden und eigentlichen Shop-Nutzer Einkäufe tätigen können. Da zu Anfang eines Projekts oftmals noch keine Staging- und Development-Umgebung existiert, muss die CI-Strategie flexibel sein und an die Branching-Strategien verschiedener Projekten in unterschiedlichen Phasen angepasst werden können.

In Abbildung 6 wird eine konzipierte CI-Pipeline in Beachtung der verschiedenen möglichen Umgebungen eines Shopware-Projekts aufgezeigt. Hierbei werden verschiedene Arten von Branches in einem VCS dargestellt, welche zunächst die gleiche Aktion auslösen, den Build-Prozess. Dort werden die CI-Tools und die Abhängigkeiten von Shopware für die anschließende Nutzung in der Testing-Stage installiert. Die Testing-Stage wird nach erfolgreichem Durchlaufen der Build-Stage ausgeführt und ist in der Abbildung in zwei Teile separiert. Im oberen Teil werden die gewählten Testing- und QA-Tools für JavaScript-Code der Strategie aufgezeigt, während der untere Teil die Tools für PHP darstellt. Wenn alle Jobs der Testing-Stage erfolgreich durchgelaufen sind, wird, je nach Art des Ausgangs-Branches, ein Deployment-Prozess für die jeweilige Umgebung angestoßen. Die Deployment-Stage liefert dann den Code des Ausgangs-Branches an die Development-, Staging- oder Produktions-Umgebung aus. Die für die Pipelines ausgewählten Tools unterteilen sich in folgende Kategorien:

PHP:

- **Unit-Testing:**
  - PHPUnit
- **Mutation-Testing:**
  - Infection
- **Static-Code-Analysis:**
  - PHP_CodeSniffer
  - PHP Mess Detector
  - PHPStan
  - Deptrac
  - License-Checker
  - Security-Checker

JavaScript:

- **Unit-Testing:**
  - Jest
- **End-to-End-Testing:**
  - Cypress
- **Static-Code-Analysis:**
  - Eslint
  - Danger JS

Die Auswahl der genannten Tools basiert auf verschiedenen Faktoren. Bei der Auswahl der Tools für das Unit-Testing wurde auf PHPUnit und Jest zurückgegriffen, da diese weit verbreitet sind und eine umfangreiche Dokumentation sowie eine aktive Community bieten. Für das Mutation-Testing wurde Infection ausgewählt, da es eine direkte Integration mit PHPUnit bietet. Die ausgewählten Tools für die statische Code-Analyse, PHP_CodeSniffer, PHP Mess Detector, PHPStan, Deptrac, License-Checker und Security-Checker für PHP sowie Eslint und Danger JS für JavaScript, helfen dabei, den Code auf Einhaltung von Coding-Standards, potenzielle Fehler und Sicherheitsprobleme zu überprüfen. Cypress wurde für das End-to-End-Testing ausgewählt, da es eine einfach zu verwendende und leistungsstarke Lösung für das Testen von Webanwendungen bietet und bereits eine Schnittstelle zur Anbindung an Shopware existiert.[76] Die konzipierte CI-Pipeline ist flexibel und kann an die spezifischen Anforderungen und die Branching-Strategie eines Projekts angepasst werden. Die gewählten Tools bilden hierbei eine Grundlage für eine möglichst breite Abdeckung des zu testenden Quellcodes und der festgelegten Konventionen des Projekts, sind aber nicht gänzlich vorgeschrieben. Die Pipeline soll im weiteren Projektverlauf um weitere Tools ergänzt und angepasst werden können.

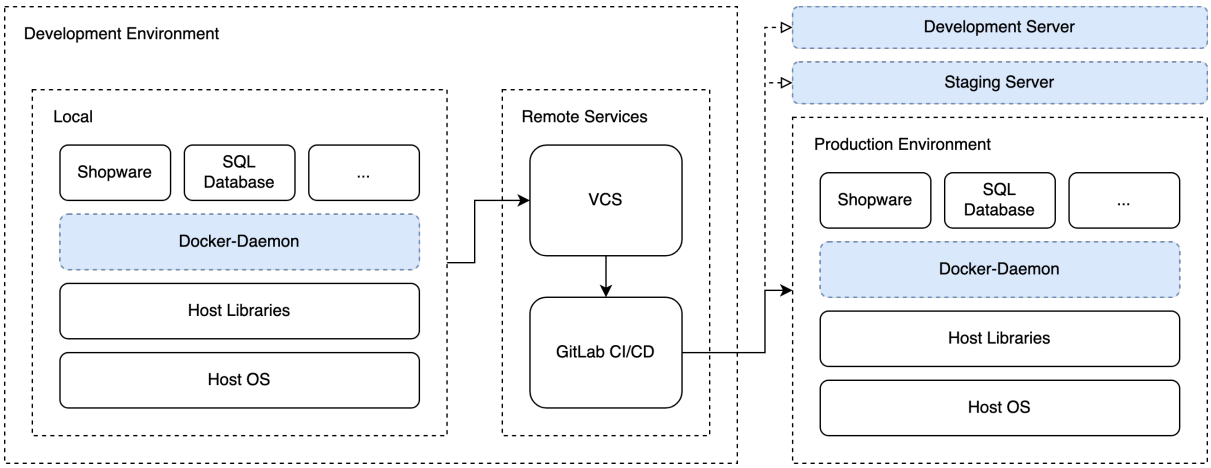
Durch den Einsatz der CI-Pipeline sollen Entwicklerteams effizienter arbeiten, die Codequalität erhöhen und die interne Kommunikation können. Wichtig für die Kommunikation ist hierbei, dass die CI-Tools und Prozesse kontinuierliches Feedback über den Stand des Codes gewähren. GitLab CI/CD kann zum Versenden von Nachrichten konfiguriert werden, welche bei Abbruch einer Pipeline auf Fehler im Quellcode aufmerksam machen. Durch die Weboberfläche wird hierbei Feedback zur Art des Fehlers und dessen Position in den zu testenden Dateien gegeben. Kontinuierliches Feedback bietet also die Möglichkeit, Entwicklern mehr Informationen über den aktuellen Status der Software mehrerer Projekte zukommen zu lassen. Diese Projektinformationen können dabei aggregieren werden, um mit der Zeit Trends innerhalb eines Projektes feststellen zu können.[77] Das kontinuierliche Feedback durch die CI-Pipeline an die Entwickler des Teams ist somit zum Erreichen des Ziels Z_3 der Strategie relevant, da es zu einer Steigerung der internen Kommunikation führen kann.

Continuous Delivery & Deployment

Neben der kontinuierlichen Prüfung des Quellcodes in der Testing-Stage und dem Erstellen einer Release-Fähigen Software durch den Build-Prozess ist die Deployment-Stage ein weiterer wichtiger Aspekt beim Betreiben einer CI-Strategie. Um die automatisiert gebaute und getestete Software auch direkt mit auszuliefern zu können, ist in der Strategie ein Deployment-Prozess angedacht. Hierbei wird die Software nach erfolgreichem Durchlaufen der Testing-Stage in der jeweils für den Ausgangs-Branch der Pipeline definierten Umgebung ausgerollt. Da die Kunden der best it GmbH verschiedene Arten von Umgebungen einsetzen, muss die Möglichkeit bestehen, den Prozess an die Bedürfnisse unterschiedlicher Projekte anzupassen. GitLab CI/CD bietet hierfür teilweise eingebaute Lösungen und Tools und kann außerdem benutzerdefinierte Skripte ausführen, womit allerlei Arten von Umgebungen abgedeckt werden können. Um traditionelle Linux-Server-Umgebungen als Hosting-Plattform zu unterstützen, soll das PHP-basierte Tool

[76] Vgl. Shopware. *E2E Platform Testsuite for Shopware 6*. 2023.
[77] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 203–211.

„Deployer“ in die Strategie mit einbezogen werden. Hiermit können Aktionen auf verschiedenen Servern durchgeführt werden, wie zum Beispiel das Übertragen der Dateien und Verzeichnisse der Software, das Starten und Stoppen von Services oder das Durchführen von Datenbankmigrationen.[78] Das Deployment in Container-Umgebungen wird durch GitLab CI/CD standardmäßig unterstützt. GitLab stellt hierfür einen eigenen Kubernetes-Agent bereit, welcher zum automatischen Ausliefern von Docker-Images an verteilte Container-Umgebungen genutzt werden kann.[79] Das Deployment kann hierbei, ungeachtet der Art der Umgebung, in GitLab manuell ausgeführt oder vollständig automatisiert werden. Da Änderungen am Produktions-System oftmals vorher durch den Kunden in einem designierten System getestet und abgenommen werden müssen, setzt die Strategie primär auf Continuous Delivery, ermöglicht aber optional auch Continuous Deployment durch das vollautomatisierte Ausrollen der Software. Wichtig ist hierbei, dass das Austauschen der laufenden mit der neuen, auszuliefernden Software so schnell und reibungslos wie möglich erfolgt, sodass die Nutzer des Shops keine Beeinträchtigungen beim Einkauf erfahren. Außerdem muss bei versehentlichem Ausrollen einer fehlerhaften Version die Möglichkeit bestehen, die veröffentlichte Software zu einem vorherigen, stabilen Stand zurückzuführen.[80]



Quelle: Eigene Darstellung nach Combe, Martin und Di Pietro (2016)

Abbildung 7: Visualisierung der Architektur der konzipierten Strategie

In Abbildung 7 wird die Architektur der konzipierten CI-Strategie grob visualisiert. Die Darstellung zeigt die Struktur der lokalen Entwicklungsumgebung so wie der späteren Deployment-Umgebungen in Zusammenhang mit dem VCS und GitLab CI/CD auf. Optionale Services und Umgebungen werden in Blau dargestellt, der Lebenszyklus von Integrationen in der Strategie wird durch die Pfeile zwischen den Teilbereichen verdeutlicht. Die lokale und die Deployment-Umgebungen basieren dabei auf den Hardware-Ressourcen, auf denen jeweils ein Betriebssystem (Host OS) installiert ist, welches das Nutzen von Host Libraries ermöglicht. Als Host Libraries werden in diesem Fall die installierten Programme und Services auf dem Betriebssystem bezeichnet, welche zum Beispiel für das Ausführen des Docker-Daemons und dessen Containern oder für das direkte Ausführen der Shopware-Services und dessen Abhängigkeiten genutzt werden.

[78] Vgl. Deployer. *Deployer - The deployment tool for PHP | Deployer*. 2023.
[79] Vgl. GitLab B.V. *GitLab Agent for Kubernetes*. 2023.
[80] Vgl. Humble und Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010, S. 259–261.

4 Entwicklung der CI-Strategie

Im folgenden wird die konzipierte CI-Strategie des vorherigen Kapitels praktisch umgesetzt. Zunächst wird eine Projektumgebung geschaffen, in der die geplanten CI-Tools und eine Shopware-Instanz zum Testen installiert werden. Daraufhin werden die installierten Tools für das Testen des lokal aufgesetzten Shops konfiguriert und angepasst, wobei der Code eines eigens dafür angelegten Beispiel-Plugins als Test-Grundlage verwendet wird. Anschließend werden verschiedene Deployment-Umgebungen für das Testen des Shops angelegt und für diese Umgebungen automatisierte Deployment-Konfigurationen zum Ausliefern durch die zu entwickelnden Pipelines erstellt. Zuletzt werden anhand der konfigurierten Tools und dem Beispiel-Shop die eigentlichen Pipelines zum automatisierten Bauen, Testen und Ausliefern von Shopware-Projekten erstellt.

4.1 Aufsetzen der Projektumgebung

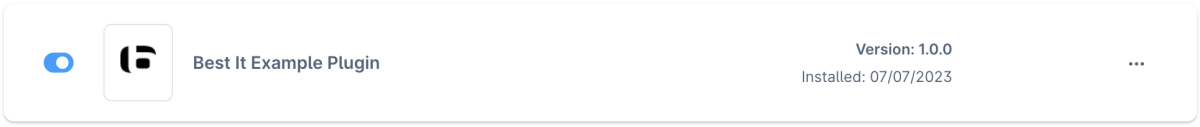
Für das Implementieren der konzipierten Strategie muss zunächst eine lokale Entwicklungsumgebung bestehen, in der ein Shopware-Projekt und die geplanten CI-Tools installiert werden können. Hierbei wurde sich für die Nutzung des von Shopware gestellten Docker-Containers entschieden, welcher die Abhängigkeiten für das Betreiben der Plattform gesammelt bereitstellt.[81] Das Ausführen der Shop-Software ohne Docker ist dabei weiterhin möglich, indem dessen Abhängigkeiten direkt auf dem Entwicklungssystem installiert werden. Zunächst wurde ein Ordner für die Shopware-Installation angelegt und Package-Konfigurationen für Composer und NPM erstellt. In der Composer-Konfigurations-Datei `composer.json` werden die Pakete für die PHP-basierten CI-Tools PHPUnit, Infection, PHPCS, PHPMD, PHPStan, Deptrac und License Checker verwaltet. Die Paket-Abhängigkeiten für die JavaScript-basierten Tools Eslint, Jest, Cypress und Danger JS werden in der Konfigurations-Datei `package.json` hinterlegt. In den beiden Konfigurations-Dateien wurden zunächst lediglich der Projekt-Titel, Autoren und die Version und die Art der Lizenz des Projekts definiert. Um die Software bei der Entwicklung zwischenspeichern und fehlerhafte Änderungen zurückrollen zu können, wurde an dieser Stelle bereits ein VCS eingeführt. Hierbei wurde sich für GitLab entschieden, da das zur Konzipierung der Strategie genutzte GitLab CI/CD bereits in der Versionsverwaltung integriert ist.

Bevor der Beispiel-Shop im Shopware-Verzeichnis aufgesetzt werden kann, muss zunächst die geplante Docker-Umgebung definiert werden. Docker bietet hierfür ein eingebautes Plugin namens „Docker Compose“, welches die Konfiguration von verschiedenen Containern in YAML-Dateien ermöglicht. Dies hat den Vorteil, dass verschiedene Services wie der Shop-Container selbst, die SQL-Datenbank für den Betrieb des Shops und weitere Dienste wie Message Queue, Search Engine und Caches gesammelt konfiguriert und mit einem einzigen Befehl angelegt werden können. Für den Betrieb des Shops wurde also zunächst die Datei `docker-compose.yml` erstellt und die Container-Konfigurationen für das Shopware-Image und eine MySQL-Instanz angelegt. Das Docker-Image `shopware/development` wird hierzu mit dem Tag `8.2-composer-2` hinterlegt, sodass der daraus resultierende Shopware-Container PHP in Version 8.2 und Composer in Version 2 zur Verfügung stellt. Um den Quellcode der zu betreibenden Shop-Software in den Container laden zu können, wird das Shopware-Verzeichnis als Volume in den Container eingebunden. Dies

[81] Vgl. Shopware. *shopware/development - Docker Image*. 2023.

ermöglicht, dass Änderungen am Quellcode, die auf dem Host-System vorgenommen werden, direkt im Container sichtbar sind und umgekehrt. Hierfür wird in der Datei `docker-compose.yml` unter dem Schlüssel `volumes` der Pfad zum lokalen Shopware-Verzeichnis und der entsprechende Pfad im Container angegeben, in diesem Fall der Order `/app`. Zusätzlich wird der Port `80` des Shopware-Containers auf den entsprechenden Port des Host-Systems weitergeleitet, wodurch der Zugriff auf den Shop über einen Webbrowser ermöglicht wird. Die MySQL-Instanz wird über das offizielle MySQL-Image `mysql` in der Version 8 bereitgestellt.[82] Ein Docker Volume wird für das Datenverzeichnis der MySQL-Instanz hinterlegt, um Datenverlust beim Wechseln der Version oder dem Herunterfahren des Containers zu vermeiden. Der Port `3306` des MySQL-Containers wird dabei auch auf den entsprechenden Port des Host-Systems weitergeleitet, um Zugriff auf die Datenbank zu ermöglichen. Die Zugriffs-Informationen und Einstellungen können in der Form von Umgebungsvariablen in der Compose-Konfiguration gespeichert werden. Beide Container sind dabei Teil des gleichen Netzwerks, welches in der Konfigurations-Datei unter dem Schlüssel `networks` definiert ist. Dies ermöglicht die Kommunikation zwischen den Containern über das interne Docker-Netzwerk. Mit dieser Konfiguration ist es möglich, eine lokale Entwicklungsumgebung mit Docker aufzusetzen, die den Anforderungen der meisten Shopware-Projekte gerecht wird. Die vollständige Container-Konfiguration in der Datei `docker-compose.yml` kann in Anhang A eingesehen werden.

Nach dem Aufsetzen der lokalen Entwicklungsumgebung und dem Bereitstellen der Abhängigkeiten von Shopware wurde der Beispiel-Shop im `shopware`-Verzeichnis installiert. Hierzu wurde der Paket-Manager Composer genutzt, welcher den Befehl `composer create-project` bereitstellt, um verschiedene Software-Projekte zu erstellen. Shopware pflegt eine eigene Vorlage zum Erstellen eines Shop-Projekts, welche unter dem Namen `shopware/production` durch den Create-Project-Befehl genutzt werden kann. Anschließend kann der Service bereits im Webbrowser erreicht werden und die lokale Installation des Shops durch den Web-basierten Shopware-Installer durchgeführt werden. In diesem Schritt werden die Stammdaten des Shops festgelegt und die Datenbankverbindung mit dem zuvor angelegten MySQL-Service hergestellt. Anhand der hier angegebenen Daten erstellt die Software eine `.env`-Datei, in der die Umgebungsvariablen für die Verbindung mit der lokalen Datenbank gespeichert werden.



Quelle: Eigene Darstellung

Abbildung 8: Beispiel-Plugin zur Konfiguration der CI-Tools

Um die geplante CI-Strategie umsetzen zu können, muss neben der lokalen Shopware-Umgebung auch eigener Quellcode existieren, welcher zur Konfiguration der Test- und Analyse-Tools genutzt werden kann. Da die Strategie das Erweitern der Shopware-Platform durch eigens entwickelte Plugins abdecken soll, wurde zunächst ein Beispiel-Plugin erstellt. Hierzu wurde der Plugin-Ordner `BestItExamplePlugin` im Verzeichnis `custom/static-plugins` der Shopware-

[82] Vgl. Oracle Corporation. *mysql - Official Image | Docker Hub*. 2023.

Installation angelegt. Die Konfiguration des Plugins erfolgt dabei über den Paket-Manager Composer, wobei eine eigene Konfigurations-Datei `composer.json` im Plugin-Ordner angelegt wird. Hier werden Grund-Informationen über das Plugin wie dessen Autor, Version, Abhängigkeiten und der Position des Quellcodes im Plugin-Ordner definiert:

```
{
  "name": "best-it/example-plugin",
  "version": "1.0.0",
  "authors": [
    {
      "name": "Frederik Bußmann",
      "email": "frederik.bussmann@bestit.de",
      "homepage": "https://www.bestit.de",
      "role": "Developer"
    }
  ],
  "type": "shopware-platform-plugin",
  "autoload": {
    "psr-4": {
      "BestIt\\ExamplePlugin\\": "src/"
    }
  },
  "extra": {
    "shopware-plugin-class": "BestIt\\ExamplePlugin\\BestItExamplePlugin",
    "label": {
      "de-DE": "Best It Beispiel-Plugin",
      "en-GB": "Best It Example Plugin"
    }
  }
}
```

Um sowohl testbaren PHP- als auch JavaScript-Code bereitzustellen, wurden jeweils eigene Code-Beispiele in beiden Sprachen angelegt. Für das Einführen und Konfigurieren der geplanten PHP-basierten CI-Tools wurden zunächst ein Beispiel-Controller und ein Beispiel-Service angelegt. Der Controller befindet sich im Unter-Verzeichnis `src/Controller` des Plugin-Ordners und wird durch das Öffnen der Route `http://localhost/example` im Webbrowser aufgerufen. Der Controller ruft hierbei den Beispiel-Service auf, welcher eine Berechnung durchführt und diese anschließend zurück an den Controller übergibt, sodass das Ergebnis im Frontend ausgegeben werden kann. Die Konfiguration der Controller und Services wird innerhalb eines Shopware-Plugins über die Datei `services.xml` im Ordner `src/Resources/config` des Plugin-Verzeichnisses gesteuert. Für das Konfigurieren der JavaScript-Tools der Strategie wurde innerhalb des Beispiel-Plugins eigener JavaScript-Code eingeführt. Die JavaScript-Erweiterung nutzt wie auch der zuvor definierte PHP-Code einen eigenen Beispiel-Service, welcher eine Berechnung durchführt, und diese an die aufrufende Skript-Instanz zurückgibt. Um das Skript inklusive des Services in die Shopware-Storefront einzubinden, wurde die Datei `main.js` im Verzeichnis `src/Resources/app/storefront/src` verwendet, welche für das Laden von JavaScript-Abhängigkeiten durch die Software automatisch referenziert wird. Das Beispiel-Plugin bietet somit eine Grundlage zum Analysieren von JavaScript- und PHP-Code und ermöglicht das Anlegen von Tests für die eingeführte Logik, wodurch die geplanten Testing-Tools konfiguriert und auf ihre Funktion geprüft werden können. In Abbildung 8 wird ein Screenshot aus dem Shopware-Backend aufgezeigt, in welchem das installierte und aktivierte Beispiel-Plugin zu sehen ist.

4.2 Konfiguration der Testing- und QA-Tools

Um die geplanten CI-Tools für das automatisierte Prüfen des im Projekt eingeführten PHP- und JavaScript-Codes installieren zu können, muss die Umgebung in der diese ausgeführt werden, an die Umgebung des Shops angeglichen werden. Hierbei könnte zunächst der Ordner mit den Paket-Manager- und Tool-Konfigurationen in den bestehenden Shopware-Container eingebunden werden. Da das Ausführen der geplanten Tools im späteren Pipeline-Umfeld jeweils isoliert in einem eigenen Container und parallel ausführt wird, wurde sich allerdings für eine eigene Container-Instanz als Tool-Umgebung entschieden. Um einen Container im Hauptverzeichnis des Projekts zu starten, welcher die Abhängigkeiten der Shopware-Platform bietet, kann weiterhin das von Shopware gestellte Development-Image lokal genutzt werden:

```
docker run -it -v ".:/app" shopware/development:8.2-composer-2 /bin/bash
```

Durch diesen Konsolenbefehl wird eine Instanz des Development-Images gestartet, welche das aktuelle Verzeichnis in den Ordner `/app` einbindet. Der Befehl nutzt dabei das Argument `-v`, um ein Volume vom Host-System zum Container zu schaffen, welches die spezifizierten Ordner-Pfade miteinander verbindet. Die Argumente `-i` und `-t` sorgen dabei für das interaktive Starten des Containers im TTY-Modus, wodurch ein Terminal bereitgestellt wird. Durch das Angeben der ausführbaren Datei `/bin/bash` am Ende des Befehls wird nach dem Starten des Containers ein Bash-Terminal ausgeführt. In diesem Terminal stehen anschließend die benötigten Abhängigkeiten für das Installieren und Ausführen der CI-Tools und Shopware bereit. Nachfolgend werden die geplanten CI-Tools für das automatisierte Prüfen des im Shopware-Projekt eingeführten PHP- und JavaScript-Codes installiert und konfiguriert:

PHPUnit

Das Testing-Framework PHPUnit wird als Entwicklungs-Abhängigkeit mit dem Paket-Namen `phpunit/phpunit` in der Datei `composer.json` installiert. Durch das Framework ist es möglich, eigene Unit-Tests in Form von PHP-Klassen und Funktionen in einem Shopware-Plugin anzulegen. Anschließend wurde für das Erstellen von Unit-Tests im Beispiel-Plugin die Verzeichnisse `Tests` und `Tests/Unit` angelegt, in welchem die PHP-Klassen der Tests abgelegt werden. Um die in diesem Ordner befindlichen Tests ausführen zu können, muss das Verzeichnis in der Composer-Konfigurations-Datei des Plugins referenziert werden:

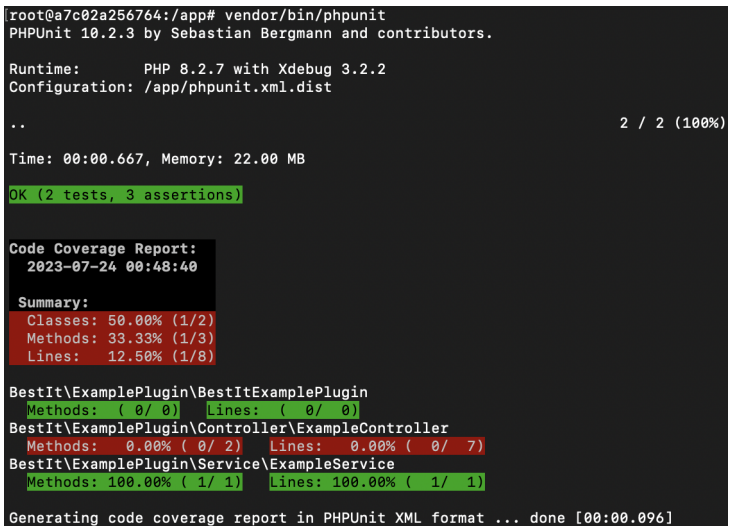
```
1 "autoload-dev": {
2   "psr-4": {
3     "BestIt\\ExamplePlugin\\Tests\\Unit": "Tests/Unit/"
4   }
5 }
```

Anschließend wurde für die Konfiguration des Test-Frameworks die Datei `phpunit.xml.dist` angelegt. In dieser Datei werden Einstellungen für die Position der Test-Dateien und auszuschließenden Dateien, die Erzeugung von Auswertungen zur Test-Abdeckung, dem Erstellen von Log-Dateien und dem Definieren des Cache-Verzeichnisses für durchgeführte Tests vorgenommen. Nach dem Erstellen der Datei wurden zum Prüfen der Konfiguration Unit-Tests für das Beispiel-Plugin kreiert. Die Namenskonvention von Unit-Tests wurde für die CI-Strategie als Na-

me der zu testenden Klasse mit dem Suffix „Test“ festgelegt. Zunächst wurde ein Unit-Test für die Datei `src/BestItExamplePlugin.php` angelegt, welche das Nutzen des Plugins in Shopware ermöglicht. Dieser Test prüft zunächst nur, ob die zu testende Klasse existiert und erfolgreich instanziiert werden kann:

```php
<?php

namespace BestIt\ExamplePlugin\Tests\Unit;

use BestIt\ExamplePlugin\BestItExamplePlugin;
use PHPUnit\Framework\TestCase;

class BestItExamplePluginTest extends TestCase
{
    public function testThatClassExistsAndCanBeInstanced(): void
    {
        static::assertTrue(class_exists(BestItExamplePlugin::class));

        static::assertInstanceOf(
            BestItExamplePlugin::class,
            new BestItExamplePlugin(true, __DIR__),
        );
    }
}
```

Der Test erweitert hierbei die von PHPUnit gegebene Klasse `TestCase` und nutzt zum Vergleichen der erwarteten und der tatsächlichen Ergebnisse von aufgerufenen Funktionen verschiedene statische Funktionen des Frameworks. Neben dem Test der Plugin-Klasse wurde ein weiterer Test für den Beispiel-Service angelegt. Die Tests können innerhalb des CI-Containers anschließend mit dem Befehl `vendor/bin/phpunit` aufgerufen werden. In Abbildung 9 wird ein Terminal aufgezeigt, in dem PHPUnit-Tests durchgeführt werden. Hierbei generiert das Framework auch Informationen zur Test-Abdeckung:



Quelle: Eigene Darstellung

Abbildung 9: Ergebnisse der PHPUnit-Tests inklusive Code-Coverage

Infection

Das Mutation-Testing-Framework Infection wird, wie auch PHPUnit, in der Konfigurations-Datei `composer.json` des Projekts installiert. Nach der Installation des Pakets durch Composer unter dem Namen `infection/infection` steht das Tool zum Ausführen von Mutation-Tests bereit. Das Programm verändert dabei den Quellcode des zu testenden Projekts und führt anschließend PHPUnit aus, wobei geprüft wird, welche Änderungen durch die gegebenen Tests unbemerkt blieben. Da für das Projekt bereits Tests angelegt wurden, musste Infection lediglich für die Prüfung der bestehenden Test-Suite konfiguriert werden. Hierzu wurden im Hauptverzeichnis des Projekts zwei Dateien angelegt: `infection.json` und `infection-bootstrap.php`.

4.3 Deployments und Umgebungen

4.4 Implementierung der Pipelines

5 Evaluierung

6 Schlussfolgerungen und Ausblick

A Anhang I: Lokale Docker-Konfiguration

Im folgenden wird die in Kapitel 4.1 aufgeführte Datei `docker-compose.yml` dargestellt, welche in der erarbeiteten CI-Strategie für das Verwalten lokaler Container-Services zum Ausführen von Shopware verwendet wird:

```yaml
version: "3"

services:
  shop:
    container_name: shop
    image: shopware/development:8.2-composer-2
    ports:
      - "80:80"
      - "8888:8888" # Administration watcher
      - "9999:9999" # Storefront watcher
    volumes:
      - ./shopware:/app
    networks:
      - web
    environment:
      - XDEBUG_ENABLED=1

  database:
    container_name: database
    image: mysql:8.0.33
    ports:
      - "3306:3306"
    volumes:
      - "database_volume:/var/lib/mysql"
    networks:
      - web
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_USER=shopware
      - MYSQL_PASSWORD=shopware
      - MYSQL_DATABASE=shopware

volumes:
  database_volume:
    driver: local

networks:
  web:
    external: false
```

Literaturverzeichnis

[1] eCommerceDB. *The Most Commonly Used Shop Software Among Online Shops in Germany - Shopware and Salesforce Share Rank No. 1.* eCommerceDB GmbH. 2023. URL: https://ecommercedb.com/insights/the-most-commonly-used-shop-softwares-among-online-shops-in-germany-shopware-and-salesforce-share-rank-no-1/4210 (aufgerufen am 22. 06. 2023).

[2] Mojtaba Shahin, Muhammad Ali Babar und Liming Zhu. „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5 (2017), S. 3909–3943. DOI: 10.1109/ACCESS.2017.2685629.

[3] Brian Fitzgerald und Klaas-Jan Stol. „Continuous software engineering: A roadmap and agenda". In: *The Journal of Systems and Software* 123 (2017), S. 176–189. DOI: 10.1016/j.jss.2015.06.063.

[4] Kent Beck, Mike Beedle, Arie van Bennekum et al. *Manifesto for Agile Software Development.* 2001. URL: https://agilemanifesto.org/ (aufgerufen am 30. 06. 2023).

[5] Lucas Gren und Per Lenberg. „Agility is responsiveness to change". In: *Proceedings of the Evaluation and Assessment in Software Engineering.* ACM, Apr. 2020. DOI: 10.1145/3383219.3383265.

[6] Martin Fowler. *Continuous Integration.* Fowler, Martin. 2006. URL: https://martinfowler.com/articles/continuousIntegration.html (aufgerufen am 26. 06. 2023).

[7] Jez Humble und David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Signature Series (Fowler). Pearson Education, 2010. ISBN: 978-0-321-67022-9.

[8] Grady Booch. *Object oriented design with applications.* 1. Aufl. Benjamin/Cummings Pub. Co, 1991. ISBN: 978-0-805-30091-8.

[9] Kent Beck. „Extreme programming: A humanistic discipline of software development". In: *Fundamental Approaches to Software Engineering.* Springer Berlin Heidelberg, 1998. DOI: 10.1007/bfb0053579.

[10] Omar Elazhary, Colin Werner, Ze Shi Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices". In: *IEEE Transactions on Software Engineering* 48.7 (Juli 2022), S. 2570–2583. DOI: 10.1109/tse.2021.3064953.

[11] Michael Hilton, Timothy Tunnell, Kai Huang et al. „Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects". In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* ASE '16. Singapore: Association for Computing Machinery, 2016, S. 426–437. ISBN: 978-1-450-33845-5. DOI: 10.1145/2970276.2970358.

[12] Sheikh Fahad Ahmad, Mohd Haleem und Mohd Beg. „Test Driven Development with Continuous Integration: A Literature Review". In: *International Journal of Computer Applications Technology and Research* 2 (Mai 2013), S. 281–285. DOI: 10.7753/IJCATR0203.1013.

[13] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto et al. „How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines“. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 2017, S. 334–344. DOI: 10.1109/MSR.2017.2.

[14] Paul M. Duvall, Steve Matyas und Andrew Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 1. Aufl. Addison-Wesley Signature Series. Pearson Education, 2007. ISBN: 978-0-321-63014-8.

[15] Omar Elazhary, Margaret-Anne Storey, Neil A. Ernst et al. „ADEPT: A Socio-Technical Theory of Continuous Integration“. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2021, S. 26–30. DOI: 10.1109/ICSE-NIER52604.2021.00014.

[16] Theo Combe, Antony Martin und Roberto Di Pietro. „To Docker or Not to Docker: A Security Perspective“. In: *IEEE Cloud Computing* 3.5 (2016), S. 54–62. DOI: 10.1109/MCC.2016.100.

[17] Cristian Constantin Spoiala, Alin Calinciuc, Corneliu Octavian Turcu et al. „Performance comparison of a WebRTC server on Docker versus virtual machine“. In: *2016 International Conference on Development and Application Systems (DAS)*. 2016, S. 295–298. DOI: 10.1109/DAAS.2016.7492590.

[18] Eliane Collins, Arilo Dias-Neto und Vicente F. de Lucena Jr. „Strategies for Agile Software Testing Automation: An Industrial Experience“. In: *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*. 2012, S. 440–445. DOI: 10.1109/COMPSACW.2012.84.

[19] Paul Ammann und Jeff Offutt. *Introduction to Software Testing*. 2. Aufl. Cambridge University Press, 2016. DOI: 10.1017/9781316771273.

[20] Yue Jia und Mark Harman. „An Analysis and Survey of the Development of Mutation Testing“. In: *IEEE Transactions on Software Engineering* 37.5 (2011), S. 649–678. DOI: 10.1109/TSE.2010.62.

[21] Shopware. *The story behind Shopware AG*. Shopware AG. 2023. URL: https://www.shopware.com/en/company/story/ (aufgerufen am 25. 06. 2023).

[22] Symfony SAS. *Projects using Symfony - Popular PHP projects using Symfony components or based on the Symfony framework*. Symfony SAS. 2023. URL: https://symfony.com/projects (aufgerufen am 24. 06. 2023).

[23] Gunnard Engebreth und Satej Kumar Sahu. „Introduction to Symfony“. In: *PHP 8 Basics: For Programming and Web Development*. Berkeley, CA: Apress, 2023, S. 273–283. ISBN: 978-1-4842-8082-9. DOI: 10.1007/978-1-4842-8082-9_15.

[24] Shopware. *Einblicke in die Core Architektur von Shopware 6*. Shopware AG. 2019. URL: https://www.shopware.com/de/news/einblicke-in-die-core-architektur-von-shopware-6/ (aufgerufen am 03. 07. 2023).

[25] Pickware GmbH. *Shopware 6 – Ein Blick auf die neue Architektur*. Pickware GmbH. 2019. URL: https://www.pickware.com/de/blog/shopware-6-neue-architektur (aufgerufen am 03. 07. 2023).

[26] Shopware. *Plugins for Symfony developers*. Shopware AG. 2023. URL: https://developer.shopware.com/docs/guides/plugins/plugins/plugins-for-symfony-developers (aufgerufen am 04. 07. 2023).

[27] Shopware. *Requirements*. Shopware AG. 2023. URL: https://developer.shopware.com/docs/guides/installation/requirements (aufgerufen am 06. 07. 2023).

[28] GitHub, Inc. *Features · GitHub Actions · GitHub*. 2023. URL: https://github.com/features/actions (aufgerufen am 07. 07. 2023).

[29] GitLab B.V. *GitLab CI/CD | GitLab*. 2023. URL: https://docs.gitlab.com/ee/ci/ (aufgerufen am 07. 07. 2023).

[30] Atlassian Corporation. *Bitbucket Pipelines - Continuous Delivery | Bitbucket*. 2023. URL: https://bitbucket.org/product/features/pipelines (aufgerufen am 07. 07. 2023).

[31] Jenkins. *Jenkins - Build great things at any scale*. 2023. URL: https://jenkins.io (aufgerufen am 07. 07. 2023).

[32] Idera, Inc. *Travis CI - Build Faster*. 2023. URL: https://www.travis-ci.com/ (aufgerufen am 07. 07. 2023).

[33] Sebastian Bergmann. *PHPUnit – The PHP Testing Framework*. 2023. URL: https://phpunit.de/ (aufgerufen am 08. 07. 2023).

[34] Meta Platforms, Inc. *Jest · Delightful JavaScript Testing*. 2023. URL: https://jestjs.io/ (aufgerufen am 08. 07. 2023).

[35] Cypress.io, Inc. *JavaScript Component Testing and E2E Testing Framework | Cypress*. 2023. URL: https://www.cypress.io/ (aufgerufen am 08. 07. 2023).

[36] Software Freedom Conservancy. *Selenium*. 2023. URL: https://www.selenium.dev/ (aufgerufen am 08. 07. 2023).

[37] Maks Rafalko. *Introduction - Infection PHP*. 2023. URL: https://infection.github.io/guide/ (aufgerufen am 08. 07. 2023).

[38] OpenJS Foundation. *Find and fix problems in your JavaScript code - ESLint - Pluggable JavaScript Linter*. 2023. URL: https://eslint.org/ (aufgerufen am 09. 07. 2023).

[39] Orta Therox und Danger JS contributors. *Danger JS*. 2023. URL: https://danger.systems/js/ (aufgerufen am 09. 07. 2023).

[40] Soner Sayakci. *Danger PHP*. 2023. URL: https://github.com/shyim/danger-php (aufgerufen am 09. 07. 2023).

[41] Squiz Labs. *PHP_CodeSniffer*. 2023. URL: https://github.com/squizlabs/PHP_CodeSniffer (aufgerufen am 09. 07. 2023).

[42] Manuel Pichler. *PHPMD - PHP Mess Detector*. 2023. URL: https://phpmd.org/ (aufgerufen am 09. 07. 2023).

[43] Ondřej Mirtes. *Find Bugs Without Writing Tests | PHPStan*. 2023. URL: https://phpstan.org/ (aufgerufen am 10. 07. 2023).

[44] QOSSMIC GmbH. *Deptrac*. 2023. URL: https://github.com/qossmic/deptrac (aufgerufen am 10. 07. 2023).

[45] madewithlove BV. *CLI Licence checker for composer dependencies*. 2023. URL: https://github.com/madewithlove/license-checker-php (aufgerufen am 10. 07. 2023).

[46] Symfony SAS. *Installing & Setting up the Symfony Framework - Checking Security Vulnerabilities*. 2023. URL: https://symfony.com/doc/current/setup.html#checking-security-vulnerabilities (aufgerufen am 10. 07. 2023).

[47] Shopware. *shopware/development - Docker Image*. Shopware AG. 2023. URL: https://hub.docker.com/r/shopware/development (aufgerufen am 12. 07. 2023).

[48] Shopware. *E2E Platform Testsuite for Shopware 6*. Shopware AG. 2023. URL: https://github.com/shopware/e2e-testsuite-platform (aufgerufen am 19. 07. 2023).

[49] Deployer. *Deployer - The deployment tool for PHP | Deployer*. 2023. URL: https://deployer.org/ (aufgerufen am 15. 07. 2023).

[50] GitLab B.V. *GitLab Agent for Kubernetes*. 2023. URL: https://gitlab.com/gitlab-org/cluster-integration/gitlab-agent/ (aufgerufen am 15. 07. 2023).

[51] Oracle Corporation. *mysql - Official Image | Docker Hub*. 2023. URL: https://registry.hub.docker.com/_/mysql (aufgerufen am 21. 07. 2023).