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

CMS . . . . . Content Management System

DI . . . . . Dependency Injection

NPM . . . . . Node Package Manager

ORM . . . . . Object-Relational Mapping

OS . . . . . Operating System

QA . . . . . Quality Assurance

VCS . . . . . Version Control System

VM . . . . . Virtual Machine

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 bietet als eine führende E-Commerce-Plattform und eine der bevorzugten Online-Shop-Lösungen in Deutschland[1] eine solide Grundlage für Unternehmen, um im digitalen Raum erfolgreich zu agieren. Durch die gezielte Implementierung von CI-Praktiken in solchen Projekten kann der Entwicklungszyklus effizienter gestaltet und die Qualität des Endprodukts gesteigert werden. Dies unterstützt Unternehmen dabei, ihre Wettbewerbsfähigkeit zu erhöhen und eine agile und reaktionsschnelle Entwicklungsumgebung zu etablieren.

1.1 Motivation

In der heutigen schnelllebigen digitalen Welt ist die Fähigkeit, qualitativ hochwertige Softwareprodukte schnell auf den Markt zu bringen nicht nur wünschenswert, sondern oft entscheidend für den Geschäftserfolg. Die E-Commerce-Branche, geprägt durch ihre intensive Wettbewerbsdynamik, erfordert von Unternehmen eine kontinuierliche Anpassung und Innovation, um im Markt bestehen zu können. Hierbei nimmt die Effizienz und Effektivität der eingesetzten Softwareentwicklungsmethoden eine zentrale Rolle ein. 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 weiterentwickelnder Technologien und variierender Anforderungen besonders wichtig für die Sicherstellung der Softwarequalität 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 Konzeption und Entwicklung der CI-Strategie und stellen die geschäftsseitigen Anforderungen von Unternehmen in der E-Commerce-Branche an den Entwicklungsprozess mit Shopware dar:

- Z_1 **Hohe Entwicklungsgeschwindigkeit**

  Die Einführung einer umfangreichen CI-Strategie soll die Effizienz der Softwareentwicklungsteams verbessern und die Zeit bis zum Produkt-Release senken. Eine hohe Entwicklungsgeschwindigkeit sorgt für eine schnellere Auslieferung neuer Features und Fehlerbehebungen und somit zu einer niedrigeren Wartezeit für Kunden.

- Z_2 **Niedrige Fehlerrate**

  CI soll dazu beitragen, Fehler frühzeitig im Entwicklungsprozess erkennen und beheben zu können, was die Qualität des Endprodukts verbessert. Die Stabilität und Qualität der

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

Ausgelieferten Anwendung wirkt sich durch weniger Ausfälle und eine niedrigere Support-Zeit auf die Zufriedenheit von Kunden aus.

- Z_3 **Kontinuierliche Auslieferung neuer Software**

  Die CI-Strategie und die damit verbundenen Prozesse die sich für Entwicklerteams ergeben, sollen zu einer Anpassbaren Entwicklungsumgebung führen. Diese Umgebung soll durch kontinuierliche Weiterentwicklung an die ständig wechselenden Anforderungen der modernen Softwareentwicklung angepasst werden können, was die Wettbewerbsfähigkeit fördert.

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

Im Laufe dieser Arbeit wird eine CI-Strategie für Shopware-basierte Kundenprojekte konzeptioniert und entwickelt. Die Arbeit ist in fünf Hauptabschnitte unterteilt, die jeweils unterschiedliche Aspekte des Prozesses abdecken.

Fachlicher Hintergrund

In diesem Abschnitt wird zunächst der theoretische Rahmen für die Arbeit festgelegt. Dies umfasst eine Einführung in das Continuous-Software-Engineering und die Prinzipien und Praktiken von Continuous Integration, sowie eine Übersicht über die Shopware-Plattform. Der 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. Der Abschnitt dient als Brücke zwischen Theorie und Praxis und stellt sicher, dass die entwickelte Strategie sowohl fundiert als auch anwendbar ist.

Entwicklung der CI-Strategie

In diesem Abschnitt wird die Entwicklung der CI-Strategie als Fallbeispiel beschrieben. Dies umfasst die Auswahl und Konfiguration der benötigten Tools, die Definition von Prozessen und Workflows, die Implementierung von Automatisierungen und Tests und das automatisierte Ausliefern der Software. Der Schwerpunkt liegt hierbei auf der praktischen Umsetzung des zuvor entwickelten Konzepts und dessen Integration in reale Shopware-Projekte.

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 Strategie zu ziehen.

Schlussfolgerung und Ausblick

Der letzte Abschnitt fasst die Ergebnisse der Arbeit zusammen und es werden Schlussfolgerungen über die CI-Strategie und dessen Anwendbarkeit 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]

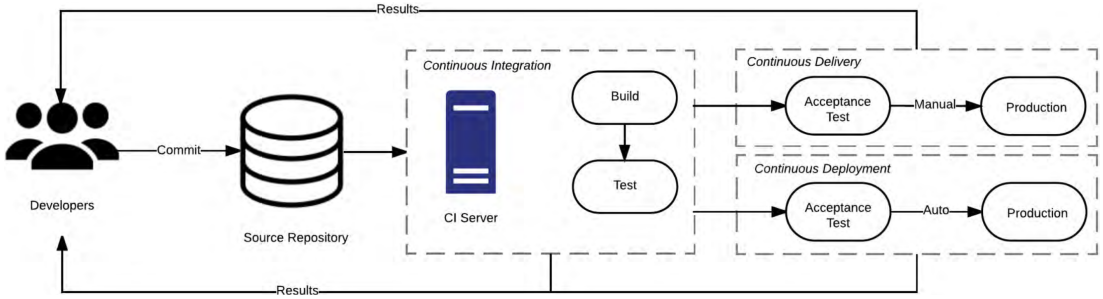
[2] Vgl. Shahin, Ali Babar und Zhu. „Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices“. 2017, S. 3910–3911.
[3] Vgl. Fitzgerald und Stol. „Continuous software engineering: A roadmap and agenda“. 2017.
[4] Vgl. Beck, Beedle, Bennekum et 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. In einem Build-Prozess erzeugte Dateien, welche die fertig gebaute Software ausmachen, werden dabei als „Artifact“ betitelt.[15] Fowler empfiehlt hierbei, dass ein entwickeltes Feature erst nach der vollständigen Integration durch einen erfolgreichen Build als fertig angesehen werden soll.[16] In der Regel finden das Bauen und Testen der Software in einer „Pipeline“ statt, welche die genutzten CI-Tools wie Test-Suites

[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. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2580.
[16] Vgl. Fowler. *Continuous Integration*. 2006.

und Static-Code-Analysis ausführt.[17]

- **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.[18] 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.[19] Automatisierten Tests in einer CI-Pipeline können insgesamt dabei helfen, Fehler in eingeführtem Quellcode zu finden.[20] 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.[21]

- **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.[22]

- **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

[17] Vgl. Elazhary, Werner, Li et al. „Uncovering the Benefits and Challenges of Continuous Integration Practices“. 2022, S. 2571.
[18] Vgl. Ahmad, Haleem und Beg. „Test Driven Development with Continuous Integration: A Literature Review“. 2013, S. 280.
[19] Vgl. Fowler. *Continuous Integration*. 2006.
[20] **benefits-and-challenges**.
[21] Vgl. Zampetti, Scalabrino, Oliveto et al. „How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines“. 2017, S. 338.
[22] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 74–75.

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 die Kommunikation und Zusammenarbeit im Team, da alle Mitglieder ständig über den Status des Projekts informiert sind.[23,24]

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“.[25] 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.[26] 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.[27]

Pipelines

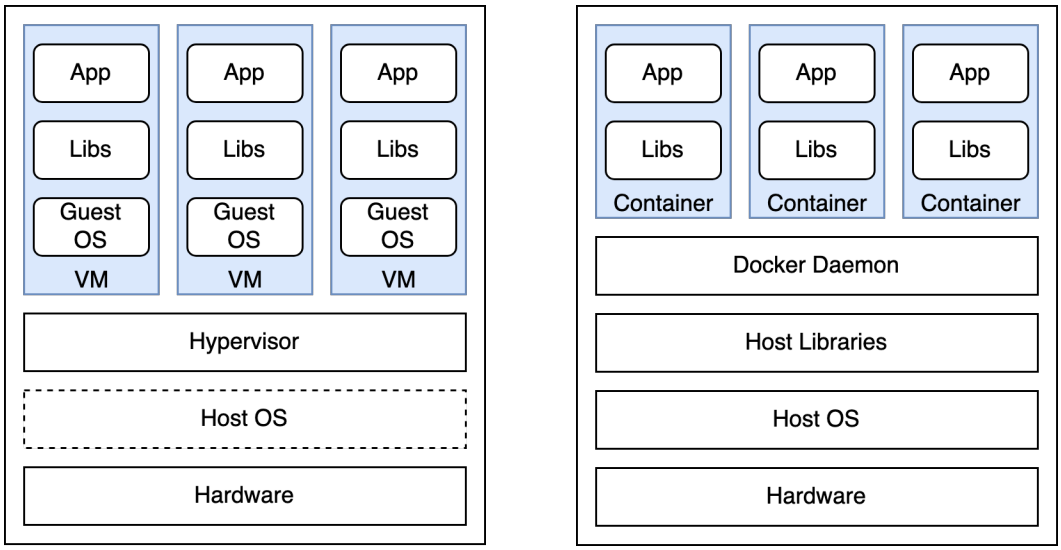
Um die in einer Versionskontroll-Software eingeführten Code-Änderungen automatisiert bauen, testen und ausliefern zu können, wird eine ausführende Umgebung für diese Prozesse benötigt. In einem CI-Prozess werden hierfür Pipelines eingesetzt, welche zum Ausführen von Befehlen und Prozessen zur automatischen Integration und dem Testen von Code genutzt werden.[28] Sie wer-

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

den durch sogenannte Pipeline- oder Task-Runner verwaltet und können verschiedene Aufgaben verrichten, welche durch das Ausführen von Befehlen innerhalb sogenannter „Jobs“ durchgeführt werden. Pipelines können mehrere Jobs starten, teilweise auch parallel, wobei nacheinander laufende Jobs voneinander Abhängig sein und Artifacts für nachfolgende Jobs zwischengespeichert werden können. Pipelines spielen durch das kontinuierliche Integrieren von entwickeltem Code eine integrale Rolle bei der Reduzierung der Cycle Time[29] und sind somit wichtig zum Erreichen des Ziels Z_1 der Arbeit. Eine Pipeline in einem CI-Prozess wird in der Regel bei jeder Änderung am Quellcode des Source-Repositories gestartet.[30]

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 sogenannten „Hypervisor“, welcher die Hardware eines physischen Computers emulieren kann, sodass mehrere Betriebssysteminstanzen und die darin verwalteten Applikationen gleichzeitig auf einem einzigen System ausführbar sind.[31] Containerization ist eine modernere Technologie, die einen Performance-Zuwachs gegenüber VMs bieten kann.[32] 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.



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

Abbildung 2: Visualisierung der Hypervisor- und Container-Architektur

[29] Vgl. Fowler. *Continuous Integration*. 2006.
[30] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 3–4.
[31] Vgl. Combe, Martin und Di Pietro. „To Docker or Not to Docker: A Security Perspective“. 2016, S. 54–55.
[32] Vgl. Spoiala, Calinciuc, Turcu et al. „Performance comparison of a WebRTC server on Docker versus virtual machine“. 2016.

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. Ein Container ist hierbei die laufende Instanz eines Container-Images, welches den Aufbau des Containers inklusive Abhängigkeiten und installierter Software speichert. 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. In Abbildung 2 werden die Architekturen von Hypervisor und Container gegenübergestellt. Die Visualisierung zeigt im linken Teil den Aufbau eines Systems, in dem ein Hypervisor zum emulieren von Hardware-Ressourcen verwendet wird (a). Die Technologie kann hierbei auf einem bereits installierten Host-Betriebssystem (Host OS) oder ganz ohne OS betrieben werden. Einzelne VMs die durch den Hypervisor betrieben werden können, nutzen hierbei jeweils ein eigenes Gast-Betriebssystem (Guest OS). Im Kontrast dazu steht die Container-Architektur, welche das Betreiben eines Host OS voraussetzt und dessen Ressourcen zur Ausführung isolierter Container-Instanzen verwendet (b). Das Ausführen der Container wird dabei durch eine Container-Engine übernommen, im Falle der Docker-Technologie wird dies durch einen eigenen Hintergrundprozess (Docker Daemon) ermöglicht.

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]
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

[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.
[37] Vgl. Ammann und Offutt. *Introduction to Software Testing*. 2016, S. 23–24.

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.

Durch diese verschiedenen Test-Arten kann ein großer Bereich der zu testenden Software und dessen Codebase abgedeckt werden. Da bereits erstellte Tests immer wieder verwendet werden können, werden diese oftmals eingesetzt um zu prüfen, ob neu eingeführter Code diese bestehenden Tests fehlschlagen lässt. Dieser Einsatz von existierenden Tests zur Sicherstellung der Funktionalität des Systems wird Regression-Testing genannt und stellt einen wichtigen Aspekt des automatisierten Prüfens von Software in einem CI-Kontext dar.[40] Um beim Ausführen von Tests etwaige Abhängigkeiten die zur Ausführung des Codes innerhalb Test benötigt werden

[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. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 53–54.

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.[41] 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.[42] 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.[43]

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.[44]

Monitoring

Monitoring, also das Überwachen von Software und dessen Metriken, ist ein unerlässlicher Bestandteil moderner Softwareentwicklungs- und Betriebsprozesse. Die Ergebnisse von Tests und statischen Code-Analysen eines CI-Prozesses können zur kontinuierlichen Überwachung von Qualitätsmetriken und Code-Standards genutzt werden.[45] Neben der Überwachung von Test- und QA-Ergebnissen können auch weitere Software-Bereiche überwacht werden, zum Beispiel durch das Ausführen von Performance- und Lasttests innerhalb der CI-Pipeline.[46] Hierbei wird die Geschwindigkeit des Durchführens von Transaktionen oder Aufrufzeiten beim Ausführen der Software mit vielen gleichzeitigen Nutzerzugriffen überwacht.

[41] Vgl. Ammann und Offutt. *Introduction to Software Testing*. 2016, S. 299–300.
[42] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 132–133.
[43] Vgl. Ammann und Offutt. *Introduction to Software Testing*. 2016, S. 242.
[44] Vgl. Zampetti, Scalabrino, Oliveto et al. „How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines“. 2017.
[45] Vgl. Duvall, Matyas und Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. 2007, S. 17–18.
[46] Vgl. Collins, Dias-Neto und Lucena Jr. „Strategies for Agile Software Testing Automation: An Industrial Experience“. 2012, S. 441.

2.3 Übersicht über die Shopware-Platform

Shopware wurde als Online-Shop-Software im Jahr 2000 durch Stefan Hamann ins Leben gerufen[47] 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 Content-Management-System (CMS) „Drupal“, dem Shop-System „Magento“ und einigen weiteren Programmen[48] 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 Verfügbarkeit und Skalierbarkeit zu gewährleisten. Nachfolgend wird der Aufbau des Symfony-Frameworks dargestellt und die Architektur der darauf basierenden Shopware-Platform aufgezeigt.

Symfony-Framework

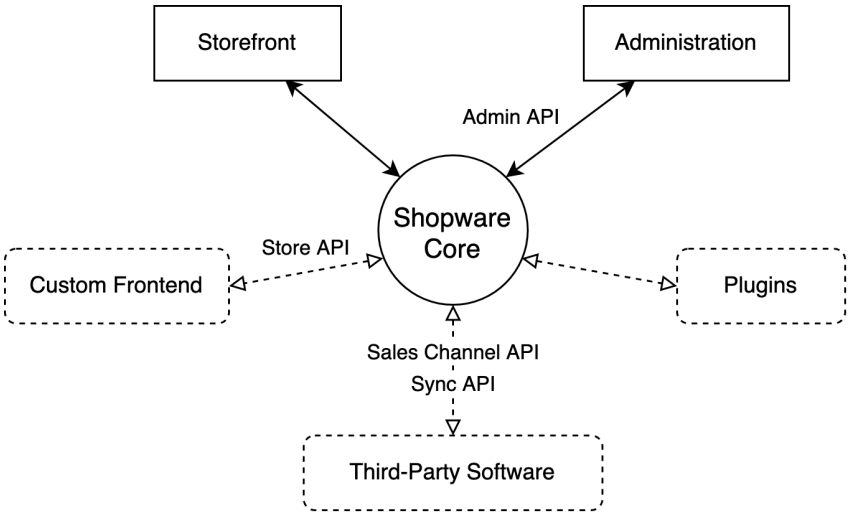
Symfony ist ein im Jahr 2005 von Fabien Potencier entwickeltes Full-Stack-Framework. Das PHP-basierte Projekt besteht aus verschiedenen Einzelkomponenten, welche unabhängig voneinander verwendet werden können, somit ist es sehr flexibel. Gleichzeitig bietet Symfony eine Reihe von Konventionen und Best 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.[49]

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

[47] Vgl. Shopware. *The story behind Shopware AG*. 2023.
[48] Vgl. Symfony SAS. *Projects using Symfony - Popular PHP projects using Symfony components or based on the Symfony framework*. 2023.
[49] Vgl. Engebreth und Sahu. „Introduction to Symfony“. 2023, S. 273–278.

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. Die Administration bietet neben der Konfiguration des Shops und dessen Daten und Produkten auch eine CMS-Funktion und kann beliebigen Content verwalten. 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 einzelnes Repository mit dem Namen `shopware/platform` auf GitHub zur Verfügung.[50] 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.[51]



Quelle: Eigene Darstellung nach Pickware GmbH (2019)

Abbildung 3: Visalisierung der Shopware-Architektur

Eine grobe Zusammenfassung der Architektur kann aus Abbildung 3 entnommen werden. Hierbei werden die Kernkomponenten von Shopware aufgezeigt und optionale Services in gestrichelt dargestellt. 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,

[50] Vgl. Shopware. *Einblicke in die Core Architektur von Shopware 6*. 2019.
[51] Vgl. Pickware GmbH. *Shopware 6 – Ein Blick auf die neue Architektur*. 2019.

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.[52] 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.

[52] 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 werden technische Anforderungen an die zu entwickelnde Strategie im Hinblick auf die geschäftsseitigen Vorgaben durch die Ziele Z_n der Arbeit definiert. Anschließend wird die Ausgangssituation in Shopware-Projekten und dessen Umgebungen und kundenspezifischer Anpassungen beleuchtet, wobei der Fokus auf den gemeinsamen Grundvoraussetzungen und der Anpassungsfähigkeit der CI-Strategie an verschiedene Architekturen liegt. Zuletzt wird die eigentliche Konzeption der Strategie anhand der zuvor erarbeiteten Methoden und Praktiken und den definierten Anforderungen vorgenommen.

3.1 Technische Anforderungen

Um ein vollumfängliches Konzept erstellen zu können, müssen einige technische Anforderungen an die Strategie definiert werden. Diese Anforderungen orientieren sich sowohl an der Zieltechnologie – in diesem Fall Shopware – als auch an den im fachlichen Hintergrund erörterten Methoden und Praktiken. Da die Strategie das Nutzen von CI in Shopware-Projekten ermöglichen soll, werden nachfolgend die technischen Anforderungen an das Konzept im Hinblick auf die Gegebenheiten der Shop-Software aufgezeigt:

- **Vollautomatisierter Software-Build**

  Der Prozess des Software-Builds muss durchweg automatisiert stattfinden. Dies umfasst das Installieren der Software aus der Versionskontrolle, dem Herunterladen und Installieren sämtlicher Abhängigkeiten und dem Erzeugen von Build-Dateien aus dem gegebenen Quellcode.

- **Automatisierte Software-Tests und Quality-Checks**

  Da Testing einen integralen Bestandteil der CI bildet, muss die zu konzipierende Strategie ein umfassendes Test-Konzept bieten. Dies schließt sowohl Unit- und Module-Tests, als auch programmübergreifende Prüfungen wie Integration- und System-Tests ein. Zudem soll die Qualitätssicherung der zu entwickelnden Software eines Projekts durch statische Code-Analyse-Tools in die CI-Strategie integriert werden.

- **Reproduzierbare CI-Umgebungen**

  Es ist von entscheidender Bedeutung, dass die CI-Umgebung konsistent und reproduzierbar ist. Dies gewährleistet, dass jeder Build und Test unter identischen Bedingungen durchgeführt wird, unabhängig davon, wann und wo der CI-Prozess initiiert wird. Die Strategie sollte daher die Verwendung von Containerisierungstechnologien, wie Docker, in Betracht ziehen, um eine einheitliche, isolierte und wiederholbare Umgebung für den Build- und Testprozess zu schaffen. Dies minimiert potenzielle Inkonsistenzen und Fehler, die durch unterschiedliche Umgebungsbedingungen entstehen könnten.

- **Hohe Anpassbarkeit und Skalierbarkeit**

  Die CI-Strategie sollte flexibel genug sein, um sich an veränderte Anforderungen und wachsende Projektgrößen anzupassen. Dies bedeutet, dass sowohl die Infrastruktur als auch die

Prozesse skalierbar gestaltet werden müssen, um mit der Evolution des Projekts Schritt zu halten. Die Möglichkeit, neue Tools und Technologien nahtlos zu integrieren, sollte ebenfalls gegeben sein, um die kontinuierliche Anpassung und Optimierung der CI-Pipeline zu gewährleisten.

- **Einheitliche Daten-Verwaltung**

  Um Konsistenz und Nachvollziehbarkeit zu gewährleisten, sollte alles, von Code über Konfigurationen bis hin zu Datenbank-Skripten und Tests, in einem Versionskontrollsystem verwaltet werden. Dies ermöglicht eine klare Historie von Änderungen und erleichtert das Rollback im Falle von Problemen. Ein zentralisiertes VCS stellt sicher, dass alle Teammitglieder stets mit der aktuellsten Version arbeiten und Änderungen nachvollziehbar sind.

- **Kontinuierliche Bereitstellung und Deployments**

  Neben dem Build- und Testing-Prozess ist es auch wichtig, dass die CI-Strategie Mechanismen für die kontinuierliche Bereitstellung und das Deployment der Software bietet. Dies ermöglicht es, Änderungen schnell in die Produktionsumgebung zu übertragen und sicherzustellen, dass die Software stets aktuell und einsatzbereit ist. Automatisierte Deployments sollten daher in den Pipelines mit eingeplant werden, um den Prozess der Softwareauslieferung zu optimieren und zu beschleunigen.

- **Hohe System-Verfügbarkeit**

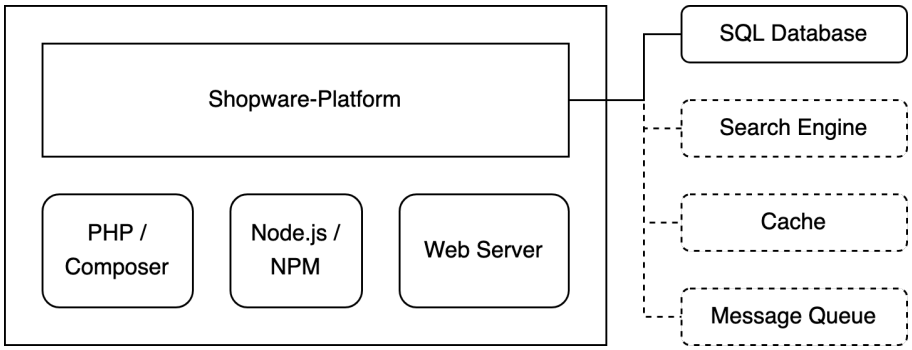
  Die CI-Infrastruktur sollte so konzipiert sein, dass sie stets verfügbar ist und minimale Ausfallzeiten aufweist. Dies ist entscheidend, um den kontinuierlichen Entwicklungsfluss nicht zu unterbrechen und eine ständige Feedback-Schleife zu gewährleisten. Redundanzen und regelmäßige Backups sollten implementiert werden, um die Systemverfügbarkeit auch bei unerwarteten Problemen zu gewährleisten.

Diese technischen Anforderungen sollen das Entwickeln einer vollumfänglichen CI-Strategie im Hinblick auf die definierten Ziele Z_n der Arbeit ermöglichen.

3.2 Analyse der Ausgangssituation

Die aktuelle Situation stellt sich wie folgt dar: In E-Commerce-Unternehmen die Shopware-Projekte betreiben, gibt es in der Regel verschiedene Kunden, die mit unterschiedlichen Versionen von der Software arbeiten. Diese Kunden können bei verschiedenen Hosting-Anbietern untergebracht sein, was zu einer Vielfalt an technischen Umgebungen führt, in denen die Software betrieben wird. Darüber hinaus verwenden Kunden meist eine individuelle Kombination aus Plugins und Eigenentwicklungen, die auf dessen spezifische 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. 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.[53]

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



Quelle: Eigene Darstellung nach Shopware (2023)

Abbildung 4: Umgebung und Abhängigkeiten der Shopware-Platform

In Abbildung 4 werden die grundsätzlichen 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 Codebase durchzuführen, wird also eine Umgebung vorausgesetzt, in der die benötigten Services und Tools installiert sind. Nachfolgend werden die in der Grafik aufgezeigten Abhängigkeiten und Services erläutert:

- **PHP**

  Da Symfony und somit auch Shopware auf der Programmiersprache PHP basieren, muss diese in der ausführenden Umgebung installiert sein.

  - **PHP-Extensions**

    Shopware benötigt für den Betrieb einige PHP-Extensions, zum Beispiel zum Erstellen von Archiven oder dem Lesen von XML-Dateien.

  - **Paketmanager „Composer“**

    Durch Composer werden sowohl die Abhängigkeiten von Symfony und Shopware, als auch Third-Party-Plugins und eigene Entwicklungen im Projekt verwaltet und zur Nutzung im PHP-Code bereitgestellt.

- **JavaScript**

  JavaScript bei der Nutzung der Shopware-Platform sowohl im Browser, als auch Serverseitig ausgeführt.

  - **JavaScript-Runtime „Node“**

    Zur Serverseitigen Ausführung von JavaScript-Code wird die Node-Runtime als Abhängigkeit vorausgesetzt.

  - **Node-Paketmanager „NPM“**

    Der Paktemanager NPM ist eine weitere Abhängigkeit von Shopware. Dieser verwaltet JavaScript-Pakete, welche zur Ausführung der Software benötigt werden.

- **Webserver**

  Um die Shop-Instanz im Browser aufrufen und API-Anfragen verarbeiten zu können, wird eine Webserver-Software benötigt.

- **Datenbank**

  Shopware schreibt zur Nutzung der Software eine Datenbank vor, welche die eigentlichen Shop-Konfigurationen, Produkte, Bestellungen, Kunden und weitere Daten verwaltet.

- **Search Engine**

  Um den Besuchern des Online-Shops eine Textsuche für Produkte und Hersteller zu bieten, unterstützt Shopware das optionale Anbinden verschiedener Suchmachinen.

- **Cache**

  Zur Optimierung der Shop-Performance kann zudem optional ein Zwischenspeicher, auch „Cache“ genannt, für Warenkorb-Inhalte und aktuelle Nutzer-Sitzungen eingeführt werden.

- **Message Queue**

  Um viele gleichzeitige Zugriffe von Usern effizient verwalten zu können, erlaubt Shopware das Anbinden einer Message Queue. Diese ermöglichen das Sammeln von Nutzer-Anfragen, welche dann in Reihe verarbeitet werden können.

Durch das Installieren dieser grundlegenden Abhängigkeiten und Services wird das Betreiben von Shopware 6 ermöglicht. Da diese sich in zukünftigen Versionen der Software ändern können, sollte bei neuen Projekten immer geprüft werden, welche Abhängigkeiten für die gewünschte Shopware-Version vorgegeben sind. Alte Projekte sollten außerdem kontinuierlich auf aktuelle Versionen upgedatet und auf die Erfüllung von dessen Voraussetzungen geprüft werden.

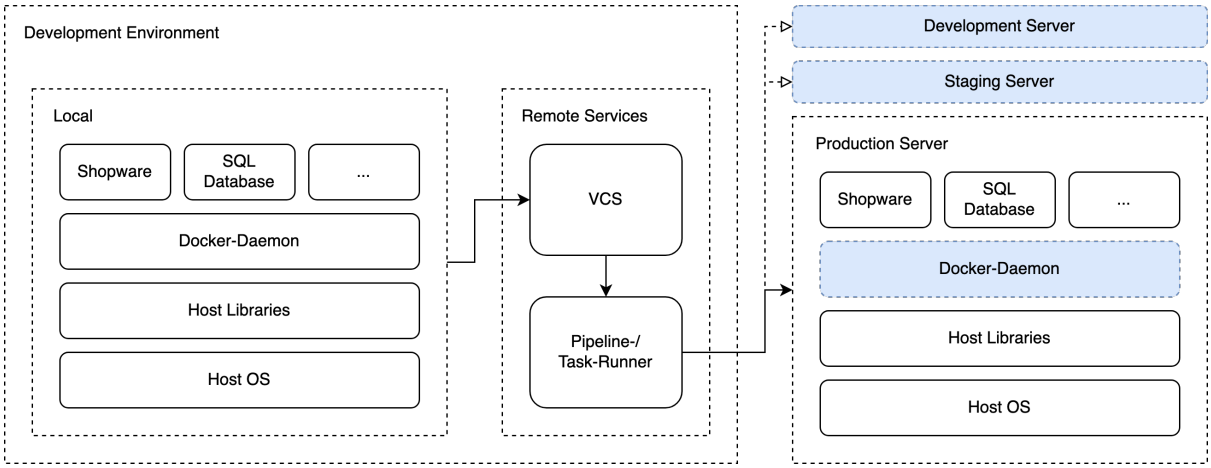
3.3 Konzeption der CI-Strategie

Im Folgenden wird die Konzeption der CI-Strategie für Kundenprojekte auf Basis der Shopware-Platform durchgeführt. Hierbei wird zunächst die ausführende Umgebung geplant, sowohl für das lokale Entwickeln mit Shopware als auch für die Integration und Ausführung der verschiedenen CI-Tools innerhalb einer Pipeline. Anschließend wird die Struktur der angedachten Pipeline aufgestellt, wobei dessen einzelne Aufgaben systematisch in eigene Phasen und Jobs unterteilt werden.

Projekt-Struktur und Umgebung

Die Anforderungen des Projekts setzten eine Umgebung, in der die Shopware-Platform ausgeführt, getestet und analysiert werden kann voraus. Eine Umgebung muss in diesem Fall sowohl lokal als auch 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. Da spätere Umgebungen, an welche die Software ausgeliefert werden soll, nicht zwangsläufig unter der Kontrolle des

Entwicklerteams stehen, wird sich in der Strategie auf die Vereinheitlichung der lokalen Entwicklungsumgebung und der Pipeline beschränkt. Hierzu empfiehlt sich die Nutzung von Containerization, wobei die Abhängigkeiten zum Ausführen der Shopware-Platform und der CI-Tools gebündelt und wiederverwendet werden können. Ein lokal entwickeltes Feature kann so nach der Fertigstellung über ein VCS integriert werden, welches dann die Pipeline-Orchestration anstößt. Die Pipeline soll zur Ausführung der einzelnen Jobs das gegebene Image instanziieren und als ausführende Umgebung verwenden, sodass diese möglichst der lokalen Entwicklungsumgebung gleicht. Nach dem erfolgreichen Durchlaufen des Software-Builds, Tests und Qualitäts-Checks soll die Software letztlich durch einen Deployment-Job innerhalb der Pipeline automatisiert an verschiedene Umgebungen ausgeliefert werden können. Hierbei kann es sich um die Produktionsumgebung oder andere Infrastrukturen handeln, wie zum Beispiel einen Development-Server zum Testen von Features oder einen Staging-Server für die Abnahme von neuen Änderungen durch Projekt-Kunden.



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

Abbildung 5: Geplante Architektur der CI-Strategie

Eine Visualisierung der geplanten Architektur für die CI-Strategie kann in Abbildung 5 eingesehen werden. Die Darstellung zeigt die Struktur der lokalen Entwicklungsumgebung so wie der späteren möglichen Deployment-Umgebungen in Zusammenhang mit dem VCS und des Pipeline-Runners 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 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.

Aufbau der Pipeline

Im Mittelpunkt der CI-Strategie steht die Pipeline zum automatischen Bauen, Testen und Ausliefern der Software. Nachdem die Umgebung für Projekte und der grundsätzliche Ablauf von

Integrationen in der Strategie definiert wurde, wird nun die eigentliche Pipeline entworfen. Da diese verschiedene Phasen durchläuft, werden nachfolgend die für die Pipeline angedachten Stages und dessen Aufbau erläutert.

- **Build-Stage**

  Ungeachtet der späteren ausführenden Umgebung muss in einem Shopware-Projekt zur Durchführung weiterer Schritte eine Build-Phase durchlaufen werden. Hierbei werden die Abhängigkeiten der Shopware-Platform selbst, zusammen mit weiteren Paketen wie Testing- und QA-Tools installiert und somit für die Testing-Phase vorbereitet. Die hierbei stattfindende Integration wird also zunächst durch den Status des Software-Builds geprüft. Ist der Build-Job erfolgreich, wird ein Artifact erzeugt welches, um die Durchlaufzeit der Pipeline insgesamt möglichst gering zu halten, in einem Cache zur weiteren Nutzung zwischengespeichert wird.

- **Test-Stage**

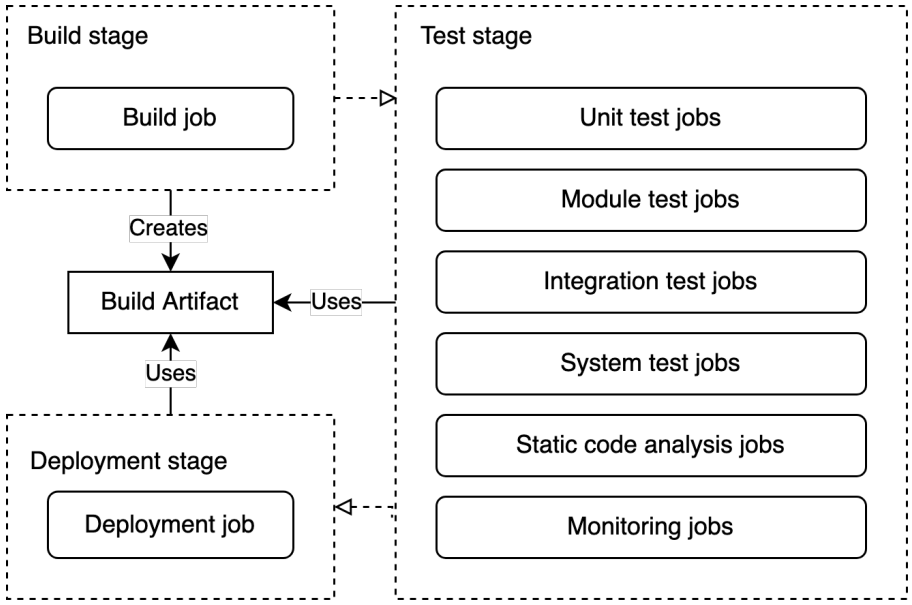
  Wenn die Build-Stage erfolgreich durchgelaufen ist und ein Artifact erzeugt wurde, beginnt die Testing-Phase. In der Strategie ist hierbei das parallele Ausführen von Tests und QA-Tools auf der bestehenden Codebase angedacht. Um Tests parallel auszuführen werden mehrere Jobs gestartet, welchen bereits alle installierten Tools und der zu testende Quellcode durch das zwischengespeicherte Artifact des Build-Jobs zur Verfügung stehen. Hierbei sollte eine möglichst große Abdeckung des Codes durch verschiedene Arten von Software-Tests bestehen. Durch Unit-, Module- und Integration-Tests kann an dieser Stelle die Funktionalität der einzelnen Anwendungs-Komponenten überprüft werden, während System-Tests die Korrektheit des Gesamtsystems anhand der Geschäftsanforderungen abdecken. Besondere Anforderungen einzelner Testing-Tools, wie zum Beispiel ein Datenbank-Service innerhalb der Pipeline zum Ausführen von System-Tests, sollten dabei nicht in der Build-Phase, sondern vor der Ausführung des Tools im jeweiligen Job installiert werden. Parallel zu den verschiedenen Test-Arten können in diesem Schritt Tools zur Static-Code-Analysis ausgeführt werden. Die Test-Stage umfasst somit die automatisierte Qualitäts- und Funktionalitäts-Prüfung der Shop-Software in der CI-Pipeline. Dabei anfallende Metriken und Stati sollen kontinuierlich überwacht werden, wobei auch Monitoring-Tools zur weiteren Prüfung der Applikations-Performance und anderen Qualitäts-Indikatoren eingesetzt werden können.

- **Deployment-Stage**

  Wenn der Software-Build inklusive aller automatisierten Tests und Qualitäts-Checks erfolgreich durchlaufen, kann eine Deployment-Stage gestartet werden. In der Strategie ist dessen Ausführung für Branches angedacht, bei denen eine Ziel-Umgebung zur Auslieferung des gebauten Artifacts existiert, wie zum Beispiel die Produktionsumgebung. Der hierbei ausgeführte Deployment-Job liefert die relevanten Daten der Shop-Software, Testing-Tools und CI-Abhängigkeiten ausgeschlossen, an die definierte Umgebung aus. Hierbei muss beachtet werden, dass die Ausfallzeit der Applikation bei der automatischen Auslieferung so gering wie möglich bleibt, um die technischen Anforderungen der Strategie zu erfüllen.

Eine Zusammenfassung des Aufbaus und der Phasen der Pipeline kann in Abbildung 6 eingesehen werden. Hierbei werden die einzelnen Phasen in gestrichelt dargestellt, diese enthalten die Arten

von Jobs, welche in der jeweiligen Phase aufgerufen werden. Der Ablauf der Pipeline wird durch gestrichelte Pfeile dargestellt, welche die Reihenfolge der Phasen verdeutlichen. Außerdem wird aufgezeigt, dass der Build-Job ein Software-Artifact erzeugt, auf welchem die weiteren Jobs der Test- und Deployment-Stage basieren. Die in einem Kundenprojekt entwickelte Pipeline soll dieser Struktur folgen, wobei allerdings keine festen Tools oder Versionen vorgegeben sind. Bei jedem neuen Projekt sollte also vorher eine Analyse der verfügbaren Testing und QA-Tools durchgeführt werden, um immer auf dem aktuellsten Stand zu sein.



Quelle: Eigene Darstellung

Abbildung 6: Phasen und Abhängigkeiten der CI-Pipeline

Diese Struktur ist in der Strategie zunächst für die Durchführung der Pipeline bei allen Ausgangs-Branches angedacht. Da Jobs allerdings eine lange Durchlaufzeit ($> 10min$) haben können, soll die Möglichkeit bestehen, diese nur unter bestimmten Bedingungen auszuführen. So können lang-andauernde System-Tests oder Performance-Monitoring zum Beispiel nur bei der Integration in einen Branch vorgenommen werden, für den eine Deployment-Umgebung definiert ist. Die Strategie soll somit Flexibilität bieten und an die individuellen Bedürfnisse eines Projekts angepasst werden können.

4 Entwicklung der CI-Strategie

5 Evaluierung

6 Schlussfolgerungen und Ausblick

A Anhang I: ...

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

[21] 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).

[22] 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.

[23] 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).

[24] 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).

[25] 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).

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