**Bachelorarbeit**

IU Internationale Hochschule für angewandte Wissenschaften

B. Sc. Wirtschaftsinformatik

Die Vorteile der High-Code Entwicklung im Vergleich zur Low-Code Entwicklung

Evsin Rahmiev

Einschreibungsnummer: 32105477

Akademiestr. 6

68159 Mannheim

Betreuer: Prof. Dr. Thorsten Fröhlich

Datum der Einreichung: 14.04.2025

Abstrakt

**Zweck:** Die vorliegende Arbeit hat die Aufgabe, die Frage zu beantworten, welche Vorteile die High-Code Entwicklung im Vergleich zur Low-Code-Entwicklung anbietet.

**Wert:** Die Studie liefert der Zielgruppe Aufschluss darüber, welche Art von Anforderungen sich effektiver über High-Code Entwicklung umsetzen lassen.

**Methoden**: Für den Vergleich zwischen den beiden Entwicklungsansätzen wird ein Case-Study mit einem A/B Test durchgeführt, der nicht nach einer Konversionsrate strebt. Damit die beiden Entwicklungsansätze verglichen werden, werden eine High-Code- und eine Low-Code-Version einer Anwendung zum Deutschlernen programmiert.

**Wichtigste Ergebnisse**: Mehr Anforderungen lassen sich über die High-Code Entwicklung realisieren.

**Schlussfolgerung:** Obwohl das Programmieren mancher Anforderungen an die Anwendung bei beide Entwicklungsansätze die Gleichen Ergebnissen erzielt haben, konnten einige Anforderungen nur über die High-Code- aber nicht über die Low-Code-Anwendung umgesetzt werden.

**Schlüsselwörter**: high-code, low-code, softwareentwicklung

Inhaltsübersicht

[Verzeichnis der Abbildungen vi](#_Toc172896932)

[Liste der Tabellen vi](#_Toc172896933)i

[Liste der Formeln vi](#_Toc172896934)ii

[Verzeichnis der Abkürzungen i](#_Toc172896935)x

[1 Einführung 7](#_Toc172896936)

[1.1 Ziel der Forschung 7](#_Toc172896937)

[1.2 Forschungsfragen 7](#_Toc172896938)

[1.3 These 7](#_Toc172896939)

[1.4 Wert und Zielpublikum 7](#_Toc172896940)

[1.5 Anwendungsbereich und Einschränkungen 7](#_Toc172896940)

[1.6 Struktur des Dokuments 7](#_Toc172896940)

[2 Literaturübersicht 8](#_Toc172896941)

[2.1 Hintergrundinformationen 8](#_Toc172896942)

[2.2 Bestehende Modelle und Theorien 8](#_Toc172896943)

[2.3 Ausgewählte Fallstudien 8](#_Toc172896944)

[2.4 Szenario Beschreibung 8](#_Toc172896945)

[3 Forschungsdesign 9](#_Toc172896946)

[3.1 Anforderungsanalyse 9](#_Toc172896947)

[3.2 Spezifikation der Anforderungen 9](#_Toc172896948)

[3.3 Technologie 9](#_Toc172896949)

[3.4 Analyse 9](#_Toc172896950)

[3.5 Fehlerbewertung 9](#_Toc172896951)

[4 Ergebnisse und Diskussion 10](#_Toc172896954)

[4.1 Seitennavigation 10](#_Toc172896955)

[4.2 Manipultaion der Seitenelemente 10](#_Toc172896956)

[4.3 Anpassbares Formular 10](#_Toc172896957)

[4.4 Inzeilige Eingabefelder 10](#_Toc172896957)

[4.5 Sitzungsunabhängige Datenspeicherung 10](#_Toc172896957)

[5 Schlussfolgerung 11](#_Toc172896958)

[5.1 Kritische Reflexion 11](#_Toc172896959)

[5.2 Empfehlungen für die zukünftige Forschung 11](#_Toc172896960)

[5.3 Ausblick 11](#_Toc172896960)

[Referenzen 15](#_Toc172896965)

[Anhang A. Leere Zoho-Seite anch Betätigen eines HTML-Links 16](#_Toc172896966)

[Anhang B. Seitennavigation über den Menüersteller 17](#_Toc172896969)

[Anhang C. Deluge Funktionen Teil 1 18](#_Toc172896970)

[Anhang D. Deluge Funktionen Teil 2 18](#_Toc172896970)

[Anhang E. Deluge Funktionen Teil 3 18](#_Toc172896970)

[Anhang F. HC-Kontaktformular 18](#_Toc172896970)

[Anhang G. Code für das HC-Kontaktformular 18](#_Toc172896970)

[Anhang H. Feldeigenschaften von Fomularfeldern 18](#_Toc172896970)

[Anhang I. HTML-Editor im Form Builder 18](#_Toc172896970)

[Anhang J. NC-Kontaktformular über Zoho Creator 18](#_Toc172896970)

[Anhang K. Anpassungsoptionen für den Formular-Baustein 18](#_Toc172896970)

[Anhang L. Präfix-, Suffix-, Pflichtfeld- und PBD-Angabe 18](#_Toc172896970)

[Anhang M. Feldtyp-, Feldgrößen- und Beschreibungstextangabe 18](#_Toc172896970)

[Anhang N. HC-Lückentest 18](#_Toc172896970)

[Anhang O. LC-Lückentext im Zoho Creator 18](#_Toc172896970)

[Anhang P. HTML- und CSS-Code als getrennte Dokumente 18](#_Toc172896970)

[Anhang Q. HTML-Snippet mit HTML- und CSS-Code 18](#_Toc172896970)

[Anhang R. Bootstrap Button im normalen Zustand 18](#_Toc172896970)

[Anhang S. Bootstrap Button mit Hover-Effekt 18](#_Toc172896970)

[Anhang T. HC-Kontaktformular bei 550 Pixel Bildschirmbreite 18](#_Toc172896970)

[Anhang U. HC-Kontaktformular bei 350 Pixel Bildschirmbreite 18](#_Toc172896970)

[Anhang V. LC-Abschlusstest bei 550 Pixel Bildschirmbreite 18](#_Toc172896970)

[Anhang W. Werte zur HC-Aufwandsermittlung 18](#_Toc172896970)

[Anhang X. Ermittlung des wahrscheinlichen HC-Schadens 18](#_Toc172896970)

[Anhang Y. Werte zur LC/NC-Aufwandsermittlung 18](#_Toc172896970)

[Anhang Y. Ermittlung des wahrscheinlichen LC/NC-Schadens 18](#_Toc172896970)

Liste der Abbildungen

[Abbildung 1. (Iron Triangle) 13](#_Toc172895425)

[Abbildung 2. (Magisches Viereck) 13](#_Toc172895425)

[Abbildung 3. (Tagesaufwand für die Fehlerbehebung) 13](#_Toc172895425)

[Abbildung 4. (Link zu einer Inhaltsseite) 13](#_Toc172895425)

[Abbildung 5. (Sidebar-Menu für globale Navigation) 13](#_Toc172895425)

[Abbildung 6. (Input-Feld für Übungsantwort gefolgt von einem Prüf-Button) 13](#_Toc172895425)

[Abbildung 7. (Input-Feld für E-Mail-Adressen) 13](#_Toc172895425)

[Abbildung 8. (Eigendefinierte CSS-Klasse für das Kontaktformular) 13](#_Toc172895425)

[Abbildung 9. (Inzeilige Eingabefelder mittels HTML) 13](#_Toc172895425)

[Abbildung 10. (CSS-Klasse zum Entfernen vob Stichpunkten) 13](#_Toc172895425)

[Abbildung 11. (Teilaufgabe zur sitzungsunabhängigen Datenspeicherung) 13](#_Toc172895425)

[Abbildung 12. (Funktion zur browserseitigen Speicherung von Daten) 13](#_Toc172895425)

[Abbildung 13. (Abrufen von zuvor gespeicherten Werten) 13](#_Toc172895425)

[Abbildung 14. (Bootstrap Klasse für runde Ecken) 13](#_Toc172895425)

[Abbildung 15. (Bootstrap-Klassen für Buttons) 13](#_Toc172895425)

[Abbildung 16. (Responsiver Bootstrap-Container) 13](#_Toc172895425)

Liste der Tabellen

[Tabelle 1. Funktionale Anforderungen 13](#_Toc172895426)

[Tabelle 2. Nicht-funktionale Anforderungen 13](#_Toc172895426)

[Tabelle 3. Technische Anforderungen 13](#_Toc172895426)

Liste der Formeln

Formel 1. ROI 12

Formel 2. Aufwand 12

Formel 3. Wahrscheinlicher Schaden 12

Liste der Abkürzungen

Abkürzungen alphabetisch sortiert (mit Stil *Abkürzung*):

DOM Document Object Model

HC High Code

IDE Integrated Development Environment

LCAP Low-Code-Applikationsplattform

LC/NC Low-Code/No-Code

ROI Return on Investment

VSC Visual Studio Code

MDE Model-Driven Engineering

# Einführung

Die Technologie der Low-Code/No-Code Entwicklung bietet einen alternativen Ansatz zur Softwareentwicklung, mithilfe derer die Entwicklungszeit für eine ausgereifte Softwarelösung drastisch reduziert werden kann. Dies ist möglich dank der Art und Weise der LC/NC-Entwicklung, bei der Systemkomponenten und Module zum Teil durch das einfache Auswählen bzw. Drag-and-Drop von vorprogrammierten UI-Elementen wie z. B. Auswahlfelder, Tabellen oder Grafiken implementiert werden können. Dies stellt eine Erleichterung im Vergleich zu der bereits etablierten und mehr bekannten High-Code Entwicklung, bei der die Umsetzung ähnlicher Systemkomponenten in jedem Fall das Schreiben von Programmcode erfordert.

Die Unterschiede der beiden Programmieransätze lassen sich bei der Entwick-lung von LexiCode, eine Anwendung zum Deutschlernen, auch bemerkbar. Die Anwendung, die grob beschrieben eine reine Frontend Anwendung ist und unter anderem Theorie zu den grammatikalischen Grundlagen der deutschen Sprachen übermitteln und diese durch inter-aktive Übungsaufgaben überprüfen soll. LexiCode kann in der Theorie sowohl über HC- als auch LC/NC-Entwicklung umgesetzt werden. Die Fragen, die in der Praxis allerdings beantwortet werden müssen, sind, welcher der beiden Ansätze ermöglicht, dass die konzipierten Systemmerkmale näher an die Anforderungen programmiert werden können und welcher der Ansätze bietet breiter Anpassungsmöglichkeiten für LexiCode an.

## Ziel der Forschung

Die vorliegende Arbeit hat die Aufgabe, nachzuweisen, dass die klassische HC-Entwicklung besser geeignet ist, um die Anforderungen an LexiCode technisch umsetzen zu können, also die geplanten Systemkomponenten im Sinne von UI-Elementen und Funktionalitäten möglichst nah an die geplanten Komponenten zu halten. Es wird erforscht, welcher der beiden Entwicklungsansätze eine Umsetzung ermöglicht, die die Anwendung LexiCode möglichst interaktiv, also für Interaktionen zwischen Nutzern und der Anwendung sorgt, visuell und funktionell anpassungsfähig macht, eine dynamische Navigation zwischen den Seiten der Anwendung ermöglicht, eine Manipulation der UI-Elemente anhand der Nutzeraktivitäten bereitstellt, den Einbau von inzeiligen Eingabefeldern erlaubt und sitzungsunabhängig Nutzerdaten im Browser speichern kann.

Darüber hinaus wird geprüft, welche der beiden Ansätze bessere Anpassungsmöglichkeiten bietet, wenn die Anwendung für mobile Endgeräte erweitert wird.

## Forschungsfragen

Um dieses Ziel zu erreichen, werden die folgenden Forschungsfragen beantwortet:

1. Können sowohl über HC- als auch über LC/NC-Entwicklung Hyperlinks angelegt werden, die die User auch vom Inhalt einer Seite, also nicht nur von der globalen Navigation heraus, auf einer anderen Seite innerhalb der Anwendung umleiten können?

Anwendungen verfügen normalerweise über eine globale Navigation, die die Benutzer auf die Hauptseiten der Anwendung verleiten können. Oftmals werden Links zu anderen Seiten auch im Inhalt einer Seite eingebaut, damit nicht alle verwandten Inhalte auf der gleichen Seite stehen und somit eine Übersichtlichkeit erreicht wird.

1. Über welchen der Entwicklungsansätze können die Seitenelemente bzw. -komponenten effektiver oder überhaupt manipuliert werden, um mehr Nutzungseffekte zu erreichen?

Bei der HC-Entwicklung ist in der Programmiersprache JavaScript das Konzept bzw. Werk-zeug des DOMs bekannt, welches die Möglichkeit bietet, dass an bestehende Elemente der der Anwendung neue Elemente wie Überschriften, Zeilen oder Bilder angeknüpft oder bestehende Elemente gelöscht werden[[1]](#footnote-1). Eine Unterfrage zu dieser Forschungsfrage ist, ob die gleiche Manipulation bei LC/NC vorhanden ist.

1. Ist die Auswahl an Anpassungsmöglichkeiten für Webformulare breiter bei HC- oder LC/NC-Anwendungen?

Auf Webanwendungen sind häufig Formulare zu finden. Diese erfüllen die gleiche Aufgabe wie Formulare in Papierform, mit dem Unterschied, dass diese digitalen Formulare u.a. die abgefragten Daten schneller und direkt an die abfragende Instanz schicken können. Die Anpassungsmöglichkeiten sollen die optische Gestaltung der Formulare und ihre Funktionalität umfangen.

1. Lassen sich sowohl über HC als auch über LC/NC inzeilige Eingabefelder einbauen und wie gut lassen sich diese anpassen?

Beim Sprachenlernen sind Übungsaufgaben üblich, bei denen ein Wort, mehrere Worte oder komplette Satzteile ausgelassen werden und stattdessen sich Eingabefelder auf deren Plätzen befinden.

Solche Felder sind eine Anforderung an LexiCode und lassen die Frage stellen, ob sich diese über HC oder LC/NC einbauen lassen und wenn ja, ob sie angepasst werden können wie z. B. in der Breite, in der Höhe, in der Randdichte oder die Hintergrundfarbe.

1. Können Funktionen in beiden Versionen der Anwendung umgesetzt werden, die benutzerdefinierte Inhalte lokal und sessionsunabhängig, also sitzungsunabhängig speichern können?

Bei der Arbeit mit einer Anwendung werden oft Daten in der Anwendung angegeben, die nach einer kurzen oder sogar längeren Zeit wieder abrufbar sein sollten. Obwohl eine Datenbankanbindung eine gängige Lösung für Datenspeicherung ist, ist für die Speicherung von Daten aller Arten nicht zwingend erforderlich. Antworten zu Übungsaufgaben oder Notizen können z. B. auf dem Browser gespeichert werden und sind trotzdem wieder in der An-wendung zu finden

## These

Die gestellten Forschungsfragen setzen Erwartungen an die Forschung sowie eine klare Vorstellung für die Endergebnisse. Unabhängig von diesen Erwartungen kann die Aussage formuliert werden, dass HC-Entwicklung aus der Sicht der breiten Verfügbarkeit von Werkzeugen zur Umsetzung der jeweiligen Anforderungen einen einfacheren Entwicklungsprozess zulässt. Unter einfacher ist gemeint, dass ein Entwicklungsprozess erwartet wird, bei dem sich die Anforderungen, die im späteren Teil dieser Forschung vorgestellt werden, wie geplant implementieren lassen, d. h. die geplanten Systemkomponenten sehen wie geplant aus und die angeforderte Programmlogik verhält sich wie vorgestellt.

In einer Konstellation, in der sich die Anforderungen nicht erfüllt werden konnten oder mehrere unterschiedliche Methoden und Werkzeuge ausprobiert werden müssen, bis die gewünschten Ergebnisse erreicht werden, kann die These nicht bestätigt werden.

## Wert und Zielpublikum

Die Ergebnisse dieser Forschung leisten einen Beitrag zur Debatte über den effektiveren Entwicklungsansatz. Die verteidigte Ansicht, dass die LC/NC-Entwicklung die Zukunft der Softwareentwicklung wäre, beschränkt sich nur auf den Aspekt der Benutzerfreundlichkeit von LC/NC-Entwicklungsplattformen bzw. die Einfachheit, über solche Plattformen eine Software ins Leben zu rufen.

Zwei Aspekte, die allerdings bei dieser Ansicht oft ignoriert werden, sind die Kompatibilität von LC/NC-Anwendungen mit anderen Systemen und die Verfügbarkeit von Werkzeugen, die weitere Konfigurations- und Anpassungsmöglichkeiten bieten oder auch für Wiederverwendbarkeit des entwickelten Programmcodes, Skalierbarkeit und Wartung ermöglichen. Dafür werden Bibliotheken, Frameworks und IDEs eingesetzt. Die Ergebnisse der Forschung sind für Softwareentwickler interessant und zwar unabhängig von der Erfahrung, Ausbildung, Branche oder Fachgebiet.

## Anwendungsbereich und Einschränkungen

Die Relevanz der Forschungsergebnisse schränkt sich auf kleinere bis mittelgroßen Anwendungen ein, weil dies, zumindest im Rahmen dieser Forschung, die vorgesehene Größe der Anwendung LexiCode ist. Ferner sind die Ergebnisse für Personen interessant, die über wenig Erfahrung als Softwareentwickler verfügen, in eher kleinere Teams oder sogar allein die Entwicklungsarbeiten durchführen werden, mit weniger hardware- und softwaretechnischen Ressourcen arbeiten werden, eine überschaubare Anwendung entwickeln möchten und sich überlegen, ob sie sich für HC oder LC/NC entscheiden sollen. Die Ergebnisse der Forschung sind auch für Lernanwendungen relevant, die andere Inhalte als eine Fremdsprache übermitteln wollen.

Die Forschung befasst sich nicht mit Prozessen und Diensten, die die Speicherung oder das Abrufen von Anwendungs- oder Nutzerdaten von einem System oder Schnittstelle, die im Hintergrund der Anwendung laufen, wie z. B. eine Datenbank, ein Server oder eine Schnittstelle, weil die Umsetzung solcher Lösungen zusammen mit den in den vorherigen Kapiteln beschriebenen Funktionalitäten den vorgesehenen Rahmen dieser Forschung übersprengen würde. Insofern sind die Ergebnisse der Forschung eher für Frontend-Lösungen relevant.

Im Laufe der Forschung und v.a. in dem Kapitel der Ergebnisse wird zugunsten der HC-Entwicklung argumentiert, weil der Entwickler von LexiCode gegnüber diesem Entwicklungsansatz voreingenommen eingestellt ist. Diese Voreingenommenheit stammt aus der Tatsache, dass der Entwickler in seiner bisherigen beruflichen Laufbahn die Glegenheit hatte, mit Anwendungen zu arbeiten, die sowohl entweder über HC- oder LC-NC programmiert wurden. Der Entwickler hat die Erfahrung sammeln können, dass LC-Anwendungen Integrations- und Skalierbarkeitsschwierigkeiten aufbereiten, während HC-Anwendungen mit genügend Programmierkenntnissen diverse Erweiterungsmöglichkeiten bieten.

Für die vorliegende Forschung wurde keine empirische Studie durchgeführt und es wurden auf keiner anderen Art und Weise empirische Daten erzeugt. Stattdessen werden für die Analyse und die Argumentation als Daten die Ergebnisse der Entwicklung der Anforderungen an LexiCode verwendet.

## Struktur des Dokuments

In dem darauffolgenden Teil des Dokuments werden die konkreten Schritte abgewickelt, die zum Forschungsergebnis führen. Im zweiten Kapitel wird der aktuelle Stand der Literatur zum Forschungsthema analysiert, um herauszufinden, wie LC/NC-Entwicklung momentan wahrgenommen wird, inwiefern der Ansatz im Vergleich zu HC-Entwicklung bevorzugt wird, für was für einen Typ Anwendungen er angewendet wird und von welchem Typ Entwickler. Darüber hinaus werden auch wichtige Begriffe im Bereich der Softwareentwicklungen genannt und erklärt.

Im dritten Kapitel wird die Methodik bzw. das Forschungsdesign erläutert und es wird näher darauf eingegangen, wie die Idee von LexiCode einerseits mit HC und andererseits mit LC/NC umgesetzt wird und mit welchen konkreten Technologien bei beiden Ansätzen gearbeitet wird.

Im darauffolgenden Kapitel, im 4. Kapitel, werden die Forschungsergebnisse vorgestellt und genau erläutert, welche der angeforderten Komponenten von LexiCode mit welchem der beiden Entwicklungsansätzen inwiefern oder überhaupt umgesetzt werden konnten. Im nächsten Abschnitt, der Diskussionsabschnitt, werden die Ergebnisse der Forschung kurz zusammengefasst, interpretiert und mit den Forschungserwartungen verglichen. Zusätzlich werden die Beschränkungen der Forschung besprochen.

Im letzten und abschließenden Kapitel wird auf die Forschung kritisch reflektiert und es werden Ideen für weiterführende Forschungen empfohlen.

## 2. Literaturübersicht

Im Folgenden werden relevante Beiträge zum Diskurs über LC/NC-Entwicklung vorgestellt. Die Beiträge unterscheiden sich einerseits nach der Umsetzung, weil manche von denen konkrete Fallstudien präsentieren, während andere rein theoretisch sind und nur die Literatur von anderen Autoren bzw. Forschern, die sich mit dem Thema befasst haben, vergleichen.

Die berücksichtigte Literatur stammt aus den vergangenen drei Jahren, da die LC/NC-Entwicklung erst in dieser Zeitspanne etwas mehr an Bedeutung zugewonnen hat.

Die Auswahl der Beiträge erfolgte nach einer einfachen Suche nach dem Begriff *Low-Code Entwicklung* und ergab auch Quellen, die sich zwar auf No-Code Entwicklung konzentrieren, allerdings über KI-Chatbots und andere ähnliche Technologien. Da die Entwicklung mittels solcher Technologien keinen Einsatz in dieser Forschung finden, wurden solche Quellen auch nicht berücksichtigt.

## 2.1 Hintergrundinformationen

Nachfolgend werden einige wichtige Begriffe definiert, die im Laufe der Forschung auftreten werden.

**Citizen Developer**: Anwender, die über formale Programmierkenntnisse nicht verfügen[[2]](#footnote-2).

**LCAP**: Anwendungen, die durch visuelles Zusammenfügen vorgefertigter Software-Bausteine erstellt und individuell angepasst werden können[[3]](#footnote-3).

**Time-to-Market**: die Zeit, die vergeht, bis die Entwicklung einer Produktidee oder ein Serviceangebot reif genug ist, damit eine Platzierung des Produktes bzw. der Dientleistung am Markt erfolgen kann[[4]](#footnote-4).

**Minimum Viable Product (MVP)**: der minimalst mögliche Funktionsumfang eines Produkts oder einer Dienstleistung. Das Produkt / die Dienstleistung werden erst dann veröffentlicht, wenn diese Entwicklungsstufe erreicht wird, was an dem Treffen zwischen minimalem Aufwand und qualitativem Feedback zu erkennen ist[[5]](#footnote-5).

**Earned-value-Ansatz**: Softwareentwicklungsansatz, bei dem die einzelnen Anforderungen mit *Value-Points* gewichtet werden, anstatt gleich behandelt zu werden. Die Anforderungen mit den meisten Wertpunkten werden vorgezogen, während die mit den wenigsten Punkten zurückgestellt werden. (ähnlich zu der agilen Softwareentwicklung, bei der Anforderungen je nach ihrem Wert für den Kunden abgearbeitet werden) (Sneed & Jungmayr, 2011).

**Value-Driven Software-Engineering (wertgetriebenes Software-Engineering)**: Ein Softwareentwicklungsprojekt soll wirtschaftlich gerechtfertigt werden, indem die Kosten mit dem Nutzen eines Produkts abgeglichen werden (Sneed & Jungmayr, 2011).

## 2.2 Bestehende Modelle und Theorien

Ein Konzept, welches für die vorliegende Forschung Relevanz hat, ist das magische Dreieck des Projektmanagements bzw. der sog. *Iron Triangle* (Atkinson, 1999). Dieses Dreieck umfasst die Dimensionen Kosten, Zeit und Qualität und hat eine Bedeutung für die vorliegende Forschung, da der Vergleich der beiden Entwicklungsansätze auch ein Projekt darstellt, für welches Ressourcen wie Zeit eingesetzt werden und eine gewisse Qualität des Endergebnisses erwartet wird:

Ein Bild, das Reihe, Screenshot, Dreieck, Astronomie enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 1. (Iron Triangle)[[6]](#footnote-6)

Baumgarten et al. (2024) haben für ihre Forschung ein Modell eingesetzt, welches auf dem *Iron Triangle* von Atkinson basiert und dieses sogar erweitert. Sie bilden in ihrer Forschung ein sog. magisches Viereck ab, das neben den drei Dimensionen des Dreiecks noch die vierte Dimension der Flexibilität einschließt und somit ein sog. *magisches Viereck* darstellt (Baumgarten et al., 2024):



Abbildung 2. (Magisches Viereck)[[7]](#footnote-7)

Die von Baumgarten et al. (2024) vorgestellte Abbildung des magischen Vierecks hat auch für die vorliegende Forschung eine Bedeutung, weil auch die Ergebnisse der vorliegenden Forschung auf dem magischen Viereck abgebildet werden können, um zu veranschaulichen, in welchen Dimensionen die HC-Entwicklung mehr Vorteile bietet im Vergleich zu der LC-Entwicklung.

Auf dem magischen Viereck ist unter der Dimension der Flexibilität auch die Kompatibilität des jeweiligen Entwicklungsansatzes mit Drittanbietersystemen oder Schnittstellen bzw. Entwicklungsumgebungen für Backend-Lösungen wie z. B. Server oder Datenbanken zu verstehen. Solche Lösungen, Systeme und Komponenten werden im Rahmen der vorliegenden Forschung nicht entwickelt oder eingesetzt.

Die von Baumgarten et al. eingesetzte Abbildung des magischen Vierecks veranschaulicht die Ergebnisse der literarischen Forschung, die sie durchgeführt haben und bei der sie unterschiedliche Beiträge zum Thema LC/NC-Entwicklung berücksichtigt haben.

Diese sind z. B. Foren, in denen Low-Code bezogene Diskussionen stattfinden, eine Forschung, die nach dem Grund des Einsatzes von Low-Code fragt, eine Forschung, die die Treiber und Bremsfaktoren einer LCAP-Einführung untersucht und eine Forschung, die sich mit den technischen Herausforderungen einer LCAP-basierten Applikationsentwicklung beschäftigt. Je nachdem, wie die Auswirkungen der LC/NC-Entwicklung von den Autoren dieser Forschungen wahrgenommen worden sind, haben Baumgarten et al. die identifizierten Praxiserfahrungen der entsprechenden Dimension auf dem Viereck zugeordnet und mit einem Faktor von „+1“ auf die Dimension abgebildet, wenn die Praxiserfahrung als Verbesserung zur HC-Entwicklung wahrgenommen wurde. Mit „-1“ wurde eine entgegengesetzte Praxiserfahrung abgebildet und mit „0“ eine neutrale (Baumgarten et al., 2014).

Auf Abb. 2. sind die Ergebnisse der von Baumgarten et al. referenzierten Forschungen geschildert, die auf Zugewinne in der Dimension der Zeit hindeuten.

Dabei haben sich Baumgarten et al. darauf konzentriert, wie lang die Iterationszyklen bei der Entwicklung einer LCAP sind, wie schnell das MVP und wie schnell das Time-To-Market erreicht werden kann. Die Ergebnisse der vorliegenden Forschung werden im späteren Kapitel über die Ergebnisse ebenfalls visuell auf dem magischen Viereck dargestellt, damit veranschaulicht werden kann, in welchen Dimensionen die LC/NC- und in welchen Dimensionen HC-Version von LexiCode zu- oder abgewonnen hat.

Ein weiterer theoretischer Hintergrund, der die vorliegende Forschung unterstützt, äußert sich in der Betrachtung der Messung eines Softwareentwicklungsprojektes, um die (weitere) Durchführung des Projektes zu rechtfertigen. Sneed und Jungmayr (2011)[[8]](#footnote-8) heben die Vorteile des Earned-Value-Ansatzes hervor, bemängeln jedoch, dass bei diesem Ansatz die Entscheidung nicht unterstützt wird, ob ein (Teil)Projekt überhaupt angegangen werden soll und falls ja, welche Anforderungen erfüllt werde sollen und welche nicht.

Als Lösung bieten sie den Ansatz von Boehm und Huang (zitiert nach Boehm & Huang, 2003), die sog. Value-Driven Software-Engineering, der sich auf die wirtschaftliche Rechtfertigung der Softwareentwicklung konzentriert. Der Ansatz besagt, dass jede Aufgabe in einem Projekt nur so viel wert wie ihr Ergebnis ist und dieser Wert größer als die Kosten sein muss, was wiederum bedeutet, dass der ROI positiv sein muss.

Boehm und Huang (2003) berechnen den ROI wie folgt:

Bei der Entwicklung einer Software ist es angesichts der Kosten wichtig, möglichst fehlerfrei zu entwickeln, da die nachträgliche Fehlerbehebung zusätzliche Kosten verursacht, die 17-26 % der gesamten Lebenszykluskosten einer Software ausmachen können. Umfangreiches Testen der Software vor der Veröffentlichung kann zwar die späteren Kosten für die Fehlerbehebung reduzieren, verlängert allerdings die Entwicklungszeit vor der Veröffentlichung, was ebenfalls zu einem Werteverlust führt.

Daher hat jede Projektleitung abzuwägen, ob der durch die zusätzlich benötigte Entwicklungszeit verursachte oder der durch die Fehlerhaftigkeit bedingte Werteverlust größer ist[[9]](#footnote-9).

Es werden mehrere Ansätze empfohlen, um die Fehler bei der Entwicklung einer Software zu reduzieren und zwei von denen, die in der vorliegenden Studie angewendet wurden, sind einerseits die Wiederverwendung bestehender Softwarekomponenten, die im Fall von LexiCode Systemdesigns, Komponenten wie UI-Elemente wie Eingabefelder, Forms oder Tests zur Wissensabfrage sind.

Ein weiterer Ansatz ist das Testen dieser Systemkomponenten, das sog. Value-Driven Testing, das als Maßnahme stark empfohlen wird, weil umfangreiches Testen in der Entwicklungsphase, sowohl für Modul- als auch für Systemtests, die Kosten für die Fehlerbehebung in der Wartungsphase reduzieren kann. Dies ist vor allem daran zu erkennen, dass die durchschnittlichen Fehlerbehebungskosten als zeitlicher Aufwand in der Testphase innerhalb der Entwicklung bis zu maximal 15 Tagen dauern kann.

In der produktiven Phase eines Softwareprodukts je nach der Größe des Betriebes, indem die Software eingesetzt wird, mindestens 13 und im schlimmsten Falle 92 Tage dauern kann[[10]](#footnote-10):

Ein Bild, das Screenshot, Diagramm, Reihe, Astronomie enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 3. (Tagesaufwand für die Fehlerbehebung)[[11]](#footnote-11)

Die These von Sneed und Jungmayr (2011), dass allein das Finden von Fehlern den Aufwand für das Testen in Bezug auf den eingesparten Folgekosten rechtfertigen kann, weist eine Schwachstelle auf, weil die Wahrscheinlichkeit besteht, dass die Fehlerkosten niedriger als die Testkosten sind, was den Testaufwand nicht rechtfertigt und daher den Bedarf auslöst, zuerst Information über die Information zu verfügen, wie der Testnutzen ermittelt wird und wie die Testkosten im Voraus zu berechnen sind. Um dies zu erreichen bzw. die Testkosten zu ermitteln, haben sie die folgende Formel zusammengestellt:

Die wichtigsten Parameter dabei sind die Anzahl der Solltestfälle (TF), die angestrebte Testüberdeckung (TU), die manuelle Testproduktivität (TP), der Grad der Testautomatisierung (TA), die Testbarkeitsmetrik (TB), die mittlere Testbarkeit (MT) der Testskalierungsexponent (TE) und der Justierungsfaktor (AF).

Das Ergebnis ermittelt eine geschätzte Anzahl von 700 Personentagen, die nötig wäre, um das System während der Entwicklung und dementsprechend vor der Veröffentlichung zu testen. Darüber hinaus wird es für sinnvoll gehalten, den Test-ROI zu ermitteln. Für den Test-ROI ist die Anzahl der erwarteten Fehler zu berücksichtigen, für die ein Wert von 6 Fehlern pro 1000 Zeilen Code, also insgesamt 600 zu erwartende Fehler angenommen wird[[12]](#footnote-12). Laut einer weiteren Annahme werden mindestens zwei Drittel der vorhandenen Fehler durch systematische Tests aufgedeckt, was die Fehleranzahl auf 200 reduziert und den größeren Anteil von 400 Fehlern nicht erst der Wartungsphase entfernt werden muss.

Die Fehler lassen sich nach Ursprung wie folgt verteilen: 40% Anforderungsfehler, 30 % Entwurfsfehler und 30 % Codefehler. Bei der Behebung dieser Fehler wird mit einem geschätzten Aufwand nach Personentagen gearbeitet, der folgendermaßen abläuft: 2,5 Personentage für Anforderungsfehler, 1,0 Personentag für Entwurfsfehler und 0,5 Personentage für Codefehler[[13]](#footnote-13).

Die Berechnung des geschätzten Aufwands für die jeweilige Fehlerquelle ergibt die Fehlerbehebungskosten, die zusammen mit den Kosten des wahrscheinlichen Schadens als Summe den Einsparungspotenzial ergeben, aus dem die Kosten des Testaufwands (Personentage x Tagessatz) subtrahiert und noch einmal durch die Testkosten dividiert werden[[14]](#footnote-14):

ROI = (Nutzen – Kosten) / Kosten.

## 2.3 Ausgewählte Fallstudien

Für die vorliegende Forschung spielte es eine Rolle, bestehende Literatur zum Thema HC- und LC/NC-Entwicklung zu berückichtigen. Zu der berücksichtigten Literatur zählen weitere eigene experimentenbasierte Forschungen, Literaturanalysen und weitere Quellen.

Ajimati, Carroll und Maher (2025) beschäftigen sich mit der Einführung von LC/NC-Technologien, die Bedeutung von Citizen Developern und deren Beitrag zur digitalen Transformation. Sie analysieren die Vorteile der LC/NC-Entwicklung wie die Effizienzsteigerung und die Herausforderungen wie Sicherheitsrisiken. Sie entwickeln auch ein theoretisches Rahmenwerk und identifizieren Forschungslücken für zukünftige Untersuchungen.

Aveiro, Freitas, Cunha, Quintal und Almeida (2023) vergleichen die HC- und LC/NC-Entwicklung anhand des Systems nexusBRaNT, welches in einem Forschungsprojekt für kognitive Rehabilitation eingesetzt werden soll. In der Studie wird festgestellt, dass durch LC der Arbeitsaufwand stark reduziert und die Systemkomplexität verringert wird. Dies ermöglicht Unternehmen, effizienter und flexibler Anwendungen zu entwickeln.

Böhler (2023) thematisiert die Revolution in der Softwareentwicklung durch den LC/NC-Ansatz. Er argumentiert, dass die LC-Technologie die Entwicklungszeit verkürzt, IT-Ressourcen entlastet und Unternehmen mehr Flexibilität bietet. Er setzt sich damit auseinander, dass Nicht-Programmierer durch LC-Entwicklung Anwendungen erstellen und somit in Unternehmen die Effizienz gesteigert und die Digitalisierung beschleunigt wird.

Di Ruscio et al. (2022) vergleichen die LC-Entwicklung und das MDE und zeigen, dass beide Entwicklungsansätze Modellbasierung und Automatisierung nutzen, sich jedoch anhand der Flexibilität und der Zielgruppe unterscheiden. Während die LC/NC-Entwicklung mehr Wert auf Benutzerfreundlichkeit und schnelle Entwicklung setzt, bietet MDE mehr Anpassungsfähigkeiten und erfordert tiefergehendes Fachwissen.

Elshan und Binzer (2024) zeigen in ihrer eigenen Studie, dass LC/NC-Plattformen die digitale Transformation unterstützen, indem sie für eine sog. Demokratie in der Softwareentwicklung sorgen, Innovationsprozesse beschleunigen und die Arbeit der IT-Abteilungen erleichtern. Obwohl bei der LC-Technologie gewisse technische Einschränkungen und Governance-Herausforderungen vorhanden sind, können Unternehmen durch transparente Strategien, Schulungen und KI-Integration, die in der vorliegenden Forschung nicht näher behandelt wird, das Potenzial dieser Technologie vollständig ausschöpfen.

Frank (2023) erklärt, dass mittels LC/NC-Entwicklung ERP-Systeme schneller und flexibler angepasst und implementiert werden können, womit maßgeschneiderte Lösungen effizienter realisiert werden können. Damit LC/NC-Plattformen erfolgreich eingesetzt werden, müssen klare Ziele definiert werden und die Plattformen sorgfältig gesteuert werden. Dadurch können unkontrollierte Anpassungen und Ineffizienzen vermieden werden.

Gialitakis, Tsakalidis, Nikos und Vergidis (2024) fokussieren sich auf die Beschleunigung der Prototypenerstellung prozessorientierter Anwendungen über LC-PLattformen im griechischen öffentlichen Sektor und die Effizienzsteigerung dabei. Sie sind der Überzeugung, dass Low-Code als Lösung viel Potenzial für öffentliche Einrichtungen verspricht und digitale Transformationsziele flexibler und kosteneffizienter erreicht.

Hensen (2023) erläutert die schnellere und kosteneffiziente Softwareentwicklung durch LC/NC-Plattformen und fördert die Zusammenarbeit zwischen der IT und den Fachabteilungen und betont, dass trotz der Vortile der LC-Entwicklung klare Governance und Integrationsstrategien erforderlich sind. Laut seiner Forschung hängt eine erfolgreiche Implementierung von Schulungen, strategischer Planung und kontinuierlicher Evaluierung ab. Nur so können Qualität und Sicherheit gewährleistet werden.

Jit, Chen, Ean, Pei, King und Chek (2024) haben die Zugänglichkeit von LC/NC-Plattformen für Lerner untersucht und welchen Lernpotenzial sie bieten. Den Ergebnissen ist zu entnehmen, dass aus diesen Plattformen der Zugang zur Softwareentwicklung erleichtert werden kann.

Liu, Jiang, Guo, Chen und Qiao (2023) analysieren 974 aufgefundene Fehler in den vier LC/NC-Plattformen OutSystems, Mendix, Appsmith und Budibase und identifizieren häufige Ursachen und Auswirken der Bugs. Laut den Ergebnissen treten 60 % der Fehler in der Design- und Spezifikationsphase auf, mehr als 37 % der Bugs verursachen unerwartetes Verhalten ohne klare Fehlermeldungen und UI-Fehler sind besonders problematisch, da LC-Plattformen spezifische Merkmale haben.

Schreiner (2024) stellt in seiner Forschung die verkürzten Entwicklungszeiten und den reduzierten Bedarf an spezialisierten Entwickler durch die visuellen Entwicklungswerkzeugen und vorgefertigte Komponenten der LC-Plattformen. Er ist der Ansicht, dass Unternehmen von einer höheren Flexibilität, schnelleren Anpassungen an Geschäftsanforderungen und einer beschleunigten Markteinführung ihrer Anwendungen profitieren.

## 3. Forschungsdesign

Um die gewünschten Ergebnisse hinter der vorliegenden Forschung zu erzielen bzw. die Forschungsfragen zu beantworten und die Forschungsthese zu bestätigen, wird ein Case Study durchgeführt, das auf dem in der Softwareentwicklung bekannten A/B-Test basiert. Die bereits erwähnte Anwendung zum Deutschlernen, LexiCode, wurde von einer Person im Umfang der Anforderungen, die später erläutert werden, einmal als eine klassische HC-Lösung und parallel als LC/NC-Lösung umgesetzt. Es wurde nicht darauf abgezielt, eine vollständige, ausgereifte und veröffentlichte Anwendung zu entwickeln, sondern vielmehr wurde es angestrebt, die festgelegten Anforderungen an der Anwendung gemäß der beiden Entwicklungsansätze so umzusetzen, dass sie in einer grundlegenden Form der angeforderten Funktionalität betriebsbereit sind.

Somit hat es auch gereicht, dass jede der Anforderungen, wie z. B. die lokale Navigation über Hyperlinks, in den beiden Instanzen der Anwendung nur einmal umgesetzt wird, damit mit überschaubarem Aufwand festgestellt werden kann, ob sich die Anforderungen in der jeweiligen Entwicklungstechnologie überhaupt umsetzen lassen.

Sowohl die HC- als auch die LC/NC-Version von LexiCode wurden von der gleichen Person umgesetzt, da die Anforderungen an LexiCode innerhalb dieser Forschung von der Anzahl und der Schwierigkeit der Umsetzung her überschaubar sind und somit sich die Beteiligung von weiteren Entwicklern erübrigt hat. Die Implementierung durch einen einzigen Entwickler hat den Kommunikationsaufwand auf null reduziert und dadurch mehr Zeit zur Umsetzung des Projekts gewonnen, weil ein großer Teil der Arbeit in einem Team aus Kommunikation besteht, welche in manchen Fällen nicht zwangsläufig zu produktiven Ergebnissen führt.

Die Zeit, die in den Entwicklungsprozess eingeflossen ist, wurde für diese Forschung nicht gemessen, da es nicht vorhersehbar war, ob sich alle Anforderungen überhaupt umsetzen lassen. Die in den nächsten Kapiteln beschriebenen Anforderungen wurden so konzipiert, dass sie eine grundlegende Digitalisierung des Prozesses des Deutschlernens befähigen, aber sich an den Erfahrungs- und Kenntnisstand des Entwicklers richten, damit die Entwicklungszeit sich nicht durch den Erwerb neuer Programmierkompetenzen zu stark verlängert.

Das Konzept des A/B-Testens wurde an den Bedingungen der vorliegenden Forschung angepasst. Anstatt die beiden Versionen von LexiCode unterschiedlichen Menschen vorzustellen, um dann anhand der Konversionsrate dieser Probanden zu prüfen, welche der beiden Versionen effektiver ist (Siroker & Koomen, 2013), werden innerhalb der vorliegenden Forschung die beiden Anwendungsversionen von der von dem Entwickler verglichen und selbstverständlich nicht um die Konversionsrate des Entwicklers zu prüfen, sondern einfach um herauszufinden, ob sich eine leistungsfähigere Anwendung mittels HC oder LC/NC bauen lässt.

Da der A/B-Test zu einem zunehmend allgegenwärtigen und kritischen Teil von Onlinegeschäften geworden ist[[15]](#footnote-15) und LexiCode mit etwaigem Ausbau das Potenzial hat, kommerziell bei einer hohen Nutzerzahl eingesetzt zu werden, eignet sich der A/B-Test als Forschungsmethodik. Des Weiteren eignet sich das A/B-Testen, insbesondere das Werk von Siroker und Koomen (2013) für die Methodik der vorliegenden Forschung, weil sie in ihrer Herausgabe die Bedeutung grundlegender Komponenten vieler Anwendungen und Websites, wie z. B. Calls-to-Action, Inhalte, visuelle Medien, Seitennavigationen und Formulare, erklären und zu jeder Komponente Ratschläge liefern, wie diese am besten gebaut und getestet werden kann.

## 3.1 Anforderungsanalyse

Die Anforderungen, die LexiCode als Anwendung zum Lernen von Deutsch zu erfüllen hat, richten sich nach dem typischen Lernweg von Menschen, die nicht nur die deutsche Sprache, sondern jede für sie fremde Sprache beherrschen wollen. Diese potenziellen Lernenden stellen die wichtigsten Stakeholder von LexiCode nach dem Entwickler dar. Dieser soz. *typischer* Lernweg ist bei allen Lernenden hinsichtlich der Reihenfolge der gelernten Inhalte gleich.

Die gelernten Inhalte teilen sich bei dem Fremdsprachenerwerb nach Niveaus auf, variieren von Anfänger (A1) bis zu Fortgeschrittenen (C1) und schließen Grundlegende Sprachkompetenzen ein, wie einfache Begrüßungen und sich vorstellen als Anfänger bis zum Schreiben komplexer Aufsätze und Studieren von Fachliteratur als Fortgeschrittener. Demnach muss eine Anwendung zum Deutschlernen, in diesem Fall, LexiCode, so konstruiert sein, dass sie ihren Nutzern einen Lernbereich bereitstellt, in dem die Lerninhalte nach Niveau aufgeteilt sind, über Theorieeinheiten zuerst als reine Information verinnerlicht und danach durch entsprechende Praxisaufgaben geübt werden können, die idealerweise auch eine Bewertung auf die Leistung liefern.

Da in dem bis heute am meisten genutzten Medium zum Lernen von Fremdsprachen, das Lehrbuch, durch die einzelnen Seiten einfach durchgeblättert werden kann, muss auch LexiCode ermöglichen, dass Benutzer mit wenig Anstrengung die Inhalte durchstöbern können. Die Schreibfelder in jedem Lehrbuch befähigen die Lernenden ihre Notizen, Vermerke und Übungsaufgaben direkt im Lehrbuch zu hinterlassen – eine Eigenschaft, die auch in LexiCode vorhanden sein sollte. Fragebögen und Übungen, bei denen die Antworten in Schreibfeldern innerhalb der Fragezeile eingetragen werden, stellen häufige Aufgabenarten in herkömmlichen, gedruckten Lehrbüchern, die entsprechend von einer Anwendung zum Deutschlernen auch bereitgestellt werden sollen.

Die bereits erwähnten Vermerke, die in einem Lehrbuch hinterlassen werden, werden von den Lernenden wiederholt aufgeschlagen, um Gelerntes zu wiederholen oder Übungen nochmal neu zu bearbeiten. Daher müssen diese schriftlichen Informationen auch in LexiCode zu jeder Zeit abrufbar sein.

## 3.2 Spezifikation der Anforderungen

Die in dem letzten Kapitel aufgezählten Anforderungen lassen sich konkretisieren und in den Unterkategorien der funktionalen, der nicht-funktionalen und der technischen Anforderungen untergliedern. Die funktionalen Anforderungen beschreiben die spezifischen Funktionen und Verhaltensweisen, die die Software erfüllen muss:

Tabelle 1. Funktionale Anforderungen



F1 muss sicherstellen, dass die Benutzer von LexiCode über Navigationsmenus von einer Inhaltsseite zu einer anderen navigieren können und bei Bedarf noch zur ursprünglichen Seite zurücknavigieren können. Diese Navigation soll uneingeschränkt möglich sein und optimalerweise über Hyperlinks erfolgen.

Unter der Anforderung F2 ist zu verstehen, dass die Anwendung den Nutzern nicht nur ermöglicht, Inhalte statisch lesen zu können, sondern diese zusätzlich zu ergänzen und zu erweitern. Diese Anforderung lässt sich mit der nächsten Anforderung, F3, weiter spezifizieren, da F3 leisten soll, dass die Benutzer auf der Anwendung Daten eingeben können. Diese Daten sind Antworten zu den Übungsaufgaben, die von LexiCode bewertet werden sollen, damit die Benutzer aktiv erfahren können, ob sie die Aufgaben richtig oder falsch bearbeitet haben.

Die Datenspeicherung, die mit F4 angefordert wird, soll nur die Speicherung der besagten Antworten zu den Übungsaufgaben ermöglichen. Diese Speicherung soll unabhängig von einem Server erfolgen, weil dies, wie in einem Früheren Kapitel erwähnt, nicht im Rahmen der vorliegenden Case Study liegt. Die gespeicherten Daten sollen bei späteren Sitzungen bzw. Lernvorgängen weiterhin aufrufbar sein und in unveränderter Form vorliegen.

F5 soll die User-Experience bequemer gestalten, indem die Benutzer die Möglichkeit haben, den Hintergrund der Anwendung auf einer dunklen Farbe umzuschalten und somit den sog. *Dark Mode* aktivieren. Diese Umschaltung wird per Button erfolgen, der nicht tief in einem Einstellungsmenu verborgen, sondern gleich auf der Benutzeroberfläche zu finden sein wird.

Die nachfolgend beschriebenen nicht-funktionalen Anforderungen beschreiben die Qualität von LexiCode in Bezug auf Benutzerfreundlichkeit und Zweckmäßigkeit:

Tabelle 2. Nicht-funktionale Anforderungen



Die Responsivität in Anforderung NF1 soll zum Ergebnis führen, dass die Anwendung, die als eine Desktop-First Anwendung, d. h. primär für die Nutzung mit Desktop-Rechnern und Laptops, entwickelt wird, sich ebenfalls an den Bildschirmauflösungen kleinerer mobilen Endgeräte wie Smartphones oder Tablets so anpassen kann, dass die Inhalte noch gut ersichtlich und lesbar sind.

Da die gezielte Nutzergruppe aus Menschen bestehen soll, die bereits eine oder mehrere Fremdsprachen gelernt haben oder erst mit LexiCode eine lernen und die App zu privaten Lernzwecken verwenden werden, soll die Anwendung nach NF2 intuitiv zu bedienen sein. Somit soll sich die Anwendung von großen, beruflich eingesetzten Lösungen wie ERP- oder Enterprise Systeme unterscheiden, die oftmals intensive Schulungen voraussetzen, bevor sie produktiv verwendet werden können.

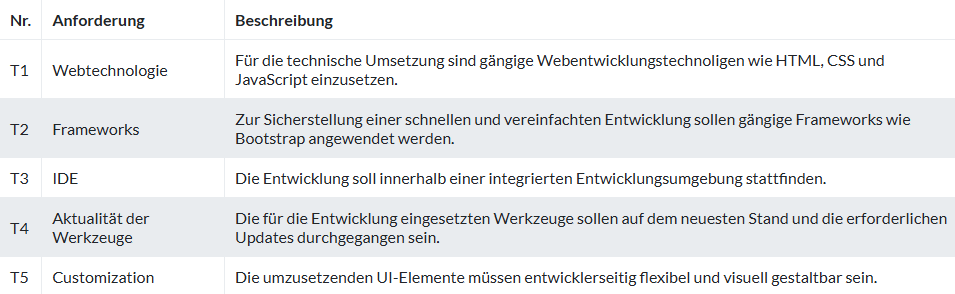
NF3 fordert an, dass die Anwendung auf der Benutzeroberfläche eine gute Kombination aus einfachem, nachvollziehbarem Design, geeigneten Farben, passender Schriftgröße und überlegter visueller Hierarchie bietet.

In NF4 wird angefordert, dass LexiCode vom Design und von dem Benutzererlebnis her an den herkömmlichen, gedruckten Lehrbüchern erinnert, da dieses Medium meistens von Sprachlernenden bereits eingesetzt worden ist. Somit wird verlangt, dass die Theorieinhalte und Praxisübungen so aussehen und strukturiert sind, wie sie in Lehrbüchern in Papierform zu finden sind.

Die letzte, nicht-funktionale Anforderung, NF5, hat den Zweck, Lernenden unabhängig von ihrem Hintergrund zu ermöglichen, mit LexiCode arbeiten zu können. Da Englisch die am meisten gelernte Fremdsprache ist, werden Benutzer mit LexiCode lernen können, da die Inhalte auf Englisch übersetzt sein werden.

Die letzte Gruppe der Anforderungen an LexiCode, die technischen Anforderungen, bezieht sich auf die Technologien, Werkzeuge und Architekturen, die für die Entwicklung der Software verwendet werden:

Tabelle 3. Technische Anforderungen



Anforderung T1 wurde festgelegt, weil LexiCode als Webanwendung über einen Browser zugänglich sein soll, ohne dass Nutzer die Anwendung auf ihre Rechner installieren müssen. Diese grundlegende Entwicklung der Anwendung über die in den Anforderungen angegebenen Webentwicklungstechnologien wird auch für zukünftige Projekte ermöglichen, dass LexiCode als eine native Applikation für mobile Endgeräte erweitert werden kann.

Über T2 wird verlangt, dass bei der Entwicklung Frameworks eingesetzt werden, weil diese schnelleren Lösungen für gängige Aufgaben in der Entwicklung einer Anwendung bieten. Solche Aufgaben sind z. B. die Einstellung der Textausrichtung, Innen- und Außenabstände der UI-Elemente und UI-Elemente responsiv machen.

Die Entwicklung soll zudem, wie in Anforderung T3 angegeben, in einer integrierten Entwicklungsumgebung oder auch *DIE* genannt stattfinden, da diese durch diverse Eigenschaften den Entwicklungsprozess erleichtern und unterstützen. Diese Eigenschaften sind z. B. farbliche Hervorhebung der Code-Schlüsselwörter, Ergänzungsvorschläge und verkürzte Schreibweise von Code-Befehlen. Das Schreiben von Programmcode ist auch über reine Texteditoren wie Notepad oder MS Office Word möglich, aber da diese über die als Beispiel aufgezählten Eigenschaften einer *IDE* nicht verfügen, werden sie in diesem Case Study für die Entwicklung nicht eingesetzt.

Die eingesetzten Werkzeuge sollen zudem, wie in T4 angegeben, auf einem aktuellen Stand sein, weil die aktuellen Versionen von Programmiersprachen, Frameworks und *IDE*s alle Standardfeatures weiterhin unterstützen und sogar neue anbieten, die die Entwicklung vereinfachen und die Performance optimieren.

Die letzte technische Anforderung, T5, benötigt, dass bei der Entwicklung der Anwendung ausreichend Möglichkeiten zur visuellen und funktionalen Anpassung vorhanden sind.

## 3.3 Technologie

Die im Rahmen der vorliegenden Forschung eingesetzten Technologien lassen sich in zwei Bereiche unterteilen: die Technologien, die für die Entwicklung der HC-Version von LexiCode herangezogen wurden, und die Technologien, die für die LC/NC-Version verwendet wurden. Um die HC-Version zu verwirklichen, wurden die gängigen Webentwicklungstechnologien HTML, CSS und JavaScript ausgewählt.

Die Verwendung von HTML begründet sich dadurch, dass die Markup-Sprache Elemente für viele gängige Anwendungskomponenten bietet wie Formulare, Listen, Texte, Buttons, Überschriften und andere. Zudem können über HTML semantische Elemente wie *<nav>* für Navigation, *<article>* für Artikel oder *<section>* in Anspruch genommen werden, die den Code leichter zu verstehen machen und den gebauten Elementen auch eine semantische Bedeutung zuweisen.

Da mittels HTML gestaltete Anwendungen mit CSS ergänzt werden, wurde logischerweise CSS auch in dem vorliegenden Projekt verwendet. CSS bietet vielfältige Anpassungsmöglichkeiten für HTML-Elemente inklusive Breite und Höhe der Elemente, Zeilenabstände der Texte, Schriftgröße, Farben, Schatteneffekte, Animationen und viele andere. Dieses breite Spektrum an Anpassungsmöglichkeiten eignen CSS zur Gestaltung einer ansprechbaren und modernen Benutzeroberfläche.

Damit LexiCode auch Funktionen ausführen kann, wurde die Funktionslogik der Anwendung über die Programmiersprache JavaScript geschrieben, die oft in Kombination mit HTML und CSS verwendet wird und aufgrund ihrer modernen ES6 Eigenschaften einen vereinfachten Codesyntax wie Pfeilfunktionen ermöglicht. Des Weiteren lässt sich über JavaScript die bereits angekündigte DOM-Manipulation realisieren, die aufgrund ihrer Möglichkeiten, die UI-Elemente zu klonen, löschen oder neu zu erstellen, ein mächtiges Werkzeug der Webentwicklung ist, und aus diesem Grund JavaScript häufig den Ruf eines bevorzugten Webentwicklungstools verleiht.

Die Entwicklung der HC-Version von LexiCode erfolgt in der Entwicklungsumgebung Visual Studio Code, die, wie andere bekannte Entwicklungsumgebungen, viele Funktionen zur Verfügung stellt, die gängige Projektverwaltungsaufgaben vereinfachen. Diese sind z. B. das Erstellen von neuen Projektordnern und Dokumenten, die Verknüpfung mit externen Systemen zur Versionsverwaltung wie *Git*, die Ausführung von Terminal Befehlen bei der Arbeit mit Paketmanagern wie *npm* und die Arbeit mit Extensions.

Zu den Extensions, die in diesem Projekt angewendet wurden, zählen *Go Live,* der dafür sorgt, dass das aktuelle Projekt nach einem einzelnen Mausklick im Browser ausgeführt wird, *Monokai Pro*, mithilfe deren benutzerdefinierte Farben für die Schlüsselwörter im Programmcode hinterlegt werden können, und *Prettier*, der den Programmcode automatisch formatiert, damit dies der Entwickler nicht tun muss.

Die Programmierung der LC/NC-Version von LexiCode wurde auf der Plattform *Zoho Creator* durchgeführt. Diese Plattform wurde für die Low-Code-Anwendung ausgewählt, da sie, laut Angaben auf der Homepage des Anbieters *Zoho*, als die „beste Low-Code-Plattform“ bewertet wird[[16]](#footnote-16).

Zoho Creator bietet die bisher angekündigten Funktionalitäten, die eine typische LC/NC-Plattform kennzeichnen, wie exemplarisch die Möglichkeit, Layouts und Komponenten über Drag-and-Drop zu erzeugen.

In der integrierten Entwicklungsumgebung von Zoho Creator, die als *Page Builder* bekannt ist, können einerseits über Drag-and-Drop Diagramme, Suchfelder, Formulare, Berichte, Buttons und andere gängige Komponenten erstellt werden. Andererseits können über Programmcode Anpassungen vorgenommen werden, die über die Drag-and-Drop Bausteine hinausgehen. Dies ist in Bezug auf die Seitenlayouts und -designs über HTML-Snippets möglich. Über *Deluge*, die Programmiersprache von Zoho Creator, lässt sich Funktionslogik programmieren.

Von dem Anbieter Zoho wird auf seiner Homepage angegeben, dass mittels seiner Plattform Anwendungen für jeden Verwendungszweck erstellt werden können und die Entwicklung mittels Zoho Creator 10-mal schneller in der jeder Phase der Lebenszyklusverwaltung von Apps ist bei anderen Tools[[17]](#footnote-17). Zoho erklären diese Geschwindigkeit im Vergleich zu anderen Anbietern mit der Tatsache, dass für eine durch ihre Plattform gebaute Anwendung kein Setup benötigt wird, eine Bereitstellung mit einem Klick möglich und kein Wartungsaufwand nötig ist.

Die aufgezählten Merkmale einer LC/NC-Plattform, die auch bei Zoho Creator zugänglich sind, zusammen mit der umfassenden Dokumentation, die Tutorials zu der Plattform und die Geschwindigkeit bei der Entwicklung sind die Gründe, warum Zoho Creator eine gute Auswahl für die Entwicklung der LC/NC-Version von LexiCode ist.

## 3.4 Analyse

Der in der theoretischen Fundierung beschriebene Ansatz des Value-Driven Testings wird auch bei der Entwicklung von LexiCode angewendet. Wie bereits erläutert, ist es bei diesem Ansatz von Bedeutung, den Aufwand, den wahrscheinlichen Schaden und aufgrund dieser Metriken den Test-ROI zu ermitteln. Der Aufwand, der sich über die in der theoretischen Grundlage angegebenen Formel ermitteln lässt, wird in Personentagen gemessen.

Der wahrscheinliche Schaden wird als die Summe des minimalen und maximalen Schadens geteilt durch 2 erklärt[[18]](#footnote-18):

Für den maximalen Schaden rechnen Sneed und Jungmayr (2011) mit den als Beispiel bereits erwähnten 600 Fehlern, die sich als angenommene Fehlerrate von 6 Fehlern pro 100 Zeilen Programmcode äußern. Sie rechnen außerdem mit Sachbearbeitern als Systemnutzern, die auch einen fixen Stundengehalt erhalten, um die Schadenshöhe zu ermitteln. Da allerdings vorgesehen ist, dass LexiCode von privaten Nutzern verwendet wird, werden hier fiktive Daten verwendet, die sich auf private Nutzer beziehen. So werden hier anstatt Sachbearbeiter private Nutzer als die Haupt-User-Gruppe und anstatt Stundengehalt für die Arbeit mit der Anwendung verlorene Arbeitszeit berücksichtigt. Diese verlorene Arbeitszeit ist die Zeit, die ein lernender Nutzer für sein Hauptberuf einsetzen kann, bei dem er oder sie 20 Euro pro Stunde verdient.

Wenn zu einem gegebenen Zeitpunkt 100 Nutzer mit LexiCode arbeiten und ein Systemausfall vorkommt, dann ist es in einem einstündigen Ausfall mit 100 verlorenen Arbeitsstunden zu rechnen und bei einem leichten, halbstündigen Ausfall mit 50. Im schlimmsten Falle, sprich bei einem maximalen Schaden, dauert der Ausfall bei jedem Nutzer 0,5 Stunden, die sich auf 50 verlorene Arbeitsstunden hochrechnen lassen.

Im besten Fall sind nur zehn Nutzer betroffen und können eine Stunde mit LexiCode nicht arbeiten, die in 10 verlorenen Arbeitsstunden resultieren und einen minimalen Schaden darstellen[[19]](#footnote-19).

Bei den angenommenen 600 Fehlern ist bei einer Ausfallquote von 10 % mit mindestens 60 Ausfällen und bei 50 % mit 300 Ausfällen zu rechnen.

Der maximale Schaden lässt sich folgenderweise ermitteln:

60 x (100 h a 20 €) = 120.000 €

+ 300 x (50 h a 20 €) = 300.000 €

420.000 €

Der minimale Schaden liegt dann bei dem folgenden Ergebnis:

60 x (50 h a 20 €) = 60.000 €

+ 300 x (10 h a 20 €) = 60.000 €

120.000 €

Für den wahrscheinlichen Schaden ergibt sich somit der folgende Wert:

(420.000 € + 120.000 €) / 2 = 270.000 €

Wenn ein Systemtest mindestens zwei Drittel der existierenden Fehler aufdeckt, kann mit einer Verminderung der Fehleranzahl von 600 auf 200 gerechnet werden, die 400 Fehlern entspricht, die in der Wartung, also in der Betriebsphase nach der Veröffentlichung der Anwendung, nicht behoben werden müssen[[20]](#footnote-20).

Wenn nach den Angaben gerechnet wird, die auch bereits in der theoretischen Fundierung bekanntgemacht wurden, ist für die Behebung der Fehler die folgende Anzahl von Personentagen zu erwarten:

160 Anforderungsfehler x 2,5 = 400 Personentage

120 Entwurfsfehler x 1,0 = 120 Personentage

120 Codefehler x 0,5 = 60 Personentage

Summe: 580 Personentage.

Bei einem Preis von beispielhaften 800 € pro Entwicklerpersonentag liegen die Fehlerbehebungskosten bei 464.000 €. Zusammen mit den Fehlerfolgekosten von 270.000 € ergibt sich ein Einsparungspotenzial von 734.000 €.

Ein auf 700 Personentagen geschätzter Testaufwand, bei dem die Kosten pro Tester bei z. B. 720 € pro Tag liegen, berechnen sich die Testkosten wie folgt:

700 x 720 = 504.000 €

Das ROI berechnet sich dann wie folgt:

ROI = (Nutzen – Kosten) / Kosten

ROI = (734.000 – 504.000) / 504.000 = 0,45.

## 3.5 Fehlerbewertung

Für den Entwicklungsprozess von LexiCode wurden bestimmte Ansätze eingehalten, um die Genauigkeit, Präzision, Wiederholbarkeit und Reproduzierbarkeit der Ergebnisse zu gewährleisten. Um Richtigkeit sicherzustellen, wurden die Anforderungen an die Anwendung passend überlegt, sodass sie, wie in dem Abschnitt über die Spezifikation der Anforderungen spezifiziert, ein herkömmliches Lehrbuch zum Sprachlernen visuell nachmacht und durch digitale Funktionen ergänzt, digitalisiert und dematerialisiert.

Die Anforderungen wurden durch den Entwickler selbst formuliert, der selbst Erfahrung mit dem Erwerb von zwei Fremdsprachen hat und somit eine genauere Vorstellung davon hat, wie der Lernprozess beim Fremdsprachenerwerb abläuft und wie eine digitale Anwendung diesen am besten abbilden kann.

Die entwickelten Softwarekomponenten wurden isoliert, nachdem sie entwickelt wurden, getestet, um zeitnah sicherzustellen, dass sie wie geplant funktionieren, ein Ansatz, der vom Prinzip her dem *Test-Driven Development* (*TDD*) ähnelt, nur mit dem Unterschied, dass, anstatt den Testprozess zu automatisieren die Systemkomponenten manuell getestet werden. Bei der Entwicklung wurde auch darauf geachtet, dass bereits angelegte Komponenten wie Buttons oder Container wiederverwendet werden, um in dem Programmcode eine Konsistenz zu sichern.

Es wurde selbstverständlich auch auf Präzision geachtet, damit LexiCode fehlerfrei arbeitet, vor allem wenn es um die Verarbeitung von Daten und Berechnungen geht. Um dies sicherzustellen, wurde zum einen beim Anlegen der Komponenten, in denen vom Benutzer Daten angegeben werden, sichergestellt, dass die Komponenten genau mit dem Datentyp arbeiten können, den der Benutzer angeben wird. Z. B. bei Komponenten auf LexiCode, die Übungsaufgaben darstellen und textuelle Antworten wie Wörter, Satzteile oder ganze Sätze erwarten, wurde im HTML-Code sichergestellt, dass diese Komponenten mit dem Datentyp „Text“ angelegt werden. Entsprechend wurden Funktionen in der Logik, die die der Richtigkeit von Übungsantworten zu prüfen haben, so geschrieben, dass diese den Datentyp *String* erwarten.

Zum Zweck der Wiederholbarkeit wurde die Versionierung kontrolliert, indem der aktuelle Programmcode nach jeder Änderung wie z. B. eine neugebaute oder geänderte Komponente auf der Web-Plattform *GitHub* gespeichert wird. Die Versionskontrolle über GitHub ermöglicht, dass bei fehlerhaften Entwicklungsarbeiten, die den Programmcode schwer beeinträchtigen und Korrekturen erschweren, die fehlerfreie Version des Codes als Kopie heruntergeladen und an der Anwendung wie geplant weitergearbeitet werden kann.

Außerdem hat der Entwickler mittels GitHub die Möglichkeit, von überall und von jedem Gerät auf den Programmcode zuzugreifen und an der Anwendung zu arbeiten. Darüber hinaus wurden die Testszenarien mit den genauen Testparametern dokumentiert, sodass sie unter den gleichen Bedingungen beliebig oft wiederholt werden können.

Damit die Ergebnisse der Teste unter den gleichen Bedingungen wieder durchgeführt werden können, wurden auch Maßnahmen für eine gewisse Reproduzierbarkeit ergriffen. Zum einen wurde die Entwicklungsumgebung dokumentiert, damit sie nachgebildet werden kann. Zum anderen wurden die externen Abhängigkeiten, in diesem Fall das einzige eingesetzte Framework, auch dokumentiert. Es wurden Kriterien für die Erzeugung von unterschiedlichen Testdaten eingesetzt, damit unterschiedliche Testszenarien getestet werden. Auch die Testdaten wurden dokumentiert.

## 3.6 Technologische Sachzwänge

Die im letzten Kapitel beschriebenen Ansätze zur Genauigkeit und Präzision sowie die Anwendbarkeit und Verfügbarkeit des Projektes weisen gewisse Einschränkungen auf, die zu erwähnen sind, weil sie die Interpretation der Ergebnisse beeinflussen können. Die Einschränkungen liegen größtenteils in der eingesetzten Technologie.

Die in dem Abschnitt über die Technologie beschriebenen Werkzeuge, die für die Entwicklung der HC-Version von LexiCode (HTML, CSS und JavaScript) eingesetzt wurden, sind ausführlich dokumentiert und können durch die Lektüre diverser Lernmedien wie Bücher, Blog-Artikel, Zeitschriften, Foren, YouTube Tutorials und Video-Kurse auf Online-Plattformen wie Udemy angeeignet werden. In manchen dieser Quellen sind kann auch die vollständige Umsetzung kleinerer bis mittelgroßer Projekte verfolgt werden, was ein großes Lernpotenzial für unerfahrene Entwickler ist.

Da Zoho Creator nicht so lange auf dem Markt wie herkömmliche HC-Entwicklungstechnologien und LC/NC-Entwicklung allgemein eine relativ neue Technologie ist, sind entsprechend weniger Lernmaterialien und Beispielprojekte zu finden. Dies verlangsamt den Lernprozess und erschwert ihn zusätzlich.

Der Entwickler, der sich mit der Entwicklung beider Versionen von LexiCode engagiert hat, kann mehr Erfarung mit HC- als mit LC/NC-Entwicklung aufweisen. Dies kann dazu geführt haben, dass der Entwickler bei im Zoho Creator gewisse Anforderungen nicht umsetzen konnte, für die es bereits Lösungen oder zumindest Workarounds gibt.

Andererseits kann die Tatsache, dass ein einzelner Entwickler an LexiCode gearbeitet hat, die Richtigkeit negativ beeinflusst haben.

In Softwareentwicklungsprojekten, an denen mehrere Entwickler beteiligt sind, können Code Reviews stattfinden, bei denen die Entwickler den Code ihrer Teamkollegen auf Fehler prüfen, damit die Wahrscheinlichkeit der Logikfehlern und Inkonsistenzen verringert wird.

Der Erfahrungsmangel bei der Entwicklung der LC/NC-Version von LexiCode kann natürlich auch bei der Entwicklung der HC-Version eine Rolle gespielt haben.

In einem Team haben die Entwickler normalerweise unterschiedliche Erfahrungsgrade und die eher erfahrenen können die weniger erfahrenen bei der Entwicklung unterstützen. Die reine Höhe des Aufwands kann die Qualität der Arbeit des Entwicklers beeinflusst haben, da Menschen tendieren, Fehler zu begehen, wenn sie von zu viel Arbeit überfordert werden.

Der Umfang des Case Study kann die Aussagekraft und Qualität der Forschung beeinflusst haben. Wie bereits erwähnt, ist für die vorliegende Forschung die Entwicklung einer reinen Frontend-Anwendung geplant worden, ohne dass serverseitige Prozesse, Schnittstellen oder Datenbankanbindungen und soz. ein Backend auch Teil der Anwendung werden.

Eine Anwendung, die über einen Backend verfügt, hat die zweifache Funktionalität einer reinen Frontend-Anwendung und verlangt auch, dass noch mehr Technologien verwendet werden. Dadurch öffnen sich in so einem Case Study weitere Möglichkeiten für einen Vergleich zweier Anwendungsansätze, was die Aussagekraft der Forschung erhöhen kann, weil mehr Anpassungsmöglichkeiten für eine HC-Fullstack-Anwendung im Vergleich zu LC selbstverständlich mehr aussagt als mehr Anpassungsmöglichkeiten für eine Frontend-HC-Anwendung.

Es ist zu erwähnen, dass im Rahmen dieser Studie die kostenfreie Probeversion von Zoho Creator eingesetzt wurde. Obwohl in den kostenpflichtigen Abonnements mehr Features zu finden sind, beziehen sich viele dieser Features auf Merkmale, deren Umsetzung in der vorliegenden Forschung nicht angestrebt wird, wie z. B. KI-Modelle, Integration mit weiteren Zoho Apps oder Business Intelligence und Analytics[[21]](#footnote-21). Die Features, die in dieser Forschung eigentlich gebraucht werden, wie bspw. Bausteine (Formulare, Berichten, Seiten) und Grundlegende Felder, sind in dem kostenlosen Abonnement vorhanden.

Ferner ist darauf hinzuweisen, dass das Entwicklungsprojekt innerhalb der vorliegenden Forschung im Bereich der LC-Entwicklung klarstellen soll, welche Komponenten und Funktionalitäten nur mit den Werkzeugen der Plattform Zoho Creator implementiert werden können, ohne nach Ergänzungswerkzeugen wie andere Programmiersprachen oder Plattformen zu suchen und über diese zu versuchen, die Anforderungen umzusetzen. Dies hängt mit dem Ziel zusammen, die Möglichkeiten von Zoho Creator als eigenständiges Werkzeug zu präsentieren.

## 4. Ergebnisse und Diskussion

Die Ergebnisse der Entwicklung und somit des Vergleichs der HC- und LC/NC-Version von LexiCode werden in der gleichen Reihenfolge vorgestellt wie die Reihenfolge der Forschungsfrage, die zum Anfang der Studie vorgestellt wurden. Bei der Vorstellung der Ergebnisse werden pro Forschungsfrage, die teilweise auch die Anforderungen nachbilden, im Abwechsel zuerst die Ergebnisse der HC- und danach die Ergebnisse der LC/NC-Anwendung erläutert, bevor es gleichermaßen mit der nächsten Anforderung bzw. mit dem nächsten Vergleichspunkt fortgesetzt wird.

## 4.1 Seitennavigation

Für die Navigation wurde das Ziel formuliert, sie lokal für die Inhalte innerhalb eines Sprachniveaus und global für den Wechsel zu einem anderen Niveau umzusetzen. Die bereits erwähnte Lehrbuchnähe wurde zum Teil implementiert, indem die Lerninhalte innerhalb eines Sprachniveaus wie z. B. C1 visuell als ein Inhaltsverzeichnis angelegt werden, auf dem jeder Punkt im Verzeichnis ein Link zu einer anderen Seite innerhalb der Anwendung ist, auf der sich der entsprechende Inhalt befindet. Diese Struktur wurde für jedes Sprachniveau von A1 bis C1 umgesetzt teilt die Kapitel in Unterkapitel, die die eigentlichen Inhaltsseiten sind.

In der HC-Version von LexiCode wurde die lokale Navigation als ein Navigationsmenü anhand des HTML-Elements *<nav>* angelegt, innerhalb dessen eine ungeordnete Liste (*<ul*>) liegt, in der der Link zu jeder Inhaltsseite als Ankerelement (*<a>*) innerhalb eines Listenelementes (*<li>*) eingebettet ist.

Über das Attribut *href* wird im Code der Link zu der entsprechenden Seite hinterlegt:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 4. (Link zu einer Inhaltsseite)

Die globale Navigation zur Fortbewegung auf die Inhaltsseiten anderer Sprachniveaus ist Teil der Seitenleistennavigation oder auch *Sidebar-Navigation*. Außerdem bietet die Seitennavigation Links zu allgemeinen Seiten der Anwendung wie z. B. das Kontaktformular oder das appinterne Forum:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 5. (Sidebar-Menu für globale Navigation)

Auch hier sind die Links Ankerelemente, die in den Listenelementen des größeren Navigationselements angelegt wurden.

Obwohl auf Zoho Creator auch HTML-Code geschrieben werden kann, werden paar klassische HTML-Attribute, wie eben das *href* Attribut, von der Plattform nicht unterstützt, was in diesem Fall das Anlegen von Hyperlinks und somit das Inhaltsverzeichniskonzept von LexiCode verhindert.

Da in der LC/NC-Version der Anwendung der gleiche Code wie in der HC-Version geschrieben wurde, erscheinen die Menüpunkte zwar als Links auf der UI auf, aber nachdem sie angeklickt werden, liefert Zoho eine Fehlermeldung, dass die Webseite nicht gefunden werden konnte[[22]](#footnote-22).

Die Seitennavigation konnte zumindest über den App-Menüersteller[[23]](#footnote-23) von Zoho Creator angelegt werden. Diese Funktionalität ermöglicht das Erstellen von Menü- und Untermenüpunkten, indem Abschnitte per Drag-and-Drop in ein Menü reingezogen werden, welches die Seitennavigation darstellt. Eine solche Navigation ermöglicht zwar auch die Umleitung zu anderen Seiten der Anwendung, kann aber dazu führen, dass dieses Seitenmenü unübersichtlich wird, wenn zu viele Links auf der Navigation angelegt werden.

Somit stellt die HC-Entwicklung Optionen zur Seitenverlinkung und -navigation zur Verfügung, die den Bau von Navigationselementen und Links außerhalb einer allgemeinen Hauptnavigation ermöglichen, wie es im Fall der LC/NC-Version von LexiCode ist. Da solche Optionen in Zoho fehlen, eignet sich diese Entwicklungsplattform nicht zur Umsetzung von Navigationselementen wie eine Header-Navigation oder ein *Footer*. Ohne solche Komponenten ist die Nützlichkeit einer Webanwendung eingeschränkt.

Es wurde vor der Durchführung der vorliegenden Forschung erwartet, dass einfache HTML-Hyperlinks im Zoho Creator umgesetzt werden können, weil diese grundlegende HTML-Elemente sind, die in sehr vielen Webanwendungen Anwendung finden. Wenn Hyperlinks auch auf anderen LC-Plattformen nicht unterstützt werden, dann stellt dies einen bedeutsamen Nachteil für die LC/NC-Technologie dar.

Die Ergebnisse von dieser Anforderung sprechen gegen die Studie von Elshan und Binzer (2024), in der behauptet wird, dass die technischen Einschränkungen der LC/NC-Plattformen durch klare Governance-Strategien und Schulungen überwunden werden können. Obwohl Governance die LC-Nutzung steuert und überwacht und Schulungen den Citizen Developern mehr Kompetenz bei der Entwicklung mittels LC-Plattformen übermitteln können, bietet keine dieser Maßnahmen eine Alternativlösung für die technischen Einschränkungen dieser Plattformen und in dem Falle der vorliegenden Forschung, keine Alternative für die Hyperlinks.

## 4.2 Manipulation der Seitenelemente

Der in der nächsten Forschungsfrage formulierte Vergleichspunkt, der sich auf die Manipulation der Seitenelemente konzentriert, hat für den Umfang dieses Projektes die spezifischere Anforderung, die von den Benutzern angegebenen Daten auf deren Richtigkeit zu prüfen und den Benutzern eine entsprechende Rückmeldung zu liefern.

Die Systemkomponente, in der diese Funktionalität erfolgen soll, ist z. B. eine Übung zur Grammatik auf Niveau A1. Nach jeder Frage befindet sich der Button „Check“, der die Richtigkeit der Antwort prüft, nachdem er angeklickt wird. Als Beispiel kann die erste Frage in der ersten Übung genannt werden, bei der der Benutzer in das leere Feld vor der Frage „Wie geht es dir?“ eine Begrüßung schreiben muss, die morgens ausgesprochen wird („Guten Morgen“).

In der HC-Version von LexiCode wurde die Richtigkeitsüberprüfung durch die Check-Buttons implementiert, für die eine JavaScript-Funktion geschrieben wurde, dieauf die Input-Felder nach ihrer Klasse zugreift. In dem untenstehenden Bild greigt die Funktion auf die Klasse „check-ex1q1“:.

Ein Bild, das Screenshot, Text, Schrift, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 6. (Input-Feld für Übungsantwort gefolgt von einem Prüf-Button)

Im weiteren Ablauf der Funktion wurde eine *if*-Abzweigung gebaut, die das erste Teilaufgabenfeld nach dem Inhalt überprüft. Wenn der Inhalt „Guten Morgen“, also richtig ist, wird der Hintergrund des Eingabefeldes mit dem *Hexcode* „#94d82d“ grün und die Buchstaben weiß gefärbt.

Wenn die Antwort falsch ist, dann wird der Feldhintergrund mit dem Hexcode „#e03131“ rot und die Buchstaben wieder weiß gefärbt.

In der Zoho-Version von LexiCode ist die Umsetzung einer DOM-Manipulation nicht möglich. Unter den Optionen von der zoho-eigenen Programmiersprache, *Deluge*, sind die Möglichkeiten vorhanden, Formulare, Reports oder Widgets einzubetten, Kontrollstrukturen wie if-Abzweigungen zu schreiben, und Daten zu manipulieren[[24]](#footnote-24). Zwar sind Optionen vorhanden, Strukturen wie Listen, Maps, Webdaten[[25]](#footnote-25) oder die XML zu manipulieren, aber in Bezug auf DOM-Manipulation scheinen keine Möglichkeiten zu finden zu sein[[26]](#footnote-26).

Als Antwort auf die Forschungsfrage, über welchen Entwicklungsansatz die Anwendungskomponenten manipuliert werden können, bleibt die HC-Entwicklung als alleinstehende Antwort. Obwohl in Zoho-Anwendungen diverse Funktionen implementiert werden können, ist die fehlende Technologie zur Manipulation der UI-Komponenten anhand des Nutzerverhaltens ein Mangel, der das Nutzererlebnis einschränkt.

Vorgänge, die sich auf Elemente beziehen, können bei einem Entwicklungsansatz, der sie nicht direkt unterstützt, nur durch mehrere Workarounds umgesetzt werden. Die Erwartung, dass innerhalb des Zoho Creators eine zu der DOM-Manipulation ähnliche Technologie verfügbar ist, wurde hier nicht erfüllt.

Diese Ergebnisse widersprechen der Forschung von Frank (2023), weil die Anforderung mittels LC-Entwicklung nicht maßgeschneidert und effizienter sondern überhaupt nicht realisiert werden konnte. Die von ihm vorgeschlagenen Maßnahmen, klare Ziele zu definieren und die Plattform sorgfältig zu steuern, können das Ergebnis nicht optimieren, weil das grundlegende Werkzeug zur Umsetzung der Anforderung fehlt.

## 4.3 Anpassbares Formular

Die Anforderung nach einem anpassungsfähigen Formular konnte in der HC-Version von LexiCode gut umgesetzt werden.

Dafür wurde im HTML-Code ein *<form>*-Element angelegt (Bühler, Schlaich & Sinner, 2023), innerhalb dessen Formulars getrennte *<div>*-Elemente angelegt wurden. In jedem *<div>* befindet sich ein Eingabefeld sowie das dazugehörige Bezeichnungselement, *<label>*.

In dem Formular, das als Kontaktformular dienen soll, wurde neben dem Typ *text* auch ein Input-Feld mit dem Typ *E-Mail* angelegt, damit dieses Feld prüfen kann, ob die eingegebenen Daten dem typischen Format von E-Mail Adressen entsprechen:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 7. (Input-Feld für E-Mail-Adressen)

In den CSS-Einstellungen wurde für das Formular eine 1 Pixel breite, feste, blaue Außenkante definiert. Darüber hinaus wurden die Ecken des Formulars gerundet und es wurden Innen- und Außenabstände konfiguriert.

Für die Eigenschaft *margin*, die für Außenabstände zuständig ist, wurde als zweiter Wert „auto“ angegeben, um die UI-Komponente zu zentrieren. Die Breite des Formulars wurde auch nach eigenem Wunsch definiert und auf 600 Pixel gesetzt. Um einen 3D-Effekt zu erreichen, wurde dem Formular zudem ein Schatten-Effekt vergeben. Diese CSS-Einstellungen wurden unter der Klasse „*contact-form*“ definiert:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 8. (Eigendefinierte CSS-Klasse für das Kontaktformular)

Im Anhang F ist die finale Version des HC-Formulars ersichtlich und im Anhang G der vollständige HTML-Code des Formulars.

Obwohl das über den Zoho Creator angelegte Formular optisch ansprechbar ist, waren in dem Zoho Formulargenerator namens *Form Builder* fast keine Möglichkeiten vorhanden, das Formular visuell anzupassen. Ein Anpassungsversuch wurde z. B. auf die Überschrift des Formulars, „Schreiben Sie uns“, in dem Feld-Editor[[27]](#footnote-27) unternommen, in dem die Anpassungen als HTML-Code geschrieben werden können.

Die Überschrift wurde mittig platziert, eine Änderung, die sowohl in der Vorschau[[28]](#footnote-28), als auch in der finalen Version des Formulars übernommen wird[[29]](#footnote-29). Weitere Anpassungsoptionen beziehen sich nur auf den Namen des Abschnitts, in dem das Formular enthalten ist, den Feldlinknamen, die Option, die Kopfzeile für den Abschnitt anzuzeigen und den Feldtyp über ein Auswahlmenü zu ändern (siehe Anhang K).

Für einzelne Felder existieren Anpassungsoptionen wie Feldname und –linkname, die Angabe, ob das Feld ein Pflichtfeld sein soll, Felder für Präfixe oder Suffixe, die Angabe, ob das Feld personenbezogene Daten enthält (Siehe Anhang L), den Feldtyp, der nicht auswählbar ist, die Feldgröße und den Beschreibungstext (siehe Anhang M).

Im Vergleich zu den Ergebnissen der letzten zwei Anforderungen, konnte bei der Anforderung nach einem anpassbaren Webformular ein Ergebnis erzielt werden, welches, abgesehen von kosmetischen Anpassungen, die gewünschte Funktionalität abdeckt. Für das Formular konnten in beiden Versionen der Anwendung Felder für die gewünschten Daten und jeweils ein Button zum Absenden angelegt werden.

Allerdings konnte die Anwendung in Bezug auf die optische Anpassung die Anforderung nicht decken. Dieser Fehlversuch hat zwar im Umfang dieses Case Study keine großen Folgen, wird aber in einem Szenario, in dem LexiCode nach Benutzeranfragen eine Modifikation des Designs unterzogen werden muss, ebenfalls zu Einschränkungen führen.

Die Forschungsfrage, welche der beiden Entwicklungsansätze mehr Anpassungsmöglichkeiten für Webformulare bietet, lässt sich mit der HC-Entwicklung beantworten. Dieses Ergebnis entspricht den Erwartungen vor der Entwicklung, dass LC/NC-Plattformen eher das schnelle Zusammenbauen von Anwendungskomponenten anstatt Anpassungsmöglichkeiten priorisieren.

Das Ergebnis dieser Anforderung entspricht der Forschung von Di Ruscio et al. (2022), weil auch in der vorliegenden Forschung festgestellt wurde, dass LC/NC-Entwicklung die Benutzerfreundlichkeit fördert aber einen Mangel an Anpassungsfähigkeiten aufweist. Wie von ihnen formuliert, ist das MDE und sogar die HC-Entwicklung die zu bevorzugende Entwicklungstechnologie.

Die Erstellung des Kontaktformulars durch reines Auswählen von Eingabefeldern bestätigt die Benutzerfreundlichkeit der LC-Plattformen, die Jit et al. (2024) in ihrer Studie thematisiert haben. Die niedrigere Einstiegshürde im Zoho Creator ermöglicht, dass neben dem Kontaktformular noch andere Systemkomponenten von Entwicklern ohne herkömmlichen Programmierkenntnissen entwickelt werden. Die schnelleren ersten Erfolge bei der LC-Entwicklung auf Zoho können dazu führen, dass Anfänger Entwickler schnelles Vertrauen in ihren eigenen Programmierfähigkeiten sammeln und schneller an größeren Projekten arbeiten.

## 4.4 Inzeilige Eingabefelder

Die zweite Aufgabe im Abschlusstest für das Niveau A1, ein Lückentest, hat in jeder Teilaufgabe ein Eingabefeld an einer unterschiedlichen Stelle in dem Satz. Dafür wurde hier eine ungeordnete Liste angelegt, in der jede Teilaufgabe ein einzelnes Listenelement ist. In jedem dieser Listenelemente liegen Input-Felder vom Typ *text*, weil hier textuelle Daten als Antworten anzugeben sind:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 9. (Inzeilige Eingabefelder mittels HTML)

Das *<ul>*-Element, welches die Listenelemente umschließt, besitzt die vom Entwickler definierte Klasse „bullet-free“, die Stichpunkte entfernt, die Listenelemente in einer untergeordneten Liste in HTML automatisch erhalten:

Ein Bild, das Schrift, Text, Grafiken, Typografie enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 10. (Eigene Klasse zum Entfernen von Stichpunkten)

Je nach Teilaufgabe wurden die Satzteile sowohl vor als auch nach den Input-Feldern geschrieben, damit sie wie gewünscht auf der UI auftauchen[[30]](#footnote-30).

Die Umsetzung der LC/NC-Version hat nahezu das gleiche Ergebnis geliefert. Der HTML- und CSS Code, der für den Abschlusstest im Kapitel 1. des Niveaus A1 geschrieben wurde, wurde ohne Änderung in einem leeren HTML-Snippet im Zoho Creator übernommen. Das Endprodukt ist vom Aussehen her nahezu identisch mit der HC-Version der Aufgabe[[31]](#footnote-31).

Da im Unterschied zu Entwicklungsumgebungen für HC-Entwicklung wie bspw. VSC in einem HTML-Snippet in dem Zoho Creator keine Dokumente oder Ordner angelegt werden können[[32]](#footnote-32), müssen alle für den Abschlusstest relevanten Codezeilen in dem gleichen Snippet eingefügt werden. Dies wird erreicht, indem der CSS-Code in dem HTML-Tag *<style>* eingefügt und der HTML-Code gleich daruntergeschrieben wird[[33]](#footnote-33), was bei zunehmender Anzahl an Komponenten bzw. Codezeilen die Übersichtlichkeit verringern wird.

Abgesehen von dem Unterschied in der Dokumentenstruktur der beiden Versionen der Anwendung, ist die Anforderung nach inzeiligen Eingabefeldern die erste in diesem Case Study, die sowohl mit HC als auch mit LC/NC erfüllt werden konnte, also die erste Forschungsfrage, die sich vollständig positiv beantworten lässt.

Die beiden Entwicklungsansätze bieten daher, zumindest hinsichtlich Eingabefelder, ungefähr die gleichen Entwicklungsmöglichkeiten, die ein kleiner Vorteil für die LC/NC-Entwicklung darstellen, da das Anlegen solcher Felder bei der Entwicklung für den durchschnittlichen Entwickler nicht mit Bedenken nach der Umsetzung verbunden ist.

Die Erwartung für diese Forschungsfrage ist die Annahme gewesen, dass inzeilige Eingabefelder sowohl über HC als auch LC/NC angelegt werden können, da dafür in beiden Ansätzen HTML-Code mit CSS-Einstellungen genügt.

Die Ergebnisse dieser Anforderung stimmen mit den Hauptvorstellungen der Studie von Ajimati et al. (2025) überein. Anforderungen, die auch über LC/NC eingehalten werden können, führen zu einer Effizienzsteigerung bei der Entwicklung und können auch von Citizen Developern realisiert werden, die sog. Demokratisierung der Softwareentwicklung. Obwohl Ajimati et al. (2025) die möglichen Qualitätsmangel bei LC/NC-Anwendungen hervorheben, konnte in der vorliegenden Forschung nachgewiesen werden, dass einfachere Anforderungen wie die inzeiligen Eingabefelder mittels LC-Plattformen in der gewünschten Qualität erbracht werden können.

## 4.5 Sitzungsunabhängige Datenspeicherung

Für diese Anforderung wurde, ähnlich zu der letzten, ein Lückentest gebaut. Jede Teilaufgabe wurde als separates Unterelement in einem Paragrafenelement (*<p>*) angelegt, um sicherzustellen, dass jede Teilaufgabe in einer getrennten Zeile steht. In jedem dieser Paragrafenelemente wurden jeweils zwei Input-Elemente angelegt: das erste als Eingabefeld vom Typ *text* und das zweite vom Typ *button*, damit die Benutzer die Möglichkeit haben, ihre Antworten manuell zu speichern.

Die Buttons verfügen über das Attribut *onclick*, das eine Event-Handler-Funktion ausführt:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 11. (Teilaufgabe zur sitzungsunabhängigen Datenspeicherung)

In der Funktion wird jedes Feld als Variable hinterlegt und der querySelector greift auf das Objekt *value*, in dem der Wert des jeweiligen Feldes gespeichert wird. Im nächsten Schritt wird über das Objekt *localStorage* die Funktion setItem() aufgerufen, in der über den querySelector auf den Wert der jeweiligen (Eingabe-)Komponente zugegriffen wird:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 12. (Funktion zur browserseitigen Speicherung von Daten)

Die Eigenschaft *localStorage* dient der fortbestehenden Speicherung von *Strings*, und macht sie nach Neuladen der Seite für den Benutzer immer noch verfügbar (Flanagan, 2020). Damit die Daten in der gleichen oder in einer zukünftigen Sitzung von der *localStorage* Eigenschaft geladen werden, wurde die separate Funktion „holen()“ geschrieben.

In dem nächsten Schritt sorgt eine if-Abzweigung dafür, dass, wenn in einem Feld ein Wert eingegeben worden ist, der Wert in das Feld geladen wird:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 13. (Abrufen von zuvor gespeicherten Werten)

Ähnlich zu der Elementenmanipulation kann die Forschungsfrage, ob Daten sitzungsunabhängig in LexiCode gespeichert werden können, nur in der HC-Version positiv beantwortet werden. Da JavaScript eine breite Auswahl an Funktionen und Objekten bietet, konnte diese nützliche Anforderung mit wenig Aufwand umgesetzt werden.

Auf ist Zoho eine solche Funktion allerdings nicht vorhanden. Dies entspricht den Erwartungen vor der Entwicklung, dass eher gängige programmiertechnische Funktionalitäten wie if-Abzweigungen in so einer LC/NC-Plattform zugänglich sind. Dementsprechend müssen bei der LC-Anwendung andere Methoden zur Speicherung von benutzerdefinierten Daten herangezogen werden, wie z. B. das Aufbauen einer eigenen Datenbank.

Da so ein Aufwand auch bei der Entwicklung einer HC-Anwendung nötig ist, wenn Daten in einer Datenbank gespeichert werden müssen, und angesichts der Einschränkungen, die bei den bisher vorgestellten Ergebnissen festgestellt wurden, sprechen in diesem Zusammenhang außer der potenziellen Zeitersparnis wenig weitere Argumente dafür, eine mit eigenem Backend ausgestattete Anwendung über Zoho Creator zu bauen.

Die Äußerung von Schreiner (2023), dass LC-Plattformen den Einsatz von HC-Entwicklern erübrigen, steht im Widerspruch zu den in diesem Kapitel behandelten Ergebnissen. Da spezifische Anforderungen wie die sitzungsunabhängige Datenspeicherung in vielen Softwareentwicklungsprojekten vorkommen, wird auch bei einem LC-Projekt eine HC-basierte Zusatzlösung erforderlich sein, die nur von einem HC-Entwickler in die Tat umgesetzt werden kann.

## 4.6 Anforderungen außerhalb der Forschungsfragen

Diese Anforderungen beziehen sich auf Komponenten, die mittels CSS-Bootstrap gebaut wurden, um die Entwicklung zu beschleunigen und zu vereinfachen. Zwei Beispiele dafür sind u. a. bei jeder Teilaufgabe in Übung 1 des bereits vorgestellten Abschlusstests zu finden. Jedes Input-Feld besitzt die Bootstrap Klasse *rounded*, welche die Ecken der Komponente rundet, der sie zugewiesen worden ist:



Abbildung 14. (Bootstrap Klasse für runde Ecken)

Der „Abgeben“ Button im gleichen Abschlusstest besitzt die Klasse *btn*, welche dem Button eine Standardhöhe, -breite und Innenabstand vergibt.

Die zweite Bootstrap-Klasse, *btn-primary*, sorgt dafür, dass der Button eine Farbe erhält, in diesem Fall die blaue Farbe[[34]](#footnote-34), und ein Hover-Effekt, bei dem sich die Farbe auf dunkelblau ändert, wenn der Benutzer mit dem Mauszeiger über den Button schwebt[[35]](#footnote-35):



Abbildung 15. (Bootstrap-Klassen für Buttons)

Um über Bootstrap Komponente responsiv zu programmieren, werden zusammenhängende Unterkomponenten bzw. Kindelemente von einem Containerelement umgegeben. Am Beispiel des Kontakformulars hat das Formular bei der Standardbildschirmauflösung eine Breite von 600 Pixel und wird mittig auf dem Display platziert. Damit der Container des Formulars responsiv wird, wird ihm die Bootstrap Klasse *container* zugewiesen:



Abbildung 16. (Responsiver Bootstrap-Container)

Bei der Desktop-Browser Bildschirmbreite von über 1700 Pixel steht das Formular, mittig auf dem Display mit ausreichend Abstand zu beiden Bildschirmkanten. Wenn die Bildschirmbreite reduziert wird, passt sich der Container des Formulars so an, dass sowohl links als auch rechts ein gewisser Außenabstand zwischen dem Formularrahmen und der Browserfensterkante bleibt.

Bootstrap sorgt dafür, dass zwischen jedem Eingabefeld im Formular und den Formularrahmen nach links und rechts Innenabstand vorhanden ist. Das erste Beispiel im Anhang veranschaulicht die Responsivität des Formulars bei 550 Pixel[[36]](#footnote-36), eine Bildschirmbreite, die geringer als die für das Formular festgelegte Breite von 600 Pixel ist.

Das zweite Beispiel zeigt die Responsivität bei einer geringeren Bildschirmbreite von 350 Pixel, auf welche das responsive Formular sich ebenfalls geeignet anpasst[[37]](#footnote-37). In den im Anhang gezeigten Beispielen wurde die Entwicklerkonsole von Google Chrome geöffnet, um die Bildschirmbreite zu reduzieren.

Da das Kontaktformular auf Zoho Creator über NC gebaut wurde und der Abschlusstest über Bootstrap gebaut wurde, wird die Responsivität von Zoho anhand des Abschlusstests geprüft. Das Ergebnis deutet darauf hin, dass auch der Abschlusstest in Zoho sich an einer reduzierten Bildschirmbreite anpasst[[38]](#footnote-38).

Im Sinne der Verwendung von Bootstrap und die Werkzeuge dieses Frameworks, konnte Zoho Creator unerwartete Ergebnisse liefern, da es aufgrund eines Erfahrungsmangels dem Entwickler nicht bewusst war, ob Bootstrap mit Zoho kompatibel ist. Dieses unerwartete Ergebnis spricht für die Entwicklung mittels Zoho, weil diese nicht nur durch vorgefertigte Bausteine, sondern auch durch die Verwendung von Bootstrap schneller durchlaufen kann.

Ähnlich zu der Forschung von Gialitakis et al. (2024) können für LexiCode mittels Bootstrap viele vorgefertigte Systemkomponenten verwendet werden, die zur Prototypenerstellung dienen. Obwohl Bootstrap auch mit der HC-Entwicklung kompatibel ist, kann in einem alternativen Entwicklungsprojekt, in dem LexiCode in einer kurzen, vorgegebenen Zeit ins Leben gerufen werden muss, über LC entwickelt werden, um die allgemeine Effizienz des Projektes zu steigern.

## 4.7 Ermittlung der Testkosten

Die Unterschiede in der Anzahl der Codezeilen, die für die Entwicklung der HC- und der LC/NC-Version von LexiCode geschrieben wurde, führen dazu, dass beide Versionen der Anwendung unterschiedlich hohe Testkosten in Anspruch nehmen werden. Für die in den bisher behandelten Bereichen von LexiCode wurden für die HC-Version der Anwendung 1435 Zeilen Code gezählt, die mit dem Faktor 0,05 multipliziert wurden, um 71,75 Solltestfälle zu erhalten. Der Faktor stellt das Verhältnis der 5000 Solltestfälle, die laut Sneed und Jungmayr pro 100.000 Zeilen Code anfallen, da 5000/100.000 = 0,05 ergibt.

Die im Anhang ausgewiesenen Werte der Aufwandsberechnung für die HC-Version von LexiCode ergeben den folgenden Aufwand[[39]](#footnote-39):

Aufwand = 1 x [(71,75) x 0,67/4 + (4 x (1 – 0,4)1,03 x (0,5/0,45)))]

Aufwand = 1 x ((48,07/6,4)1,03 x 1,11)

Aufwand = 1 x (81,03 x 1,11)

Aufwand ≈ 8,51 Personentage

Für die Schadensermittlung wird von 8,61 Fehlern nach der Auslieferung der Anwendung ausgegangen. Für die Ermittlung dieser Zahl wird angenommen, dass pro 100 Zeilen Code 6 Fehler auftreten werden, und es wird die Gesamtzahl von 1435 Codezeilen der HC-Version von LexiCode berücksichtigt. Die Schadensermittlung liefert als Ergebnis einen wahrscheinlichen Schaden von 3.871 €[[40]](#footnote-40), wenn bei einer Ausfallquote von 10 % 0,861 Fehler (8,61 x 0,1 = 0,861) und bei einer Ausfallquote von 50% 4,3 Fehler auftreten (8,61 x 0,5 = 4,3). Der Entwicklerpersonentag wird anders als bei Sneed und Jungmayr auf 200 € festgelegt und führt somit zu Fehlerbehebungskosten i. H. v. 1.722 € (200 x 8,61 = 1.722). Wenn die Fehlerfolgekosten von 3.871 € darauf gerechnet werden, entsteht durch das umfangreiche Testen ein Einsparungspotenzial von 5.593 €. Der auf 8,51 Personentage geschätzte Testaufwand ergibt zusammen mit den angenommenen Kosten für eine Testperson von 150 € pro Tag die folgenden Testkosten:

8,61 x 150 € = 1.291,5 €.

Das ROI für die HC-Version von LexiCode berechnet sich wie folgt:

ROI = (5.593 - 1.291,5) / 1.291,5 = 3,33.

Für die LC/NC-Version von LexiCode wurden 1487 Zeilen Code gezählt und ergeben mit dem Faktor 0,05 eine Anzahl von 74,35 Solltestfällen (1487 x 0,05 = 74,35). Auch hier wurden im Anhang die Werte für die Aufwandsabrechnung ausgewiesen[[41]](#footnote-41) und diese führen zu der folgenden Schätzung:

Aufwand = 1 x [(74,35) x 0,67/4 + (4 x (1 – 0,4)1,03 x (0,5/0,45))]

Aufwand = 1 x ((49,81/6,4)1,03 x 1,11)

Aufwand = 1 x (81,03 x 1,11)

Aufwand ≈ 8,51 Personentage

Für die LC-Anwendung wird davon ausgegangen, dass 8,9 Fehler nach der Veröffentlichung im Code zu finden sein werden. Die Anzahl wurde nach dem gleichen Ansatz wie bei der HC-Anwendung ermittelt. Der wahrscheinliche Schaden für die LC-Anwendung liegt bei 4.005 €[[42]](#footnote-42), wenn bei einer Ausfallquote von 10% 0,89 Fehler (8,9 x 0,1 = 0,89) und bei einer Ausfallquote von 50% 4,45 (8,9 x 0,5 = 4,45) Fehler auftreten. Auch hier gilt der gleiche Entwicklerpersonentag von 200 € und ergibt mit der erwarteten Fehleranzahl Fehlerbehebungskosten i. H. v. 1.780 € (200 x 8,9 = 1.780). Wenn diese mit dem wahrscheinlichen Schaden summiert werden, entsteht ein Einsparungspotenzial von 5.785 €. Der Testaufwand von 8,9 Personentagen ergibt mit den gleichen Testkosten von 150 € pro Person/pro Tag die folgenden Testkosten:

8,9 x 150 € = 1.335 €.

Das ROI lässt sich wie folgt berechnen:

ROI = (5.785 - 1.335) / 1.335 = 3,33

Einerseits hat die Schätzung nach dem Value-Driven Testing festgestellt, dass ausführliches Testen in der Entwicklungsphase von LexiCode wirtschaftlich berechtigt ist, weil bei beiden Versionen der Anwendung ein positiver ROI kalkuliert wurde. Außerdem ist dieser ROI höher als der Wert, den Sneed und Jungmayr in ihrem Beitrag berechnet haben. Allerdings ist zu beachten, dass sie in ihrem Beitrag über eine Konstellation sprechen, in der eine viel umfangreichere Anwendung mit deutlich mehr Codezeilen zu testen ist, was u. a. mehr Solltestfälle und höhere Testkosten verursacht.

Andererseits hat die Schätzung für beide Versionen von LexiCode den gleichen ROI bekommen, der ein Widerspruch zu den Erwartungen darstellt, weil es davon ausgegangen wurde, dass der ROI für die LC/NC-Version einen höheren ROI haben wird. Diese Erwartung wurde nicht erfüllt und ist auf die Anzahl der Codezeilen der beiden Versionen der Anwendung zurückzuführen, die ein Paradoxon darstellt, der auch gegen Erwartungen spricht. Es wurde vor der Entwicklung selbstverständlich erwartet, dass die LC/NC-Version von LexiCode aus insgesamt weniger Codezeilen bestehen wird, daher auch der Begriff „Low-Code/No-Code“.

Die fehlende Dokumenten- und Verzeichnisstruktur, die bei der Vorstellung der inzeiligen Eingabefeldern angesprochen wurde, führt dazu, dass auf der Zoho-Anwendung viel Coderedundanz entsteht, weil sich wiederholender Code, wie z. B. die CSS-Einstellungen, nicht in einer separaten Datei abgelegt werden kann, aus der die Einstellungen in dem jeweiligen HTML-Snippet ausgelesen werden können.

Obwohl manche Komponenten, wie z. B. das Kontaktformular oder die Sidebar-Navigation, komplett über NC programmiert wurden, ist die LC-Version von LexiCode insgesamt als Codebasis umfangreicher, obwohl die Sidebar-Navigation auf jeder Seite der Anwendung vorhanden sein muss und ihre Umsetzung durch vorgefertigte Bausteine mehrere Codezeilen gespart hat.

Wenn eine zukünftige Version von Zoho Creator die Dokumentenstruktur einer Entwicklungsumgebung für HC-Entwicklung unterstützt, dann können diese Coderedundanzen vermieden werden und das vorliegende Case Study würde, wie erwartet, eine geringere Codezeilenanzahl für die LC/NC-Version von LexiCode aufweisen. In der aktuellen Version von Zoho Creator wird die Anzahl der Codezeilen zwischen den beiden Anwendungsversionen möglicherweise größer und die Codebasis der HC-Anwendung doch wie erwartet größer, wenn noch weitere Komponenten und ein Backend Teil der Anforderungen werden.

Liu et al. (2023) vertreten die gleiche Ansicht wie Sneed und Jungmayr, weil sie festgestellt haben, dass 60 % der Fehler einer Anwendung in der Design- und Spezifikationsphase auftreten. Diese Feststellung unterstützt den Ansatz des Value-Driven Testing, weil es für ein Softwareentwicklungsprojekt kosteneffektiver ist, durch umfangreiches Testen in der Entwicklungsphase viele Fehler aufzudecken und diese zu beheben, bevor das System veröffentlicht wird.

## 4.8 Bestätigung der These

Die zu Anfang dieser schriftlichen Ausarbeitung vorgestellte These lässt sich nach der Darlegung der Ergebnisse und die Diskussion über ihre Bedeutung sowie der Vergleich der Ergebnisse mit den Erwartungen bestätigen. Im Unterschied zu der LC/NC-Version von LexiCode konnten die Anforderungen auf der HC-Version von LexiCode wie geplant umgesetzt werden. Das gewünschte minimalistische Design der Anwendungskomponenten, die Nutzerinteraktionen und ihre Erge-bnisse konnten nah an den Anforderungen ins Leben gerufen werden.

Dadurch, dass sich die geplanten Komponenten in der HC-Version der Anwendung wie geplant programmieren haben lassen, ist kein zusätzlicher Aufwand für die Suche nach alternativen Werkzeugen wie andere Funktionen, Frameworks, Programmiersprachen, Entwicklungsumgebungen oder alternative Komponentenideen entstanden. Die Implementierung der LC/NC-Version dagegen hat einige unerwartete Hürden bereitgestellt.

## 4.9 Die Ergebnisse auf dem magischen Viereck abbilden

Die zum Anfang dieser Forschung vorgestellte Studie von Baumgarten et al. hat auf dem sog. magischen Viereck die Ergebnisse mehrerer weiterer Studien zum Thema LC/NC-Entwicklung abgebildet, die entweder eigene Praxiserfahrungen aus eigenen Projekten mitteilen oder meinungsbasierte Beiträge zum Thema LC/NC-Entwicklung geleistet haben. Die Ergebnisse der vorliegenden Studie zeigen auf einen Zugewinn in Bezug auf Qualität, weil sich die Anforderungen so wie gewünscht haben umsetzen lassen. Außerdem erzielte das vorliegende Projekt eine Wertesteigerung in der Dimension der Flexibilität, weil manche Anforderungen über einfachere und kreativere Wege als die gängigen Lösungen implementiert werden konnten, wie die sitzungsunabhängige Speicherung der Daten, die sich über die localStorage Eigenschaft des Browsers speichern lassen, anstatt eine Anbindung an eine Datenbank zu benötigen. Somit erhält die HC-Version von LexiCode den Faktor „+1“ auf den Dimensionen der Qualität und der Flexibilität auf dem Viereck:



Abbildung 17. (Magisches Viereck mit den Ergebnissen von LexiCode)

## 4.10 Zusamenfassung

Der Studie ist zu entnehmen, inwiefern die HC-Entwicklung einen vereinfachten Entwicklungsprozess und eine genauere Umsetzung der Systemanforderung zulässt im Vergleich zu der LC/NC-Entwicklung. Die zum Anfang der vorliegenden Forschung aufgezählten Forschungsfragen lassen sich wie folgt beantworten.

**Anlegen von Hyperlinks über der globalen Navigation hinaus, die den Benutzer von einer Seite innerhalb der Anwendung zu einer anderen verleiten sollen.**

So eine Funktionalität lässt sich über die HC-Anwendung aber nicht über die LC/NC-Anwendung umsetzen. Obwohl Hyperlinks eine der Grundeigenschaften von HMTL ist, werden sie vom Zoho Creator nicht unterstützt.

**Eine Manipulation der Seitenelemente und –komponenten, um mehr Nutzereffekte zu erreichen**

Über die HC-Entwicklung und noch genauer gesagt, die DOM-Manipulation von JavaScript, kann die Grundlage für solche Effekte programmiert werden. Die Standardfunktionen von Zoho Creator unterstützen keine ähnlichen Eigenschaften und lassen eventuell nur über Workarounds realisieren.

**Breitere Auswahl an Anpassungsmöglichkeiten für Webformulare**

Die HC-Entwicklung bietet mehr Möglichkeiten zur Gestaltung eines Webformulars mit Eigenschaften wie Randfarbe, Innen- und Außenabstände und Platzierung auf der Seite. Die Option des Zoho Creators, über vorgefertigte Bausteine Formulare zu erstellen, bietet weniger Änderungsmöglichkeiten.

**Einbau inzeiliger Eingabefelder und deren Anpassung**

Dies ist eine der Anforderungen, bei denen HC- und LC/NC-Entwicklung innerhalb dieses Case Study gleichgestellt sind. Diese Art von Feldern ergibt sich aus einer Kombination zwischen HTML und CSS und der Code lässt sich über beide Entwicklungsansätze ausführen.

**Sitzungsunabhängige Speicherung benutzerdefinierter Daten**

Auch bei dieser Anforderung ist eine Lösung nur über die HC-Version von LexiCode möglich. Die Standardfunktionen von dem Zoho Creator können diese Anforderungen nicht verwirklichen lassen, was den Einsatz von Workarounds erfordert.

## 5. Fazit

Die HC-Entwicklung erweist sich im Rahmen der vorliegenden Forschung als mehr vorteilhaft für einen Entwicklungsprozess. Diese Aussage basiert darauf, dass innerhalb des vorliegenden Entwicklungsprojektes die vorgesehene Anwendung, die über herkömmliche Programmiersprachen programmiert wurde, über die eigenen Entwicklungswerkzeugen dieser Programmiersprachen programmiert werden konnte, ohne auf zusätzliche Werkzeuge greifen zu müssen. Obwohl es festgestellt wurde, dass bei der Gestaltung des Frontends die LC/NC-Entwicklung mit vielen der Instrumente der HC-Entwicklung arbeiten kann, haben manche teilweise einfache Instrumente auch gefehlt.

Darüber hinaus hat die LC-Entwicklung und, konkret formuliert, Zoho Creator, Einschränkungen im Bereich der Funktionslogik ergeben, da die Plattform nicht genug eigene Funktionen für die Implementierung gängiger Anforderungen anzubieten scheint, was den Entwicklungsprozess verlängern und erschweren kann, weil dadurch Alternativwerkzeuge und -lösungen gesucht werden müssen, um die Anforderungen einzuhalten. Außerdem hat die LC/NC-Entwicklung im Rahmen der vorliegenden Forschung sein Hauptvorteil, das Schreiben von weniger bis viel weniger Programmcode, nicht einhalten können, weil die LC-Version von LexiCode aus mehr Codezeilen als die HC-Version besteht.

## 5.1 Kritische Reflexion

Die bei der Umsetzung der LC-Version von LexiCode festgestellten Funktionsmängel sind, laut den Ergebnissen der Forschung, für Zoho Creator spezifisch. Da die Auswahl der LC-Plattform auf der eigenen Beschreibung von Zoho basiert, die auf ihrer Homepage zu finden ist und allgemeine Vorteile für die Entwicklung einer Anwendung über den gesamten Lebenszyklus aufzählt, war die Auswahl der Plattform nicht auf die für das vorliegende Entwicklungsprojekt relevanten Plattformeigenschaften fokussiert, die eher mehr Vorteile im Frontend-Bereich anbieten sollten.

Bei der Interpretation der Ergebnisse wurde zwar erklärt, welche der Anforderungen nicht umgesetzt werden konnten, und es wurden keine alternativen Lösungen oder zumindest Ansätze vorgestellt, die zu einer Lösung führen können. Obwohl dies getan wurde, weil das im Forschungsdesign schon so festgelegt war, ist es für die Plausibilität und Effektivität der LC-Entwicklung trotzdem ein Vorteil, Alternativlösungen für technische Einschränkungen zumindest in der Theorie zu erwähnen. Wenn eine Anforderung über Zoho Creator in Kombination mit anderen Werkzeugen implementiert worden ist, hat Zoho Creator trotzdem einen Beitrag zum finalen Ergebnis und hat die Implementierung ermöglicht.

Bezüglich des LC-Entwicklungsansatzes ist auch zu erwähnen, dass die beiden Versionen von LexiCode ungefähr die gleiche Menge an Code beinhalten, weil für die LC-Anwendung viel Code überhaupt verwendet wurde. Da der Programmcode für das Aussehen der Komponenten, der für die HC-Version von LexiCode geschrieben wurde, unverändert für die LC-Version übernommen wurde, hat dies dazu geführt, dass beide Versionen der Anwendung eine nahezu gleiche Anzahl an Codezeilen haben.

Obwohl der Ansatz des Value-Driven Testing eine ungefähre Schätzung der Wirtschaftlichkeit hinter dem systematischen Testen von LexiCode vor der Veröffentlichung liefert, stellt sich die Frage, ob er für LexiCode ein geeigneter Analyseansatz ist, da er, zumindest in der Forschung von Sneed und Jungmayr, für die Kalkulation der Testwirtschaftlichkeit betrieblich eingesetzter Software verwendet wird, bei der von einem wirklichen wirtschaftlichen Schaden gesprochen werden kann, da vergütete Sachbearbeiter mit der Software in der Ausfallzeit nicht arbeiten können und gegen diese vergütete Arbeitszeit kein betriebswirtschaftlicher Wertezuwachs generiert wird.

## 5.2 Empfehlungen für zukünftige Forschungen

Damit unterschiedliche Ergebnisse, als die in dieser Forschung erzielt werden können, muss sie unter unterschiedlichen Bedingungen ausgeführt werden. Zum einen muss bei einem vergleichendem Case Study wie das vorliegende die auszuwählende LC/NC-Plattform vor der Auswahl anhand der Dokumentation auf der Seite des Anbieters sowie weitere verfügbare Materialien wie Online-Tutorials grundlegend studieren werden und entwicklerseitig ist es zu analysieren, ob mit der jeweiligen LC-Plattform bereits ähnliche Lösungen umgesetzt worden sind.

Da manche der Anforderungen allein über Zoho nicht programmiert werden konnten, können weitere über die Integrationsmöglichkeiten andere Plattformen wie andere Entwicklungsumgebungen wie VSC oder *Eclipse* oder HC-Programmiersprachen wie JavaScript oder Java, die die Funktionen unterstützen, die in Zoho fehlen, an die Zoho Anwendung angebunden werden.

Damit in der LC/NC-Version der Anwendung die Anzahl der Codezeilen geringer als die in der HC-Anwendung ist, müssen bei einem zukünftigen Case Study die vorgefertigten Bausteine des Zoho Creators bzw., wenn eine andere Plattform als Zoho ausgewählt wird, die vorgefertigten Bausteine der ausgewählten LC-Plattform verwendet werden. Im Falle von Zoho müssen entsprechend der Vorlagen für Formulare, Seiten und Berichte eingesetzt werden, weil diese sogar über Mausklick anstatt über Drag-and-Drop erstellt werden können.

## 5.3 Ausblick

Trotz den genannten technischen Einschränkungen hat die LC/NC-Entwicklung Potenzial im Bereich der schnelleren einfacheren Entwicklung von kleineren Anwendungen, die mit der Zeit nicht skaliert werden müssen.

Die Zeitersparnis, die LC-Plattformen bei der Entwicklung bieten, äußern sich als vorgefertigte Bausteine aus, die die optische Entwicklung der Systemkomponenten beschleunigt. Die technischen Einschränkungen in Form von fehlenden Funktionen können bedeuten, dass es auf LC-Plattformen zurzeit einfach noch keine vorgefertigten Bausteine dafür entwickelt worden sind. Da es Bausteine für die UI-Komponenten gibt und sogar Bausteine für allgemeine, einfachere Funktionen, ist es zu erwarten, dass in der Zukunft auch Bausteine für komplexe Funktionen gibt, bei denen jeder Teilvorgang der Funktion ebenfalls per Mausklick oder Drag-and-Drop erzeugt werden kann. Eine zukünftige Entwicklungslandschaft, in die auch komplexere, größere Systeme über LC-Plattformen geschrieben werden, werden den Bedarf an HC-Entwicklern reduzieren und dazu führen, dass diese sich mit anderen Aufgabenbereichen beschäftigen müssen.

Referenzen

Ajimati, M. O., Carroll, N. & Maher, M. (2025). Adoption of low-code and no-code development: A systematic literature review and future research agenda. The Journal of Systems & Software, 225.

Aveiro, D., Freitas, V., Cunha, E., Quintal, F. & Almeida, Y. (2023): Traditional vs. low-code development: comparing needed effort and system complexity in the NexusBRaNT experiment. *2023 IEEE 25th Conference on Business Informatics (CBI)*, 1-10, https://doi.org/10.1109/CBI58679.2023.1.0187470.

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, it’s time to accept other success criteria. Elsevier Science Ltd.

Baumgarten, C., Rainer, E. & Stich, S. (2024). Professionelle Softwareentwicklung mit Low Code optimieren – eine Fallstudie. In: HMD Praxis der Wirtschaftsinformatik. Springer Verlag.

Boehm, B. et al. (2000). Software Cost Estimation with COCOMO-II. Prentice Hall.

Brandt-Pook, H. & Kollmeier, R. (2020). Softwareentwicklung kompakt und verständlich. Wie Softwaresysteme entstehen (3. Auflage). Springer Verlag.

Bühler, P., Schlaich, P. & Sinner, D. (2023). HTML und CSS. Semantik. Design. Responsive Layouts (2. Auflage). Springer Verlag.

Böhler, T. (2023). Software einfacher, flexibler und leichtfüßiger entwickeln. In: Produktion, (12), 16.

Di Ruscio, D., Kolovos, D. & De Lara, J. (2022). Low-code development and model-driven engineering: two sides of the same coin?

Elshan, E. & Binzer, B. (2024). Mehr als ein Trend?: Wie Low-Code die digitale Transformation unterstützt. In: HMD Praxis der Wirtschaftsinformatik, 1070-1087.

Flanagan, D. (2020). JavaScript: The Definitive Guide: Master the World’s Most-Used Programming Language. O‘Reilly Media, Incorporated.

Frank, B. (2023). ERP: The next big thing: Sind Lösungen auf Low-Code-Basis die Zukunft? In: IT-Management, 50-54.

Gabler Wirtschaftslexikon (2025). Minimum Viable Product (MVP). https://wirtschaftslexikon.gabler.de/definition/minimum-viable-product-mvp-119157/version368108.

Gabler Wirtschaftslexikon (2025). Time to Market. https://wirtschaftslexikon.gabler.de/definition/time-market-54271/version-277318.

Gialitakis, M.-K., Tsakalidis, G., Nikos, N. & Vergidis, K. (2024): Rapid prototyping of process-driven applications using low-code development platforms: A case study from the Greek public sector. 2024 International Conference Automatics and Informatics (ICAI) Automatics and Informatics (ICAI), 2024 International Conference. 650-656. https://doi.org/10.1109/ICAI63388.2024.10851543.

Hensen, U. (2023). Maßgeschnittene Software mittels Low Code. In: Factory Innovation, 33-37.

Jit, T. L., Chen, K. L., Ean, H. L., Pei, V. W., King, T. Y. & Chek, Y. O. (2024). Preliminary Study on the Accessibility and Learning Experience of Low-Code Development Platforms. 2024 5th International Conference on Artificial Intelligence and Data Sciences (AiDAS). 486-491. https://doi.org/ 10.1109/AiDAS63860.2024.10730306.

Liu, D., Jiang, H., Guo, S., Chen, Y., & Qiao, L. (2023). What's wrong with low-code development platforms? An empirical study of low-code development platform bugs. IEEE Transactions on Reliability, 695–709. https://doi.org/10.1109/TR.2023.3295009.

Schreiner, C. (2024). Entwicklungszyklen halbieren mit Low Code. In: Digital Engineering, 44-45.

Siroker, D. & Koomen, P. (2013). A/B Testing. The Most Powerful Way to Turn Clicks into Customers. Wiley.

Spierling, D. (2023). Mehr IT-Nachhaltigkeit und Klimaschutz durch Low-Code Development. In: Wirtschaftsinformatik & Management, 107-113.

Sneed, H. M. & Jungmayr, S. (2011). Mehr Testwirtschaftlichkeit durch Value-Driven-Testing. In: Informatik-Spektrum: Organ der Gesellschaft für Informatik e. V. und mit ihr assoziierter Organisationen. Springer-Verlag.

Zoho (04 März 2025). Dokumentation. https://www.zoho.com/creator/help/#documentation.

Zoho (04 März 2025). Preisvergleich. https://www.zoho.com/de/creator/pricing-comparison.html.

##### Leere Zoho-Seite nach Betätigen eines HTML-Links

Ein Bild, das Text, Screenshot, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Bearbeitung der Seitennavigation über den App-Menüersteller

Ein Bild, das Text, Screenshot, parallel, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Deluge Funktionen Teil 1

Ein Bild, das Text, Screenshot, Software, Computer enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Deluge Funktionen Teil 2

Ein Bild, das Text, Screenshot, Software, Computer enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Deluge Funktionen Teil 3

Ein Bild, das Text, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HC-Kontaktformular

Ein Bild, das Text, Rechteck, Diagramm, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HC-Kontaktformular

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Feldeigenschaften von Formularfeldern

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HTML-Editor im Form Builder

Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### NC-Kontaktformular über Zoho Creator

Ein Bild, das Screenshot, Text, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Anpassungsoptionen für den Fomrular-Baustein

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Präfix-, Suffix-, Pflichtfeld- und PBD-Angabe

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Feldtyp-, Feldgrößen- und Beschreibungstextangabe

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HC-Lückentest

Ein Bild, das Text, Software, Computersymbol, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### LC-Lückentest im Zoho-Creator

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HTML- und CSS-Code als getrennte Dokumente

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HTML-Snippet mit HTML- und CSS-CODE

Ein Bild, das Text, Screenshot, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Bootstrap Button im normalen Zustand

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Bootstrap Button im normalen Zustand

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HC-Kontaktformular bei 550 Pixel Bildschirmbreite

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### HC-Kontaktformular bei 350 Pixel Bildschirmbreite

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### LC-Abschlusstest bei 550 Pixel Bildschirmbreite

Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Werte zur Aufwandsermittlung bei der HC-Version von LexiCode

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Ermittlung des wahrscheinlichen Schaden bei der HC-Version

Ein Bild, das Text, Schrift, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Werte zur Aufwandsermittlung bei der LC/NC-Version von LexiCode

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

##### Ermittlung des wahrscheinlichen Schadens bei der LC/NC-Version

Ein Bild, das Text, Schrift, Reihe, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Erklärung der Echtheit

|  |  |  |
| --- | --- | --- |
| Hiermit erkläre ich, dass ich diese Bachelor-/Masterarbeit selbständig und ohne fremde Hilfe angefertigt habe. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und alle Quellen angegeben, denen ich Text und Inhalt entnommen habe. Diese Arbeit oder Teile davon wurden noch nie einem anderen Prüfungsausschuss vorgelegt. Ich erkläre mich mit einer Plagiatsprüfung meiner Arbeit durch einen Plagiatserkennungsdienst einverstanden. | | |
| Ort, Datum |  | Unterschrift |

1. Flanagan, 2020 [↑](#footnote-ref-1)
2. Di Ruscio, Kolovos & De Lara, 2022 [↑](#footnote-ref-2)
3. Baumgarten, Rainer & Stich, 2024 [↑](#footnote-ref-3)
4. Gabler Wirtschaftslexikon, 2025 [↑](#footnote-ref-4)
5. Gabler Wirtschaftslexikon, 2025 [↑](#footnote-ref-5)
6. Atkinson, 1999 [↑](#footnote-ref-6)
7. Baumgarten et al., 2024 [↑](#footnote-ref-7)
8. zitiert nach Boehm, 2000 [↑](#footnote-ref-8)
9. Sneed & Jungmayr, 2011 [↑](#footnote-ref-9)
10. Sneed & Jungmayr, 2011 [↑](#footnote-ref-10)
11. Sneed & Jungmayr, 2011 [↑](#footnote-ref-11)
12. Sneed & Jungmayr, 2011 [↑](#footnote-ref-12)
13. Sneed & Jungmayr, 2011 [↑](#footnote-ref-13)
14. Sneed & Jungmayr, 2011 [↑](#footnote-ref-14)
15. Siroker & Koomen, 2013 [↑](#footnote-ref-15)
16. Zoho, 2025 [↑](#footnote-ref-16)
17. Zoho, 2025 [↑](#footnote-ref-17)
18. Sneed & Jungmayr, 2011 [↑](#footnote-ref-18)
19. Sneed & Jungmayr, 2011 [↑](#footnote-ref-19)
20. (Sneed & Jungmayr, 2011, S. 207) [↑](#footnote-ref-20)
21. (Zoho, 2025) [↑](#footnote-ref-21)
22. Siehe Anhang A [↑](#footnote-ref-22)
23. Siehe Anhang B [↑](#footnote-ref-23)
24. Siehe Anhang C [↑](#footnote-ref-24)
25. Siehe Anhang D [↑](#footnote-ref-25)
26. Siehe Anhang E [↑](#footnote-ref-26)
27. Siehe Anhang H [↑](#footnote-ref-27)
28. Siehe Anhang I [↑](#footnote-ref-28)
29. Siehe Anhang J [↑](#footnote-ref-29)
30. Siehe Anhang N [↑](#footnote-ref-30)
31. Siehe Anhang O [↑](#footnote-ref-31)
32. Siehe Anhang P [↑](#footnote-ref-32)
33. Siehe Anhang Q [↑](#footnote-ref-33)
34. Siehe Anhang R [↑](#footnote-ref-34)
35. Siehe Anhang S [↑](#footnote-ref-35)
36. Siehe Anhang T [↑](#footnote-ref-36)
37. Siehe Anhang U [↑](#footnote-ref-37)
38. Siehe Anhang V [↑](#footnote-ref-38)
39. Siehe Anhang W [↑](#footnote-ref-39)
40. Siehe Anhang X [↑](#footnote-ref-40)
41. Siehe Anhang Y [↑](#footnote-ref-41)
42. Siehe Anhang Z [↑](#footnote-ref-42)