1. Motivation und Ziele

Die Begriffe No-Code, Low-Code und High-Code müssen voneinander abgegrenzt werden. High-Code bezieht sich auf die klassische quelltextbasierte Softwareentwicklung, unter Berücksichtigung unterschiedlicher Programmierparadigmen (Imperativ, Objektorientiert, Funktional, …). Typisch ist der Einsatz quelltextorientierter Programmiersprachen wie z.B. Cobol, C, C++, Java, JavaScript oder auch Python. Mit Hilfe von Compilern bzw. Interpretern werden diese Quelltexte semantikerhaltend in ausführbare Softwareanwendungen überführt. Neben einer entwicklungsbegleitenden Qualitätssicherung (Softwaretest) bedarf es letztlich des Deployments in physische oder auch virtualisierte Laufzeitumgebungen.

Die Begriffe No-Code und Low-Code werden durch [Bitkom 2021] in folgender Weise definiert:

> *„Als No Code (NC) bezeichnet man Plattformen, mit denen ganz ohne Programmieraufwand Applikationen erstellt werden können. Ein »No-Coder« fügt seine Anwendung per Mausklick aus vorkonfektionierten Bestandteilen zusammen, ohne dass er den Code manuell ergänzen oder anpassen kann.“*

> *„Bei Low Code (LC) ist das Grundprinzip, dass in einer grafischen Entwicklungsumgebung Applikationen per Drag-and-Drop aus vorgefertigten Bausteinen zusammengesetzt werden. Doch der Clou bei Low Code ist, dass bestehende Bausteine mittels klassischen Codes von der Entwicklungsabteilung ergänzt und individuell angepasst werden können.“*

Im Rahmen dieses Buches soll auf den Low-Code-Ansatz fokussiert werden. Dem entsprechend können die Themen folgende Rollen adressieren:

- Klassische IT-Experten (Anforderungsmanagement, Softwarearchitektur, Programmierung, Softwaretest, …)
- Informationsmanagement (Management von Geschäftsprozessen und Geschäftsdaten bzw. Unternehmensarchitekturen)
- Business Developer (Operative und strategische Unternehmensentwicklung im Diskurs fachlicher Domänen)
- Citizen Developer (Fachexperten als Softwareentwickler mit geringen IT- und Programmierkenntnissen).

Darüber hinaus gilt das Interesse dem Qualitätsmanagement und damit einhergehenden Testaspekten bzw. letztendlich der Installation entwickelter Low-Code-Lösungen im Rahmen konkreter Serversysteme.

Abbildung 1-1: KI-generierte Abbildung mit Hilfe von Stichwörtern

Abbildung 1-1 wurde mit Hilfe einer KI generiert (speziell https://playground.com/create - KI-Modell playground v2 – 02. Februar 2024). Als Grundlage wurden der KI die Stichwörter „No-Code, Low-Code, High-Code und Software Development“ übergeben. Während im Hintergrund der Abbildung die Sprachanweisungen der originären Rechnerplattform zu erahnen sind, finden sich im Vordergrund vielfältig vernetzte Komponenten unterschiedlicher Ausprägung, welche die fachlichen benötigen Funktionen der Softwareanwendung repräsentieren. Aus Sicht der Autoren entstand eine durchaus gelungene Abstraktion dieser Begriffswelten, gilt es doch mit Hilfe des No-Code- bzw. Low-Code-Ansatzes eine kompositorische Softwareentwicklung zu etablieren. Statt textlicher Anweisungen werden grafische Repräsentationen vordefinierter Komponenten verwendet, die so für Eingabedialoge, Prozessabläufe, Datenbankschnittstellen oder auch Integrationen zu externen Systemen verwendet werden. Benötigte Verbindungen zur Abbildung funktionaler Interaktionen werden mit Hilfe von Kanten realisiert, die ihrerseits hinsichtlich der auszutauschenden Daten im Detail zu spezifizieren sind.

1.1 Historische bzw. evolutionäre Entwicklung

Die Idee, Software ohne Zuhilfenahme zuweilen kryptisch anmutender Programmiersprachanweisungen zu entwickeln, kann auf viele Jahrzehnte zurückblicken. Vorschläge wie das Rapid Application Development (kurz RAD) versuchten zum Anfang der 90er Jahre eine prototypen- und modellgetriebene Entwicklung im Diskurs datenbankorientierter Anwendungen zu etablieren. Weitere Ansätze finden sich mit modellgetriebenen Vorgehensweisen, die insbesondere von Standardnotationen wie UML[2] profitieren sollten. Zumeist wurde dabei allerdings ausschließlich auf das Element der Klassendiagramme (Attribute und Methoden) zurückgegriffen, woraus sich massive Limitierungen bezüglich der so abbildbaren Algorithmen (geringe Komplexität) ergaben. Darüber hinaus war es bei industriellen Projekten schwierig, die Konsistenz zwischen Modellnotationen und Quellcodefragmenten über einen längeren Zeitraum zu gewährleisten. Obwohl mit den vor ca. 20 Jahren eingeführten Komponentenmodellen (z.B. Enterprise Java Beans – jetzt Jakarta Enterprise Beans) noch keine gravierende Reduktion des benötigten Quellcodes einherging, konnten auf dieser Grundlage realisierte Projekte deutlich effizienter und vor allem fachorientierter (d.h. Konzentration auf Geschäftslogiken) realisiert werden. Neben den genutzten Frameworks, die mögliche Architekturansätze durch die Bereitstellung korrespondierender Pattern begrenzten, profitierten diese Lösungen von umfänglich eingesetzten Quellcodegeneratoren (z.B. Abstraktion benötigter Netzwerkskommunikationen). In diesem Zusammenhang erfolgte auch die Bereitstellung von ausschließlich grafisch orientierten Entwicklungswerkzeugen, die eine „LEGO“-orientierte Komposition vorgefertigter Bausteine erlaubte. Ein entsprechendes Beispiel unter Verwendung des Java-Studios findet sich unter [Schmietendorf et al. 2002] ab S. 114.

Eine drastische Veränderung ergab sich durch die Bereitstellung webbasierter Services, die mit Hilfe des Cloud-Paradigmas über das Internet angeboten wurden. Erstmals ergaben sich so bereits in Ausführung befindliche Daten- und Logik-Services (allg. Web Services bzw. Web-APIs) mit einer intern „engen“ Bindung und „loose“ gekoppelten Schnittstellen. Auf der Grundlage der wiederverwendbaren Services war es nunmehr möglich, benötigte Kompositionen grafisch zu konfigurieren (anfänglich häufig mit dem Begriff der Mashups verbunden) und in entsprechenden Laufzeitcontainern zur Ausführung zu bringen. Aktuell wird die Low-Code- (wie auch die No-Code-) Entwicklung nicht selten mit webbasierten Apps in Verbindung gebracht. Allerdings existieren auch Ansätze, die kompositorisch er-

2 https://www.uml.org/index.htm

stellte Web-APIs und deren Spezifikationen bzw. prozessorientierte Integrationslösungen im Back-End in den Mittelpunkt der Entwicklung stellen. In diesem Zusammenhang benötigte Plattformen werden entsprechend [Gartner 2021] die Form der Softwareentwicklung gravierend beeinflussen:

> *„By 2025, 70% of new applications developed by organizations will use low-code or no-code technologies, up from less than 25% in 2020. The rise of low-code application platforms (LCAPs) is driving the increase of citizen development, and notably the function of business technologists who report outside of IT departments and create technology or analytics capabilities for internal or external business use.“*

1.2 KI-basierte Ergebnisse zum Low-Code-Begriff

Im Zusammenhang mit der Verfügbarkeit einer generativen KI (u.a. Large Language Models, kurz LLMs), die auf der Basis durch Menschen erstellter Inhalte (allgemein Trainingsdaten) u.a. Texte, Bilder, Videos oder auch Quellcode generativ bereitstellen kann, stellt sich grundsätzlich die Frage nach der Sinnfälligkeit, ein neues Buch bereitzustellen. Das gilt umso mehr im Diskurs der Informationswissenschaften bzw. korrespondierender Ingenieurdisziplinen.

Aus Sicht der Autoren führt an der Verwendung generativer KI-Ansätze zukünftig kein Weg vorbei. Obwohl uns die vielfältigen Nachteile (u.a. ungeprüfte Inhalte, unzureichende Zitierfähigkeit, unklare Urheberrechte) durchaus bewusst sind, halten wir die sich daraus ergebenden Möglichkeiten dennoch für gravierend. Mit Hilfe der KI-basierten Auseinandersetzung konnte das Themengebiet frühzeitig strukturiert bzw. bezüglich der betrachteten Sachverhalte abgegrenzt werden.

Im Zusammenhang mit der Erstellung des vorliegenden Buchs wurde das ChatGPT-System zu den folgenden für das Buch zentralen Themenkomplexen befragt:

- Wie entstand die Low-Code-Idee in den letzten 30 Jahren, welche Meilensteine gab es?
- Welche Möglichkeiten und Grenzen bieten sich mit einer Low-Code-basierten Softwareentwicklung?
- Für welche Anwendungsszenarien eignet sich eine Low-Code-basierte Softwareentwicklung und für welche eher nicht?
- Welche Anforderungen bzw. Kriterien sollten Low-Code-Entwicklungsplattformen erfüllen?

- Wie könnte eine Klassifizierung bestehender Low-Code-Plattformen aussehen?

Die bereitgestellten Antworten (vgl. Anlage A) können für eine erste eher oberflächliche Reflektion der Themenstellung durchaus überzeugen. Dem entsprechend stellt sich die Frage nach darüber hinaus gehenden Inhalten, die dem Buch eine Berechtigung geben bzw. entsprechende Alleinstellungsmerkmale verleihen. Aus Sicht der Autoren finden sich diese mit den folgenden Sachverhalten:

- Empirisch gesicherte Bewertung der durch die KI bereitgestellten Aussagen durch die Reflektion zitierfähiger Fachbeiträge bzw. verfügbarer Expertenaussagen. → Verifikation KI-Aussagen
- Bereitstellung von im Umgang mit Low-code-Lösungen gewonnen Erfahrungen, welche durch dokumentierte Fallstudien nachvollzogen und vor allem reproduziert werden können. →Praxis-Erfahrungen
- Bereitstellung personifizierter Aussagen, die einen Beitrag zum Meinungsdiskurs in der einschlägigen Community leisten und so einer kritischen Diskussion unterzogen werden können. →Praxis-Erfahrungen
- Reflektion praktischer Erfahrungen aus Industrie und Forschung, so dass z.B. hinsichtlich der KI auch Beziehungen unbekannter Einflussfaktoren wiedergegeben werden können. →Praxis-Erfahrungen
- Vermeidung allgemeiner Aussagen (aktuell typisch für KI-ChatBots) durch die Belegung bzw. Referenzierung konkret eingesetzter Modelle, Methoden oder auch Architekturansätze. →Vermeidung oberflächlicher Allgemeinaussagen
- Kreative Reflektion der Low-code-Themenstellung, so dass sich einhergehende Möglichkeiten, aber auch Risiken leichter auf potentiell eigene Anwendungsszenarien abbilden bzw. bewerten lassen. → Anwendungsfälle Low-Code

Die hier gewählte offensive Auseinandersetzung mit KI-basierten Ergebnissen soll im Sinne von Erfolgskriterien für die Bewertung der Buchinhalte herangezogen werden. Darüber hinaus erhoffen wir uns die Etablierung eines kritischen Diskurses (webbasierte Diskussionsplattform), so dass die Buchinhalte eine sukzessive Verbesserung in nachfolgenden Auflagen erfahren können. Für die inhaltliche Bearbeitung der folgenden Kapitel wurde also das ChatCPT-System durchaus zu Rate gezogen, eine unreflektierte Übernahme der generierten Inhalte erfolgte allerdings an keiner Stelle!

1.3 Treiber der Low Code Softwareentwicklung

Unternehmen sehen sich entsprechend dem folgenden Zitat von [Gartner 2022] einem extremen Spannungsfeld im Diskurs der digitalen Transformation ausgesetzt:

> *"Organizations are increasingly turning to low-code development technologies to fulfill growing demands for speed application delivery and highly customized automation workflows," said Varsha Mehta, Senior Market Research Specialist at Gartner. "Equipping both professional IT developers and non-IT personas — business technologists — with diverse low-code tools enables organizations to reach the level of digital competency and speed of delivery required for the modern agile environment."*

Auf der einen Seite stehen vielfältige Herausforderungen des Marktes im Zusammenhang mit den Erwartungshaltungen der potentiellen Kunden, auf der anderen Seite gilt es die unternehmensinterne Organisation effizient und flexibel zu gestalten. Häufig können historisch gewachsene IT-Organisationen den Erwartungen externer wie interner Kunden (allgemein Fachabteilungen) nicht standhalten. Gerade bei kleineren und schnell benötigten Softwarelösungen stellt sich die Frage, inwieweit ggf. Standardlösungen bzw. sehr einfach zu implementierende Anwendungen in unmittelbarer Verantwortung der Fachabteilungen zum Einsatz kommen können, vgl. Abbildung 1-2.

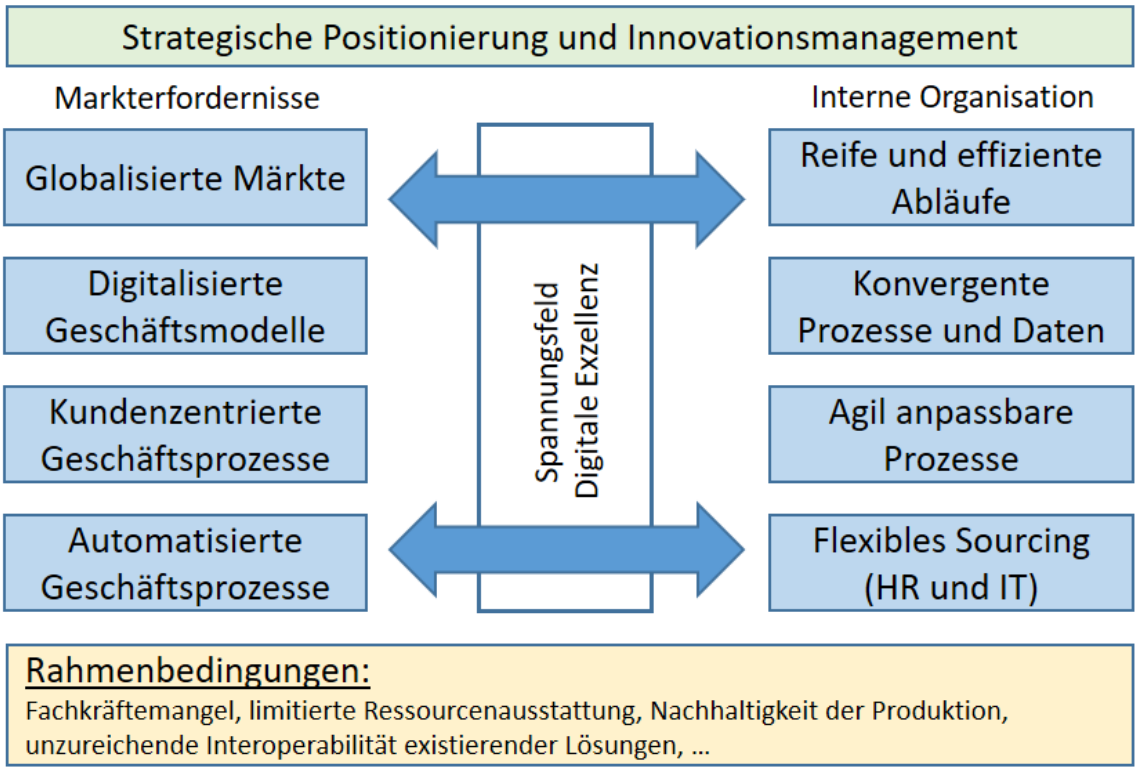

Abbildung 1-2: Unternehmerisches Spannungsfeld im Diskurs der digitalen Transformation