Low-Code == Low Quality?

Low-Code verspricht eine große Zeitersparnis, da Entwickler weniger Code sc
ben müssen – im Extremfall praktisch sogar keinen. Ist deshalb die Zeit gekor
in der sich Entwickler neue Beschäftigungen suchen müssen?

Know-how 14.08.2018 08:01 Uhr – Lofi Dewanto, Manuel Klein – 422 Kommentare

Im Zeitalter von Digitalisierung, IT 4.0 und ähnlichen om-
nipräsenten Themen lässt sich festhalten: Die Geschwindigkeit in der IT nimmt fort

fend zu. Vorbei sind die Zeiten beispielsweise von langen Architekturphasen oder F
pen. Agile Softwareentwicklung ist schon deshalb kaum vermeidbar, weil es nicht r
möglich ist, auf die Fertigstellung eines klassischen Softwareprojekts zu warten un
dann in Produktion zu gehen. Seien es die Anpassung des Auftragsmanagements o
Erweiterungen zur genaueren Datenanalyse, am liebsten möchte der Fachbereich
gebnis bereits gestern sehen.

Im vorliegenden Artikel schauen die Autoren auf die Geschichte der Low-Code-Ans
betrachten die aktuellen Angebote und bewerten, welchen (Zeit-)Gewinn sie bring
nen sowie die dabei in Kauf zu nehmenden Kosten und Einschränkungen.

RAD als Vorläufer Nummer eins

Bereits in den 90er-Jahren entstanden erste Versionen von RAD-Produkten (Rapid A
cation Development), etwa Visual Basic, Delphi oder auch Oracle Forms. Mit dieser
grammierumgebungen können Entwickler ihre Desktop-Anwendung visuell "zusar
setzen". Sowohl die Elemente der Benutzeroberfläche als auch die Logik lassen sic
Komponenten in einer Komponentenpalette ablegen. Die grafische Oberfläche ste
Mittelpunkt und wird Schritt für Schritt um die Geschäftslogik angereichert. So zeic
sich RAD-Tools als einfach zu erlernende Umgebungen aus. Die Oberfläche als zent
Element ist anschaulich und ermöglicht frühe Vorstellungen von der Anwendung.

Darüber hinaus wurden andere wichtige Aspekte für Geschäftsanwendungen beda
Zur Anwendungsverteilung auf dem Desktop lässt sich eine einzige integrierte "fat'
Datei zum Verteilen oder eine schlankere EXE mit separaten DLL-Bibliotheken gene

Allerdings hatten und haben RAD-Tools auch Einschränkungen. Sie sind in der Rege
prietär. Das heißt, ein Verlassen der Umgebung des Tools ist nicht oder nur eingesc
möglich. Die Zielumgebung ist häufig festgelegt, beispielsweise Microsoft Windows
sual Basic und Delphi, ein Oracle-Applikationsserver nebst Datenbank für Oracle F
Embarcadero hat allerdings angefangen, Delphi von einer reinen Windows- in Rich
ner Cross-Plattform-Software zu erweitern. So lassen sich neben Windows heute a
macOS, Linux und insbesondere mobile Apps generieren. Die gemeinsame Arbeit a
Anwendung ist kaum oder zumindest nur mit Einschränkungen möglich, da sie me
geringfügig modular aufgebaut sind. Eine Trennung ist somit höchstens anhand ve
dener Masken realisierbar.

RAD-Tools wie Visual Cafe und Borlands JBuilder hatten zudem mit der Geschwind
beziehungsweise dem Ressourcenbedarf zu kämpfen. Getreu dem Motto "eat your
dog food" waren sie selbst in Java geschrieben, was zur damaligen Zeit einen enor
Ressourcenhunger bedeutete.

Im 21. Jahrhundert verschwanden die meisten genannten RAD-Umgebungen vom

Java ist heute die meistgenutzte Programmiersprache und Eclipse immer noch die verbreitetste Entwicklungsumgebung. "Klassische" UI-Toolkits wie Swing und SWT den jedoch zunehmend bedeutungslos und Anwendungen praktisch ausnahmslos Web geschrieben. Der in Eclipse enthaltene WYSIWYG-Designer (What You See Is W Get) WindowBuilder fristet somit ein Schattendasein.

Für den Webbereich gibt es einige kommerzielle visuelle Editoren wie MyEclipse od Open-Source-Produkte wie die JBoss Tools sowie NetBeans. Oracles Application E (APEX) bietet ebenfalls eine Weboberfläche an, allerdings befinden sich Benutzer d Abhängigkeit zur Oracle-Datenbank.

Die Oberflächengestaltung erfolgt zunehmend mit HTML, CSS und JavaScript, wor gleichzeitig der zusätzliche Berufszweig des Webdesigners entstanden ist. Untersc che Webbrowser, Betriebssysteme wie Linux und macOS und Devices vom PC über Smartphone bis zu Wearables kommen als Zielsysteme in Frage. Schließlich sind N wicklungen nur noch in den seltensten Fällen wirklich autark, in aller Regel sind an Systeme und/oder Datenquellen anzubinden.

An dieser Stelle ist es spannend, dass Desktop-Anwendungen auf mobilen Geräten gewisse Renaissance erleben. Mobile Anwendungen, "Apps" unter Android und iOS fen komplett lokal. Die Entwicklungsumgebungen bieten entsprechend einen gute ellen Editor mit Android Studio und Xcode an. Allerdings sind auch sie durch Weba dungen bedroht, sei es durch responsive Webapps oder Techniken wie Progressive Apps (PWA).

MDSD als Vorläufer Nummer zwei

Die Idee der modellgetriebenen Softwareentwicklung (MDSD, Model Driven Softwa velopment) ist, den Abstand zwischen Fachbereich und Entwicklung mit einer ode reren Modellierungssprachen zu verkleinern. Modelle sollen einen ganzheitlichen u meinsamen Blick auf die Domäne ermöglichen, die sowohl den technischen als ins dere auch den fachlichen Anforderungen Rechnung trägt. UML (Unified Modeling L ge) und BPMN (Business Process Modeling Notation) haben sich in Geschäftsanwe gen durchgesetzt. In Anwendungen, in denen Prozesse (BPMN), Struktur (UML-Klassenmodell) und Zustände (UML-Zustandsmodell) nicht trivial sind, ist man froh elle und ausführliche Modelle zu besitzen. Als Nebenprodukt des zu generierenden dells erhält man so eine jederzeit aktuelle Dokumentation.

Generatoren und Interpreter sowohl zur Entwicklungs- als auch zur Laufzeit werde schließend verwendet, um den Code soweit wie möglich aus dem Modell zu generi beziehungsweise zu interpretieren. Der restliche Anteil, der nicht (sinnvoll) als Mod stellbar ist, wird zusätzlich manuell und oft textuell kodiert.

MDA war über Jahre, wenn nicht gar Jahrzehnte, eines der Themen, die die Softwa wicklung revolutionieren sollten. Allerdings ist das immer wieder erneute Generier besondere bei kleinen Änderungen wie dem Hinzufügen eines neuen Attributs mül und verlangt in einem Projekt mit mehr als einem Entwickler schnell nach einem g ren Koordinationseinsatz. Auch die Nahtstelle zwischen generiertem und eigenem bleibt problematisch. So gab es lange Zeit größeres Interesse an UML-Produkten w droMDA. Heute spielen sie jedoch nur noch eine untergeordnete Rolle. BPMN erfre nach wie vor großen Interesses, da Fachbereiche gut mit der Darstellungsweise zu kommen. Als populäres Beispiel sei das Open-Source-Produkt Camunda genannt. bildet jedoch nur einen relativ schmalen Ausschnitt aus einer Software ab und läss somit kaum als vollständiger Ansatz verstehen.

Neben den genannten (und vielen weiteren ähnlichen) Plattformen dürfte so zieml der Entwickler mit ein paar Berufsjahren auf dem Buckel ambitionierte Versuche e erlegenden Wollmilchsau kennengelernt haben. Also den Versuch, ein Framework, Plattform zu implementieren, die im Rahmen des eigenen Geschäftskontexts einer teil der Softwareentwicklung erübrigt, standardisiert oder in irgendeiner Form vere facht. Dazu gehören dann Design- und Architekturvorgaben für den Entwicklungsp und die IT-Landschaft.

Wie so häufig steckt der Teufel dabei im Detail. Trotz aller guten Ansätze, die zur En dung für eine eigenentwickelte Plattform geführt haben, werden über kurz oder la ge auftauchen, die so gar nicht zu dem überlegten Modell passen. Das können, mü aber keine eigenen Fehlentscheidungen sein. Wer hat 2007 oder sogar noch eher d gedacht, das eine Software auch in der eingeschränkten Umgebung eines Smartph laufen muss?

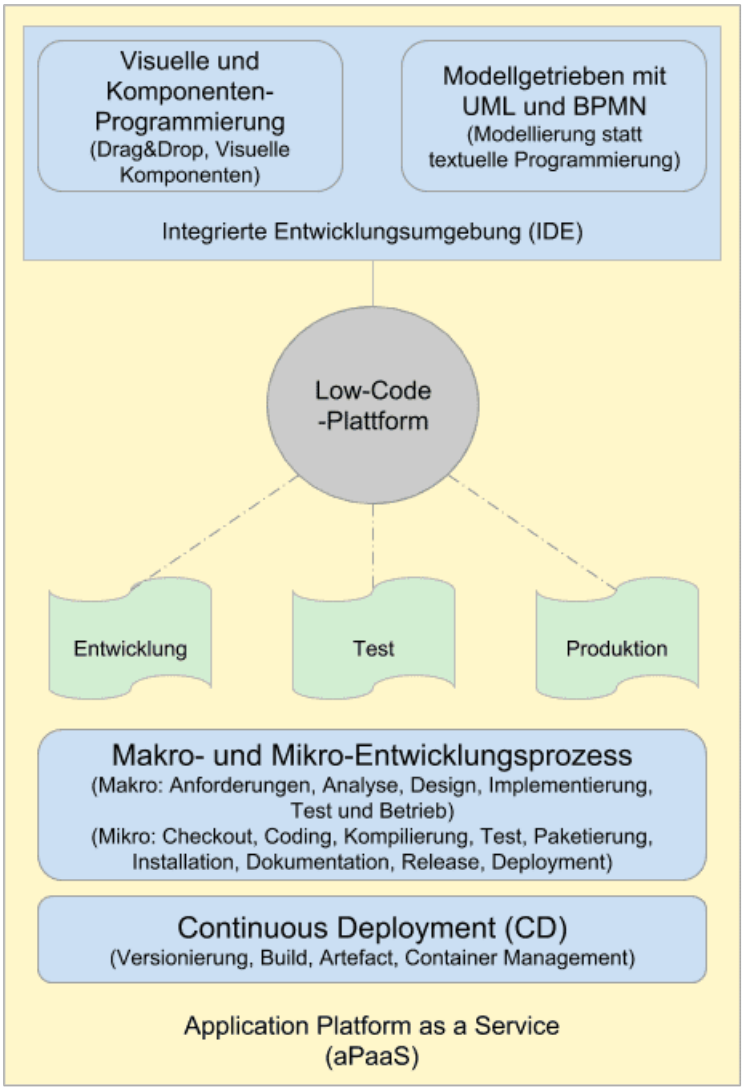
Low-Code- beziehungsweise No-Code-Plattformen

Unter anderem durch eine Veröffentlichung der Forrester Group ist der Begriff Low publik geworden. Im Prinzip geht es darum, möglichst viele Konzepte unter einen bekommen, die das Schreiben von Code erübrigen oder zumindest deutlich reduzi So enthalten die Low-Code-Plattformen die Vorgehensweisen von RAD und MDSD rücksichtigen ALM (Application Life Cycle Management) sowie Continuous Integrat beziehungsweise Continuous Deployment (CD) für Entwicklungs-, Test- und Produ umgebungen – kurzum den Makro- und Mikro-Entwicklungsprozess.

Um eine "One-Click"-Erstellung der gesamten Umgebung zu ermöglichen, muss ei sprechende PaaS (Platform as a Service) zur Verfügung stehen. Dazu existieren On- premise-Angebote wie die Container-Plattform OpenShift, in der Regel jedoch ents chende Cloud-Services. Diese aPaaS (Application Platform as a Service) bieten alle was eine Anwendung für ihren gesamten Lebenszyklus benötigt – von der ersten P

nungsphase über die Entwicklung, Abnahme bis in den Betrieb. Abbildung 1 zeigt d chitektur einer Low-Code-Plattform.

Wer Interesse an der Geschichte der Softwareentwicklung hat, sei an der Stelle auf Ähnlichkeiten der Low-Code-Plattformen zu den CASE-Tools (Computer Aided Soft Engineering) hingewiesen. Hier gilt wohl der Spruch "Wer zu früh ist, den bestraft d ben", CASE-Werkzeuge stammen aus einer Zeit weit vor Cloud & Co. und damals ga viele der Möglichkeiten von heute noch nicht.



Komponenten und Architektur einer Low-Code-Plattform (Abb. 1)

Eine umfassende Low-Code-Plattform sollte sämtliche in Abbildung 1 dargestellte pekte der Softwareentwicklung abdecken. Entwicklungswerkzeug(e), Entwicklung und Produktionsumgebung sowie der Entwicklungsprozess sind in einer Plattform

sammengefasst. Die Möglichkeit, sämtliche Funktionen als Cloud-Service (aPaaS) z
wenden, vereinfacht den Einstieg in solche Low-Code-Plattformen. Eigene Server u
stallationen sind nicht erforderlich.

Aus Sicht der Anwendungsentwickler haben sich die Elemente einer Anwendung u
Prozess der Entwicklung seit den 90er-Jahren bis heute konzeptionell nicht veränd
(siehe folgenden Kasten zur Struktur einer Anwendung und des Entwicklungsproze

Präsentationsschicht:

- <Desktop-, Web- und Mobilen-Benutzeroberfläche>: Visuelle Editoren für Benutzerol fläche
- <Präsentationslogik>: Textueller Code

Geschäftslogikschicht:

- <Geschäftsprozesse>: Visuelle Editoren für BPMN und/oder UML-Aktivitätsmodelle
- <Geschäftsprozesslogik>: Textueller Code
- <Geschäftsentitäten>: Visuelle Editoren für UML-Klassenmodelle
- <Geschäftsentitätenlogik>: Textueller Code

Datenschicht:

- <Datenbank>

Basis:

- <Security>
- <Benutzermanagement>
- <Authentifizierung: Login (Single Sign On)>
- <Autorisierung: Zugriffsmanagement>
- <Token-Generierung für API-Zugriffe>

Makro- und Mikro-Entwicklungsprozess:

- User Stories, Aufgaben-, Version-Tracking und Dokumentation
- Integrierte Entwicklungsumgebung
- Metriken-Plattform für den Code-Analyse
- Continuous Deployment für Entwicklungs-, Test- und Produktionsumgebung: Quellc Versionierungssystem, Build-Server, Artefakt-Management, Server- und Container-

Management

Die aktuellen Low-Code-Plattformen bieten folgende Unterstützung bei der Anwer
entwicklung:

1. Präsentationsschicht: Mit visuellen Editoren werden die UI-Elemente einfach zusammengekli
wunschgemäß arrangiert. Eine Abstimmung mit dem Fachbereich anhand der "echten" Maske
jederzeit möglich.
2. Geschäftslogikschicht: Mit visuellen Editoren für BPMN und UML lassen sich Prozesse und
Geschäftsentitäten entwerfen. Diese Editoren können die detaillierte Logik und einzelne
Prozessschritte vergleichbar mit Flowcharts umsetzen.
3. Datenschicht: Entweder werden die SQL-Befehle aus den Geschäftsentitäten komplett generie
die Datenbankstruktur wird visuell mit einem ERM-Editor (Entity Relationship Model) erstellt.
4. Basisdienste wie Authentifizierung und Autorisierung werden meistens innerhalb einer Menüs
angeboten.

In der Theorie sehr sinnvoll, stellen sich in der Praxis jedoch schnell Fragen: So ist e
tiv einfach, Elemente übersichtlich zu arrangieren. Wie aber sollen sich diese bei ei
Veränderung der Auflösung oder gar einem anderen Anzeigegerät verhalten? So m
möglich sein, entweder das Verhalten oder mehrere Maskenvarianten für verschied
Auflösungen und Geräte zu hinterlegen.

Auch bei der Modellierung der Geschäftsprozesse stellen sich schnell einige Fragen
es allein aus Verständnis- und Dokumentationszwecken sicherlich sinnvoll, die Pro
im Großen abzubilden. Ob dagegen eine schnell zu programmierende if-then-else-
Anweisung aufwendig in einem Modell beschrieben werden muss, ist fraglich. Eber
sind Generatoren beim Erstellen der Datenbankobjekte und -zugriffsschichten be-
schränkt, zudem sollte auch der Umgang mit bestehenden Strukturen möglich sein

In der Theorie sollte jedes Unternehmen klare Vorgaben bezüglich des Makro- und
Entwicklungsprozesses haben und die nötige Infrastruktur idealerweise per Knopf
zumindest aber per Checkliste erzeugen können. In der Praxis zeigt sich jedoch, da
Rad – oder Teile von selbigem – jedes Mal neu erfunden wird. Macht sich das zwei T
dauernde Onboarding neuer Entwickler bei einer Projektlaufzeit von sechs Monate
mehr kaum bemerkbar, sieht die Rechnung bei häufig wechselnden Teams und ku
Projektlaufzeiten ganz anders aus. Der Vorteil, die gesamte Umgebung von einer L
Code-Plattform zur Verfügung gestellt zu bekommen, ist daher nicht zu unterschät
Dies ist gegebenenfalls mit weniger Flexibilität zu bezahlen.

Tabelle 1 stellt einige Low-Code-Plattformen mit ausgewählten Eigenschaften dar.
genschaften wurden hierbei untersucht:

- Besitzt die Plattform visuelle Editoren und lassen sich aus Modellen Code transparent generie
- Umfasst die Plattform den kompletten Makro- und Mikro-Entwicklungsprozess?
- Lässt sich das Ergebnis exportieren und in die andere Plattform importieren beziehungsweise in einer Entwicklungsumgebung öffnen?
- Welche Programmiersprachen werden für die detaillierte Anpassung unterstützt?

Produkt	Beschreibung	Visuelle Editoren und modellgetrieben	Kompletter Entwicklungsprozess	Ex Pl
Zoho Creator	Low-Code-Plattform der bekannten CRM-Plattform. Große Auswahl an Tools wie: – Workflow-Unterstützung mit Zoho Flow – Bugtracker mit Zoho Bugtracker Diese Plattform kommt fast ohne Programmieren aus. Die Programmiersprache Deluge ist eine leicht zu lernende visuelle Programmiersprache.	Ja	Ja	Ne
Google App Maker	Low-Code-Plattform von Google, derzeit nur für G-Business-Kunden vorhanden. Für komplexe Logik werden Programmierkenntnisse in JavaScript benötigt. Einfache Anwendungen mit Google-Produkt-Integrationen sind sehr einfach und elegant umgesetzt.	Ja	Nein	Ne
Microsoft PowerApps	Diese Plattform bietet drei unterschiedliche Ansätze an:	Ja	Nein	Ne

	– Canvas App: Mit Drag & Drop kann eine generelle App erstellt werden. – Modellgesteuerte App: Auf Basis von Prozessen und Daten wird eine App entwickelt. Hierbei spielt das Layouting einer App keine Rolle, da diese fest verankert ist.			
Outsystems	Marktführer bei Low-Code nach Forrester und Gartner. Umfangreiche Plattform, auch in der Cloud. Programmierkenntnisse erforderlich.	Ja	Nein	Ja
Mendix	Ebenso Marktführer bei Low-Code nach Forrester und Gartner. Mendix bietet komplette Makro- und Mikroentwicklungsprozess aus einer Hand an. Die Plattform nutzt gängige offene Standards wie BPMN, UML, OData, SAML2. Programmierkenntnisse erforderlich.	Ja	Ja	Ja

Low-Code als Lösung aller Softwareentwicklungsprobleme?

Ansätze der Low-Code-Programmierung sind bereits Jahrzehnte alt und haben die Zielsetzung einer vereinfachten Oberflächenentwicklung (à la RAD) beziehungsweise der ganzheitlichen Softwarearchitektur (à la MDA). Da mit dem technischen Fortschritt die Anforderungen an die Software deutlich gestiegen sind, hat sich jedoch an den Problemstellungen bei diesen beiden Aspekten grundsätzlich wenig verändert. Lässt sich durch Visualisierung vieles

vereinfachen, gibt es weiterhin komplexe Sachverhalte,
die zu kodieren sind.

An dieser Stelle sei auf das meistens geltende Paretoprinzip in der Softwareentwicl
hingewiesen: Low-Code-Plattformen erreichen 80 Prozent der Ergebnisse eines So
projekts, für die verbleibenden 20 Prozent sind aber leider 80 Prozent des Aufwanc
leisten. Die zeitintensiven komplexesten Teile einer Software lassen sich auf absel
Zeit nicht generieren.

Einen großen Vorteil können Low-Code-Plattformen dank der heutigen Möglichkei
Application Platform as a Service (aPaaS) bieten. Der Aufbau einer kompletten Ent
lungsumgebung ist somit deutlich vereinfacht. Allerdings gibt es in diesem Bereich
wie Red Hats OpenShift und in Verbindung mit Docker-Containern Open-Source-Pr
wie Confluence, JIRA, Jenkins, SonarCube und Nexus, die teilweise deutlich vielsei
sind. Die Investition für den Aufbau der Entwicklungsautobahn auf Basis von Open
Source-Produkten muss vorher getätigt werden, für eine langfristige digitale Strate
diese jedoch sinnvoll, da Know-how und Personal aufgebaut werden können. Hers
unabhängigkeit lässt sich mit diesem Ansatz ebenfalls erreichen.

Insgesamt ist somit eine detaillierte Kosten-Nutzen-Betrachtung nötig, sofern der I
einer Low-Code-Plattform in Erwägung gezogen wird. Dabei sind auch "weiche" Fa
in Betracht zu ziehen. So stoßen Low-Code-Plattformen häufig auf Ablehnung, was
benenfalls auch im eigenen Entwicklerteam für Unruhe sorgen kann. (ane)

Dr. Lofi Dewanto
arbeitet als Teamleiter der Softwareentwicklung beim Umweltdienstleister Intersero
Dienstleistungs GmbH in Köln. Er engagiert sich insbesondere für "javanische" Open-
Source-Software sowie MDA.

Manuel Klein
arbeitet als Fachbereichsleiter Enterprise Java Development bei der MT AG in Ratinge
ben der Programmierung mit Java EE beschäftigt er sich gemeinsam mit seinem Tea
den aktuellen JavaScript-Frameworks sowie hybriden Apps.

Kommentare lesen (422 Beiträge)

Forum bei heise online:

Tools

Themenseiten:

SOFTWAREENTWICKLUNG

https://heise.de/-4134288

Security mit KI: Vorausscha
32-Zöller von EIZO: Erste Te
Wie intelligenter Speicher d
Die Bundeswehr wird digita
c't <webdev > - DIE Fronten
Studie: Was macht CISOs er