

# Low-Code-Canvas

---

Gestaltungsaspekte von Low-Code-Plattformen

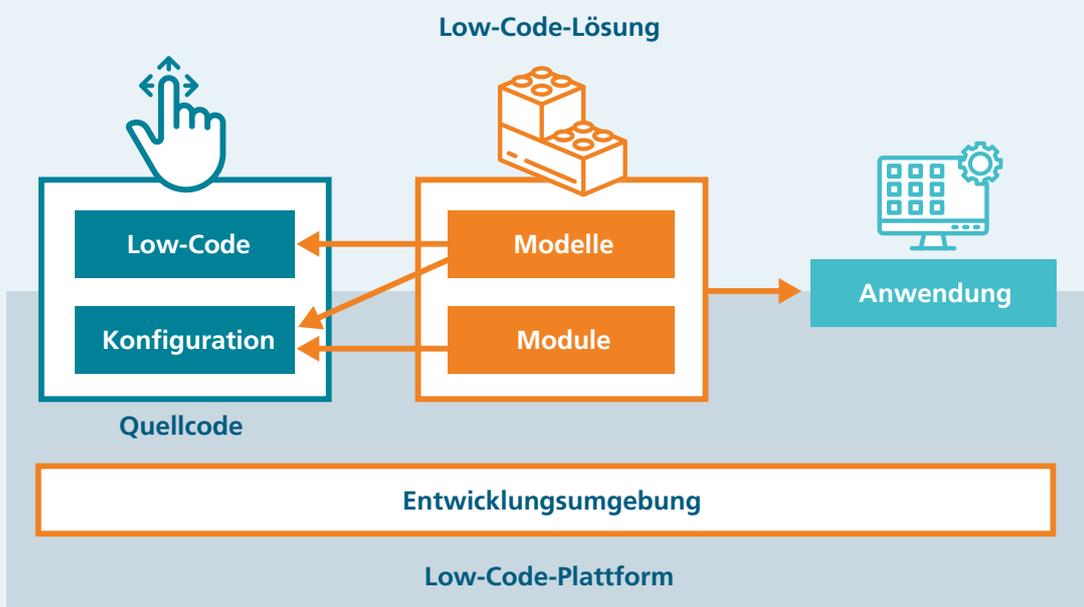
# Gestaltung von Low-Code-Plattformen

Bei der Planung des Einsatzes von Low-Code-Plattformen, sind viele technische und nicht-technische Aspekte zu berücksichtigen. Das Low-Code-Canvas ist ein Vorschlag für eine systematische Betrachtung, um daraus individuelle Anforderungen an einsatzbereite Softwaresysteme abzuleiten.

Software ohne klassische Programmierkenntnisse zu erstellen, erfreut sich steigender Beliebtheit. Was steckt eigentlich dahinter? Low Code kann als weitere Stufe der Digitalisierung durch IT gesehen werden, die vor allem bei der Erstellung und Pflege von Geschäftsanwendungen ansetzt. Das Low-Code Manifest der Low-Code Association [<https://www.lowcodeassociation.org/manifest/>] beschreibt unter anderem folgendes Merkmal:

**»Low-Code ermöglicht es Unternehmen und Organisationen, für viele ihrer Geschäftsprozesse maßgeschneiderte und damit besser passende Softwarelösungen von hoher Akzeptanz einzusetzen und das kostengünstig und schnell.«**

*Allgemeines Konzept für eine Low-Code-Lösung*





Anforderungsbereiche an eine Low-Code-Plattform

Primär geht es darum, typische Anwendungen von Fachabteilungen zur Abwicklung von Geschäftsprozessen oder Verwaltungsvorgängen einfach und schnell anpassen und auch möglichst erweitern zu können. Dazu können in Anwendungen bei einer Low-Code-Lösung wiederverwendbare Module durch die Anwender (Fachabteilung, ...) leicht eingebunden und fachlich selbst konfiguriert werden. Ferner gibt es Low-Code-Modelle, bspw. Entscheidungen, die Anwender:innen selbst erstellen und konfigurieren können. Diese werden dann innerhalb der Low-Code-Plattform automatisch in angepasste Module umgesetzt. Die Modellierung, Konfigurierung und Bereitstellung der Anwendung erfolgt durch eine spezielle Entwicklungsumgebung.

### Allgemeines Konzept für eine Low-Code-Lösung

Eine Low-Code-Lösung muss nicht immer auf einer umfassenden Low-Code-Plattform basieren. Das Konzept von Low Code kann sich auch nur auf Teile von Anwendungen beschränken, beispielsweise zur einfachen Anpassung von Berechnungen oder Texten innerhalb eines Gesamtprozesses. Damit kann Low Code die vielfach üblichen »Office«-Formulare auf Basis von Tabellenkalkulationsvorlagen oder Textverarbeitungsmakros – die sogenannte »Schatten-IT« – ersetzen und besser in die IT einer Organisation einbinden. Im Nachfolgenden verwenden wir zur Vereinfachung auch den Begriff »Low-Code-Plattform« für jede Art von Low-Code-Lösung.

### Anforderungsbereiche an eine Low-Code-Plattform

Bei der Auswahl und Umsetzung einer Low-Code-Lösung müssen – wie bei jeder Software-Lösung – vorab auch die Anforderungen umfassend betrachtet werden. Diese lassen sich in Anforderung an die Technik, aber auch an die eigene Organisation der Anwender:innen selbst gliedern. Ferner unterteilen sich diese bei einer Low-Code-Lösung deutlicher und stärker in Anforderungen an die Plattform, die Anwendung und die Mitarbeitenden. Um die verschiedenen Aspekte einer Low-Code-Plattform besser verstehen und einordnen zu können, sind im nachfolgend abgebildeten Low-Code-Canvas wesentliche Themen strukturiert aufgeführt, die man insbesondere bei der Auswahl und Beurteilung einer Low-Code-Plattform im Kontext betrachten sollte. Rechtlich relevante Themen sind zusätzlich mit  markiert.

Für den IT-Betrieb relevante Themen sind mit  gekennzeichnet.

Für das Personal des Anwenders sind die relevanten Themen mit  markiert, wobei bei Bedarf auch die jeweiligen Themen der Modellierung von spezifischem Interesse sind.

## Themengruppen

Das Low-Code-Canvas ist in inhaltliche Themengruppen aufgeteilt:

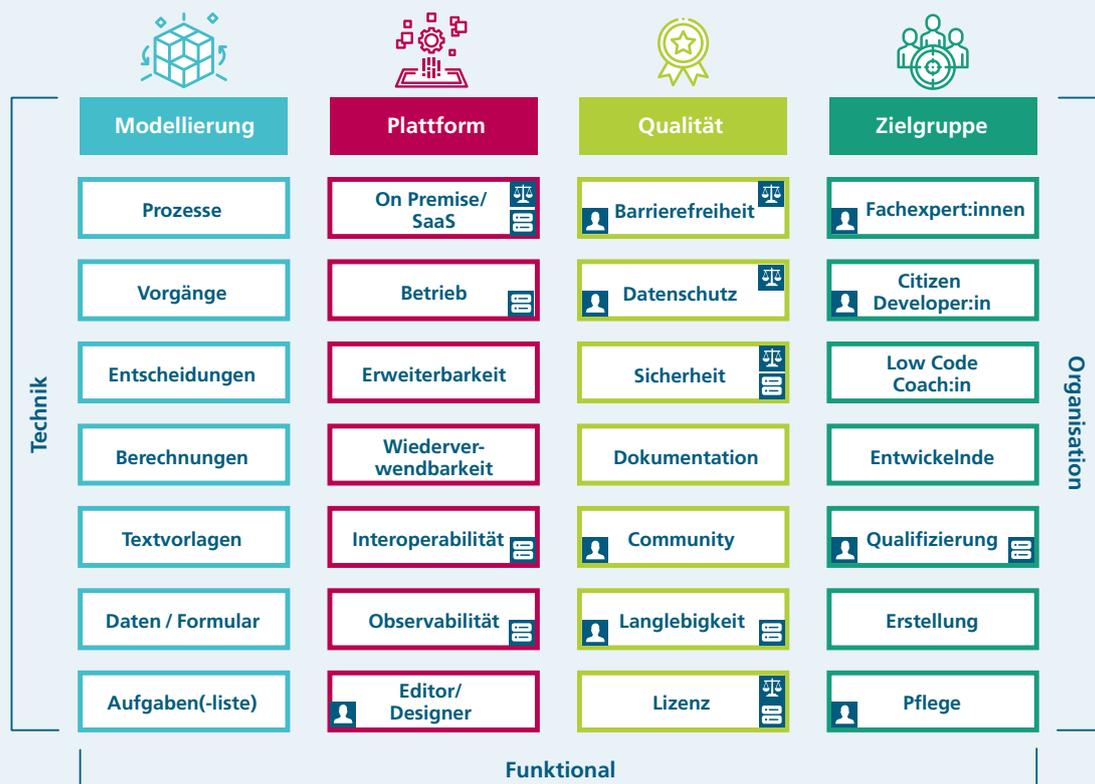
- »Modellierung« umfasst die typischen fachlichen Aspekte, die Anwender:innen gerne anpassen und erweitern wollen. Je nach eigenem Anwendungsgebiet kann diese Liste an Modellen erweitert werden, z. B. um Geo-Modelle u. v. m.
- »Plattform« beinhaltet den technischen Kern, auf dem eine Low-Code-Lösung basiert.
- »Qualität« umfasst technische, wirtschaftliche und rechtliche Rahmenbedingungen, die implizit vorausgesetzt werden.
- »Zielgruppe« umfasst, welche Rollen bei einem Low-Code-Projekt ausgefüllt werden sollten, bzw. wie man die notwendigen personellen Voraussetzungen schaffen kann. Ferner ist zu klären, in welchen Projektphasen Low Code genutzt werden soll.

»Modellierung« und »Plattform« sind eher die Sicht des Anwenders auf die technischen Eigenschaften einer Low-Code-Plattform. »Zielgruppe« betrachtet die organisatorischen Aspekte und personellen Folgen, die mit der Einführung einer Low-Code-Plattform einhergehen. »Plattform« und »Qualität« definieren wichtige funktionale und qualitative Aspekte, die man fordern sollte. Nachfolgend werden alle Themen des Low-Code-Canvas kurz erläutert.

**Prozesse:** Die internen Abläufe der digitalisierten Vorgänge sollten explizit in Form von Prozessen visualisiert (Transparenz) und auch angepasst (Varianten) werden können. Dafür sollten gängige Modellierungsstandards wie BPMN (Business Process Model and Notation) zur Verfügung stehen.

**Vorgänge:** Die manuell zu unterstützenden Vorgänge (Workflows), die durch die Nutzer:innen/Nutzenden gesteuert werden und nicht automatisiert sind, sollten ebenfalls explizit adäquat visualisiert (Transparenz) und bei Bedarf durch die Organisationseinheiten der Nutzenden angepasst werden können. Dafür sollten Modellierungsstandards wie CMMN (Case Management Model and Notation) evtl. auch BPMN zur Verfügung stehen.

Vorschlag eines Low-Code-Canvas



**Entscheidungen:** Ein wesentlicher Kern von Anwendungen, der häufig Änderungen unterworfen ist, sind Entscheidungen. Eine Entscheidung kann einfache Bedingungen und Verknüpfungen enthalten, aber auch eine Kombination vieler Teilentscheidungen mit unterschiedlichen Kombinationsstrategien. Darüber können Entscheidungen strukturiert und schrittweise kaskadiert/berechnet werden. Dafür sollten gängige Modellierungsstandards wie DMN (Decision Model and Notation) zur Verfügung stehen.

**Berechnungen:** Ein weiterer Kern von Anwendungen, der sich häufig ändern kann, sind Berechnungen. Diese können aus einfachen Formeln, aber auch komplexeren mathematischen Operatoren (Schleifen, Bedingungen, Filtern, Projektionen) bis hin zu Funktionen bestehen. Dafür sollten gängige und standardisierte Sprachen wie FEEL (Friendly Enough Expression Language; als Teil von DMN) zur Verfügung stehen.

**Textvorlagen:** Für die Kommunikation mit externen Nutzer:innen oder Organisationen sowie internen Notizen werden vordefinierte Textvorlagen benötigt, aus denen automatisch die Zieltexte generiert werden. Dies beschleunigt einerseits die Kommunikation, und andererseits werden dadurch rechtliche Vorgaben sichergestellt. Die Textvorlagen benötigen Platzhalter, um konkrete Daten/Information bei der Generierung einfügen zu können. Ferner sollten Textblöcke bei Bedarf oder mehrfach mittels Kontrollstrukturen eingefügt werden können. Die Textvorlagen können bspw. mit erweiterten web-basierten Texteditoren erstellt werden, die Markups für Platzhalter und Kontrollstrukturen nutzerfreundlich einfügen und Änderungen möglich machen.

**Daten/ Formular:** Für die Eingaben bei Formularen, Daten-Transformationen, Datenspeicherung etc. müssen Datenstrukturen definiert werden, inklusive der Validierung von Daten. Eine Nutzung von Schemata (wie JSON-Schema) und deren Spezifikationsumfang sollte die Basis sein. Die Pflege kann z. B. über grafische Formular-Editoren oder Baumstrukturen erfolgen.

**Aufgaben(-liste):** Manuelle Aufgaben in Vorgängen und Prozessen werden später den Nutzenden der Anwendung typischerweise als Aufgabenlisten präsentiert. Ob diese Aufgabenlisten in einer speziellen Komponente in der Anwendung integriert sind oder durch externe, spezialisierte Anwendungen (bspw. Trouble-Ticket-Systeme, E-Akten u.v.m.) realisiert werden, sollte konfigurierbar sein. Relevant ist, wie die Aufgaben modelliert und gegebenenfalls an bestehende Systeme angebunden werden.

**On Premise/ SaaS:** Man kann eine Low-Code-Plattform auf den IT-Ressourcen der eigenen Organisation betreiben oder als Dienstleistung beim Hersteller – in der Regel in einer Public Cloud. Die erste Variante garantiert größere digitale Souveränität bzgl. technischer Unabhängigkeit und/oder Datenschutz. Die zweite Variante erfordert wenig technischen Aufwand in der Organisation, bedingt aber in technischer, rechtlicher, organisatorischer und wirtschaftlicher Sicht absolutes Vertrauen in den Anbieter. Außerdem muss man die Rahmenbedingungen des Anbieters bei Aktualisierungen der Software einhalten.

**Betrieb:** Der Betrieb muss vor allem die organisationsweite »Governance« umsetzen, bspw. Rollenkonzepte, Sicherheit, u.v.m. Für den technischen IT-Betrieb ist der Aufwand unter Berücksichtigung der Service Level Agreements ein wichtiges Entscheidungskriterium: Wieviel Wissen und Personal muss für den Betrieb (Installation, Pflege und Problembeseitigung) vorgehalten werden? Ergänzend ist zu klären, wie gut die Unterstützung durch die Community ist bzw. ob/wieviel Support durch den Hersteller/Anbieter benötigt wird. »Managed Services« bieten die Variante, dass der Betrieb auf den IT-Ressourcen der Organisation vollständig durch den Anbieter übernommen wird.

**Erweiterbarkeit:** Jede IT-Lösung muss fachlichen, gesetzlichen und technischen Änderungen angepasst werden. Insbesondere kann es notwendig sein, die Low-Code-Plattform, um zusätzliche Funktionalitäten oder Module zu erweitern. Wenn dies durch den Anwender selbst oder Dritte erfolgen soll, werden in der Plattform offene, standardisierte Schnittstellen für Module benötigt, die eigenständig genutzt werden können. Grundsätzlich sollte der Anwender die Möglichkeit haben, die Low-Code-Modelle jederzeit selbstständig ändern zu können.

**Wiederverwendbarkeit:** Einerseits sollten Low-Code-Modelle, Bausteine, Module und Lösungen wiederverwendet werden können, bspw. durch Konfiguration. Insbesondere sollten ähnliche Anwendungen die Modelle und Module komplett übernehmen und entsprechend anpassen/konfigurieren können. Dies bedingt auch, dass Basis-Modelle bzw. Bausteine oder Module für das anvisierte Anwendungsgebiet bereits standardmäßig verfügbar sind. Andererseits sollten Modelle und Module anderer Hersteller ohne große Änderungen auf Low-Code-Plattformen übernommen werden können. Die Anbindung an einen »Marktplatz« kann helfen, auf einfache Weise Anwendungen wiederzuverwenden und weiterzuentwickeln.

**Interoperabilität:** Da ein Hersteller nicht für alle Anwendungsgebiete eine Low-Code-Plattform anbieten kann, sollte diese möglichst offen gestaltet sein und die Nutzung von externen (Funktions-)Modulen sowie idealerweise Low-Code-Modellen anderer Hersteller erlauben. Schnittstellen und



## Wir sind überzeugt: Low-Code revolutioniert die Konzeption, Entwicklung und langfristige Pflege.«

Karsten Noack, Vorsitzender, Markus Bernhart, Stv. Vorsitzender  
Low-Code Association e. V.

Modelle sollten daher standardisiert und interoperabel sein: Interoperabilität ermöglicht die Entstehung eines Low-Code-Ökosystems und -Marktes. Für die Anwender:innen senkt dies die Herstellerabhängigkeit und das Risiko beim Einsatz einer spezifischen Low-Code-Plattform.

**Observabilität:** Vorgänge, wie die Bearbeitung eines Falls, bestehen aus vielen manuellen und vor allem auch automatisierten Arbeitsschritten. Um die (erfolgreiche) Abarbeitung eines Vorgangs beobachten zu können, müssen die Arbeitsschritte protokolliert und visualisiert werden können. Dies ist wichtig, um Störungen zu erkennen, ob aufgrund unerledigter Aufgaben eines Bearbeitenden oder in der automatisierten Verarbeitung. Darüber hinaus ist bedeutend, dass die technische Abarbeitung durch die internen Services protokolliert und spezifische Kennzahlen erhoben werden. Diese Informationen müssen für den »Fachlichen Support« angemessen aufbereitet und visualisiert werden, um frühzeitig Probleme und Engpässe in der Anwendung erkennen zu können.

**Editor/Designer:** Das Aushängeschild einer Low-Code-Plattform ist eine möglichst einfach zu bedienende Entwicklungsumgebung, um die Anwendung zu erstellen und zu pflegen. Die »Designer« umfassen vor allem die Modell-Editoren, um die verschiedenen Low-Code-Modelle zu ändern bzw. zu konfigurieren. Sinnvollerweise sollten die »Designer« die Modell-Editoren nur in der Struktur bereitstellen, die dem Anwendungsfall – und damit den zugrunde liegenden Prozess – entspricht.

**Barrierefreiheit:** Die Benutzungsoberflächen der Low-Code-Plattform selbst und vor allem auch der entstehenden Anwendungen müssen barrierefrei sein und mindestens die gesetzlichen Standards BITV 2.0 und WCAG 2.1 umsetzen. Generell sollte auch die Benutzerfreundlichkeit geprüft und gewährleistet sein.

**Datenschutz:** Die Low-Code-Plattform muss alle Aspekte des Datenschutzes berücksichtigen und entsprechend auch in den entstehenden Anwendungen umsetzen. Dies umfasst sowohl den Schutz personenbezogener Daten selbst als auch die Protokolle über die konkrete Bearbeitung/Nutzung personenbezogener Daten. Damit verbunden ist die Auskunftspflicht über gespeicherte Daten einer Person und die gesetzlich vorgeschriebene Löschung der Daten. Generell sollte die Low-Code-Plattform das Prinzip »Privacy-by-design« unterstützen und umsetzen.

**Sicherheit:** Generell muss eine Low-Code-Plattform das Prinzip »Security-by-design« konsequent und vollständig umsetzen, um die Schutzziele Integrität, Vertraulichkeit und Verfügbarkeit von Daten und Anwendung zu gewährleisten. Die relevanten Vorgaben des BSI-Grundschatzes müssen konsequent unterstützt und umgesetzt werden. Für Authentisierung und Autorisierung sollten offene Standards und/oder gängige Lösungen eingesetzt werden, Beispiele dafür sind »OAuth« und »Keycloak«. Für die zentrale Pflege von Zugriffsregeln ist ein beispielhafter Standard »Open Policy Agent«.

**Dokumentation:** Bei einer Low-Code-Plattform muss es nicht nur eine gute Benutzeranleitung für die Nutzenden der Anwendung geben, sondern auch eine ausführliche und vollständige Entwicklungsdokumentation. Diese sollte umfassend und verständlich beschreiben, wie die mit der Low-Code-Plattform entstandene Anwendung angepasst und erweitert werden kann. Die klassische Dokumentation sollte zudem durch interaktive Online-Lernressourcen ergänzt werden.

**Community:** Low Code bedingt zusätzliches Wissen für Entwickelnde und Organisationen. Daher ist in der eigenen Organisation von strategischem Interesse, dass das Low-Code-Wissen möglichst breit gestreut und auf möglichst viele Mitarbeitende verteilt wird. Der interne wie externe Austausch sollte forciert werden. Daher ist auch die Größe der externen Community für eine Low-Code-Plattform relevant.

**Langlebigkeit:** Generell ist man bei jedem Software-System von einem Hersteller abhängig. Wenn aber eine größere Anzahl von Anwendungen einer Organisation auf einer Low-Code-Plattform basieren, ist die langfristige Verfügbarkeit entscheidend. Daher sollten der Anpassungsaufwand wie Offenheit, Nutzung von Standards, Nutzung von Open Source, Größe der Community, Langzeit-Support und Aufwand bei einer Weiterentwicklung der Low-Code-Plattform sowie der Portierung auf eine andere Low-Code-Plattform betrachtet und bewertet werden. Größere Änderungsaufwände oder eine Herstellerabhängigkeit (Vendor Lock-in) sind weitgehend zu vermeiden.

**Lizenz:** Generell ist eine möglichst freie Nutzung der Low-Code-Plattform innerhalb der Organisation und bei der Zusammenarbeit mit deren Dienstleistern wichtig. Wenn man einen zentralen IT-Dienstleister hat, können auch einige Open-Source-Lizenzen für einen Einsatz kritisch sein. Ferner ist die Frage zu klären, wer die Rechte an den Low-Code-Modellen besitzt – und ob dieser in Form von Quellcode überhaupt exportierbar ist. Fraglich ist auch, wer die (exklusiven) Nutzungsrechte an den (beauftragten) Erweiterungen (Modulen) der Low-Code-Plattform hat.

Fachexpert:innen kennen die Fachlichkeit und fachlichen Anforderungen am besten und sollten die fachlich wichtigsten Low-Code-Modelle bzw. No-Code-Modelle innerhalb bestehender Anwendungen möglichst selbst pflegen können: typischerweise Workflows, Entscheidungen, Berechnungen, Texttemplates sowie bei Bedarf auch Datenmodelle.

Citizen Developer:in sind meist technisch affine Fachexpert:innen und vor allem daran interessiert, ihre Anwendungen selbst zu pflegen und teilweise selbst erweitern zu können. Sie kommen auch mit komplexeren, kniffligen Problemen und Techniken zurecht. Ihr technisches Wissen eignen sie sich selbst oder durch gezielte Fortbildungen an.

Low Code Coach:in ist Mediator:in zwischen Fachexpert:innen/Citizen Developer:innen und dem Anbieter bzw. den Entwickelnden. Mit fundiertem IT-Wissen als Architekt:in und Spezialist:in für Low-Code-Modelle unterstützen Low Code Coaches die Fachexpert:innen/Citizen Developer:innen aktiv bei der (Weiter-)Entwicklung und beratend bei der Pflege. Sie können der Fachabteilung, der IT-Abteilung, dem IT-Dienstleister oder dem Anbieter angehören.

**Entwickelnde:** »Deep Coder:innen« entwickeln und pflegen bei Low Code vor allem die Low-Code-Plattform bzw. Low-Code-Komponenten für eine Anwendung selbst. Bei Bedarf entwickeln sie auch Erweiterungen, bspw. in Form von zusätzlichen, wiederverwendbaren Modulen oder neuen Typen von Low-Code-Modellen.

**Qualifizierung:** Wenn Fachexpert:innen ihre Anwendungen/Fachsysteme selbst pflegen sollen, benötigen sie das notwendige technische Wissen. Durch gezielte Fortbildungen und/oder durch Wissensaustausch mit der internen, organisationsübergreifenden oder auch externen Community kann dieses erlangt werden.. Damit ändert sich ggf. auch ihr Stellenprofil. Weiterqualifizierung ist auch wichtig, um Zukunftsängste bei der Einführung von Low Code zu begegnen.

**Erstellung:** Dass eine Anwendung »kinderleicht« von Fachexpert:innen und Citizen Developer:innen selbstständig erstellt werden kann, ist in der Praxis eher seltener der Fall. Die Erstellung und Weiterentwicklung sollten aber gemeinsam von diesen mit »aktiver« Unterstützung einer bzw. eines Low Code Coach:in möglich sein. Diese unterstützen vor allem bei der »Architektur« und der Grundstruktur der Low-Code-Lösung. Außerdem leisten sie den notwendigen Wissenstransfer an die Fachexpert:innen und Citizen-Developer:innen.

**Pflege:** Ein wesentliches Ziel von Low Code ist die selbstständige Pflege der Low-Code-Modelle durch die internen Fachexpert:innen und Citizen Developer:innen. Diese müssen nicht aus dem spezifischen Fachbereich kommen, sondern können auch organisationsweit intern unterstützen.

Das Low-Code-Canvas soll eine erste Unterstützung bei der Auswahl einer Low-Code-Plattform bieten. Auf dieser Basis können Entscheider:innen Fragen an Hersteller/Anbieter auswählen, Herstellervergleiche durchführen und konkrete Anforderungen an eine Low-Code-Plattform formulieren.

## Kontakt

---

Dipl.-Inf. Nadja Menz  
Gruppenleiterin Fachprozesse und sichere Infrastrukturen  
Geschäftsbereich DPS  
Tel. +49 30 3463-7320  
[nadja.menz@fokus.fraunhofer.de](mailto:nadja.menz@fokus.fraunhofer.de)

Fraunhofer FOKUS  
Kaiserin-Augusta-Allee 31  
10589 Berlin

[www.fokus.fraunhofer.de](http://www.fokus.fraunhofer.de)

Wir  
vernetzen  
alles