April 2021 Deutschland € 18,90
www.TecChannel.de

IDG

TECCHANNEL COMPACT

IT IM MITTELSTAND VON IDG

RATGEBER ■ TIPPS ■ PRAXIS

Software-Entwicklung

Basics

- Neue Rollen und Gehälter für Softwareentwickler
- Das steckt hinter API, IaC, UX, NLP und No Code
- Was JavaScript von TypeScript unterscheidet
- Das müssen Sie über DevOps wissen

Prototyping

- Wie Prototyping bei der Softwareentwicklung hilft
- Design Thinking: Vom Problem zur Idee
- UX-Technologien, die Sie im Blick haben sollten
- Application Security: Software sicher gestalten

Coding

- Einstieg: Python lernen leicht gemacht
- Diese Sprachen eignen sich für KI-Projekte
- Collaborative Coding: Programmieren im Team
- Wie KI die Software-entwicklung vereinfacht

IT-Projekte richtig managen

Drei Vorgehensmodelle für Low-Code-Projekte

Low-Code- und andere Rapid-Development-Technologien ermöglichen eine direkte Zusammenarbeit von Softwareentwicklern und Anwendern. In der Praxis bieten sich drei unterschiedliche Vorgehensmodelle für Low-Code-Projekte an.

Bevor ein IT-Projekt losgeht, sollte man sich ein paar grundlegende Fragen stellen: Ist das in den 90er Jahren entwickelte Scrum-Vorgehensmodell noch das richtige und agil genug für die kommende neue Generation von Softwareentwicklungsmethoden? Sollte man Agilität neu denken, und wenn ja, dann wie? Vor dem Hintergrund eines entschlackten und wiederbelebten Wasserfallmodells vielleicht? Oder sollte es bei künftigen Projekten in eine ganze andere Richtung gehen?

Agile Softwareentwicklung nach Scrum

Die Vorteile von Scrum sind hinlänglich bekannt. Der Zeitraum von der Definition fachlicher Vorgaben bis zu deren Produktivsetzung wird von mehreren Jahren auf einige Monate verkürzt. Das allein ist ein großer Gewinn, der viel mehr Freiheitsgrade in der IT-Unterstützung der Fachbereiche eines Unternehmens ermöglicht. Wenn man genau hinschaut, dann ist die agile Softwareentwicklung nach Scrum aber nur eine vom Product Owner gesteuerte Aufeinanderfolge kleiner Wasserfallmodell-Projekte, in denen man sich von Ausbaustufe zu Ausbaustufe einer Software bewegt. Eigentlich ist es mehr ein agiles Projektmanagement als agile Entwicklung im engeren Sinne.

Das Modell ist optimal zugeschnitten auf die selbstständige Entwicklung der eigenen Kernsoftware, beispielsweise einer Online-Plattform. Es ist ideal, um so schnell wie möglich mit einer zunächst überschaubaren Minimallösung produktiv zu gehen, und diese dann mit dem verfügbaren Personal schrittweise im Monatsrhythmus weiter auszubauen. Sehr gut geeignet ist Scrum auch für die laufende Weiterentwicklung fertiger Software im Rahmen eines konstanten Softwarepflegebudgets. Im Backlog werden die Anpassungswünsche gesammelt und dann, nach Priorität und Machbarkeit priorisiert, in die laufende Weiterentwicklung eingetaktet und jeweils in eine der nächsten Versionen der Software eingebaut.

Kombiniert mit einer modernen DevOps-Organisation (mehr dazu ab **Seite 12**) kann man so seine IT merklich auf Trab bringen. Und es entspricht sehr gut dem laufenden Erkenntnisgewinn aus dem Betrieb der vorherigen Versionen. Zumindest wenn die Kosten keine Rolle spielen, kann dies aus Managementsicht der richtige Weg sein, und aus Sicht eines Personentage verkaufenden IT-Dienstleisters allemal. Nach Aufwand bezahlte Scrum-Projekte sind insbesondere für den Auftragnehmer von Vorteil, da risikolos und gut planbar.

Das verbesserte Wasserfallmodell

Geht es um die Neuentwicklung einer komplexen Software, womöglich als Festpreisprojekt in einem Auftraggeber-Auftragnehmer-Verhältnis (mehr dazu ab **Seite 103**), liegen die Dinge anders. Niemand braucht jeden Monat eine funktionsfähige, aber allzu rudimentäre Programmversion, die ohnehin noch nicht einsetzbar ist. Der Monatsturnus steht zudem der Abarbeitung umfangreicherer Arbeitspakete im Weg.

Da nicht eindeutig zugeordnet werden kann, ob der Product Owner ein Vertreter der Fachabteilung, der IT-Abteilung, des Auftraggebers oder des Auftragnehmers sein soll, denn beides passt nicht optimal in die reale Welt, sind die Erfahrungen mit extern vergebenen agilen Entwicklungsvorhaben sehr durchwachsen. Wenn man genau weiß, was man will, und eine optimale und termingerechte Umsetzung wünscht, dann wäre das Wasserfallmodell eigentlich ideal – wären da nicht die altbekannten Probleme eines überbordenden Formalismus und unzureichender Aktualität, und vor allem das Stille-Post-Phänomen, welches die Nutzeranforderungen auf dem langen Papierweg teilweise grotesk verzerrt, so dass manchmal regelrecht am Bedarf vorbei entwickelt wird.

Die Frage ist, ob und wie man den vernünftigen Grundansatz „Erst denken, dann handeln" in die agile Welt der Low-Code-Softwareentwicklung übertragen kann. Iteratives Prototyping allein, wie es beispielsweise im V-Modell-XT der Öffentlichen Verwaltung festgeschrieben ist, genügt hierfür definitiv nicht. Entscheidend für ein verbessertes Wasserfallmodell ist vielmehr, die Nutzeranforderungen viel direkter und schneller an die Entwickler heranzutragen, idealerweise im direkten Dialog. Dies aber wiederspräche der Grundstruktur klassischer Projektentwicklung mit vorgeschalteten Konzeptionsphasen – lange bevor die Programmierer ins Spiel kommen. Ineffizient ist dies allemal, da der Aufwand, die Anforderungen detailliert genug aufzuschreiben, oftmals weit höher ist als der, sie mit einer hochentwickelten Low-Code-Plattform umzusetzen.

Für normale Projekte, bei denen sich die konkreten Anforderungen immer erst dann herauskristallisieren, wenn man schon was auf dem Bildschirm sieht, ist der vorgelagerte Aufwand eher ein Hemmnis als ein Vorteil, und erhöht zudem unnötig die Projektkosten. Also dann doch lieber agil – aber wie genau?

Design Thinking als Grundprinzip

Wenn es heute möglich ist, Computerprogramme in Echtzeit vor den Augen der Anwender, oder gar mit diesen gemeinsam live am Bildschirm umzubauen, dann ist der Scrum-typische Monatsrhythmus der agilen Softwareentwicklung eindeutig zu langsam. Eine direkte Interaktion zwischen Anwendern und Entwicklern, etwa um gemeinsam optimale (das heißt für die Anwender sinnvolle und zugleich technisch gut umsetzbare) Lösungswege zu finden, ist nicht vorgesehen. In diesem Sinne ist Scrum eindeutig nicht agil genug. Außerdem stellt sich in der Praxis immer wieder die Frage, ob es diesen einen allwissenden Product Owner überhaupt gibt. Wissen nicht die einzelnen Fachanwender selbst am besten, was sie brauchen? Die Projektleiter sollten sich daher nicht so sehr als Owner des entstehenden Produktes sehen, sondern mehr als Moderatoren zwischen Anwendern und Entwicklern, um unter den gegebenen Rahmenbedingungen, wie etwa begrenztes Budget oder restriktive Zeitvorgaben, dennoch zu einem optimalen Ergebnis zu kommen.

Das wesentlich bessere Medium als ein Backlog sind regelmäßige Design-Thinking-Workshops, und dies nicht nur in der Anfangsphase eines Projekts, sondern über den gesamten Projektzeitraum hinweg. Design Thinking steht für interdisziplinäre Zusammenarbeit in gemischten Teams, für Offenheit und Kreativität, und zugleich für deutlich weniger Tool-Gläubigkeit. Die Tools des Design Thinking sind keine Computerprogramme, sondern Papier und Bleistift, Whiteboards und Stehtische, mit und auf denen man gemeinsam Bildschirmentwürfe malen oder Abläufe skizzieren kann. Und die Offenheit, immer wieder alles zu hinterfragen, frei nach dem Prinzip „Ist das wirklich das, was Sie brauchen?". Als Diskussionsgrundlage hat man hier stets die laufenden Arbeitsstände auf dem Bildschirm: keine Sprints, keine fertigen Module, sondern laufende Arbeitsstände – unfertig und ungetestet, aber realitätsnah in der Handhabung und mit mehr oder weniger echten Daten.

Das Projekt immer horizontal, und nicht vertikal zuschneiden

Auf der anderen Seite ist Scrum aber auch zu agil, weil es der Versuchung Tür und Tor öffnet, die Dinge in der falschen Reihenfolge zu tun. Für typische Datenbankanwendungen ist es sinnvoll, in einer frühen Projektphase zumindest die Grundstrukturen des Datenmodells verbindlich und endgültig festzulegen, um ein kostenintensives Ändern der bereits fertigen Programmteile zu vermeiden, wenn neue Use Cases Einfluss auf das Datenmodell haben. Zudem kann man so gleich zu Beginn Altdaten anonymisiert übernehmen oder realitätsnahe Testdaten generieren, und auf diese Weise wirklichkeitsnäher und unter Last entwickeln.

Zudem legt das Low-Code-Prinzip mit seinem enormen Implementierungstempo und seinem 98-Prozent-Ansatz nahe, bereits zu Projektbeginn die Dinge zu identifizieren und zu initiieren, die vielleicht doch programmiert werden müssen, und die sonst als zu spät gestartete „langlaufende Tasks" die Endtermine gefährden könnten.

Schließlich lässt sich noch vor der Erarbeitung der konkreten Inhalte und Bedienabläufe der komplette Programmrahmen mit Benutzerverwaltung, Layout und UUX-Richtlinien im Vorfeld entwickeln, bevor man sich mit den Anwendern in die Einzelheiten der Programme vertieft. Auf diese Weise werden die später zu besprechenden Arbeitsstände plastischer und realitätsnäher, was die besonders wichtige kreative Zusammenarbeit mit den späteren Anwendern erheblich vereinfacht. Außerdem kann man dann die etwas umfangreichere Arbeit an den eigentlichen Inhalten noch etwas weiter nach hinten schieben, was die Planbarkeit verbessert und die Aktualität erhöht.

Das phasenagile Vorgehensmodell

Daraus ergibt sich ein optimales Vorgehensmodell für Low-Code-Entwicklungen: in geordneten Projektphasen von wohldefiniertem Umfang und Inhalt, und zugleich so agil wie möglich in den einzelnen Phasen. Wie diese Phasen genau aussehen, mag von den konkreten Rahmenbedingungen, vom Geschäftsfeld des Anbieters, und vielleicht auch von den konkreten Eigenschaften der jeweils eingesetzten Low-Code-Plattform abhängen. Für Datenbankanwendungen in einem kostenbewussten und termingebundenen Projektumfeld erweist sich ein 5-stufiges Phasenmodell als optimal. Die einzelnen Phasen kann man etwa wie folgt definieren:

- **Initialisierung**
- **Datenbasis**
- **Programmrahmen**
- **Fachmodule**
- **Finalisierung**

Für die fünf Phasen ist jeweils etwa der gleiche Zeitbedarf zu veranschlagen. Die Hauptarbeit aber liegt in Phase 4, in der unter Umständen mehrere Design-Thinking-Teams gleichzeitig aktiv sind. In den Anfangsphasen sind etwas mehr die Informatiker gefragt, während weiter hinten im Projekt verstärkt die **Citizen Developer** (mehr dazu ab **Seite 69**) aktiv werden, also Projektmitwirkende, die mindestens ebenso tief im Fachlichen stecken wie in der IT.

Der agile Festpreis

Das Phasenmodell mit ausgeprägtem Design Thinking ist zugleich auch der vermutlich beste Weg, um agile Festpreisprojekte zu ermöglichen. Um einem Ausufern der Benutzerideen entgegenzuwirken, setzt man darauf, dass die Fachanwender und die Entwickler geeignete und zugleich bezahlbare Lösungswege gemeinsam erarbeiten und budgetkonform aushandeln. Die Projektleiter beider Seiten wirken bei Bedarf kompromissfördernd und mäßigend auf ihre Kollegen ein, da sie am gemeinsamen Projekterfolg gemessen werden. Bei Bedarf werden zusätzlich Plus-Minus-Listen geführt, über die Mehr- und Minderaufwände verrechnet werden können.

Entscheidend für den Projekterfolg ist dabei die strikte Planung der Design-Thinking-Workshops, und dass die Projektleiter effektiv durchsetzen, anstehende Entscheidungen auch wirklich zu treffen.

Das „Immer-Dienstags-Prinzip“

Für Low-Code-Projekte hat sich für die Design-Thinking-Workshops ein Wochenturnus als optimal erwiesen, und zwar über die gesamte Projektlaufzeit hinweg, von der ersten bis zur letzten Woche. Das ist genau der Zeitraum, nach dem man bei der Arbeit mit Low-Code-Plattformen jeweils hinreichend viel Neues zur Diskussion stellen kann, sodass die Besprechungs- und Doing-Aufwände in einem sinnvollen Verhältnis stehen, und sich die Reisebelastung der Mitarbeiter externer Dienstleister in zumutbaren Grenzen hält. Zwar nehmen je nach Projektphase teils andere Personen an den Meetings teil, aber dennoch ist es sinnvoll, sofort zu Projektbeginn zwischen Auftragnehmer und Auftraggeber dafür einen festen Wochentag zu vereinbaren, wie beispielsweise jeden Dienstag. Das reduziert den Organisationsaufwand, und alle Beteiligten können besser planen.

Die instantane Entwicklungsmethodik

Wenn für den Auftraggeber nur das bestmögliche Ergebnis zählt – in möglichst kurzer Zeit, und vor allem dann, wenn die Aufgabenstellung zu Projektbeginn noch unklar ist, wodurch eine systematische Anforderungsanalyse weder im Vorfeld, in bestimmten Projektphasen, noch in der laufenden Fortschreibung eines Backlogs unmöglich ist –, kann es sinnvoll sein, die technisch mögliche Flexibilität von Low-Code-Produkten bis an die Grenze auszureizen. Dann wird fast durchgängig vor den Augen der Endanwender und gewissermaßen auf Zuruf entwickelt. Das bedeutet, dass der Entwicklungsprozess zu großen Teilen durch spontane Eingebungen der Anwender gesteu-

ert wird. Wenn die Technologie in der Lage ist, die Probleme abzufangen, die durch Try and Error zwangsläufig entstehen, dann liefert dieses Verfahren den unschätzbaren Vorteil, eine Anwendungssoftware schrittweise immer mehr an das anzupassen, was tatsächlich gebraucht wird. Hier macht es Sinn, zuerst über Datenstrukturen und Inhalte nachzudenken und danach über Prozesse und Programmoberflächen.

Nachteilig ist, dass Projekte bei instantaner Entwicklungsmethodik faktisch nicht kalkulierbar sind, und deshalb schwerlich mit fixem Budget oder gar als Festpreisprojekte aufgesetzt werden können. Wenn man instantan vorgehen möchte, dann muss man den finanziellen Spielraum haben. In der gesamten Kosten-Nutzen-Bilanz kann die instantane Vorgehensweise aber durchaus die wirtschaftlichste Methode sein, und sie kann bei guter Zusammenarbeit der Projektleiter beider Seiten für die Fachbereiche zu den besten erzielbaren Ergebnissen führen.

Wasserfall oder Scrum oder phasenagil?

Alle drei Vorgehensmodelle haben ihre jeweilige Berechtigung, und es kommt auf die konkreten Rahmenbedingungen an, welches am besten geeignet ist. Während das **klassische Wasserfallmodell** an sich gut und richtig, aber für Low-Code-Entwicklungen eigentlich zu langsam ist, sind das schon etwas in die Jahre gekommene Scrum-Konzept und das neuere phasenagile Vorgehensmodell zwei sinnvolle Alternativen – Ansätze zur agilen Softwareentwicklung, die unterschiedlicher nicht sein könnten.

Während **Scrum** für die Eigenentwicklung von Softwareprodukten und Onlineangeboten, sowie für die kontinuierliche Verbesserung vorhandener Software als besser geeignet erscheint, ist das **phasenagile Konzept** viel besser auf budget- und termingebundene Neuentwicklungen ausgerichtet und ermöglicht agile Festpreisprojekte. In bestimmten Konstellationen, insbesondere bei etwas kleineren Projekten oder abgekoppelten Modulen bzw. Microservices von überschaubarer Größe mit relativ offenem Budget, macht es durchaus auch Sinn, noch einen Schritt weiter zu gehen, und gänzlich instantan zu entwickeln – eine Vorgehensweise, die ohne moderne Low-Code-Technologien kaum möglich wäre.

Karsten Noack

Karsten Noack ist Gründer und CEO der Scopeland Technology GmbH. Als Visionär entwickelte er bereits Mitte der 90er Jahre die Grundlagen der Technologie, die heute als ‚Low-Code' und als Schlüsseltechnologie der Digitalisierung bekannt ist. Karsten Noack verfügt über Erfahrungen im Einsatz von Low-Code-Plattformen in großen Unternehmen und Behörden.