Bekannt und bewährt

Agile Softwareentwicklung anhand von Best Practices

Daniel Noz

Es werden immer wieder die gleichen Fehler in der Softwareentwicklung begangen. Die folgenden Erfolgsrezepte helfen, diese typischen Fehler zu vermeiden. Diese Best Practices sind keine Neuigkeit in der Softwareentwicklung, sie finden sich in den Modellen der agilen Entwicklung, beim eXtreme Programming oder in Scrum wieder.

Im Rahmen meiner IT-Laufbahn bin ich als Entwickler und Berater immer wieder an Großprojekten mit mehrjähriger Laufzeit in der Softwareentwicklung beteiligt. Das Ziel, dem Kunden eine zufriedenstellende Software abzuliefern, wird durch eine Vielzahl von Faktoren beeinflusst, wie zum Beispiel Budgetkürzungen in Krisenzeiten, Vorgaben an Technologie und Entwicklungswerkzeugen, die Wahl der Mitarbeiter, das Vorgehensmodell, die Entwicklung an getrennten Standorten und viele mehr. Diese Faktoren können den Projektverlauf zu jeder Zeit in eine bestimmte Richtung wenden oder sogar zum Scheitern bringen.

Es scheint unmöglich, das eine Rezept für erfolgreiche Softwareentwicklung herauszufinden. Darum geht es in diesem Artikel auch nicht. Ich möchte nicht über die Gründe schreiben, warum so viele Projekte scheitern, sondern die meiner Meinung nach wichtigsten Best Practices aufzeigen, mit deren Hilfe man die typischen Probleme in der Softwareentwicklung verhindern kann. Diese Best Practices sind besonders wichtig in Großprojekten, persönlich empfehle ich sie aber für jedes Softwareentwicklungsprojekt.

Erstellung einer Machbarkeitsstudie mit Lasttests

Anhand einer Machbarkeitsstudie, eines sogenannten Proof of Concept (PoC), wird mit einer kleinen Gruppe von Experten geprüft, ob die umzusetzenden Anforderungen prinzipiell durchführbar sind. Die Grundlage dafür bildet die Spezifikation des Kunden. Inwieweit diese ausgearbeitet ist, soll den PoC nicht beeinflussen. Wichtig ist, den Fokus nicht auf die fachlichen Anforderungen zu legen, sondern auf die technische Realisierbarkeit. Der PoC soll zeigen, ob die ausgewählte Technologie und die Werkzeuge, für die man sich entschieden hat, den technischen Anforderungen gewachsen sind.

Der Aufwand für einen PoC kann je nach Umfang der zu integrierenden Systeme unterschiedlich groß sein. Aus eigener Erfahrung rate ich aber, den Technologieentscheid noch einmal zu überdenken, sollte der PoC mehr als vier Wochen dauern.

Für die Wahl der Technologie gibt es eine Vielzahl von Kriterien, über die sich der entsprechende Projektverantwortliche bereits vor Beginn der Entwicklung Gedanken machen muss: der Anwendungstyp (Webanwendung, Rich-Internet-Anwendung, Fat Client, Mobile Client usw.), die Anzahl der Benutzer, Geschwindigkeitsvorgaben, parallele Zugriffe.

Der Technologieentscheid kann und sollte revidierbar sein, wenn das Ergebnis des PoCs nicht zufriedenstellend ist. Ein wichtiges Prüfmerkmal ist die Geschwindigkeit der Entwicklungszyklen: Man bedenke den Mehraufwand, der entsteht, wenn man ein Entwicklerteam mit einer Technologie arbeiten lässt, mit der die Anforderungen nur schwerfällig umsetzbar sind und als Beispiel der Deployment-Aufwand einen Anteil von mehr als dreißig Prozent eines Entwicklertages einnimmt.

Der Lasttest, welcher den PoC auf kritische Engpässe und technologische Machbarkeit prüft, ist ein wichtiger Bestandteil. Sollte schon der PoC mit einer bestimmten Last und konkurrierenden Zugriffen nicht zurechtkommen, wie soll es dann erst die Endanwendung können? Für Lasttests ist es im Übrigen nicht notwendig, die Zielinfrastruktur schon von Beginn an bereitzustellen. Anbieter von sogenannten „Clouds“ ermöglichen es, eine konfigurierbare Serverlandschaft für einen bestimmten Zeitraum anzumieten.

Aufsetzen der Testinfrastruktur vor Implementierungsbeginn

Das nächste wichtige Ziel ist, dem Entwickler die bestmögliche Infrastruktur für seine Arbeit bereitzustellen: Eine Infrastruktur, die dem Entwickler bei jeder Änderung, die er in ein *zentrales Repository* einfügt, mitteilt, ob die Anwendung noch gebaut werden kann sowie ob die entwickelten Tests noch lauffähig sind. Das Schlagwort lautet *Continuous Integration (CI)*. Stellt man diese Infrastruktur erst während der Entwicklung zur Verfügung, riskiert man nicht nur Abstriche an der Qualität, sondern läuft auch Gefahr, ein System zu entwickeln, das die Entwickler beherrscht und nicht anders herum.

Definierte Tests geben jederzeit eine Aussage, ob die Anwendung noch lauffähig ist bzw. welche Teile der Anwendung nicht lauffähig sind. Ist Letzteres der Fall, so muss definiert sein, was im Fehlerfall passiert. CI-Systeme können bei Abbruch des Builds oder der Tests verschiedene Fehlernachrichten verschicken.

Ebenso lassen sich statische Code-Auswertungstools in ein CI-System integrieren. Hierbei wird der Quelltext auf potenzielle Fehler oder Verletzungen von Programmierrichtlinien geprüft. Man sollte sich jedoch die Frage stellen, welche Auswertungen auch wirklich sinnvoll sind. Eine wichtige Auswertung ist die Codeabdeckung. Sie sagt aus, wie viel Prozent des geschriebenen produktiven Codes durch Testfälle abgedeckt ist. Je höher dieser Wert, desto geringer ist die Wahrscheinlichkeit, dass im Rahmen von Kundentests technische Fehler entstehen. Ob der Code fachlich korrekt ist, wird durch fachliche Testdaten geprüft, die vorab im Rahmen einer Testdatenabstimmung vom Fachbereich vorgegeben werden.

Eine Testinfrastruktur wie diese setzt voraus, dass die Entwickler im Rahmen ihrer Entwicklung auch Tests schreiben. Diese bedeuten zwar auf den ersten Blick mehr Aufwand, rechnen sich aber innerhalb der Entwicklung, denn *eine Entwicklung ohne Testfälle kann in einem Projekt mit mehreren Entwicklern nicht erfolgreich sein*. Diese Testinfrastruktur bietet jederzeit die Möglichkeit, die Anwendung auf einer für den Kunden erreichbaren Testumgebung bereitzustellen (deployen).

Lässt man den Kunden in iterativen Zyklen die Anwendung testen, sollte man sich über den Umgang mit Fehlern Gedanken machen. Idealerweise arbeitet der Entwickler mit einer Entwicklungsumgebung, in der ein sogenanntes „Bugtrackingtool“ bereits integriert ist, sodass der Entwickler jederzeit die Fehler sieht, die ihm zugeordnet wurden. Aus Erfahrung weiß ich, dass Entwickler versuchen, Fehlern aus dem Weg zu gehen.

Die Rolle des Architekten

In jedem größeren IT-Projekt gibt es die Rolle des Architekten. Seine Aufgabe ist es unter anderem, grundlegende übergreifende Themen zu definieren. Wie sieht der Umgang mit technischen Fehlern aus, welches Transaktionsverhalten hat die Anwendung, wie sieht die Strukturierung der Anwendungsartefakte aus und welche Testframeworks setzt man ein?

Ich habe Projekte erlebt, in denen es nur mit einer Vielzahl von Problemen möglich war, solche übergreifenden Entscheidungen zu ändern. Verbesserungsvorschläge von Entwicklern konnten nicht umgesetzt werden, weil das Risiko und der Änderungsaufwand zu groß gewesen wären. Eine Anwendung mit stabilem Continuous Integration ermöglicht größere übergreifende Änderungen an der bestehenden Anwendung. Sind zum Beispiel nach einer Umstellung auf eine neue Frameworkversion immer noch alle Tests lauffähig, kann man die Entwicklung bedenkenlos fortsetzen.

Häufig erlebe ich es, dass für die Umsetzung neuer fachlicher Anforderungen Systeme so generisch wie möglich konzipiert werden. Mit jedem Sonderfall soll die Anwendung umgehen können, auch wenn am Ende nur ein Prozent der Anwendungsfälle diese Funktionalität benötigt. Ich bevorzuge den „Keep it simple"-Ansatz: ein Problem nach dem anderen so einfach wie möglich zu lösen. Die Änderungshäufigkeit innerhalb einer Entwicklung ist aus Erfahrung zu groß, um auf Vorrat zukünftige Komponenten zu programmieren.

Auswahl der Projektbeteiligten und Aufteilung der Teams

Im Idealfall hat man nur Projektmitarbeiter, die ihre Arbeit aus Leidenschaft machen und ihr Fach verstehen. Da dies in der Regel aber eher die Ausnahme ist, halte ich es für sehr wichtig, klare Verantwortungsbereiche für in sich abgeschlossene Arbeitspakete zu vergeben. Werden die Aufgaben zu granular erstellt, kann dies zur Folge haben, dass die Verantwortung für das Ganze verloren geht. Teams zu je fünf Entwicklern, die ein in sich abgeschlossenes Thema bearbeiten, sind der Idealfall. Man fördert den Zusammenhalt und verhindert das Zustandekommen eines Wissensmonopols für bestimmte Teile der Anwendung.

Test und Implementierung

Implementierung bedeutet nicht nur Lines of Code. Der Fokus muss auf den Kriterien Stabilität des Build/Deployment, Tests sowie Testabdeckung liegen. Neue Funktionalität darf erst dann in das zentrale Repository eingefügt werden, wenn die genannten Kriterien erfüllt sind. Diese *Stabilität* zu erhalten, muss höchste Priorität haben, denn ein Fehler eines Entwicklers kann das gesamte Entwicklerteam und somit die Weiterentwicklung lahm legen. In diesem Fall muss der Fehler sofort behoben werden. Dieses Risiko kann durch die oben beschriebene Testinfrastruktur verhindert werden. Zusätzlich ist es notwendig, einen Prozess sowie einen Verantwortlichen bezüglich des Umgangs mit Fehlern zu definieren.

Vor der Implementierung muss geschätzt werden, was implementiert wird. Gerade zu Beginn einer Entwicklung ist es schwierig, eine Aussage zu treffen, wie lange man für ein bestimmtes Arbeitspaket benötigt. Tests, die Probleme vorheriger Arbeitspakete beschreiben, werden mit der Zeit helfen, die *Aufwandsschätzung* zu verbessern. Noch bevor man mit der Umsetzung beginnt, sollte man sich ausreichend Gedanken über die Aufgabenstellung machen. Dazu ist es sinnvoll, eng mit dem Fachexperten zusammenzuarbeiten.

Ein erster Schritt zum Aufbauen eines gemeinsamen Verständnisses ist das Erstellen von Testfällen, User Stories: Wie lauten die Eingabebedingungen und was soll das Programm, der Programmteil am Ende liefern? Diese Testdaten baut der Entwickler dann in einen Testfall ein und entwickelt so lange, bis der Code die Testerwartung erfüllt.

Schnittstellen mit anderen Komponenten sind auf die gleiche Weise abzustimmen. Eine gute Zusammenarbeit zwischen Auftraggeber und Auftragnehmer ist hierbei unabdingbar! Das Arbeitspaket ist erst dann abgeschlossen, wenn nicht nur die neu entwickelten Tests laufen, sondern auch der Rest der Anwendung (Build und bestehende Tests).

Ein großer Vorteil dieser testgetriebenen Vorgehensweise ist, dass jederzeit refaktoriert werden kann: Entdeckt ein Entwickler eine unsaubere Codestelle in fremdem Code, soll er diesen Code anpassen. Der geschriebene Code muss nicht direkt bei der ersten Implementierung den besten Stil haben. Die Tests geben dem Entwickler bei Änderungen immer die Aussage, ob noch alles korrekt und lauffähig ist. Sind jedoch nur noch 90 von 100 Tests lauffähig, ist es für Entwickler nur „halb so schlimm", wenn nur noch 85 laufen. Hält man die Tests hingegen immer lauffähig, gibt es keine Ausrede mehr. Nur so kann gewährleistet werden, dass die bestehende Funktionalität (der bestehende Code) im Laufe einer Weiterentwicklung noch korrekt funktioniert, die Qualität stimmt und der Spaß an der Entwicklung nicht verloren geht.

Zusammenfassung

Auch wenn man diese Best Practice-Prinzipien nicht von Anfang an befolgt, ist es möglich, sie während der Entwicklung einzustreuen. Dies ist jedoch nicht einfach, da sich häufig schon eine unsaubere Vorgehensweise in die Köpfe der Entwickler eingeschlichen hat.

Die in diesem Artikel beschriebenen Prinzipien sind keine Neuigkeit in der Softwareentwicklung, sie finden sich in den Modellen der agilen Entwicklung, XP oder Scrum wieder. Unabhängig davon, für welches Entwicklungsmodell man sich entscheidet, sind die genannten Best Practices die meiner Meinung nach wichtigsten Schritte für eine erfolgreiche Softwareentwicklung. Orientiert man sich an ihnen, so hat man die beste Basis für einen flexiblen und schlanken Entwicklungsprozess, zügigen Fortschritt sowie zufriedene Entwickler, die keine Angst vor kritischen Änderungen haben.

Daniel Noz arbeitet bei dem internationalen Dienstleister für Management- und IT-Beratung Acando als IT-Entwickler und -Berater im Bereich JavaEE, hauptsächlich in Großprojekten.
E-Mail: daniel.noz@acando.de.