Software-Entwicklung

Nachhaltige Software-Entwicklung: gebaut für den Wandel

Eine aufwendig entwickelte Software sollte möglichst lange nutzbar sein. Damit sie den jeweils aktuellen fachlichen und technologischen Anforderungen genügt, muss deshalb eine permanente Weiterentwicklung erfolgen. Dabei hat sich die Art, wie Software entwickelt wird, über die Jahre technisch und organisatorisch geändert. Wie und von wem eine Software erstellt wurde, hat eine größere Auswirkung auf die langfristige Qualität als die eingesetzten Technologien.

Die Software-Krise gibt es seit mehreren Jahrzehnten und sie betrifft Neuprojekte wie Altprodukte gleichermaßen. Schon bei der Entwicklung werden viele Kardinalfehler begangen, die eine spätere Modernisierung oder Anpassung deutlich komplizierter machen. Irgendwann ist dann der Zeitpunkt gekommen, sich von den Software-Altlasten zu verabschieden. Doch soweit muss es nicht kommen, wenn man einige wenige Erkenntnisse während der Entwicklungsphase konsequent umsetzt.

Kleine Einheiten statt großer Monolithen

Die Wiederverwendung von Komponenten und eine saubere Schichtenarchitektur galten lange als Allheilmittel gegen eine umfassende Wartung ganzer Software-Landschaften. Trotzdem machen die steigenden Wartungskosten den größten Teil der Weiterentwicklung aus.
Inzwischen zählen selbst objektorientierte Projekte zu den ungeliebten Altlasten und hierbei sind nicht nur unzeitgemäße Benutzeroberflächen gemeint. Eine Schichtenarchitektur – bestehend aus Oberfläche, Geschäftslogik und Datenhaltung – wird häufig eingesetzt, um vorhandenes Wissen und auch die Arbeit selbst besser organisieren zu können.

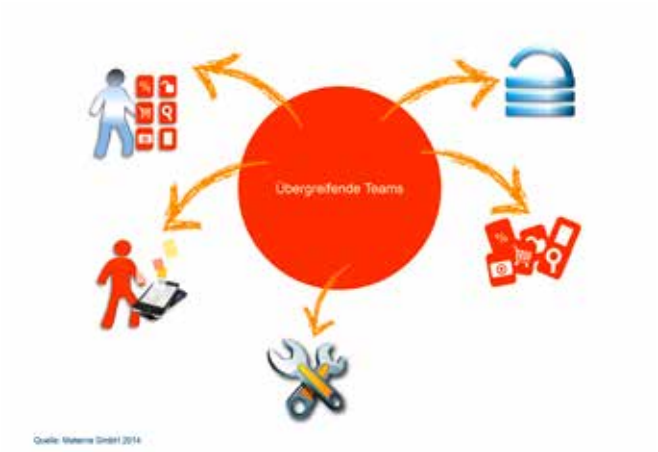

Abbildung 1: Alles Wissen ist in fachübergreifenden Teams vorhanden.

Statt die Aufgaben, wie in einem Schichtenmodell, nach technischen Aspekten zu organisieren und damit alle Ebenen unabhängig voneinander zu betrachten, sollte immer das System als Ganzes im Auge behalten werden. Dabei trägt das entsprechende Team nicht nur Verantwortung für die Entwicklung selbst, sondern auch für den produktiven Einsatz der Software. Teams sollten deshalb interdisziplinär aufgebaut sein, um sowohl die fachlichen, als auch die technischen und betrieblichen Aspekte abzudecken. Ein fachübergreifendes Team (siehe Abbildung 1) hat den Vorteil, dass es über das gesamte benötigte Wissen für die Erstellung und den Betrieb einer Anwendung verfügt. Ein solcher Ansatz reduziert die Abstimmungskosten und wirkt sich auch auf die Architektur einer Software-Lösung aus. So wird versucht, eine Anforderung ganzheitlich von der Erstellung bis zum Betrieb in kleinen Schritten zu lösen. Dies verhindert die Entwicklung unnötiger Funktionen. Da Entscheidungen dezentral getroffen werden, können Entwickler Fehler schneller und pragmatischer beheben.
Automatische Tests, Integration und Installation als Teil des Entwicklungsprozesses fördern die Entstehung kleinerer fachlicher Dienste und vertikaler Komponenten (siehe Abbildung 2). Ein kontinuierlicher Deployment-Prozess sorgt schließlich dafür, dass diese Services schneller und in verbesserter Qualität ausgeliefert werden. Auf diese Weise können Rückmeldungen früher aus dem Live-Betrieb in die Weiterentwicklung einfließen.

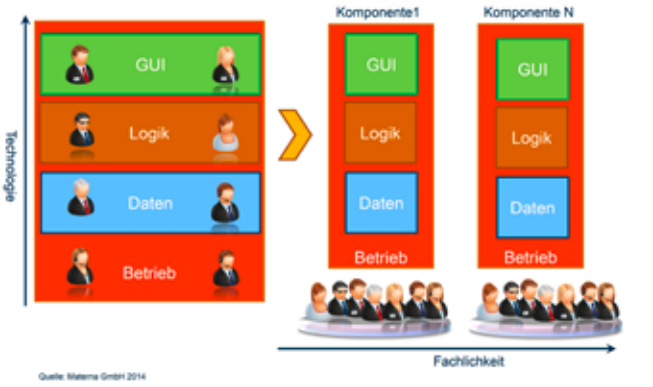

Abbildung 2: Von der Schichtenarchitektur zu vertikalen Komponenten

Miteinander statt gegeneinander

Positiv auf die Entwicklung nachhaltiger Software-Lösungen wirkt sich der DevOps-Ansatz aus. Der Begriff – zusammengesetzt aus „Dev“ für Anwendungsentwicklung (Development) und „Ops“ für IT-Betrieb (Operations) – steht für das Zusammenrücken der beiden Bereiche. Mit DevOps lassen sich Kommunikations- und Abstimmungsprobleme schon früh vermeiden und die Abteilungen lernen voneinander. Die Prozesse, Prinzipien und Werkzeuge, die in der Entwicklung eingesetzt werden, unterscheiden sich nicht – egal ob damit eine Test-, In-

tegrations- oder Produktionsumgebung aufgebaut wird. Solche Umgebungen werden in Einzelschritten aufgebaut, so dass Fehler durch Änderungen transparent nachvollziehbar sind. Um Missgriffe zu vermeiden und eine konstante Produktionsqualität zu erreichen, ist es notwendig, dass die einzelnen Schritte automatisiert und wiederholbar ausgeführt werden.
Da in fachübergreifenden Teams jeder für alles verantwortlich ist, muss dafür gesorgt werden, dass alle Mitarbeiter den gleichen Kenntnisstand haben. So lassen sich Wissensinseln und das Entstehen von Silos vermeiden. Das Entwicklungsteam kann lokal und selbstständig entscheiden, welche Maßnahmen es ergreifen möchte und welche Technologie zur Lösung des jeweiligen Problems am besten geeignet ist. Als schöner Nebeneffekt entstehen überschaubare und rasch testbare Software-Einheiten von besserer Qualität in kürzerer Zeit. Fachliche Komponenten entstehen also in vertikaler Ausrichtung und lösen das bekannte, an Technologien orientierte Schichtenmodell ab.

Randbedingungen können sich ändern

Plattformunabhängigkeit der Anwendungen mit einer Unterstützung für Internationalisierung und Lokalisierung sind gute Voraussetzungen, damit eine Software auch in anderen Bereichen als nur in der ursprünglich vorgesehenen Umgebung läuft. Diese Eigenschaften müssen jedoch auch in mindestens jeweils zwei Ausprägungen umgesetzt und vor allem permanent getestet werden, um hier Stolperfallen zu vermeiden.
Klare Strukturen und die Überprüfung der Einhaltung einheitlicher Regeln sind für die Wartbarkeit von Software nützlich. Der reine Blick auf den Source-Code versperrt aber manchmal die Sicht auf Schritte, die für den nachhaltigen Einsatz der Anwendung notwendig sind. Hierzu zählen Abläufe wie Test, Build und Deployment. Die Wahrheit liegt eben nicht allein im Code, sondern auch in der zur Verfügung stehenden Software.
Bestehende Software-Lösungen wurden oft auf Plattformen und mit Werkzeugen entwickelt, die heute nicht mehr existieren. Hier ist es hilfreich, wenn alle für den Bau benötigten Quelltexte noch vorhanden sind und mit einem Build-Werkzeug gebaut, getestet und verteilt werden können. Der Königsweg führt dabei vom kontinuierlichen Build-Prozess über die kontinuierliche Integration bis hin zum kontinuierlichen Deployment-Prozess. Der Status der einzelnen Projektschritte wird dabei durch einfache aggregierte Metriken visualisiert, die auf detaillierten Testprotokollen beruhen. Oft wird jedoch vergessen, dass implizite Abhängigkeiten zum benutzten Betriebssystem und seinen Werkzeugen bestehen. Deswegen sollten alle Abhängigkeiten explizit ausgeschlossen werden. In der Konsequenz müssen die benötigten externen Werkzeuge ebenfalls versioniert und in der Produktion als Komponente der Anwendung ausgeliefert werden. Gerade die Variantenvielfalt bei mobilen Endgeräten kann ohne einen kontinuierlichen Deployment-Prozess, der auch die Zielumgebung in der jeweils benötigten Version automatisiert und schnell zur Verfügung stellt, kaum mehr effizient abgedeckt werden.

Design-for-Replacement statt Design-for-Reuse

Kleine, unabhängige Software-Komponenten haben den Vorteil, dass diese einfacher zu ändern oder zu ersetzen sind. Hier gilt der Grundsatz, dass Design-for-Replacement wichtiger ist als Design-for-Reuse. Um sicherzustellen, dass diese Ersetzbarkeit auch erhalten bleibt, ist es notwendig, dass neben der eigentlichen Zielumgebung im automatischen Build- und Deployment-Prozess alternative Versionen oder Umgebungen mitgetestet werden. Die kontinuierliche Überprüfung der Ersetzbarkeit vermeidet später größere und riskantere Migrations- und Modernisierungsschritte. Dadurch wird späterer Aufwand und das damit verbundene Risiko von neuen Fehlern minimiert. Als angenehmer Nebeneffekt können neuere Versionen schneller ausgetauscht werden, um zum Beispiel Sicherheitslücken oder Fehler in Fremdkomponenten in der Produktion zu beheben, da diese bereits präventiv mitgetestet wurden. So bleibt die Anwendung in der Produktion länger stabil, durch die neu eingesetzten Versionen aktuell und kann auch mit wachsenden Anforderungen Schritt halten.
Um Verzögerungen durch fehlendes Wissen und die damit verbundenen Kosten mittelfristig zu vermeiden, sollten Unternehmen alle wichtigen Informationen für Mitarbeiter transparent und zentral verfügbar machen.
Für die Teammitglieder bedeutet dies ein Umlernen, sodass sie sich nicht nur in ihrem Spezialgebiet auskennen, sondern mindestens zwei weitere Fachgebiete abdecken müssen. Hier sind eine gute Grundlagen- und Methodenausbildung neben hoher Lernbereitschaft wichtige Voraussetzungen, um sich schnell in neue Fachgebiete einzuarbeiten. Man spricht dann von einem T-förmigen Mitarbeiterprofil. Vor allem bei Wartungsprojekten ist es wichtig, ein „Kopfmonopol“ zu vermeiden. Deswegen sollten Änderungen immer von zwei Personen durchgeführt werden. Oft entsteht so eine bessere Lösung. Das breiter verteilte Wissen reduziert ein mögliches Personenrisiko.
Das Verhalten der Software sollte möglichst einfach ohne viel Zusatzaufwand anpassbar sein. Funktionen ändern sich schneller als Anforderungen an Qualität. Deswegen ist es wichtig,

langfristig Wert auf die Erfüllung der nicht-funktionalen Anforderungen zu legen, denn eine Investition in Qualität zahlt sich hier aus.
Weil sich die Infrastruktur oft ändert, kann dies Auswirkungen auf die Software haben. Deswegen spielt gerade bei einer verteilten Anwendung die Robustheit gegenüber Netz- und Dienstausfällen eine große Rolle. So sollte die Software möglichst sanft mit Fehlern in der Infrastruktur umgehen und die Möglichkeit bieten, auf einen Notbetrieb umzuschalten, zum Beispiel mit dem Schutzschalter-Muster von Michael Nygard. So bleibt das System mit seinen Grundfunktionen verfügbar, auch wenn einige seiner verwendeten Dienste temporär nicht erreichbar sind. Nach einer festgelegten Zeit wird dann automatisch wieder auf den Normalbetrieb umgeschaltet.

Fazit

Alles hat seine Zeit. Deswegen sollten sich Unternehmen rechtzeitig von Dingen trennen, die auf einer früheren falschen Annahme oder Umsetzung beruhen. Wenn sie nicht mit überschaubarem Aufwand und Risiko reparierbar sind, sollte sich die IT-Abteilung von alten Ansätzen oder nicht mehr wartbaren Fremdkomponenten trennen. Es gilt, unnötigen Ballast loszuwerden oder diesen schon bei der Entstehung zu vermeiden.
Eine gezielte regelmäßige Aktualisierung der Software erleichtert spätere Anpassungen. Oft lassen sich so größere Migrations- und Modernisierungsschritte vermeiden. Notwendige Änderungen sollten frühzeitig angegangen und besser isoliert betrachtet werden. Leichen im Design- und Bomben im Code-Keller entfernt man am besten sofort, bevor sie größeren Schaden anrichten.
Das Problem der Legacy-Software sollte von seiner Entstehung her gesehen werden (ad fontes). Nach dem Gesetz von Conway werden die Strukturen von Systemen durch die Kommunikationsstrukturen in den jeweiligen Organisationen beeinflusst. Eine Microsoft Research-Studie bestätigt dieses Gesetz für Windows Vista. Die dort entstandene Komplexität und Fehlerrate lässt sich direkt aus der dafür zuständigen Organisationseinheit bei Microsoft ableiten.
Als eine Konsequenz wurde die Code-Anzahl in der Nachfolgeversion Windows 7 sogar noch unter die von Windows XP reduziert. Deswegen reicht es für eine verbesserte Wartungsqualität nicht aus, nur die Symptome innerhalb der Software zu bekämpfen. Es muss auch der Prozess, wie Software entworfen und entwickelt wird, an aktuelle Anforderungen angepasst werden. Das kann heute mit anderen Organisationsformen erfolgen, als es früher möglich war.
Kommunikation und Entscheidungen haben sich dramatisch verändert. Dies bedeutet, dass Unternehmen heute mit anderen Organisationsformen auch zu anderen Ergebnissen kommen. Statt die Architektur anhand künstlicher Technologien und Wissensgrenzen aufzubauen, sollte alles Wissen für die Erstellung und den Betrieb der Software im Projektteam hinreichend vorhanden sein. Das Team ist sowohl für die Erstellung als auch für den Betrieb zuständig. Dies vermeidet Silodenken und verbessert die Kommunikation. ■

Über den Autor

Frank Pientka ist seit mehreren Jahrzehnten in der professionellen Software-Entwicklung tätig. Seit 2009 arbeitet er als Senior Software Architect bei Materna. Als Gründungsmitglied des iSAQB (International Software Architecture Qualification Board) liegt ihm das Lernen und Vermitteln von guten Design-Prinzipien für mehr Qualität in der Software am Herzen. Sie erreichen den Autor unter: frank.pientka@materna.de.

Interessante Beiträge

- Materna Monitor 4/2011, Frank Pientka, Vom Sprint zum Langstreckenlauf: Entwicklung trifft Betrieb
- Materna Monitor 1/2014, Frank Pientka, Entwickelst du noch oder lieferst du schon: Continuous Delivery
- McKinsey, Juli 2014, Sriram Chandrasekaran, Sauri Gudlavalleti, Sanjay Kaniyar, Achieving success in large, complex software projects