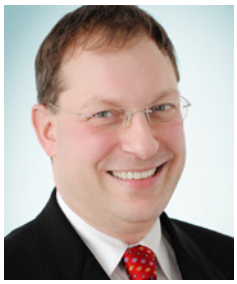

Frank Pientka
(frank.pientka@materna.de)
ist Senior Software Architect bei der Materna GmbH in Dortmund und verantwortlich für das Beratungsprodukt WebCheck Architecture. Er führt Architekturreviews durch und als Gründungsmitglied des iSAQB liegt ihm das Lernen und Vermitteln von guten Design-Prinzipien für mehr Qualität in der Software besonders am Herzen.

Wege aus der Software-Wartungsfalle – keine wilde Softwareaufzucht

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. Das Prinzip der Nachhaltigkeit kann, um Ressourcenverschwendung und teure Migrationsprojekte zu vermeiden, auch auf die Softwareentwicklung übertragen werden. In diesem Artikel werden die wesentlichen Aspekte, die nachhaltige Softwareentwicklung unterstützen, vorgestellt.

Die Softwarekrise gibt es seit mehreren Jahrzehnten und sie betrifft Neuprojekte wie Altprodukte gleichermaßen. Schon bei der Entwicklung werden oft 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.

Nachhaltige Software

Wer Unternehmensanwendungen entwickelt, möchte diese über Jahre hinweg nutzen. Die Erfahrungen der letzten Jahre haben gezeigt, dass die Kosten für die Wartung die eigentlichen Entwicklungsaufwendungen übersteigen und damit der Einsatz der Software oft nicht mehr wirtschaftlich ist.

Hier kommt der Begriff der Nachhaltigkeit ins Spiel. Als Folgen des Dreißigjährigen Krieges herrschte im 17. Jahrhundert in ganz Europa ein großer Holzmangel. Deswegen schrieb Hans Carl von Carlowitz vor über 300 Jahren eine Buch „Sylvicultura oeconomica oder Anweisung zur wilden Baum-Zucht“ für eine kontinuierliche, beständige und nachhaltende Nutzung des Holzes.

Ähnlich wie der Wald ist die Softwareentwicklung heute keine Monokultur mehr, sondern ein komplexes Ökosystem mit vielen Beteiligten. In Anlehnung an den Brundtland-Bericht der Vereinten Nationen, der 1987 erstmalig den Begriff der Nachhaltigkeit allgemein beschrieb, wurde das Konzept der *digitalen* Nachhaltigkeit entwickelt, das die langfristig orientierte Herstellung und Weiterentwicklung von digitalen Wissensgütern ermöglicht.

Die knappe Ressource für uns ist heute der Mensch mit seinem Wissen, seinen Erfahrungen und Fähigkeiten. Um diese dauerhaft zu nutzen und zu entwickeln, muss sich die Art ändern, in der wir Softwareentwicklung betreiben. Dabei müssen die Interessen von Ökonomie, Ökologie und sozialem Miteinander in Einklang gebracht werden (siehe Abbildung 1).

Ohne Open-Source-Komponenten sind heute weder das WWW noch eine kommerzielle Softwareentwicklung wirtschaftlich denkbar. Gerade wenn die Software in industriellen Produktionsprozessen oder embedded eingesetzt wird, ist es entscheidend, nicht nur den Quellcode zu besitzen, sondern den gesamten Erstellungsprozess dauerhaft durchführen zu können. Je mehr Software in die Wirtschaftspro-

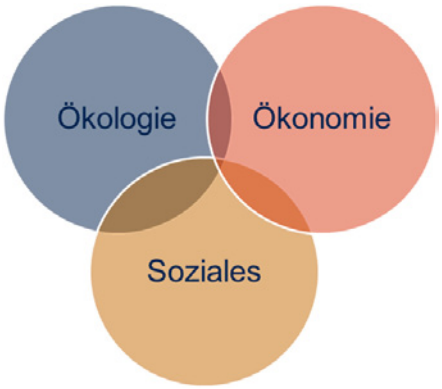

Abb. 1: Ökologie, Ökonomie und Soziales ins Gleichgewicht bringen

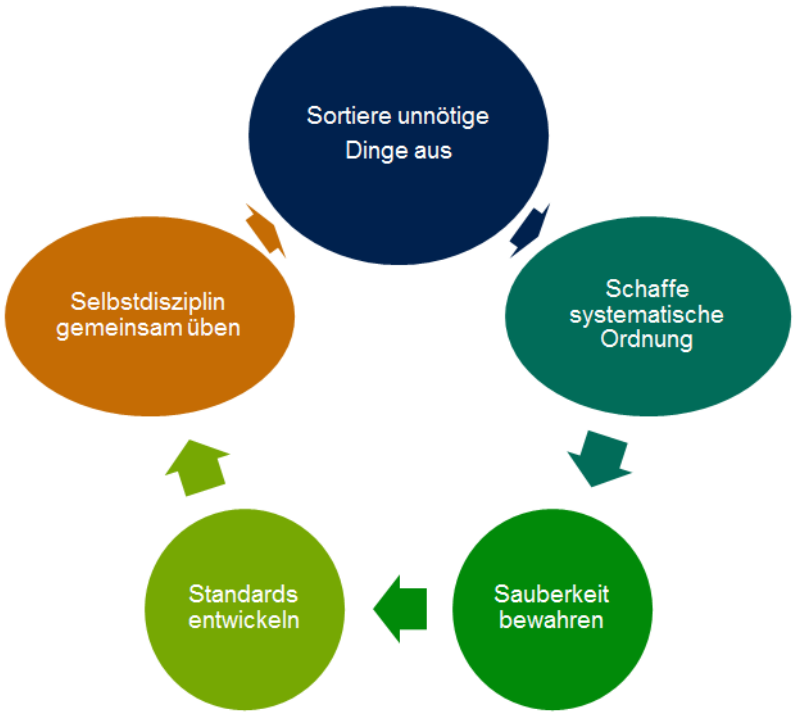

Abb. 2: 5 Schritte zur besseren Wartbarkeit

zesse eines Landes eingebunden ist, umso wichtiger wird der Faktor Open Source.

Schritte zur ganzheitlichen Produktion

Zu einer nachhaltigen Verbesserung der Produkte und ihrer Wartung wurde Anfang der 50er-Jahre in Japan das Konzept der Total Productive Maintenance (TPM) entwickelt. Es überträgt die Prinzipien von Kaizen oder Lean Production auf die Instandhaltung. Die 5S zur Arbeitsgestaltung, im deutschen Sprachraum auch 5A genannt, lassen sich gut auf die nachhaltige Weiterentwicklung der Software und der Evolution ihrer Architektur übertragen:

1. Sortiere unnötige Dinge aus
2. Schaffe systematische Ordnung
3. Sauberkeit bewahren
4. Standards entwickeln
5. Selbstdisziplin gemeinsam üben

Diese 5 Schritte, werden, wie in Abbildung 2 dargestellt, mehrfach durchlaufen.

Innere und äußere Qualität zählt

Eine gute Basis für die langfristige Nutzung von betriebswirtschaftlicher Software sind nicht-funktionale Qualitätsmerkmale nach der ISO 25010-Norm. Für die Bereiche Wartbarkeit, Änderbarkeit, Anpassbarkeit und Modifizierbarkeit gibt es ausgereifte objekt orientierte Metriken, um den Erfüllungsgrad zu messen und geeignete Korrekturmaßnahmen zur Erhaltung der inneren Qualität einzuleiten.

Meist werden diese Metriken mit der Wiederverwendbarkeit in Verbindung gebracht, die jedoch nicht automatisch dadurch erreicht wird. Wichtiger als Wiederverwendbarkeit ist für eine nachhaltige Softwareentwicklung oft die Austauschbarkeit, Übertragbarkeit oder die Interoperabilität von Softwarekomponenten.

Eine wichtige Voraussetzung dafür sind die Testbarkeit, Fehlertoleranz, Analysierbarkeit und Stabilität der verwendeten Software. Leider ist es hier schon schwieriger, allgemeine Metriken zu finden, sodass diese selbst definiert werden müssen.

Um Ziele wie die Anpassbarkeit oder die Änderbarkeit zu erreichen, ist es hilfreich, hier explizite mögliche Wachstums- und Änderungsszenarien frühzeitig zu entwickeln und diese von Zeit zu Zeit bei Entscheidungen zu überprüfen oder bei geänderten Rahmenbedingungen explizit anzupassen. Dadurch können Entscheidungen besser kontrolliert und unter konkreteren Bedingungen getroffen werden (siehe Tabelle).

Für die Übertragbarkeit, Analysierbarkeit und die Interoperabilität sind neben der Einhaltung von Standards auch die Verwendung einheitlicher Querschnittskonzepte wichtig. Nur durch eine konsistent umgesetzte Architektur und Muster kann die innere und äußere Qualität erreicht werden. Beide Qualitätsaspekte sind für die nachhaltige Produktqualität wichtig.

Nachhaltigkeit betrifft alle Phasen des Softwareentwicklungsprozesses und auch die Prozessqualität hat einen großen Einfluss auf die Produktqualität. Ein eingespieltes Entwicklungsvorgehen mit von allen gelebten Regeln, eine Kultur des Wissensaustausches und der Fehlertoleranz sind dabei eine wichtige Basis. Für einen guten Wartungsprozess muss das Wissen im Team über die Software, die dabei verwendeten Pflegeprozesse und die wichtigen Architekturprinzipien und -entscheidungen frisch gehalten werden.

Kleine Einheiten statt großer Monolithe

Die Wiederverwendung von Komponenten und eine saubere Schichtenarchitektur galten lange als Allheilmittel gegen eine umfassende Wartung ganzer Softwarelandschaften. Trotzdem machen die steigenden Wartungskosten den größten Teil des Weiterentwicklungsetats aus.

Eine Schichtenarchitektur – bestehend aus Oberfläche, Geschäftslogik und Datenhaltung – wird häufig eingesetzt, um vorhandenes Wissen und auch die Arbeit selbst besser organisieren zu können. Dabei wird oft zu wenig berücksichtigt, dass dadurch die Abstimmungskosten steigen

Art des Evolutionsszenariums	Name	Wahrscheinlichkeit
Wachstum	Ausbau der Produktionsumgebung auf 10 Knoten	mittel
Änderung	GUI-Framework austauschen	gering
Änderung	Fremdbibliothek auf Majorversion aktualisieren	hoch
Wachstum	Benutzerzahlen verdoppeln sich	mittel

Tab.: Evolutionsszenarien

und eine einheitliche Sicht auf die Architektur verloren geht, sodass z. B. Redundanzen zwischen den Schichten entstehen, die eine Gesamtoptimierung erschweren.

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 3) 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 Softwarelösung aus.

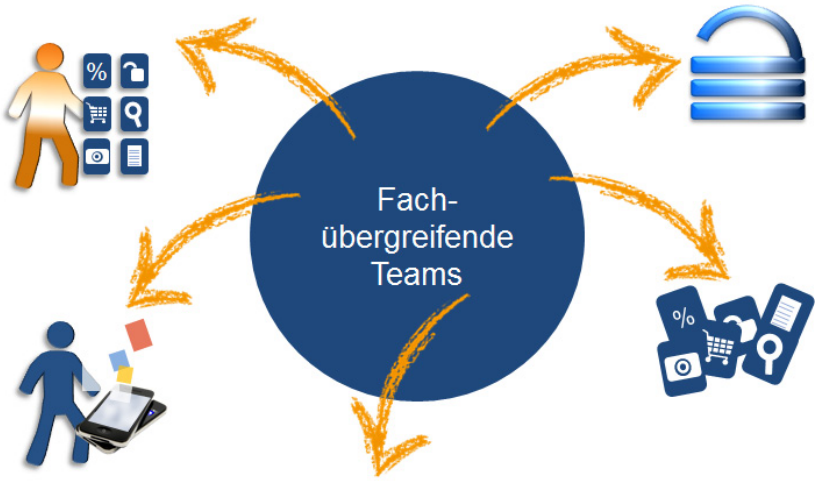

Abb. 3: Alles Wissen ist in fachübergreifenden Teams vorhanden.

Miteinander statt gegeneinander

Automatische Tests, Integration und Installation als Teil des Entwicklungsprozesses fördern die Entstehung kleinerer fachlicher Dienste und vertikaler Komponenten (siehe Abbildung 4). Positiv auf die Entwicklung nachhaltiger Softwarelösungen 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-, Integrations- oder Produktionsumgebung aufgebaut wird.

Da in fachübergreifenden Teams, nach dem Scrum Guide 2013, jeder für das Ganze verantwortlich und das benötigte Wissen im gesamten Team vorhanden ist, sollte angestrebt 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 dabei überschaubare und rasch testbare Softwareeinheiten 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

Bestehende Softwarelösungen wurden oft auf Plattformen und mit Werkzeugen entwickelt, die heute nicht mehr existieren. Oft wird zudem 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. Der Königsweg führt dabei vom kontinuierlichen Build-Prozess über die kontinuierliche Integration bis hin zum kontinuierlichen Deployment-Prozess.

Sollte sich beim eigenen Produkt mit der Zeit herausstellen, dass die Wartungsaufwendungen und Änderungszeiten unverhältnismäßig steigen, sollte man über Reparaturmaßnahmen nachdenken. Dabei sollte immer abgewogen werden, ob ein größerer Aufwand oder ein größeres Risiko für eine Reparatur gegenüber einem Austausch und Ersatz einer anderen Komponente gerechtfertigt ist.

Eine Frage, die man sich beim Weiternutzen bestehender Produkte stellen sollte, ist, ob diese auch in Zukunft den gewachsenen und veränderten Anforderungen standhalten. Können diese angepasst oder erweitert werden oder müssen sie

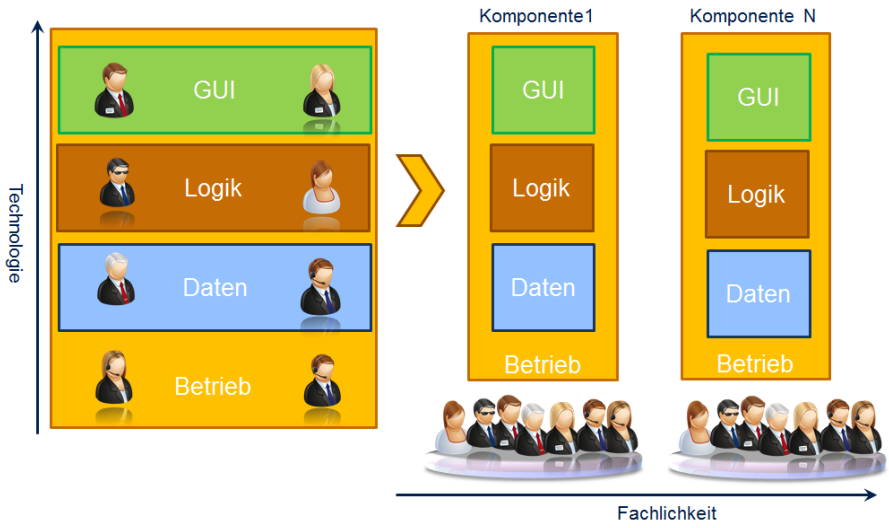

Abb. 4: Von der Schichtenarchitektur zu vertikalen Komponenten

grundsätzlich umgebaut und repariert werden?

Dann, wenn riskante und teure Umbauten nötig sind, sollte man überlegen, ob man die Funktionen oder Teile davon nicht ersetzen kann. Bevor man größere Architektur-Refactorings durchführt, ist es effizienter, kleinere Refacoring-Maßnahmen kontinuierlich anzuwenden. Hierdurch kann sowohl die Strukturierung des Codes, als auch das Architektur-Design schrittweise mit weniger Risiko und Aufwand verbessert werden.

Eine gezielte regelmäßige Aktualisierung der Software erleichtert die Weiterentwicklung. Oft lassen sich damit größere Migrations- und Modernisierungsschritte vermeiden. Notwendige Änderungen sollen frühzeitig angegangen und besser isoliert betrachtet werden. Leichen im Design und Bomben im Code-Keller entfernt man sofort, bevor sie größeren Schaden anrichten. Dabei sollte man an den Satz von Marc Aurel denken „Beachte immer, dass nichts bleibt, wie es ist, und denke daran, dass die Natur immer wieder ihre Formen wechselt."

Ökologie trifft Ökonomie – nutzen statt besitzen

Eine Grundsatzfrage, die man sich bei der Entwicklung und noch mehr bei der Weiterentwicklung der Software stellen sollte, lautet: „Will oder muss ich bestimmte Funktionen selbst entwickeln oder kann ich bestehende Funktionen nutzen und integrieren". Um Ressourcen zu sparen, geht der Trend in der Konsumwelt des „Shareconomy" immer mehr vom Besitzen zum Nutzen. Stabile APIs und eine Versionierungsstrategie für größere API-Änderungen sind für die Langzeit-Unterstützung eines Softwareprodukts immer wichtiger.

Im eigenen Projekt sollte man zwar immer nur eine Version einer Bibliothek verwenden. Doch kann es passieren, dass in der Ablaufumgebung oder in größeren Projekten mehrere Versionen zum Einsatz kommen. Um einen Parallelbetrieb von neuer und alter API zu erreichen und Konflikte möglichst zu vermeiden, sollte man sich einheitliche Versionierungsverfahren für API-Änderungen überlegen.

Design-for-Replacement statt Design-for-Reuse

Kleine, unabhängige Softwarekomponenten 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 werden späterer Aufwand und das damit verbundene Risiko von neuen Fehlern minimiert.

Als angenehmer Nebeneffekt können neuere Versionen schneller ausgetauscht werden, 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.

Für eine kontinuierliche Produktqualität ist neben dem *Was* auch das *Wie* entscheidend. Wie eine Software entworfen und entwickelt wird, muss an aktuelle Anforderungen angepasst werden.

Kommunikation und Entscheidungen haben sich heutzutage gegenüber früher stark geändert. Das bedeutet, dass wir unsere Organisationsformen an den aktuellen Bedarf anpassen.

Statt die Architektur an künstlichen 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, um Silodenken zu vermeiden und die Kommunikation zu verbessern.

Für die Teammitglieder bedeutet dies ein Umlernen, sodass sie sich nicht nur in ihrem Spezialgebiet auskennen, sondern mindestens zwei weitere Fachgebiete abdecken sollten. 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. 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.

Fazit

Die Softwareentwicklung hat sich nicht nur technologisch, sondern insbesondere organisatorisch verändert. Die Softwarebranche wird immer wieder von Hype-Wellen überrollt.

Für Projekte sind jedoch nicht die Innovation, sondern die Konsistenz und die Reife der Lösung wichtige Erfolgsfaktoren. Das Wissen sollte möglichst umfassend und gleichverteilt bei allen Beteiligten und eingesetzten Komponenten und Werkzeugen sein, um das System nachhaltig weiterentwickeln zu können.

Über die langfristige Qualität einer Anwendung entscheidet heute weniger die eingesetzte Technologie, als vielmehr, mit welcher Organisationsform die Software entsteht. 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. ■