WAS SOFTWAREENTWICKLUNG HERAUSFORDERND MACHT

Ein Artikel von Josef Wohlrab, ROI-EFESO

Die Bedeutung von Software für die zukünftige Wettbewerbsfähigkeit zeigt vor allem der disruptive Wandel der Automobilindustrie. Vor wenigen Jahren noch belächelt, hat sich Tesla zum Treiber der Branche entwickelt. Das Unternehmen macht sich nicht allzu viele Gedanken über Spaltmaße – und folgt stattdessen der Idee der „fahrenden Software", was etablierte Automobilmarken radikal herausfordert. Von der Börse beflügelt, hat Tesla die Traditionskonzerne damit in den Wettbewerb auf einem ungewohnten Terrain gezwungen. Und der Auftritt der Großen der Branche lässt bislang zu wünschen übrig. Was aber macht die Softwareentwicklung so herausfordernd und wo liegen die Besonderheiten und Unterschiede zu traditionellen industriellen F&E-Projekten?

Software lebt

Ein wesentlicher Unterschied zur Hardware ist, dass diese zum SOP (Start of Production) komplett fertig sein muss. Software hingegen „lebt" – sie wird auch nach der Auslieferung an den Kunden kontinuierlich weiterentwickelt, kann angepasst und erweitert werden – und muss das auch. Zum einen als Teil eines sich ständig ändernden Eco-Systems, zum anderen, um dem Nutzer verbesserte, erweiterte oder neue Features anzubieten und durch den Nachverkauf weiteres Geschäft zu generieren. Der limitierende Faktor hierfür ist die Kapazität der zugrunde liegenden Hardware, d.h. Speicher, Prozessor und Peripherie. Entgegen der verbreiteten Annahme, dass Software schnelllebig sei, lebt erfolgreiche Software im Kern von evolutionärer Weiterentwicklung – und kann dabei älter werden als manch eine Hardware, auf der sie läuft.

Trotz dieser Langlebigkeit ist die Entwicklung von Software sehr viel dynamischer und kurzzyklischer als die von Hardware. Dafür gibt es verschiedene Ursachen. So existieren häufig bereits Teile des gewünschten Codes und stehen sogar als Open Source kostenlos zur Verfügung. Zeitaufwendiges Prototyping ist nicht notwendig, neue Versionen können in kurzen Zyklen geliefert werden. Software lässt sich im Wesentlichen in System- und Applikationssoftware unterscheiden. Die Systemsoftware, z.B. Betriebssysteme, stellt hierbei die Grundfunktionalität für die Applikationssoftware des Nutzers sicher. Die Annahme scheint naheliegend, dass die im Hintergrund laufende Systemsoftware, sofern sie einmal gut läuft, analog zu guter Hardware keinen wesentlichen Überarbeitungsbedarf aufweist. Doch beide Softwaretypen müssen gleichermaßen gepflegt und angepasst werden, da sich die Peripherie (Hardware) und vor allem die Schnittstellen des Ökosystems weiterentwickeln. Die kontinuierliche Gestaltung, Pflege und Wartung der Softwarearchitektur ist daher eine wesentliche Kernkompetenz in der Softwareentwicklung. Die immer größer werdende Anzahl der Hardware-Varianten, die unterstützt werden müssen, und die Notwendigkeit langfristiger Softwarepflege fordern eine hohe Kompetenz im Versions- und Konfigurationsmanagement. Dafür braucht es eine saubere Dokumentation und eine stringente Nachvollziehbarkeit.

MVP is key

Hardware-Prototypen sind teuer und treiben mit langen Lead Times maßgeblich die Entwicklungsdauer von der Idee bis zum SOP. Änderungen in späten Entwicklungsphasen sind kostenintensiv und Fehler im Entwicklungszeitraum führen zu exponentiellen Folgekosten. Daher setzt man bei der Hardware nach dem Prinzip des Systems Engineering auf ein möglichst gutes Frontloading in der Erhebung und Beschreibung der Anforderungen und konzentriert sich darauf, fehlerfreie Produkte an Kunden auszuliefern. Anforderungen an die Software werden hingegen im Laufe der Entwicklung kontinuierlich verfeinert oder sie entstehen überhaupt erst während der Entwicklung.

Hardware wird mit komplexen und teuren Werkzeugen produziert, wobei Entwicklungsfehler nachträglich korrigiert werden können. Software hingegen ist nach der Entwicklung fertig. Sie wird nicht produziert, sondern kopiert. Daher ist es umso wichtiger, dass das Original bereits eine hohe Qualität und Reife aufweist, was jedoch selbst bei einfacher Software alles andere als selbstverständlich ist.

Um Fehler in der komplexen Softwareentwicklung zu reduzieren und die Qualität so früh wie möglich sicherzustellen, existieren unterschiedliche Methoden, wie z.B. automatisierte Tests, statische Code-Analysen, Test-driven Development oder Pair Programming. Der entscheidende Hebel liegt jedoch in der Möglichkeit, sehr früh Nutzerfeedback einzuholen, um eine Entwicklung am Nutzer vorbei zu verhindern. Durch die schnelle Erzeugung eines MVP (Minimal Viable Product), d.h. eines Produkts mit minimalem für den Kunden sinnvollem Funktionsumfang, kann kurzfristig das Feedback (und ggf. neue Anforderungen) abgeholt und damit die weitere Entwicklung gesteuert werden. Die Entwicklung von Software erfolgt deshalb deutlich kurzzyklischer als die von Hardware: mit einem lebenden Anforderungs-Backlog statt eines festgeschriebenen Lasten- und Pflichtenhefts.

Etwas schwieriger wird dies bei technisch komplexen Systemen, etwa bei komplexer Software, die für Hardware-basierte Produkte, wie das Automobil, produziert wird. Hier ist es notwendig, beide Projektphilosophien – den agilen Softwareansatz und den für die traditionelle Produktion typischen Wasserfall-Ansatz – im Rahmen eines hybriden Entwicklungsmodells gut zu verzahnen.

Architektur dominiert den Lebenszyklus

Besonders hoch ist die Relevanz sauberer Architekturarbeit bei technisch komplexen Systemen, die sicherheitsrelevante Funktionalitäten realisieren und daher höchsten Anforderungen an Zuverlässigkeit, Qualität und Sicherheit unterliegen.

Vor allem bei Greenfield-Ansätzen in der Softwareentwicklung liegt der Schlüssel für eine hochwertige Software in einer guten Architektur, die stimmig zur zugehörigen elektrisch-elektronischen (E/E-) und Hardware-Architektur ist und von einer logischen und funktionalen Architektur abgeleitet wurde. Gute Architekturarbeit setzt eine hohe Kompetenz und Disziplin voraus. Dabei basiert sie vor allem auf zwei Prinzipien: der Hierarchisierung und der Modularisierung. Zum einen können nur dadurch Änderungen, Implikationen und Abhängigkeiten systematisch bewertet werden. Zum anderen lassen sich Module – standardisierte Blöcke mit definierten Schnittstellen – in der Software deutlich besser skalieren als in der Hardware, da sie nicht durch physikalische Eigenschaften in ihren Einsatzmöglichkeiten limitiert sind.

Bei diesen Überlegungen gilt es, einen wichtigen Aspekt zu beachten. Software wird selten auf einem weißen Blatt entwickelt. Deshalb spielt die kontinuierliche Verbesserung der Software („Refactoring“) eine wesentliche Rolle in der Softwareentwicklung.

Dadurch wird laufend die Komplexität reduziert, die Lesbarkeit erhöht und damit die Wartbarkeit und Erweiterbarkeit sichergestellt. Erfolgt dies nicht kontinuierlich in ausreichendem Umfang, läuft man Gefahr, durch die Anhäufung technischer Schulden eine nicht mehr beherrschbare Komplexität zu erzeugen. Das würde das Ende der Software bedeuten. Für die stetige Weiterentwicklung ist deshalb ein sauberes Versions- und Konfigurationsmanagement notwendig.

Auch Hardware bzw. deren Struktur lässt sich optimieren. Doch ist dies deutlich schwieriger und aufgrund der benötigten neuen Produktionswerkzeuge mit hohen Kosten verbunden. Außerdem entsteht deutlich seltener ein umfassender Optimierungsbedarf durch neue Umgebungsbedingungen oder Funktionserweiterungen.

Same same but different

Die Entwicklung von Software folgt in mehrfacher Hinsicht anderen Logiken, Dynamiken und technologischen Prozessen als die von Hardware. Gerade für Unternehmen, die stark auf klassische Produktentwicklung ausgerichtet sind, bedeuten diese Unterschiede eine Herausforderung. Sie müssen andere Organisationsformen, Metriken und Effizienzsteigerungsinstrumente etablieren, die eine erfolgreiche Steuerung der Softwareentwicklung erlauben. Gleichzeitig gilt es, die beiden Welten zu integrieren und eine erfolgreiche Zusammenarbeit zu organisieren, um eine nachhaltig erfolgreiche Transformation ins Zeitalter der smarten, digitalen Industrie sicherzustellen.