



□ Dr. Michael Sturm

(E-Mail: michael.sturm@berner-mattner.com)
 leitet die Abteilung Software Products im Geschäftsbereich Industry der Berner und Mattner Systemtechnik GmbH. Er verfügt über langjährige Erfahrung im Produktmanagement und in der Entwicklung komplexer Softwaresysteme. Er erwarb sein Diplom in Informatik 1994 an der Technischen Universität München und promovierte 2000 am Lehrstuhl für Theoretische Informatik und Grundlagen der künstlichen Intelligenz.



□ Dr. Alexander Harhurin

(E-Mail: alexander.harhurin@berner-mattner.com)
 promovierte am Lehrstuhl für Software & Systems Engineering der TU München. Seine Kompetenzen in den Bereichen modellbasierte Entwicklung, Anforderungsmanagement und Entwurf von Facharchitekturen sicherheitskritischer Softwaresysteme hat er im Zuge seiner siebenjährigen Mitarbeit an zahlreichen Projekten in der Automatisierungs- und Automobilindustrie erworben.

Softwarearchitektur in eingebetteten Systemen

Die wachsende Komplexität softwareintensiver Steuerungssysteme und die gesetzlichen Anforderungen an die Sicherheit von Produkten stellen die Industrie vor neue Herausforderungen. Interaktion von Funktionen verschiedener Steuergeräte, geografisch verteilte Entwicklung sowie begrenzte Ressourcen von Steuergeräten sind Beispiele dieser Herausforderungen. Um diesen Rechnung zu tragen, stellt dieser Artikel die Rolle des Softwarearchitekten sowie die Rolle der Softwarearchitektur im Bereich eingebetteter Systeme dar. Wir diskutieren die wesentlichen Synergien und Unterschiede zwischen der klassischen Entwicklung von Business-Software und der von softwareintensiven Steuerungssystemen. Die Aspekte Qualitäts- und Ressourcenanforderungen, Wiederverwendbarkeit und Variabilität sowie Organisation der Entwicklung werden in diesem Artikel ebenfalls behandelt.

Einführung

Der Anteil elektronischer und softwareintensiver Steuerungssysteme ist in den vergangenen Jahren in fast allen Industriebranchen stark angestiegen. Er wird getrieben von steigenden gesetzlichen Anforderungen und Kundenanforderungen, die in vielen Fällen kosten- und ressourceneffizient durch Software realisiert werden können. Beispielsweise werden nahezu alle Funktionen in modernen Fahrzeugen inzwischen elektronisch gesteuert, geregelt oder überwacht [SuT10].

Die wachsende Komplexität softwareintensiver Steuerungssysteme und die gesetzlichen Anforderungen an die Sicherheit von Produkten stellen die Industrie vor neue Herausforderungen:

- Physikalische Prozesse, die durch eingebettete Systeme gesteuert werden, stellen besondere Anforderungen an die Softwaresysteme. Echtzeitanforderungen oder nichtfunktionale Anforderungen,

wie begrenzter Ressourcenverbrauch (Prozessorleistung, Energieverbrauch, Speicherplatz, etc.), können nur durch gemeinsames Hard- und Software Engineering (Co-Design) erfüllt werden.

- Die Sicherheits- und Zuverlässigkeitsanforderungen, die in besonderem Maße an eingebettete Systeme gestellt werden, betreffen die Hard- und Softwarequalität (mehr dazu in [HuH11]).
- Der Bedarf für eine wachsende Anzahl softwarebasierter Funktionen bei einer konstanten Anzahl von Steuergeräten führt in der Praxis zur stärkeren Interaktion von Funktionen verschiedener Steuergeräte. Die Funktionen beschränken sich nicht mehr nur auf ein Steuergerät und können somit auch nicht mehr in einer Hardware-Topologie dargestellt werden [Bro07].

- Ein System wird selten von einem einzigen Team entwickelt, vielmehr ist die Entwicklung komplexer Systeme durch den klassischen Zulieferprozess geprägt. Die Gliederung des Systems in handhabbare Teilsysteme mit fest definierten Schnittstellen ist eine der wichtigsten und gleichzeitig der schwierigsten Aufgaben eines Systemintegrators.

Um diesen Herausforderungen Rechnung zu tragen, setzen wir in allen Projektphasen Softwarearchitekten ein, die den sorgfältigen Entwurf einer Softwarearchitektur nach modernen Methoden des Systems- und Software-Engineerings ermöglichen.

Die Rolle des Softwarearchitekten sowie der Softwarearchitektur im Bereich eingebetteter Systeme werden ebenso wie die Aspekte der Qualitäts- und Ressourcenanforderungen, Wiederverwendbarkeit und Variabilität sowie Organisation der Entwicklung im Folgenden behandelt.

Rolle des Architekten

Kommunizieren und motivieren

Eingebettete Entwicklungen sind traditionell durch eine sehr technische Sichtweise geprägt. Speziell die Historie der Entwicklung aus dem Steuerungs- und Regelungsumfeld von Systemen heraus erforderte häufig keine spezialisierten Informatikkenntnisse.

Diese Situation hat sich grundlegend geändert. Die heute notwendigen Fähigkeiten erfordern ein stark heterogenes Team, das unterschiedlichste Disziplinen abdeckt. So ist neben den Kenntnissen der klassischen Regelungstechnik auch spezialisiertes Wissen über z. B. Aufmerksamkeitsprozesse zum Oberflächendesign notwendig. Das Lenken der Aufmerksamkeit auf aktuell wichtige Informationen in einer Oberfläche kann so aufgrund der Kenntnis der zugrunde liegenden kognitiven Prozesse optimal erfolgen.

Es erscheint offensichtlich, dass eine ausgeprägte Kommunikationskultur innerhalb des Teams zu den Haupterfolgsfaktoren eines Projektes zählt. Speziell eine „gemeinsame Sprache“ zu sprechen wird zur Herausforderung. Hier beginnt die Aufgabe des Softwarearchitekten, der als Verantwortlicher für die Schnittstellen zwischen den einzelnen Disziplinen Kommunikation fördern und eine gemeinsame Sicht etablieren muss.

Besondere Kommunikationsfähigkeiten sind auch gefragt, wenn es um die Basis für eine gute Architektur geht: der maßgebliche Erfolgsfaktor sind die Anforderungen! Die zugrunde liegenden Bedienabläufe aus Sicht der Endbenutzer, genauso wie die z. B. vom Produktmanager gelieferten nicht funktionalen Anforderungen etwa zur Performanz oder Wartbarkeit müssen diskutiert, verstanden und verfeinert werden. Wieder ist die Herausforderung, disziplinübergreifend zu kommunizieren!

Hinzu kommt noch die Tatsache, dass ein Entwicklerteam nur dann effektiv und effizient arbeiten kann, wenn die Zusammenhänge zwischen den ursprünglichen Anforderungen und der sie umsetzenden Architektur vermittelt wurden. Kommunikations- und Motivationsfähigkeiten sind also für die Rolle des Softwarearchitekten unerlässliche Voraussetzungen.

Erfahrungen einbringen

Doch wie trifft und motiviert ein Softwarearchitekt seine Architektur- und Designentscheidungen? Oft erschließen sich Entscheidungen – wie etwa die Auswahl einer bestimmten Technologie – nicht unmittelbar. Hier muss der Softwarearchitekt Erfahrungen einbringen: idealerweise verfügt er über eigene, aber auch fremde Erfahrungswerte in Form von Patterns sind ein wertvoller Teil seines Handwerkszeugs.

Dabei beherrscht er „Pipes and Filters“- oder „Schichten“-Architekturen genauso, wie das Design mit „Model-View-Controller“ oder „Model-View-View-Model“-Patterns! Auch den sinnvollen Einsatz von Best Practices, wie etwa KISS (Keep it simple, stupid!), oder Design-Prinzipien wie das „Open-Close-Prinzip“ motiviert er mühelos anhand konkreter Erfahrungen.

Besonders wertvoll sind natürlich Erfahrungswerte aus dem Bereich der eingebetteten Systeme. Aufgrund deren zunehmender Komplexität nimmt aber das Einbringen von Erfahrungen aus anderen Bereichen – wie zum Beispiel dem Entwurf von Enterprise-Systemen – einen immer höheren Stellenwert ein.

Auch in die Auswahl bzw. das Design geeigneter Hardware muss sich der Softwarearchitekt mit einbringen. Hier kann er Fragen nach z. B. Performanz – brauchen wir Grafikbeschleunigung? – mit konkreten Erfahrungswerten hinterlegen.

Strukturen schaffen

Die Erstellung der Softwarearchitektur muss – wie zuvor bereits dargestellt – unter Einbindung des Teams und aller Stakeholder gelöst werden. Dabei geht es um die Schaffung von Strukturen, die dies unterstützen. So können der Einsatz von Modellen, z. B. auf der Basis von UML, das Beschreiben von Use-Cases, aber auch andere – von ihm festzulegende – Methoden eine gemeinsame Diskussionsbasis schaffen.

Die Auswahl dieser Methoden und der zugehörigen Tools ist Bestandteil seiner Strukturierungsaufgabe. Diese ist in der Regel nicht nur auf die reine Architekturfindung beschränkt, sondern umfasst den kompletten Entwicklungsprozess – über das Design und die Umsetzung bis hin zum Test.

Aspekte wie Coding Conventions, automatische Build- und Testsysteme aber auch die Strukturierung des dislozierten (verteilten) Engineerings und damit die Auswahl einer Kollaborationslösung stellen wesentliche Faktoren für den Erfolg dar.

Offensichtlich sind die Dokumentation, Kommunikation und Motivation der hier gefällten Entscheidungen absolut notwendig. Nur so können diese in Form eines Prozesses umgesetzt und gelebt werden.

Nutzen der Architektur

Beherrschen von Komplexität

Die Entwicklung eingebetteter Software war bisher – geprägt durch den klassischen Zulieferprozess – auf eine Hardware-Topologie ausgerichtet. Eine Familie verwandter Funktionen entsprach vielfach eins-zu-eins einem Steuergerät. Es bestand nur wenig Interaktion zwischen den Funktionen unterschiedlicher Steuergeräte. Als Folge dessen ist der Stand der Technik in der Industrie der Einsatz Hardware-gerichteter Ansätze, die durch Kommunikationstechnologien zwischen Hardware-Komponenten geprägt sind [Bro07].

Bei der Entwicklung eingebetteter Systeme basiert jedoch ein hoher Anteil wettbewerbsrelevanter Innovationen auf Software. Eine wachsende Anzahl softwarebasierter Funktionen bei einer konstanten Anzahl von Steuergeräten eines eingebetteten Systems führt zur stärkeren Interaktion von Funktionen verschiedener Steuergeräte. Durch den hohen Verteilungsgrad der Funktionen wird die Komplexität der Entwicklung erhöht und zugleich die Möglichkeit, unerwünschte Wechselwirkungen zwischen Funktionen zu entdecken, enorm erschwert.

Eine der wesentlichen Aufgaben einer Softwarearchitektur ist daher, Umfang und Komplexität eingebetteter Systeme durch Strukturierung beherrschbar zu machen. Typische Methoden zur Beherrschung eines komplexen Problems sind die hierarchische Zerlegung des Systems in kleinere Teilsysteme einerseits und Abstraktion andererseits.

Qualitäts- und Ressourcenanforderungen

Bei der Entwicklung eingebetteter Softwaresysteme herrscht in der Regel ein sehr hoher Kostendruck. Dies hat zur Folge,

dass kostengünstige Hardware verwendet wird, die sowohl vom Speicherplatz als auch von der Rechengeschwindigkeit her sehr knapp bemessen ist. Da viele der eingebetteten Funktionen jedoch Echtzeitanforderungen erfüllen müssen, stehen sich hier widersprüchliche Anforderungsklassen gegenüber, die jedoch beide erfüllt werden müssen [Bro08].

Funktionale sowie Qualitätsanforderungen (wie z. B. Ressourcenverbrauch oder Reaktionszeiten) so zu präzisieren, dass ein Softwaresystem mit einem angemessenen Kosten-Nutzen-Verhältnis erstellt werden kann, ist eine der wichtigsten Aufgaben im Architektorentwurf.

Wiederverwendbarkeit und Variabilität
Wiederverwendbarkeit und Variabilität sind bei der Entwicklung eingebetteter Softwaresysteme von zentraler Bedeutung. Es gibt einerseits Basisfunktionen, die in allen Produkten einer Systemreihe zu finden sind, andererseits variable Funktionen, die nur in einzelne Produkte eingebaut werden.

Um Entwicklungskosten zu reduzieren, sollte es vermieden werden, dass die Funktionen für jede Variante und jede HW-Plattform vollständig neu entworfen bzw. aufwendig angepasst werden müssen [Bro08].

Zu diesem Zweck unterteilt die Softwarearchitektur das System in mehrere

Bausteine, die in verschiedenen Produktreihen wieder verwendet werden können. Darüber hinaus können einzelne Systemelemente als optional markiert werden, um eine konfigurierbare Architektur für eine ganze Produktlinie zu entwerfen.

Organisation der Entwicklung

Wie bereits erwähnt wird ein System nur selten von einem einzigen Team entwickelt, vielmehr ist die Entwicklung komplexer Systeme durch den klassischen Zulieferprozess geprägt. Die Gliederung des Systems in handhabbare Teilsysteme mit fest definierten Schnittstellen reduziert den Aufwand für die Kommunikation zwischen den Entwicklern.

Diese Zerlegung ist nicht allein aus der funktionalen Anforderungsspezifikation abzuleiten und erfordert weitere Informationen, die in die Softwarearchitektur einfließen (beispielsweise die organisatorische Struktur im Unternehmen, vgl. Conway's Law). Die Architektur bildet die Grundlage für die Organisation der Entwicklung sowie für die Aufgabenverteilung zwischen Auftraggebern und Zulieferern.

Wartbarkeit

Wenn ein Softwaresystem über einen längeren Zeitraum weiter entwickelt oder modifiziert wird, entsteht häufig ein monolithisches Programm mit einem unvollziehbaren Kontrollfluss oder vielen Variablen, auf die unkontrolliert zugegriffen

wird. Der Wartungsaufwand für solche Programme wächst kontinuierlich.

Die Wartbarkeit eines Systems, d. h. die Vermeidung der Entstehung von monolithischen, kaum änderbaren Systemen, ist eines der zentralen Ziele der Softwarearchitektur.

Traceability

Bei der Entwicklung komplexer Systeme ist es üblich, dass sich die Anforderungen sowie die technische Infrastruktur der zu entwickelnden Systeme kontinuierlich ändern. Für die Wartung solcher Systeme ist es wichtig, die Auswirkungen dieser Änderungen rückverfolgen zu können.

Die Softwarearchitektur ist ein zentrales Bindeglied zwischen Anforderungen und Elementen der Implementierung. In einem durchgängigen modellbasierten Entwicklungsprozess werden funktionale Anforderungen den Komponenten der Softwarearchitektur zugeordnet, die wiederum auf Steuergeräte abgebildet werden. Somit entstehen eindeutige Traceability-Links zwischen allen Artefakten des Entwicklungsprozesses.

Fazit

Die zunehmende Komplexität von softwareintensiven Steuerungssystemen erfordert die Erstellung einer tragfähigen Softwarearchitektur. Hierbei können Synergien mit der klassischen Entwicklung von Business-Software genutzt werden, um



Abb.: Softwarearchitektur

Aspekte wie Qualitäts- und Ressourcenanforderungen, Wiederverwendbarkeit und Variabilität sowie Organisation der Entwicklung zu adressieren.

Die Rolle des Softwarearchitekten muss dazu kompetent besetzt werden. Berner & Mattner bildet hierfür eigens Softwarearchitekten aus. Diese werden sowohl in der Produktentwicklung und bei In-House-Projekten als auch in beratender Funktion beim Kunden eingesetzt. ■

Literatur

- [SuT10]** Jörg Schäuffele und Thomas Zurawka. Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen. Vieweg+Teubner Verlag, 2010.
- [HuH11]** Alexander Harhurin und Thorsten Hiebenthal. Sicherheitsnormen als Mittel zur Qualitätssteigerung. In: ETR – Eisenbahntechnische Rundschau, Oktober 2011.
- [Bro07]** Manfred Broy, Ingolf H. Krüger, Alexander Pretschner und Christian Salzmann. Software Engineering for Automotive Systems. In: FOSE'07, IEEE, 95(2):356-373, 2007.
- [Bro08]** Manfred Broy, Martin Feilkas, Johannes Grünbauer, Alexander Gruler, Alexander Harhurin, Judith Hartmann, Birgit Penzenstadler, Bernhard Schätz und Doris Wild. Umfassendes Architekturmodell für das Engineering eingebetteter Software-intensiver Systeme. Technischer Bericht, TU-München, 2008.
- [Bäc11]** Jörg Bächtiger. Die Macht, die uns umgibt – Design Principles. Schneller und besser Software entwickeln, SEACON Kongress 2011.
- [Siw08] Peter Siwon. Denkanstöße für den Projekterfolg: Die menschliche Seite. Vogel Business Media; 1. Auflage (Juli 2008).
- [BuH11]** Holger Breitling, Ralf Hofmann. Wege aus der Identitätskrise: Die Rolle des Software-Architekten zwischen Unternehmens- und Anwendungsarchitektur, SEACON Kongress 2011.