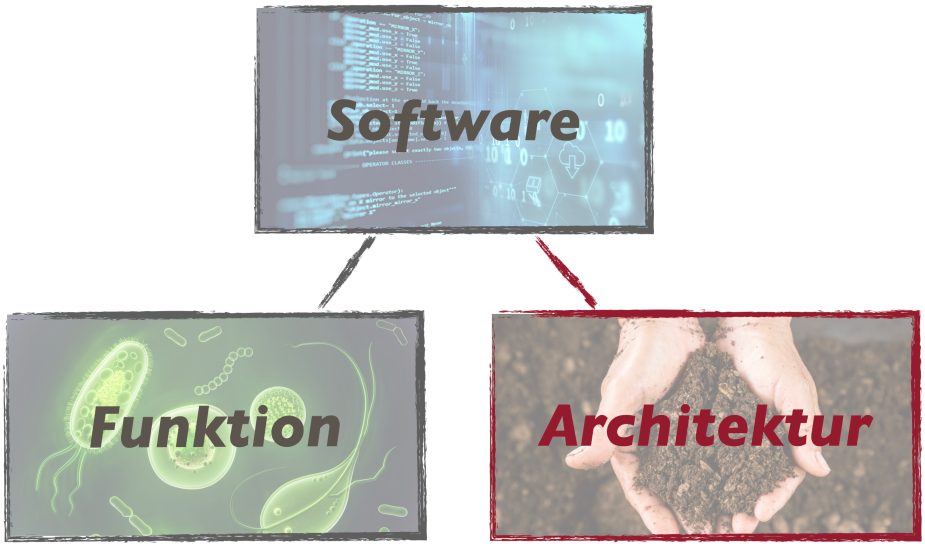
Systems und Software Architektur I

Voraussetzung für effizientes Engineering komplexer Systeme

Embedded Software Engineering Report Nr. 43



Funktion verhält sich zur Architektur wie Mikrobe zu Milieu

Nach *Robert C. Martin*[1] kann Software unterteilt werden in Funktion und Architektur. Aber was genau ist Architektur? Oder anders gefragt, was soll Architektur bewirken?

Die Aufteilung des Gesamtsystems in Komponenten mit dem Ziel der Wiederverwendung ist ein häufiger Aspekt.

Komponenten sind immer eingebettet in ein Gesamtsystem und damit in einen Kontext. Innerhalb dieses Kontextes müssen die Komponenten mit Betriebsmitteln versorgt werden, wie Energie, mechanischer Stabilität, aber auch Wärmeableitung.

Die Bereitstellung dieser Ressourcen ist ein weiteres wichtiges Merkmal der Architektur. Bezogen auf Software ermöglicht die Architektur auch die Bereitstellung von Ressourcen, wie Rechenleistung und Speicher. Der Scheduling Mechanismus sorgt dafür, dass jede Komponente *(in diesem Fall ein Task)* ausreichend Laufzeit von der CPU zugewiesen bekommt, eine Voraussetzung für zeitlich deterministisches Verhalten der darin enthaltenen Funktion.

Die Absicherung einer ausreichenden Energieversorgung, Ableitung von Wärme, Anordnung und mechanische Stabilität einzelner Komponenten im Gesamtsystem … das sind die Aufgaben, die die Architektur unter anderem auch für das richtige Funktionieren des Gesamtsystems gewährleisten muss.

In diesem Sinn habe ich die These aufgestellt *‚Funktion verhält sich zur Architektur wie Mikrobe zu Milieu‘,* die ich in dieser Ausgabe des ESER näher erläutern werde.

SODIUS WILLERT

Inhalt:

Architektur ist Struktur

Architektur entsteht durch hierarchische Dekomposition

Funktionale, logische und physikalische Architekturen

Was kosten verschiedene Sichten?

So wenig unterschiedliche Architektur-Sichten wie möglich, so viel wie notwendig.

Konzept Architektur

Komplexität kann nicht eliminiert werden

Entwurfsprinzipien und Muster

Freedom of interference

Architektur ist Milieu

Disclamer

Das Thema Architektur erscheint mir hochgradig komplex, und ich vereinfache meine Ausführungen im folgenden, indem ich die Sicht auf die Spezifikation von Systemen reduziere. Alle folgenden Betrachtungen gehen davon aus, dass eine anwenderspezifische, möglichst lösungsfreie Sicht definiert und analysiert wurde, aus der dann Spezifikationen abgeleitet werden. Wenn im Folgenden von funktionaler Architektur und funktionalen Anforderungen gesprochen wird, liegt die Sicht hauptsächlich auf der System-Spezifikation. Insbesondere in dem Beispiel zu Abbildung Nr. 7 wird vorausgesetzt, dass bereits in der funktionalen Architektur Kenntnisse über die technischen Lösungen der logischen Architektur bekannt sind.

Software besteht aus Funktion und Architektur

Nach *Robert C. Martin*[1] besteht Software aus Funktion und Architektur. Ich behaupte, das gleiche gilt auch für Systeme. Funktion *(im Sinn von allgemeiner Funktionalität und nicht im Sinn einer expliziten Funktion)* und Architektur sind zwei unterschiedliche Aspekte oder Gesichtspunkte eines Systems oder einer Software. Dem logischen Umkehrschluss zufolge beinhaltet Architektur an sich keine Funktion *(bezogen auf die funktionalen Anforderungen des Systems / der Software)*. Dieser Gedanke liegt den folgenden Überlegungen und Ausführungen zugrunde.

Das beantwortet zunächst die Frage, was Architektur nicht ist: Architektur ist nicht Funktion. Es bleibt die Frage, was Architektur dann ist und was der Sinn von Architektur ist, wenn sie doch keine Funktion liefert. Ich befürchte, es gibt keine klare und schlüssige Antwort. Hier ist nun mein Versuch einer Definition, auf der die folgenden Betrachtungen fußen, ohne Anspruch auf absolute Richtigkeit. Besser gesagt, es ist nicht der Versuch einer weiteren Definition, denn davon gibt es unzählige, sondern der Versuch, eine möglichst allgemein akzeptierte Definition zu finden.

Das sagt die ISO/IEC/IEEE 42010

> 3.2 ***architecture***
>
> ⟨*system*⟩ *fundamental concepts or properties of a system in its environment embodied in* ***its elements***, ***relationships***, *and in the* ***principles of its design*** *and evolution.*

Demzufolge ist *‚Architektur'* die grundlegende Organisation eines Systems auf Basis von *‚Komponenten und deren Beziehungen'* und den *‚Grundsätzen für seinen Entwurf'*. Das führt mich über die Unterteilung von Funktion und Architektur zu einer weiteren Untergliederung der Architektur an sich, die ich im folgenden aufgreifen und erläutern möchte.

1. ***‚embodied in its elements, relationships'*** *also* zum einen die Komponenten und deren Beziehungen, die sich aus funktionalen Aspekten ableiten *(wenn ein Fahrzeug funktional ‚bremsen' muss ist es naheliegend, daraus eine technische Komponente ‚Bremssystem' abzuleiten)*. Im folgenden sehe ich diesen Aspekt als funktionale Architektur, weil sich deren Design tendenziell aus funktionalen Anforderungen ableitet.
2. ***‚principles of its design'*** und zum anderen Prinzipien, denen beim Entwurf gefolgt werden, und aus denen sich eine so genannte *‚Konzept Architektur'* ableitet.

Der zweite Punkt kommt mir persönlich häufig zu kurz, und aus meiner Erfahrung ist das der sehr viel wichtigere Gesichtspunkt einer Architektur, wenn es um die Erfüllung von nichtfunktionalen Anforderungen und/ oder Constraints geht.

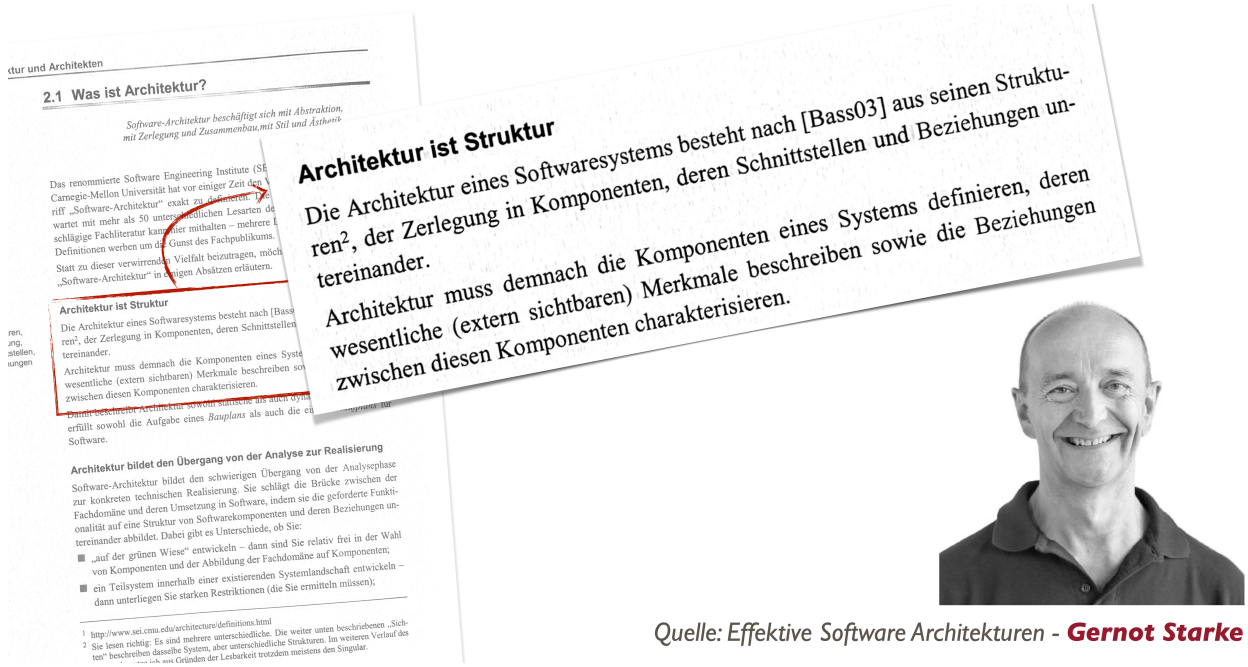
Im Verlauf dieses ESER werde ich als erstes den Blick auf die funktionale Architektur richten und das Basis-Prinzip ‚Hierarchische Dekomposition', mit dem die Architektur designt wird. Ich werde erläutern, dass dieses Prinzip beim Design von komplexen Systemen an seine Grenzen stößt und dass wir weitere Sichten / Konzepte wie logische und physikalische Architekturen benötigen.

Das Basis-Prinzip ‚Teile und Herrsche', auf dem hierarchische Dekomposition fußt, hat nicht nur Vorteile. Es gibt auch Nachteile, die wir betrachten, und ich werde Lösungen zeigen, diese zu umschiffen.

Das führt uns dann zur ‚Konzept Architektur', die ich noch anreißen, aber dann in einer weiteren Ausgabe des ESER vertiefen werde.

[1] *Quelle: Clean Architecture - Robert C. Martin - Initiierte die Entwicklung des Agilen Manifests und ist führendes Mitglied der Bewegung Software Craftsmanship*

Architektur ist Struktur



Architektur ist Struktur

Die Architektur eines Softwaresystems besteht nach [Bass03] aus seinen Strukturen[2], der Zerlegung in Komponenten, deren Schnittstellen und Beziehungen untereinander.

Architektur muss demnach die Komponenten eines Systems definieren, deren wesentliche (extern sichtbaren) Merkmale beschreiben sowie die Beziehungen zwischen diesen Komponenten charakterisieren.

Quelle: Effektive Software Architekturen - **Gernot Starke**

Wenn Systeme und/oder Software unterteilt werden können in Funktion und Architektur, was genau ist dann Architektur?

Gernot Starke beschreibt Architektur als Zerlegung in Komponenten, Definition, deren Merkmale *(Komponentenspezifikation)* und Schnittstellen.

Zusammen mit der Aussage von Robert C. Martin, dass Architektur nicht Funktion ist, können die Komponenten wie leere Container angesehen werden (Struktur), für die jeweils spezifiziert ist, welche Funktion diese Komponente beinhalten sollen. In der Komponente ist dann die realisierte Funktion eingebettet, die der Spezifikation entspricht.

Mit dieser Gliederung des Gesamtsystems in Komponenten sollen folgende Ziele erreicht werden:

- Besseres Verständnis des Systems
- Planung und Verteilung von Aufgaben an verschiedene Teams
- Zuordnung von Verantwortlichkeiten
- Einkauf und/oder Unterbeauftragung von Teilen des Systems
- Verteilung von Funktion auf physikalische, elektronische oder digitale Domänen
- Modularisierung und Wiederverwendung
- Aufteilung bzw. Zuordnung von Betriebsressourcen (Laufzeit, Speicher …)

Architektur schafft Struktur auf Basis des Mechanismus Teile & Herrsche oder anders ausgedrückt *‚Hierarchischer Dekomposition'*. Das ist ein Aspekt der Architektur, der sich überwiegend aus funktionalen Gesichtspunkten ableitet, weshalb ich ihn funktionale Architektur nenne.

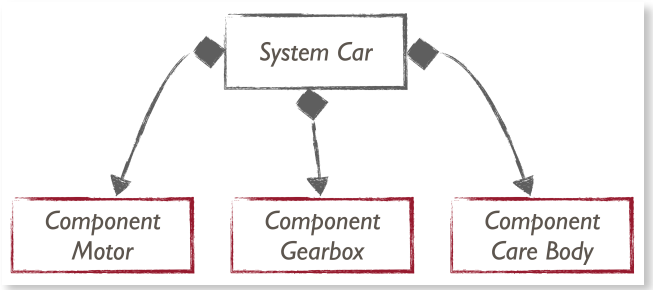


Abbildung Nr. 1: Zerteilung des Systems in Komponenten

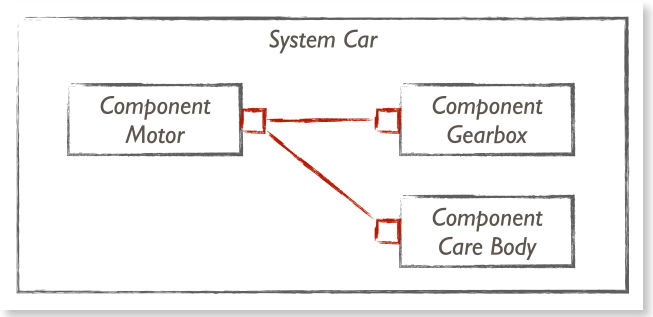


Abbildung Nr. 2: Spezifikation der Schnittstellen

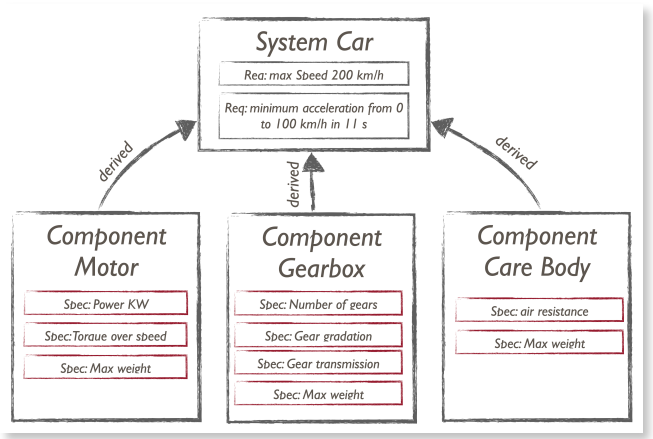


Abbildung Nr. 3: Spezifikation der Komponenten

Architektur entsteht durch hierarchische Dekomposition

In den letzten Jahren stelle ich auf Veranstaltungen oder bei Kundenterminen immer wieder die Frage, welche Phase im Engineering die problematischste darstellt. Üblicherweise bekomme ich die Antwort, dass es die Integrations-Phase ist.

Die architekturbedingte Aufteilung in Komponenten führt dazu, dass diese Komponenten später im Entwicklungsprozess wieder zusammengeführt werden müssen. Auf den ersten Blick klingt das nicht aufregend. In der Praxis scheint jedoch gerade im Zusammenfügen, also in der Integration *(auch Aggregation)* der einzelnen Komponenten zu einem Gesamtsystem die größte Herausforderung zu liegen.

Um besser zu verstehen, worin die Herausforderung der Aggregation heutiger komplexer Systeme liegt, möchte ich einen Vergleich nutzen.

Was unterscheidet die Fahrfunktion *‚Lenken'* eines 70er Jahre PKW von der äquivalenten Funktion in einem aktuellen Premium Fahrzeug?

Zum Beispiel wird das Lenken an sich heute durch *‚Servolenkung'* erleichtert. Die Grenzgeschwindigkeit zum Durchfahren einer Kurve hat sich deutlich erhöht, da in kritischen Situationen die Innenräder verzögert *(abgebremst)* werden, um damit das Fahrzeug sprichwörtlich in die Kurve zu ziehen. Das Fahrwerk wird an der Kurven-äußeren Seite gestrafft.

In Abbildung Nr . 4 sehen Sie den Versuch, beide Systeme einander gegenüber zu stellen.

Technisch betrachtet lässt sich die Fahrfunktion Lenken der 70er Jahre PKWs auf Basis eines Lenksystems abbilden, das hierarchisch dekomponiert ist. Was bedeutet das? Jede Komponente des Lenksystems dient ausschließlich der Funktion Lenken und die Fahrfunktion Lenken kann vollständig durch die Komponenten des Lenksystems abgebildet werden.

Werfen wir einen Blick auf die Fahrfunktion Lenken in 2020, dann haben wir ein so genanntes *‚Orthogonal vernetztes System'*. Was bedeutet das? Auf der funktionalen Ebene gibt es verschiedene Fahrfunktionen. Lenken, Bremsen, Licht … Diese sind orthogonal zueinander *(haben rein funktional betrachtet keine gegenseitigen Abhängigkeiten)*.

Auf der technischen Systemebene gibt es evtl. korrespondierende Systemkomponenten wie Lenksystem oder Bremssystem. Diese lassen sich aber nicht *(wie in den technischen Systemen der 70er Jahre)* aus den Fahrfunktionen hierarchisch dekomponieren. Sie liefern Teilfunktionen, aus denen sich übergeordnete komplexe Fahrfunktionen zusammensetzen und sich dazu aus unterschiedlichen Systemkomponenten bedienen und umgekehrt.

1970 hierarchisch dekomponiert

Komponenten dienen der Lenkfunktion und diese kann ausschließlich durch diese Komponenten gebildet werden.



2020 orthogonal vernetzt

Komponenten dienen mehreren Funktionen und diese zieht sich durch verschiedenste Komponenten.

Lenksystem	Lenkwinkel an Vorderrädern einstellen
Bremssystem	Innenräder verzögern
Lichtsystem	Kurvenlicht
Sitz	Sitzwangen ausfahren *(Fahrer stabilisieren)*
Fahrwerk	Fahrwerk seitlich straffen *(Neigung reduzieren)*
Hinterachse	Hinterachslenkung
…	

Abbildung Nr. 4: Vergleich Lenkfunktion PKW 1970er zu 2020er Jahre

Das System Bremsen *(technische Sicht/Ebene)* ist auch beteiligt an der Lenkfunktion *(funktionale Sicht/Ebene)* ebenso das System Fahrwerk, Licht, Sitz. Zumindest, wenn technische Funktionen wiederverwendet werden sollen, was praktisch immer der Fall ist.

Es ist also nicht mehr möglich, die Fahrfunktionen hierarchisch in Systemkomponenten zu dekomponieren. Aber wie gehen wir damit um?

Funktionale, logische, physikalische Architekturen

Bisher haben wir unser System / Software in die Anteile *‚Funktion'* und *‚Architektur'* aufgeteilt, die Architektur hinsichtlich funktionaler Aspekte betrachtet und ihr den Namen *‚Funktionale Architektur'* gegeben. Diese Sicht möchte ich nun noch einmal erweitern, um eine Logische und Physikalische Architektur, wie in Abbildung Nr. 5 dargestellt.

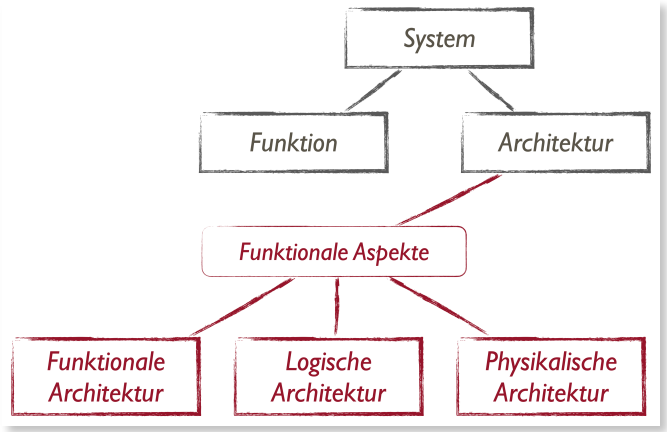


Abbildung Nr. 5: Funktionale- Logische- Physikalische Architektur

Der Trick liegt darin, dass jede dieser Sichten auf die Architektur orthogonal zu den jeweiligen anderen Sichten ist. Bezogen auf eine Sicht ist hierarchische Dekomposition möglich, je nach Gesichtspunkt funktional, logisch oder physikalisch. *(In Architekturen von sehr großen, komplexen Systemen kann es noch weitere Sichten geben)*

In jeder Sicht ergeben sich andere Teilfunktionen, die bezogen auf die jeweilige Sicht unabhängig voneinander sind.

In der funktionalen Sicht, zum Beispiel die vollständige Abbildung der Fahrfunktion Lenken. Diese beinhaltet evtl. auch anteilig Bremsaktivitäten oder Kurvenlicht … die alle in dem Gesichtspunkt Lenken beinhaltet sind.

In der logischen Sicht gibt es nun technische Systeme, wie zum Beispiel je ein Lenk- Brems- und Lichtsystem.

Auch wenn die Namensgebung in diesem Fall gleich ist, sind die Anteile sehr unterschiedlich. Wird die Fahrfunktion Lenken auf die Systemkomponenten der logischen Architektur abgebildet, dann bedient sich die Lenkfunktion neben Systemfunktionen der Systemkomponente *‚Lenksystem'* auch bei Systemfunktionen aus *‚Bremssystem'* oder *‚Lichtsystem',* um funktional vollständig abgebildet zu werden.

Eine Fahrfunktion kann sich uneingeschränkt aus allen Systemkomponenten bedienen (Orthogonal), um die Funktionalität abzubilden. Darin liegt eine grundlegende Voraussetzung zur strukturierten Wiederverwendung von Funktionalität.

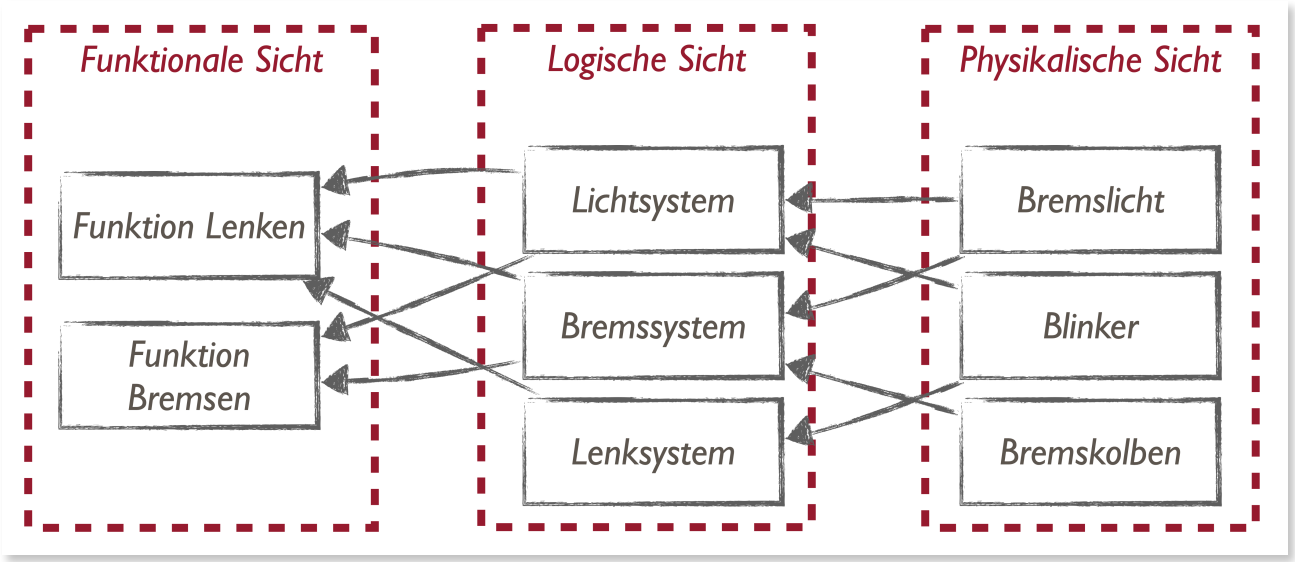


Abbildung Nr. 6: Orthogonal verlinkte Sicht auf Architekturen

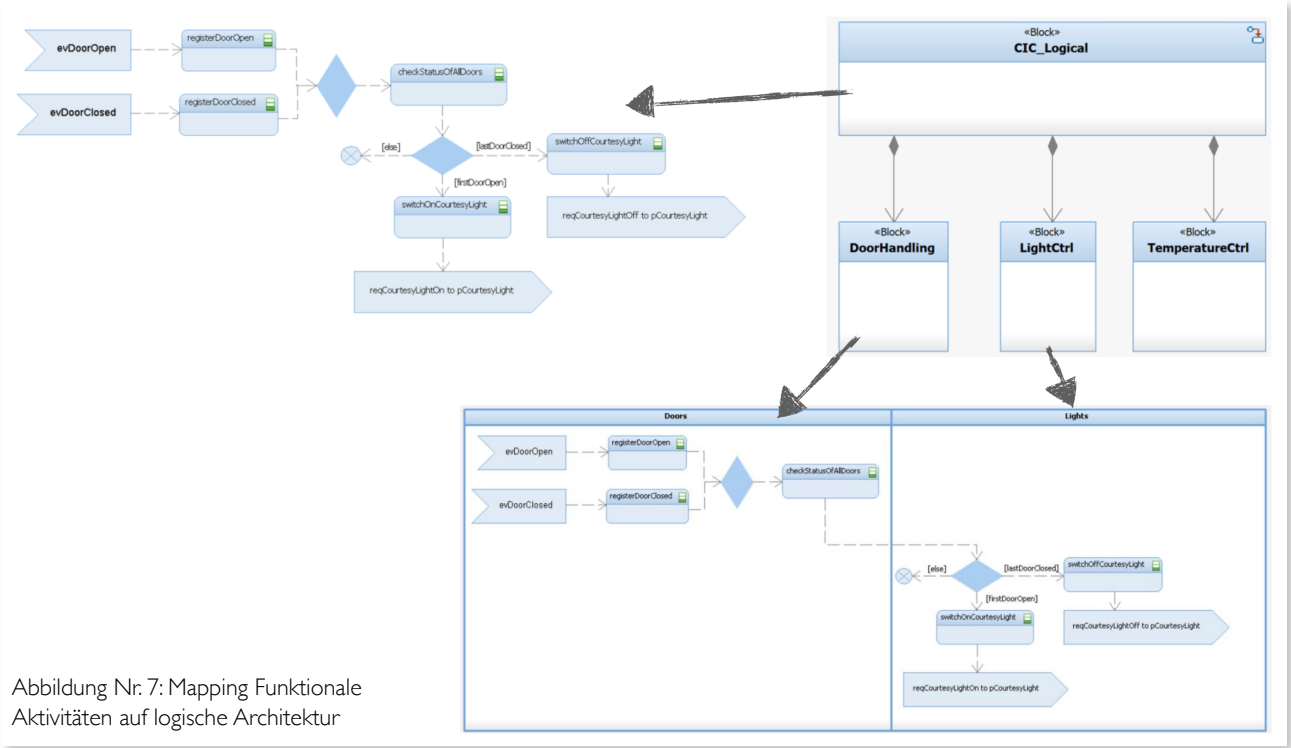


Abbildung Nr. 7: Mapping Funktionale Aktivitäten auf logische Architektur

Das gleiche gilt für die physikalische Architektur. Dort gibt es zum Beispiel das Bremslicht. Von der funktionalen über die logische Ebene wird dann abgebildet, dass das Bremslicht nicht leuchtet, wenn aus dem Kontext der Fahrfunktion Lenken eine Bremsfunktion genutzt wird. Oder dass bei einer Notbremsung über die Warnblinkfunktion alle Blinker angesteuert werden. *(Kopplung oder Vernetzung des Bremssystems mit dem Lichtsystem über eine funktionale Fahrsituation)*

In der Abbildung Nr. 6 sind die verschiedenen Architekturen mit möglichen Funktionen/Komponenten dargestellt.

Mit den verschiedenen Architektursichten haben wir nun eine Möglichkeit gefunden, orthogonal vernetzte Systeme darzustellen. Aber wie hilft uns das praktisch weiter?

Eine Darstellung wie in Abb. Nr. 6 zeigt die verschiedenen Architekturen und mögliche Abhängigkeiten, aber ein Link von Lenkfunktion zu Bremssystem erklärt sich nur dann, wenn die Kenntnis des genauen Lenkszenarios bekannt ist. Wüsste ich nicht, dass beim Einlenken evtl. die Innenräder abgebremst werden, dann würde der Link nicht zum Verständnis des Systems beitragen.

Nun kommt modellgetriebenes Engineering ins Spiel. Folgend stelle ich exemplarisch eine Möglichkeit dar, wie auf Basis eines Aktivitätsdiagramms die funktionale Sicht auf die logische Sicht abgebildet werden kann und so das volle Verständnis auch auf der logischen Sicht ermöglicht wird.

In der Abbildung Nr. 7 sehen wir ein Aktivitätsdiagramm, das eine Fahrfunktion in Bezug auf das Gesamtsystem beschreibt. Das Gesamtsystem ist in der logischen Architektur aufgeteilt in die Komponenten DoorHandling, LightCtrl und TemperatureCtrl.

Auf Basis eines Aktivitätsdiagrammes kann die Fahrfunktion auf die logische Architektur abgebildet werden, indem das Aktivitätsdiagramm für die involvierten logischen Komponenten je eine sogenannte *‚Swimlane'* bekommt. Nun können die Aktivitäten *(entsprechend der logischen Architektur)* den Swimlanes zugeordnet werden.

Das gleiche Vorgehen wird mit weiteren funktionalen Aktivitätsdiagrammen angewandt. So summieren sich in verschiedenen Diagrammen immer mehr Aktivitäten in den Swimlanes der Systemkomponenten der logischen Architektur. Diese sind orthogonal zur funktionalen Architektur. Das bedeutet, eine Aktivität, die aus funktionaler Sicht zu Lenken gehört liegt aus logischer Sicht eventuell in der Komponente Bremssystem *(abbremsen der Innenräder)*.

Die Aufteilung in verschiedene Architekturen ist ein sehr mächtiger Mechanismus, um komplexe Systeme verstehbar abzubilden. Er ermöglicht in Bezug auf jede Architektur, weiterhin das hilfreiche Prinzip *‚Hierarchische Dekomposition'* anzuwenden. Es bildet über die erhöhte Verstehbarkeit hinaus die Grundlage für systematische Wiederverwendung von Systemfunktionen.

Was kosten verschiedene Sichten?

Nichts im Leben ist kostenlos und auch die Einführung jeder weiteren Architektur beinhaltet neben neuen Aspekten redundante Anteile. Die Architektur muss erstellt und gepflegt werden, das bedeutet Aufwand.

Der Nutzen geht schnell verloren, wenn dieser Aufwand zur Pflege der Architektursichten nicht geleistet werden kann und diese inkongruent werden. Dann sind ihre Aussagen nicht mehr verlässlich und können sogar zu Fehlern führen. Aus diesem Grund sollte die Leitlinie immer sein:

So wenig unterschiedliche Architektursichten wie möglich, so viel wie notwendig, um ein ausreichendes Systemverständnis und systematische Wiederverwendung zu gewährleisten.

Folgt man einem klar definierten Prozess, zum Beispiel *‚HarmonyMBE‘*, dann können notwendige Arbeitsschritte, um die Architekturen kongruent zu halten, teilweise automatisiert werden. HarmonyMBE beinhaltet Wizards zur automatisierten Übertragung von Änderungen. Um diese Wizards anzuwenden, muss jedoch sehr präzise modelliert werden, was einen hinreichenden Reifegrad in Bezug auf den Prozess und die Modellierungsdisziplin voraussetzt.

Konzept Architektur

Nachdem wir bisher die Architektur eher aus der Sicht der Funktion betrachtet haben, werde ich den Blickwinkel nun noch einmal erweitern auf die so genannten Constraints und nichtfunktionalen Aspekte, auch QoS *(Quality of Services)*.

Dazu kommt eine weitere Architektur ins Spiel, die ich *‚Konzept Architektur‘* nenne.
Siehe Abbildung Nr. 8.

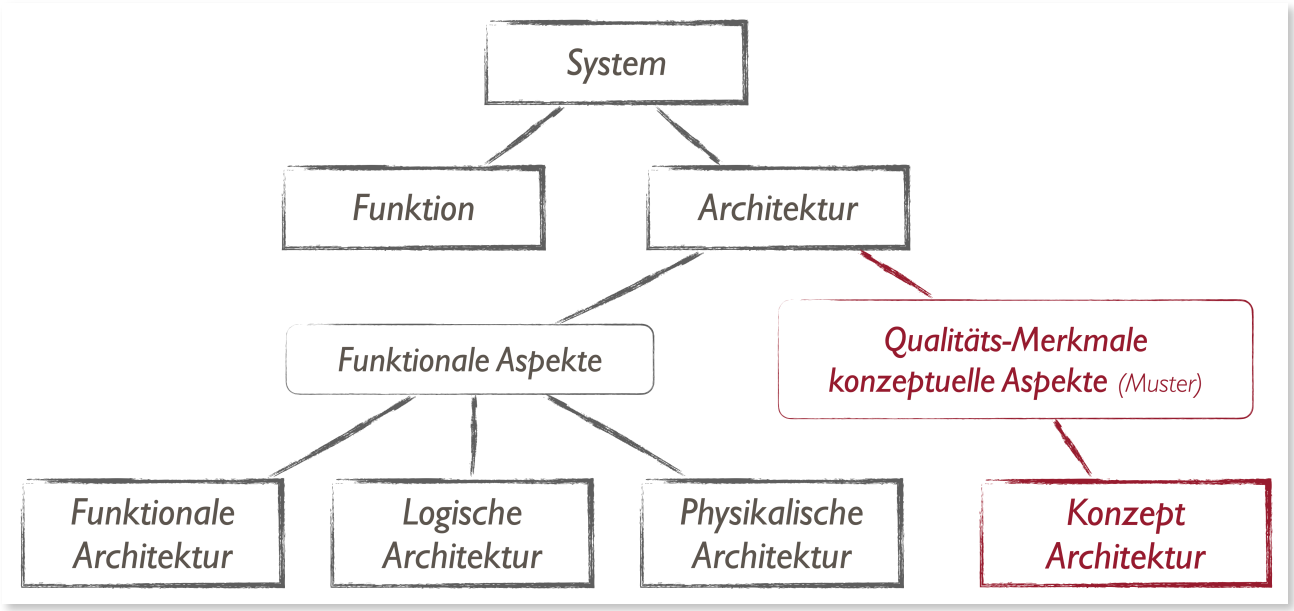


Abbildung Nr. 8: Konzept Architektur

Die Trennung von funktionalen und nichtfunktionalen Anforderungen *(NFA)* in verschiedenen Architektur-Sichten ist nicht unbedingt üblich und ich verlasse hier den Stand der Technik bzw. Stand der Wissenschaft. Die Bezeichnung *‚Konzept Architektur‘* ist von mir eingeführt und ich konnte ihn in der von mir verwendeten Bedeutung in der einschlägigen Literatur nicht wieder finden. Ich halte deren Einführung aber für hilfreich oder sogar notwendig, um den Gesichtspunkt der Contraints und NFA nicht nur zu adressieren, sondern auch explizit im Design einer Architektur abzusichern.

Komplexität kann nicht eliminiert werden

Um die Qualitätsaspekte zu betrachten, möchte ich noch einmal auf das Prinzip *Teile und Herrsche* zurückkommen. Es gibt noch einen Aspekt bei dessen Anwendung, dem häufig keine Beachtung geschenkt wird.

Wird ein System in Komponenten zerlegt, ist einer der elementaren Gesichtspunkte die Verringerung der Komplexität einzelner Komponenten gegenüber dem Gesamtsystem und damit verbunden deren bessere Handhabung. Streng genommen kann jedoch Komplexität, bei gleichzeitiger Beibehaltung der vollständigen Funktion, nicht wirklich reduziert werden.

Bezogen auf eine einzelne Komponente verringert sich nachweislich die Komplexität gegenüber dem Gesamtsystem. Aber bezogen auf das Gesamtsystem verschieben wir mit jeder Teilung Komplexität auf die Schnittstellen Ebene. Siehe Abbildung Nr. 9.

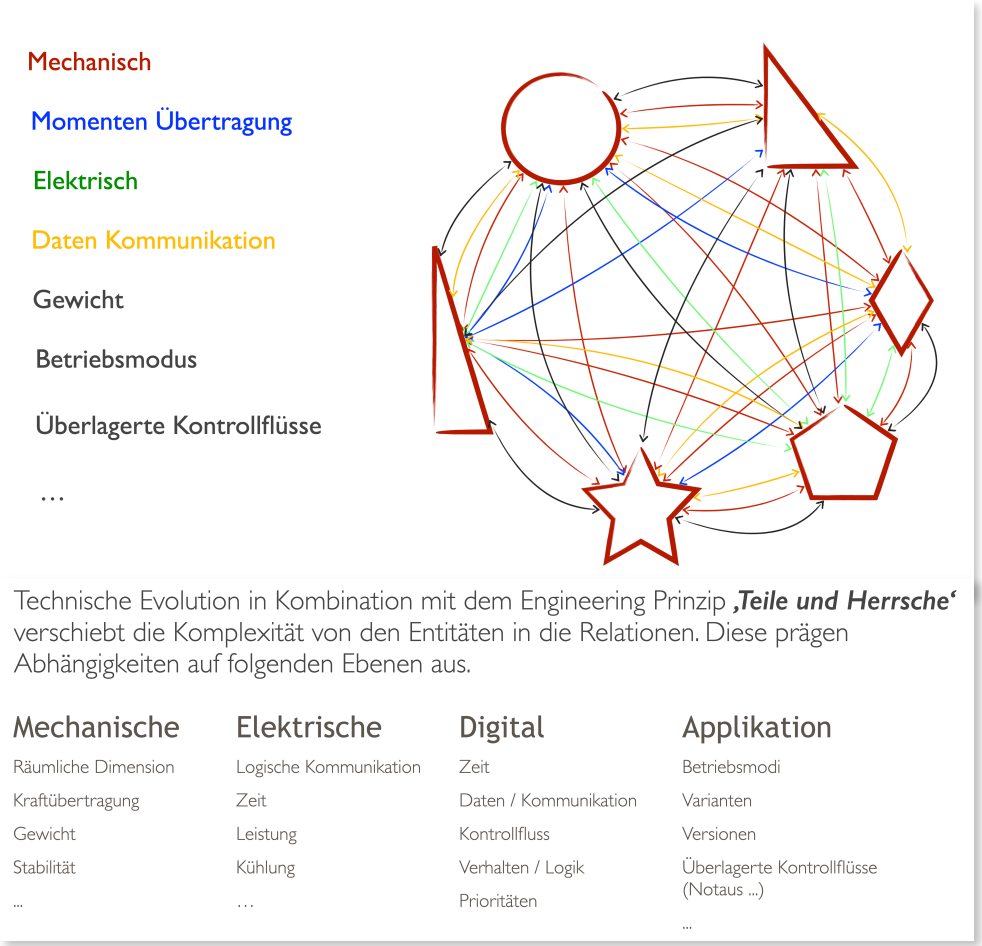


Abbildung Nr. 9: Komplexität auf Beziehungsebene

Wenn wir das Teilen auf die Spitze treiben würden, dann wäre die inhärente Komplexität eines jeden Elementes *(wir würden wahrscheinlich nicht mehr von einer Komponente sprechen)* infinitesimal klein und wir hätten die Komplexität vollständig auf die Beziehungsebene verschoben.

Im Engineering komplexer Systeme macht sich dieser Effekt heute zunehmend bemerkbar. In Automotive Projekten, in denen ich involviert bin, höre ich häufig, die Steuergeräte an sich funktionieren, aber in deren Zusammenspiel ergeben sich Probleme.

Freedom of Interference

Wenn sich die Komplexität zunehmend auf den Schnittstellen ausprägt, dann steigt die Wahrscheinlichkeit von so genannten *‚Hidden Links'*, was in der korrespondierenden Komponente zu sogenannter *‚Emergenz' (nicht definiertem Verhalten)* führt und das wiederum zu *‚Dysfunktion'*.

Um das zu verhindern, wird die Forderung nach *‚Freedom of interference' (Rückwirkungsfreiheit)* lauter. In der ISO 26262 wird sie hinsichtlich Programmfluss, Speicherzugriff und Kommunikation explizit gefordert.

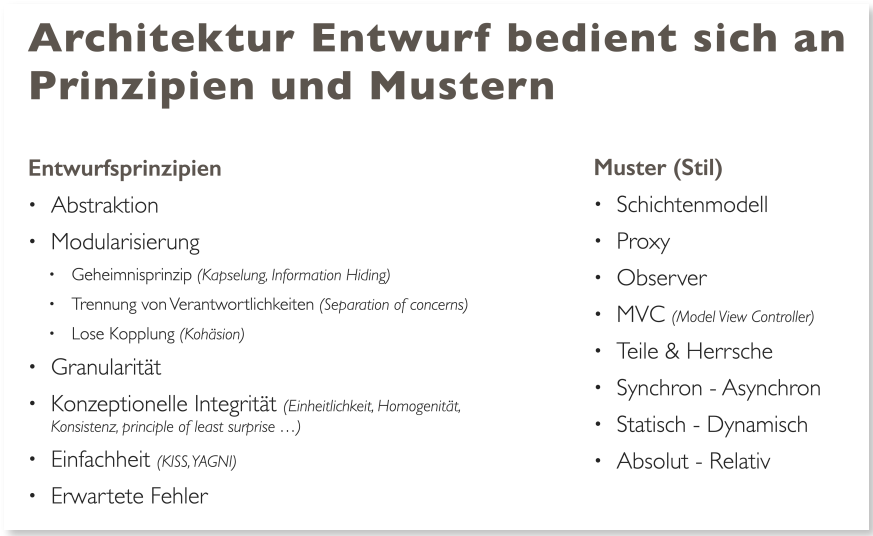


Abbildung Nr. 10: Entwurfsprinzipien und Muster

Aber wie erreicht man diese Rückwirkungsfreiheit?

Nun sind wir bei Entwurfsprinzipien und Architektur Mustern angekommen. Es gibt Entwurfsprinzipien und darauf aufbauende Muster, mit denen Qualitätsaspekte explizit adressiert werden.

In Abbildung Nr. 10 sind Beispiele für Entwurfsprinzipien und Architektur Muster aufgeführt.

Ich gehe an dieser Stelle nicht tiefer auf die Prinzipien und Muster ein und verweise auf den ESE-Report Nr. 31, *‚Architektur und Design'* und die einschlägige Fachliteratur.

www.sodiuswillert.com/de/ese-reports/software-architektur-und-design-2

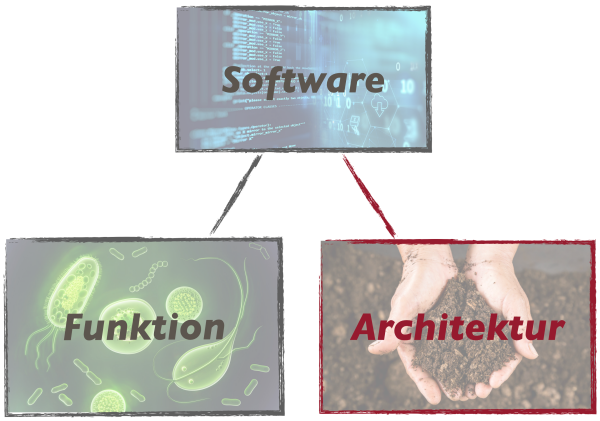
Architektur ist Milieu

Vielmehr stelle ich hier diese These auf:

Funktion verhält sich zu Architektur wie Mikrobe zu Milieu.

Damit meine ich, dass *‚Funktion' (Mikrobe)* in die *‚Architektur' (Milieu)* eingebettet ist. Wird aus Sicht der Architektur auf Basis von hierarchischer Dekomposition eine Komponente definiert, kann diese auch gesehen werden wie ein Container, in dem Funktion bereitgestellt wird.

Architektur muss nun über die schlichte Aufteilung in Komponenten (Funktionale Sicht) in so weit hinaus gehen, dass den Komponenten auch die Ressourcen bereitgestellt werden, die gewährleisten, dass sich die darin enthaltene Funktion ausprägen kann.



Das sind zum Beispiel aus Software Sicht: Speicher, Laufzeit, Daten. In anderen Engineering Disziplinen können das sein: Spannungsversorgung, statische Stabilität, Wärmeableitung …

Es gibt Konzepte für obige Aspekte. Zum Beispiel Hutschienensysteme zur flexiblen mechanischen Befestigung von Komponenten. Diese können mit der Energieversorgung und in diesem Kontext mit Überlastungsschutz, Kurzschlussschutz kombiniert werden. Die Architektur gewährleistet also, dass Komponenten mit allen wichtigen Ressourcen versorgt werden, die notwendig sind, damit die darin enthaltene Funktion wirksam bereitgestellt werden kann. Und das unter Berücksichtigung von Qualitätsaspekten wie Wartbarkeit, Austauschbarkeit, Wiederverwendbarkeit.

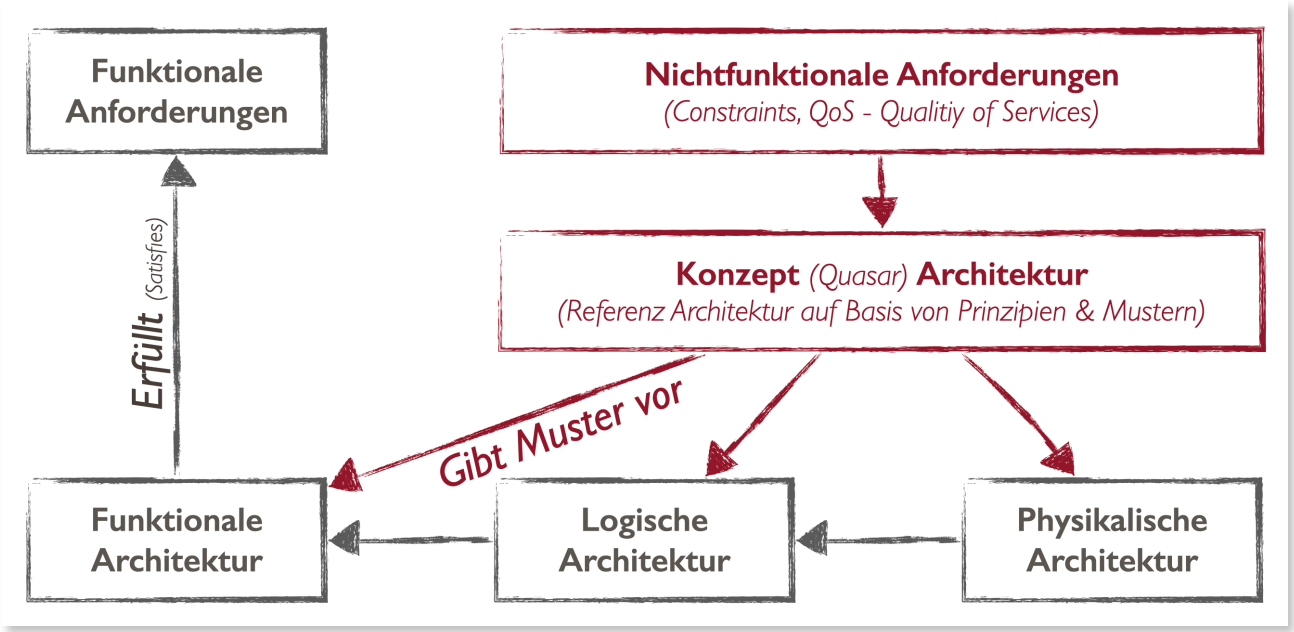


Abbildung Nr. 11: Zusammenwirken der unterschiedlichen Architekturen / Architektur Sichten

Im Kontext von Embedded Software könnte das ein Echtzeit Betriebssystem sein, das auf Basis eines Scheduling Mechanismus den Komponenten *(in diesem Kontext dem Task, Thread oder Prozess)* die notwendige Rechenleistung und Speicher bereit stellt, um eine deterministische Funktion zu gewährleisten.

Wir kommen nun zu einer weiteren Sicht auf die Architektur, wie in Abbildung Nr. 11 dargestellt.

Wir haben einerseits die funktionalen Gesichtspunkte *(Funktionale Anforderungen),* die über die funktionale, logische und physikalische Architektur adressiert werden. Andererseits haben wir die Qualitäts-Gesichtspunkte *(Constraints, Nichtfunktionale Anforderungen, QoS),* die über die Konzeptarchitektur explizit adressiert werden.

Die Konzeptarchitektur definiert Entwurfsmuster, die den Entwurfsprinzipien folgen, um die Qualitätsaspekte absichern.

Das eigentliche Design und die funktionale, logische und physikalische Architektur verwenden die in der Konzeptarchitektur vorgegebenen Entwurfsmuster bzw. folgen diesen. Dadurch werden Qualitätsziele erreicht und/oder abgesichert.

Wie eine Konzept Architektur für Embedded Software konkret aussehen kann, werde ich im nächsten ESE-Report genauer vorstellen.

Zum Abschluss

Mit diesem ESE Report habe ich *(vermutlich)* nicht nur für mich persönlich Neuland betreten. Stundenlange Internetrecherche und Wälzen der einschlägigen Fachliteratur haben mir viele Erkenntnisse gebracht, aber es sind gefühlt noch mehr Fragen offen geblieben und vor allem Unsicherheit. Unsicherheit ob ich die richtigen Begriffe richtig anwende. Ob ich in den Konzepten Aspekte nicht berücksichtigt habe … Wie das alles praktisch anwendbar ist …

Bis heute habe ich nicht herausgefunden, wie die verschiedenen Architekturen *(Funktional, Logisch, Physikalisch)* bezeichnet werden. Sind es Architekturen? Sichten darauf? Aspekte?

Den von mir eingeführten Begriff *‚Konzept Architektur'* konnte ich an einigen wenigen Stellen im Internet wieder finden. Aber nicht eindeutig verwendet. Auch die Begriffe *‚Quasar Architektur'*, *‚technische Architektur'* … scheinen etwas ähnliches zu adressieren.

Dieser ESER ist ***nicht*** der Weisheit letzter Schluss. Eher sehe ich ihn als eine Basis zur weiteren Diskussion.

Mein Interesse ist groß, die von mir dargestellten Aspekte und Sichtweisen z.B. in einer Webkonferenz zu diskutieren. Wer auch daran interessiert ist, schreibt mir bitte eine kurze Mail an awillert@sodiuswillert.com . Ich organisiere dann einen Termin.

Es ist mir ein großes Anliegen, dieses Thema gemeinsam praxisnah voran zu bringen.



Mehr zum Thema ‚Systems & SW Architektur'

- **Architekturentwurf für Embedded Systeme**
  Andreas Willert / Webinar on demand / in deutsch
  https://www.youtube.com/watch?v=_e_rNiWUyH4

- **Elaborating Functional and Logical Architectures with HarmonyMBE**
  Andy Lapping / Webinar on demand / in englisch
  https://www.youtube.com/watch?v=pxi-QFgFiv4

Tages-Seminar an der Technischen Akademie Esslingen:

- **Software-Architekturen für Embedded-Echtzeitsysteme**
  07. März 2024 - Andreas Willert
  https://www.tae.de/weiterbildung/it-digitalisierung/softwareentwicklung/software-architekturen-fuer-embedded-echtzeitsysteme/

- **Weitere Embedded Software Engineering Reports zum Thema Architektur**

  Part 2: Über den Unterschied zwischen Architekturentwurf und Design
  https://www.sodiuswillert.com/de/ese-reports/software-architektur-und-design-2

  Part 1: Das Software Architektur Design definiert, wie alles ineinander wirkt
  https://www.sodiuswillert.com/de/ese-reports/architektur-design-software

- **Voraussetzung zum effizienten Engineering von komplexen Systemen**
  Von Dokumenten zu Linked Data - ESE-Report Nr. 42
  https://www.sodiuswillert.com/ese-reports/von-dokumenten-zu-linked-data

Autor:

ANDREAS WILLERT

Herausgeber:

WILLERT SOFTWARE TOOLS GMBH

Hannoversche Straße 21

31675 Bückeburg

info@sodiuswillert.com

Tel.: +49 5722 9678 - 60

sodiuswillert.com

Willert Software Tools ist Teil der Unternehmensgruppe SODIUS WILLERT