HANSER

Leseprobe

Oliver Alt

Modellbasierte Systementwicklung mit SysML

ISBN: 978-3-446-43066-2

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-43066-2

sowie im Buchhandel.

2 Systems Engineering

Fragen, die dieses Kapitel beantwortet:

- Was ist ein System?
- Was ist Systems Engineering?

2.1 Was ist ein System?

Die Frage, was man genau unter einem System versteht, ist schwer zu beantworten. Es kommt dabei immer auf die Sichtweise an und darauf, wie man seinen Entwicklungskontext abgrenzt.

Beispielsweise ist für einen Automobilzulieferer ein Steuergerät, welches er zu entwickeln und zu liefern hat, sein System. Für den Auftraggeber, also den Automobilhersteller hingegen ist dieses Steuergerät sicherlich maximal ein Teilsystem, wenn nicht sogar nur eine Systemkomponente.

Für den Begriff des **Systems** finden sich, je nachdem für welchen Anwendungsbereich die Definition gemacht wurde, verschiedene Festlegungen, die jedoch Gemeinsamkeiten aufweisen.

Unter der deutschen Wikipedia-Seite findet sich folgende Definition [Wik11a]:

Ein **System** (von griechisch $\sigma\upsilon\sigma\tau\eta\mu\alpha$, [...] „das Gebilde, Zusammengestellte, Verbundene“; Plural Systeme) ist eine Gesamtheit von Elementen, die so aufeinander bezogen sind und in einer Weise wechselwirken, dass sie als eine aufgaben-, sinn- oder zweckgebundene Einheit angesehen werden können und sich in dieser Hinsicht gegenüber der sie umgebenden Umwelt abgrenzen.

In der ISO 26262, einer neuen Norm für die Entwicklung von sicherheitskritischen Systemen in der Automobilindustrie (vgl. Abschnitt 6.3.3), findet sich folgende Definition[1]:

System: Menge von Elementen (Systemelementen), mindestens Sensoren, Verarbeitungseinheiten und Aktuatoren, die gemäß einem Entwurf in einer Beziehung zueinander stehen.
Anmerkung: Ein Systemelement kann auch selbst wieder ein System sein.

Die zweite Definition ist schon sehr technisch gehalten, da hier bereits von Sensoren usw. gesprochen wird. Es kommt bei solchen Definitionen eben immer auch darauf an, in welchem Entwicklungskontext man sich bewegt. Sicherlich ist ein System eine Ansammlung von einzelnen Teilen, welche durch ihre besondere Art der Kombination eine neue Funktionalität oder Aufgabe erfüllen können, die die Einzelteile so nicht erfüllen konnten.

Diese besondere Kombination der Einzelteile zu finden, darin liegt die geistige Leistung der Ingenieure, die das System entwickeln. Das Systems Engineering steckt dabei den Rahmen ab, der hilft, auf systematische Art und Weise Systeme erfolgreich zu entwickeln.

Wichtig aus der zweiten oben aufgeführten Definition ist sicherlich die Anmerkung, dass ein Systemelement auch selbst ein System sein kann, also auch wieder aus Systemelementen bestehen kann. Damit ist ein System eine rekursive Struktur. Deshalb ist eine genaue Definition des Betrachtungs- bzw. Entwicklungskontextes so wichtig. Wenn man klar den Kontext abgegrenzt und definiert hat, weiß man immer genau, auf welchen Teil des Systems man sich bezieht, was Teil und auch was nicht Teil des betrachteten Systems ist.

Findet diese Kontextabgrenzung nicht oder nur unzureichend präzise statt, so kann dies zu Missverständnissen zwischen verschiedenen, an der Entwicklung beteiligten Personen führen, da man über bestimmte Systemteile oder Systemkontexte spricht, aber das Gegenüber vielleicht völlig andere darunter versteht.

Um solche Probleme zu lösen oder zu minimieren, wurden Techniken und Methodiken entwickelt, die unter dem Begriff *Systems Engineering* zusammengefasst wurden.

2.2 Systems Engineering

Systems Engineering kann man vielleicht am besten mit Systemtechnik oder auch Systementwicklung ins Deutsche übersetzen. Beide Begriffe drücken jedoch nicht ganz das aus, was man weithin unter Systems Engineering versteht, nämlich die Gesamtheit der Entwicklungsaktivitäten, die notwendig sind, um ein System zu entwickeln. Daher soll der englischsprachige Begriff hier auch noch weiterhin Verwendung finden.

[1] Originaltext:
system set of elements, at least sensor, controller, and actuator, in relation with each other in accordance with a design
NOTE: An element of a system can be another system at the same time.

Systems Engineering besteht aus drei Bausteinen. In Bild 2.1 sind diese drei Bausteine illustriert. Es sind:

- Systemarchitektur
- Systemanforderungen
- Systemverhalten

Diese drei Bausteine zusammengenommen ergeben ein Bild bzw. eine Sammlung von Arbeitsprodukten, welches das System im Rahmen des Systems Engineering vollständig definiert.

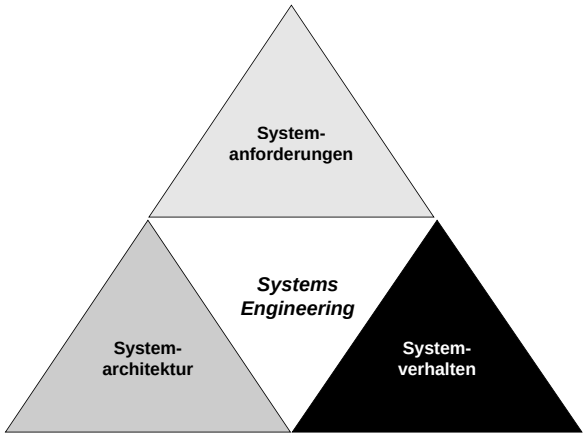

BILD 2.1 Die drei Bausteine des Systems Engineering

Im Folgenden möchte ich die einzelnen Bausteine kurz erläutern, damit Sie einen Eindruck gewinnen, was man konkret darunter versteht.

2.2.1 Architektur

Der erste Baustein ist die Systemarchitektur. Sie beschreibt die Struktur des Systems und legt die Komponenten fest, aus denen das System besteht. Außerdem definiert sie deren Schnittstellen intern zu anderen Systemkomponenten, aber auch die Schnittstellen zu den Systemgrenzen. Damit grenzt man durch die Architektur den Systemkontext ganz präzise ab. Darüber hinaus legt die Architektur fest, welche Schnittstellen einer Systemkomponente mit welcher Schnittstelle einer anderen Systemkomponente Daten, Material oder Energie austauscht.

Über die Architektur allein lässt sich ein System noch nicht ausreichend definieren. Es fehlen weitere Informationen darüber, welche Funktion das System mit dieser Architektur erfüllen soll. Die Architektur ist eine statische Sicht auf das System, aus der nicht hervorgeht, *welche* Daten oder Materialien über die Schnittstellen *wann* ausgetauscht werden sollen.

Um diesen Sachverhalt noch einmal zu verdeutlichen, wollen wir eine Analogie nutzen. Versetzen Sie sich einmal in die Rolle eines Archäologen, der eine Ausgrabung macht. Er stößt vielleicht auf ein altes Gebäude oder Gemäuer, welches eine merkwürdige Raumausrichtung

und Aufteilung aufweist. Die Architektur des Gebäudes kann zu 100 % rekonstruiert werden, jedoch wird deren genauer Zweck Raum für Spekulation bieten oder unbekannt bleiben.

Es fehlt hier die Funktionsbeschreibung, aus der hervorgeht, warum das entdeckte Gebäude in dieser Weise erbaut wurde. Wenn man diese Funktion kennt, wird auch klar, warum genau diese Architektur als Lösung gewählt wurde.

Architektur, egal ob im Bauwesen oder in der Systementwicklung, ist immer nur eine mögliche Lösung für ein gegebenes Problem. Ein gutes Beispiel aus dem Alltag für unterschiedliche Lösungen für dasselbe Problem sind Korkenzieher für Weinflaschen. Die Funktion ist dabei immer dieselbe, nämlich die Aufgabe, den Korken rückstandsfrei und ohne die Flasche zu zerstören aus dieser zu entfernen. Sicherlich kennen Sie auch verschiedene Modelle von Korkenziehern, die diese Aufgabe auf unterschiedliche Arten erledigen. Sie alle haben andere Architekturen bzw. ein anderes Design[2], lösen das gegebene Problem aber alle, vielleicht mehr oder weniger gut.

Insgesamt kann die Architektur die folgenden Fragen beantworten oder bei deren Beantwortung hilfreich sein:

- Auf welche Art und Weise wird eine geforderte Funktionalität realisiert?
- Aus welchen Teilen besteht das zu realisierende System?
- Ist das System mit der gewählten Architektur in der Lage, eine geforderte (neue) Funktionalität zu erfüllen?

Abschließend zur Architektur noch eine Bemerkung: In der Architektur gibt es keine Funktionen. Das heißt, man darf nicht versuchen, Funktion mit Hilfe der Architektur zu beschreiben. Vielmehr haben die Architekturkomponenten ein Verhalten, durch das diese eine Funktionalität erfüllen. Eine Architektur dokumentiert eine Lösung, die gewählt wurde, damit das System am Ende eine bestimmte Funktion hat. Die Architektur beschreibt das „Wie ist es gelöst?“.

Die aus der Architektur nicht hervorgehenden Informationen über das dynamische Systemverhalten stecken in den anderen beiden Bausteinen des Systems Engineering, nämlich den (System-)Anforderungen bzw. der Definition des Systemverhaltens.

2.2.2 Anforderungen

Anforderungen (engl. *Requirements*) sind Texte, die definieren, was das zu entwickelnde System können und leisten muss. Dabei unterscheidet man oftmals zwischen funktionalen Anforderungen und nichtfunktionalen Anforderungen.

2.2.2.1 Funktionale Anforderungen

Funktionale Anforderungen spezifizieren das Verhalten des Systems, bzw. der Systemkomponenten. Sie machen Aussagen über dynamisches Verhalten und definieren damit auch, wann

[2] In diesem Buch wird zwischen Architektur und Design kein Unterschied gemacht. Prinzipiell sind die Arbeitsschritte bei beiden Tätigkeiten identisch, wobei man oftmals unter Design mehr Details erwartet als bei Architektur. Daher lässt sich eine Grenze zwischen Architektur und Design auch nicht allgemein definieren, und damit sind die beiden Begrifflichkeiten immer Definitionssache.

und in welcher Weise Daten bzw. Materialien vom System verarbeitet und über Schnittstellen mit anderen Systemkomponenten oder externen Systemen ausgetauscht werden. Funktionale Anforderungen und die Architektur zusammen definieren daher ein System vollständig, da Struktur und Verhalten nun definiert sind. Immer vorausgesetzt die funktionalen Anforderungen sind vollständig, d.h. die Systemfunktionalität ist ausreichend komplett beschrieben.

2.2.2.2 Nichtfunktionale Anforderungen

Neben den funktionalen Anforderungen benötigt man auch die nichtfunktionalen Anforderungen. Diese Art Anforderungen sind nur indirekt für die Funktion des Systems entscheidend, da sie typischerweise Qualitätsanforderungen sind. In der neueren Literatur (z.B. [Poh08]) wird inzwischen häufiger von *Qualitätsanforderungen* anstelle von nichtfunktionalen Anforderungen gesprochen. Die beiden Begriffe können synonym verwendet werden. Da man in der Praxis so gut wie überall noch von nichtfunktionalen Anforderungen spricht, soll auch hier der Begriff weiter verwendet werden.

Nichtfunktionale Anforderungen spezifizieren gemäß DIN/ISO 9126 die folgenden Eigenschaften eines Systems oder einer Systemkomponente:

- Zuverlässigkeit, engl. *Reliability* (z.B. Fehlertoleranz)
- Benutzbarkeit, engl. *Usability* (z.B. Bedienbarkeit, Wartbarkeit)
- Effizienz, engl. *Efficiency* (z.B. Reaktionszeiten, Verbrauch)
- Änderbarkeit, engl. *Maintainability* (z.B. Modifizierbarkeit, Testbarkeit)
- Übertragbarkeit, engl. *Portability* (z.B. Anpassen an neue Umgebung, leichter Austausch von Komponenten)

All diese nichtfunktionalen Anforderungen haben einen Einfluss auf die Systemarchitektur. Wird beispielsweise gefordert, dass ein System leicht wartbar sein muss, so kann dies dadurch sichergestellt werden, dass das Gehäuse ohne besonderes Werkzeug geöffnet werden kann. Dies hat auf die Funktion des Systems keinen Einfluss, jedoch erfüllt die Architektur damit diese Qualitätsanforderung.

Auch die anderen oben genannten Arten von Qualitätsanforderungen beeinflussen die Architektur entsprechend. Dabei muss immer im Auge behalten werden, dass durch die Änderung der Architektur aufgrund nichtfunktionaler Anforderungen die Funktionalität (also die Erfüllung der funktionalen Anforderungen) nicht beeinträchtigt werden darf.

Auch Anforderungen wie Preis oder die Forderung, dass das System nach einer bestimmten Entwicklungsnorm zu entwickeln ist, gehören zu den nichtfunktionalen Anforderungen. Auch diese haben Einfluss auf die Systemarchitektur, da der vorgegebene Preis den Freiraum in der Entwicklung z.B. in Bezug auf Materialauswahl einschränkt. Weiterhin könnte die geforderte Einhaltung einer Entwicklungsnorm auch Einfluss auf die Architektur haben, wenn diese Norm z.B. bestimmte Architekturvorgaben macht.

2.2.2.3 Anforderungen und Architektur gehören immer zusammen

Anforderungen und Architektur gehören stets zusammen. Einerseits kann man keine Architektur erstellen, wenn man keine Anforderungen an das System hat, da die Architektur eine Lösung darstellt, welche die Anforderung erfüllt.

Andererseits kann man keine Anforderungen ohne Architektur schreiben. Sehen Sie sich beispielsweise folgende Anforderung an:

Im Falle eines Fehlers muss das System den Benutzer akustisch auf den Fehler aufmerksam machen.

Diese vielleicht etwas unpräzise formulierte Anforderung beinhaltet jedoch bereits implizit mehrere Architekturentscheidungen, die der Schreiber der Anforderung bereits getroffen hat:

1. Die Komponente, die diese Anforderung erfüllt, wird hier mit dem Namen „System“ bezeichnet. Der Schreiber der Anforderung hat also ein Bild von etwas im Kopf, das er mit System bezeichnet hat. Ein solches Bild braucht man immer, wenn man eine Anforderung schreibt. Dokumentiert man ein solches Bild auf Papier oder in einem Systemmodell, so ist dies ein (erstes) Bild der Architektur. Im Fall unserer Beispielanforderung kann man zumindest etwas als Architektur dokumentieren, was man mit „System“ bezeichnet.
2. Das System braucht eine Möglichkeit, dem Benutzer akustische Warnmeldungen auszugeben. Dies könnte durch ein Subsystem geschehen, das in der Lage ist, auf Anforderung hin akustische Signale abzugeben. Dies ist wiederum eine Architekturentscheidung, und diese kann auch als Architektur dokumentiert werden.
3. Es muss es eine Fehlererkennung geben, damit ein Fehler zunächst einmal erkannt werden kann. Auch dies kann als Architektur dokumentiert werden.
4. Und zuletzt impliziert diese Anforderung auch noch, dass das System und ein Benutzer miteinander kommunizieren. Damit hat man bereits eine erste externe Systemschnittstelle und einen Kommunikationspfad gefunden, den man auch in der Architektur darstellen kann.

Sie sehen: Aus einer einzelnen Anforderung lassen sich diverse Architekturinformationen ableiten. Der Schreiber einer Anforderung hat also immer schon ein mehr oder weniger konkretes Bild des zu entwickelnden Systems im Kopf, wenn er eine Anforderung aufschreibt.

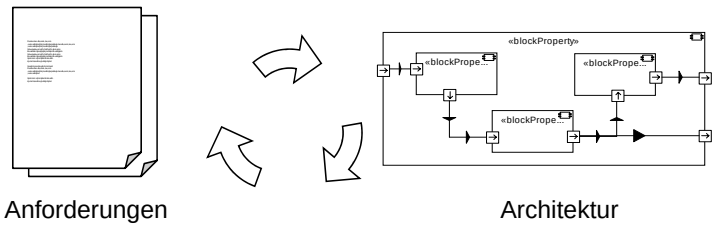

BILD 2.2 Anforderungen und Architektur als iterativer Prozess

Man kann also Anforderungen immer nur in Zusammenhang mit der Architektur schreiben. Dabei macht es zunächst keinen Unterschied, ob man die Architektur bereits zu Papier gebracht, oder nur im Kopf hat. Im Laufe der Systementwicklung muss man selbstverständlich das Wissen über Anforderungen und Architektur auch dokumentieren.

Aus Anforderungen entsteht eine Architektur als Lösung, und aufgrund dieser können sich wiederum neue Anforderungen ergeben. Beispielsweise können dies dann Anforderungen an die in der Architektur definierten Unterkomponenten und deren Schnittstellen sein. Bild 2.2 veranschaulicht diesen Sachverhalt nochmals.

In Abschnitt 7.3.3 wird das gemeinsame Entwickeln von Architektur und Anforderungen anhand des in diesem Kapitel genutzten Beispiels noch einmal ausführlich gezeigt.

2.2.2.4 Gute Anforderungen formulieren

Es gibt viele Möglichkeiten, Anforderungen aufzuschreiben. Sie können ein Lasten- oder Pflichtenheft als zusammenhängenden Text wie einen Aufsatz schreiben. Sie können aber auch Anforderungen nach bestimmten Richtlinien formulieren, um damit bereits bei der Anforderungsspezifikation eine Formalität einzuhalten, die später gewisse Vorteile bei der Systementwicklung bietet.

Bekannt geworden durch die SOPHIST-Group und Chris Rupp ist ein Vorschlag zum Formulieren von natürlichsprachlichen Anforderungen[3], der es erlaubt, auf einfache Art und Weise mit Hilfe einer Formulierungsschablone Anforderungen zu formulieren, die dann bereits eine Reihe von Qualitätskriterien an Anforderungen erfüllen [R+09].

Dieses Formulierungsschema wird im Übrigen auch für die international anerkannte Prüfung zum Certified Requirements Engineer [PR09] (www.certified-re.de) vorgeschlagen. Daher soll es an dieser Stelle auch noch einmal als Hilfestellung für die tägliche Arbeit mit Anforderungen gezeigt und genutzt werden.

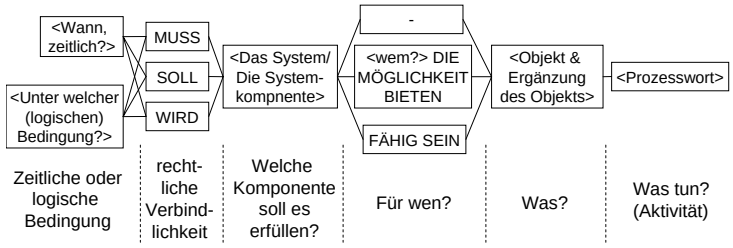

BILD 2.3 Formulierungsschablone für Anforderungen nach [R+09]

Bild 2.3 zeigt das Formulierungsschema für textuelle Anforderungen. Jede Anforderung beginnt mit einer zeitlichen (Wann soll etwas getan werden?) oder logischen Bedingung. Diese Bedingung kann dann auch wegfallen, wenn die Anforderung bedingungslos unter allen möglichen Umständen immer gilt.

Nach der Bedingung folgt die rechtliche Verbindlichkeit, also die Aussage, ob es zwingend erforderlich, nur optional bzw. zukünftig geplant ist, diese Anforderung zu erfüllen. Nun folgt die Systemkomponente, für die diese Anforderung bestimmt ist. Dies ist eine Abwandlung vom ursprünglichen Schema, wo nur immer vom *System* die Rede ist. Da der Begriff des Systems aber wie bereits oben erläutert stark kontextabhängig ist, sollte hier immer der Name der Systemkomponente stehen, auf die sich die Anforderung bezieht.[4]

Der Rest der Anforderung beschreibt dann was diese Systemkomponente tun soll, bzw. leisten muss.

Die Verwendung der Formulierungsschablone bietet eine Reihe von Vorteilen. Ich möchte Ihnen hier nur die wichtigsten aufführen. Die vollständige ausführliche Erläuterung der Schablone sowie die Schablone für englischsprachige Anforderungen finden Sie beispielsweise in [R+09].

---

[3] Auch bekannt als Teil des „SOPHIST REgelwerk“

[4] Auf neueren Präsentationen der SOPHIST-Group aus dem Jahr 2011 wird nun auch der Begriff `<Systemname>` als Erweiterung des ursprünglich veröffentlichten Schemas verwendet. (Quelle: GI e.V. RE-Fachgruppentreffen 2011, Hamburg)

Die wichtigsten Vorteile durch den Einsatz der Schablone sind:

- Eindeutigkeit der Anforderung (weniger Missverständnisse)
- Vollständigkeit der Anforderung (präzise Formulierung)
- Testbarkeit der Anforderung (einfachere Testfallerstellung)
- Jede Anforderung besteht aus nur einem Satz (keine Kettenanforderungen und dadurch bessere Zuordnung der Anforderungen zu Testfällen und Architekturkomponenten möglich)
- Semiformaler Charakter durch Schablone erleichtert die spätere Formalisierung bzw. Erstellung von formalen Verhaltensmodellen

2.2.3 Systemverhalten

Als letzter Baustein des Systems Engineering fehlt noch die Beschreibung des Systemverhaltens. Vielleicht fragen sich nun einige von Ihnen, warum man denn noch einmal das Verhalten beschreiben soll. Dies ist doch eigentlich mit den funktionalen Anforderungen bereits beschrieben.

Sie haben zunächst vollkommen recht! Die funktionalen Anforderungen beschreiben das Verhalten des Systems und seiner Unterkomponenten. Diese Anforderungen beschreiben das Verhalten als Text, also nicht formal. Das heißt, diese Anforderungen müssen von einem Techniker oder Ingenieur gelesen und entsprechend in eine Realisierung des Systems umgesetzt werden.

Beschreibt man das Verhalten des Systems nicht nur informell mit Hilfe von funktionalen Anforderungen, sondern formal, so kann man daraus automatisiert Arbeitsprodukte ableiten, die ansonsten manuell erstellt werden müssen. Typische Beispiele für solch eine Vorgehensweise sind Codegenerierung oder auch die Ableitung von Testinformationen.

Unter einer formalen Verhaltensbeschreibung soll hier verstanden werden, dass sich daraus Arbeitsprodukte und Informationen rechnergestützt mit Hilfe von Algorithmen ableiten lassen. Eine solche formale Verhaltensbeschreibung des Systems kann daher die Arbeit der Entwicklungsingenieure vereinfachen oder entlasten, da zumindest Standardaufgaben nun rechnergestützt durchgeführt werden können.

Durch eine solche rechnergestützte Entwicklung wird der Entwicklungsingenieur in keinster Weise überflüssig oder ersetzt. Es findet in einem solchen Fall nur eine Verlagerung der Entwicklungsaktivitäten auf eine andere Ebene statt. Wenn beispielsweise Code aus einer formalen Verhaltensbeschreibung generiert wird, so ist die geistige inhaltliche Arbeit immer noch die gleiche. Sie steckt nun aber nicht mehr in der Aufgabe der Erstellung des Programmcodes, sondern liegt in der Erstellung der Verhaltensbeschreibung und in den Algorithmen, die der Codegenerator benutzt.

Konzepte und Technologien, um Verhalten von Systemen formal zu beschreiben, sind zahlreich und vielfältig. Viele davon entstammen aus spezifischen Anwendungsbereichen, sogenannten Domänen, und eignen sich daher besonders gut, um Verhalten von Systemen, die aus dieser Domäne stammen, zu spezifizieren.

Dieses Buch befasst sich mit der SysML als Beschreibungssprache für Systeme. Trotzdem sollen auch kurz einige andere Methodiken zur Verhaltensbeschreibung erwähnt werden. Diese können im Systems Engineering neben oder zusätzlich zu SysML angewandt werden:

- **Specification and Description Language (SDL)**
  Die SDL-Sprache [SDL11] entstammt dem Umfeld der Telekommunikationsindustrie und wurde entwickelt, um das Verhalten von Telekommunikationssystemen zu beschreiben. Große Teile der SDL-Sprache sind inzwischen auch in die UML- und SysML-Sprachen als Teile der Aktivitäts- und Sequenzdiagramme übernommen worden, da sich gezeigt hat, dass sich die Konzepte auch über den Telekommunikationsbereich hinaus eignen, um Verhalten von technischen Systemen zu beschreiben.
- **Petri-Netze**
  Petri-Netze [Pet62] eignen sich besonders gut, um nebenläufige Prozesse zu beschreiben und zu spezifizieren. Es gibt eine Reihe von Erweiterungen des ursprünglichen Konzepts. Damit lassen sich bestimmte Dinge des Systemverhaltens besser ausdrücken. Auch Teile des Konzeptes der Petri-Netze sind inzwischen in die UML und auch SysML eingeflossen. Die Aktivitätsdiagramme definieren ihre Semantik, also wie sie zu interpretieren sind, analog zu den Petri-Netzen[5].
- **Grafische Funktionsentwicklung**
  Grafische Funktionsentwicklung wird dort eingesetzt, wo vorwiegend regelungstechnische Systeme beschrieben werden müssen. Typische Vertreter von Werkzeugen zur Funktionsentwicklung sind Matlab/Simulink [The11] und ASCET SD [ETA11]. Diese Beschreibungssprachen, die dort zum Einsatz kommen, basieren darauf, dass man Blöcke mit klar definiertem Verhalten miteinander verschaltet. Solche Blöcke sind typischerweise Dinge, die aus der Systemtheorie und der Regelungstechnik stammen, wie Übertragungsfunktionen, Addierer, Integrierer etc.

Die Liste lässt sich sicherlich noch weiter fortsetzten. Es existiert bestimmt auch eine Reihe von weniger bekannten Verhaltensbeschreibungen, die für spezielle Zwecke entworfen und eingesetzt werden. Wichtig soll nur an dieser Stelle sein, Ihnen ein Gefühl für solche formalen Verhaltensbeschreibungen zu geben.

Man kann die formale Verhaltensbeschreibung im Systems Engineering auch erst in einem zweiten Schritt in Angriff nehmen. Startpunkt sind immer die Anforderungen und die Architektur, die man auf jeden Fall braucht. Dies ist dann zunächst eine informelle Beschreibung des Systems.

Ohne formale Verhaltensbeschreibung ist die Erstellung der Arbeitsprodukte und der Realisierung eine komplett manuelle Aufgabe. Mit formaler Verhaltensbeschreibung können Aufgaben automatisiert oder auch Systemverhalten simuliert werden, um bereits in einer frühen Phase der Systementwicklung Fehler zu finden oder auch Konzepte kostengünstig zu erstellen oder bei Bedarf auch einmal zu verwerfen.

[5] Das Tokenkonzept wird aus Petri-Netzen übernommen.

2.3 Das Systems-Engineering-Schema

Systems Engineering ist ein iterativer, das heißt fortlaufender Prozess. Im Laufe der Entwicklung entstehen meist mehrere Perspektiven, die bestimmte Aspekte des zu entwickelnden Systems zeigen und definieren. Diese verschiedenen sogenannten Abstraktionsebenen (siehe auch Abschnitt 6.6) definieren jeweils Architektur, Anforderungen und bei Bedarf Verhalten und bauen jeweils aufeinander auf.

Bild 2.4 zeigt das *Systems-Engineering-Schema*, das die Zusammenhänge des Systems Engineering schematisch darstellt. Wichtig dabei ist noch einmal, dass Anforderungen, Architektur und eine eventuelle formale Verhaltensbeschreibung Hand in Hand gehen, jeweils aufeinander aufsetzen und sich iterativ weiter entwickeln.

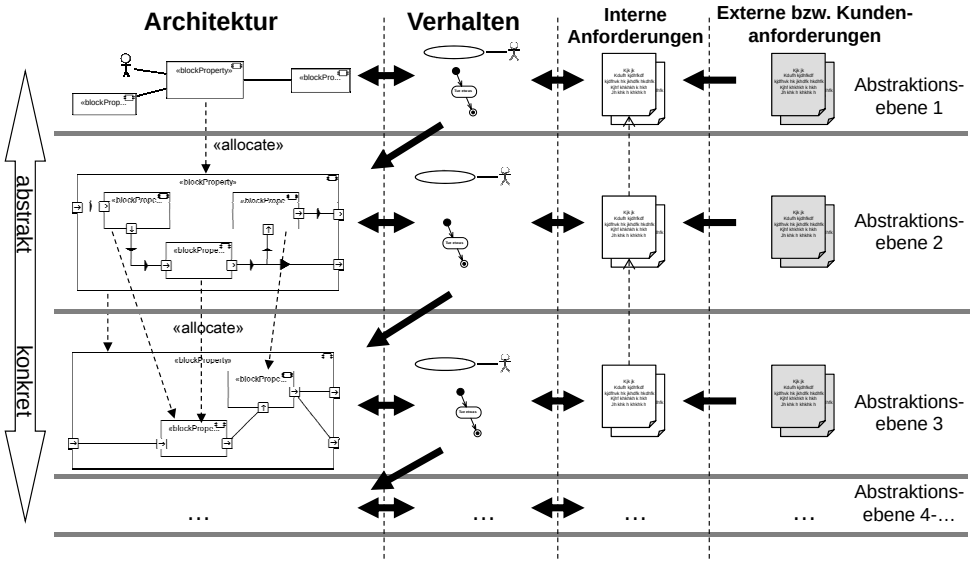

BILD 2.4 Systems-Engineering-Schema

Die Systementwicklung beginnt zum Beispiel mit einem neuen Entwicklungsauftrag des Managements, der besagt, dass das neue System eine Weiterentwicklung des bestehenden, aber mit größerem Leistungsumfang und geringerem Gewicht sein soll.

Mit dieser ersten Anforderung kann nun die Entwicklungsaktivität beginnen. Zunächst wird man damit starten, eine Kontextabgrenzung durchzuführen, um herauszufinden, was man selbst entwickeln muss und was nicht Teil des Systems ist. Eventuell nutzt man dann auch Konzepte wie Anwendungsfälle, Mind Mapping und Brainstorming. Mit dieser ersten Kontextabgrenzung findet man weitere, erste Anforderungen, die die Anforderungen des Managements verfeinern.

In der nächsten Entwicklungsstufe wird dann eine weitere Detaillierung der Architektur, der Anforderungen und des Verhaltens vorgenommen. Dies geht so lange weiter, bis das neue System soweit beschrieben und definiert ist, dass es in Produktion gehen kann.

Außerdem zeigt das Systems-Engineering-Schema noch, dass Kundenanforderungen, Marktanforderungen oder auch gesetzliche Anforderungen als externe Anforderungen in die Ent-

wicklung eingehen. Externe Anforderungen sollten aber nie direkt als Entwicklungsgrundlage dienen, da sie eventuell zu ungenau spezifiziert oder aber für die etablierten Entwicklungsmethoden des Unternehmens nicht passend sein können.

Daher werden externe Anforderungen immer in interne Anforderungen umgewandelt bzw. auf interne Anforderungen abgebildet. Typischerweise enthalten Lastenhefte externe Anforderungen und Pflichtenhefte interne Anforderungen. Diese Pflichtenhefte werden oftmals auch als rechtsverbindliche Entwicklungsgrundlage zwischen Auftraggeber und Auftragnehmer verhandelt und abgestimmt.

Das System-Engineering-Schema stellt den Prozess des Systems Engineering bewusst sehr allgemein und abstrakt dar. Dadurch wird es möglich, damit die Arbeitsweise von verschiedenen konkreten Realisierungen des Systems Engineering zu beschreiben. Mit dem Schema als Vorgabe allein lässt sich Systems Engineering im realen Umfeld nicht oder nur schlecht etablieren. Es bedarf weitere Konkretisierungen, wie man das Schema nun in der Praxis anwenden soll. Dies können Festlegungen sein, welche Abstraktionsebenen benötigt werden, um das zu entwickelnde System zu beschreiben, und wie die konkreten Arbeitsprodukte aussehen sollen. Auch ist es wichtig zu entscheiden, mit welchen Werkzeugen Systems Engineering durchgeführt werden soll.

Ein konkreter Vorschlag zur Umsetzung des Systems-Engineering-Schema findet sich weiter hinten in Teil II des Buches. Die dort beschriebenen Konzepte wurden für die Praxis entwickelt und haben sich dort in langjähriger Anwendung auch bewährt.

Qualitätssicherung gehört immer auch dazu

Das Systems-Engineering-Schema in Bild 2.4 stellt nur den Entwicklungsteil der Systementwicklung bzw. des Systems Engineering dar[6]. Natürlich ist klar, dass jedem Entwicklungsschritt auch qualitätssichernde Maßnahmen folgen müssen und diese entsprechend im Systems Engineering verankert sein müssen. Solche Qualitätsmaßnahmen können Reviews, statische und dynamische Tests sowie Prüfung und Erprobung sein – je nachdem, wie sich die Qualität eines Arbeitsprodukts überprüfen lässt.

[6] Man könnte auch sagen, es stellt nur die linke Seite des allgemeinen V-Modells (vgl. Abschnitt 6.2) dar.