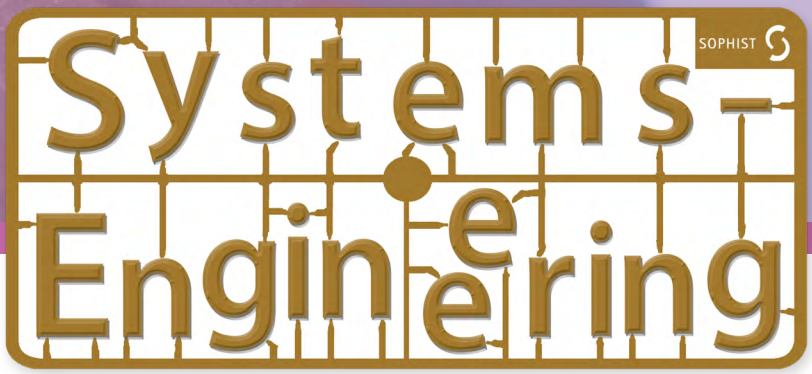


Die SOPHISTen

»Mit **Requirements-Engineering** und **Architektur** zum Erfolg«

SOPHIST

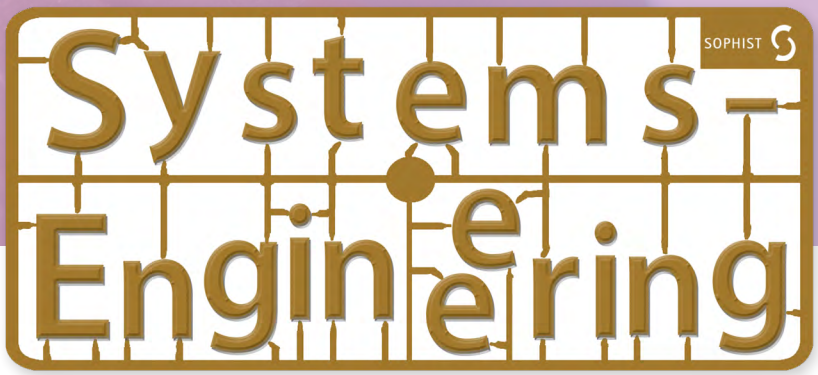
Die SOPHISTen

SOPHIST GmbH
Vordere Cramergasse 13
90478 Nürnberg
Deutschland
www.sophist.de

@ SOPHIST_GmbH
@SOPHIST.GmbH
/sophistgmbh
blog.sophist.de

1. Auflage 2021

Copy Editing & Herstellung: Roland Kluge, SOPHIST GmbH
Illustrationen: Assad Binakhahi



Die SOPHISTen

»Mit **Requirements-Engineering** und **Architektur** zum Erfolg«

SOPHIST

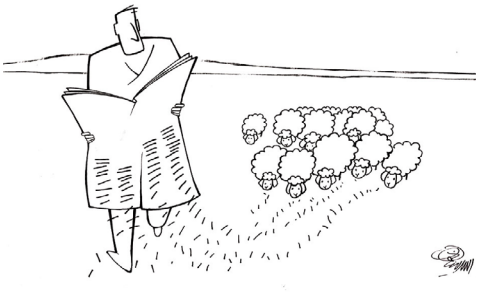
www.sophist.de

1. Einleitung

Seit über 20 Jahren beschäftigt sich SOPHIST nun mit Anforderungen. Wir haben in dieser Zeit zahlreiche Firmen bei der Einführung und Durchführung von gutem Requirements-Engineering unterstützt.

Dabei wurden wir mit den unterschiedlichsten Anwendungsgebieten konfrontiert. Diese reichten von gesamten Flughafen-Towern über reine Software-Anwendungen bis hin zu einzelnen ASICs (Application Specific Circuit).

Der Bedarf an der Unterstützung hat sich in dieser Zeit jedoch gewandelt. Neben der Ermittlung und Dokumentation von Anforderungen, die sich an den Betrieb des betrachteten Produkts richten, rücken die Anforderungen aus den anderen Lebenszyklen des Produkts immer mehr in den Fokus. Weiterhin wird die Notwendigkeit immer größer, die Analyse der Anforderungen als integralen Bestandteil der weiteren Entwicklung zu sehen.



Diese Integration wird im Bereich der reinen Softwareentwicklung mit den agilen Ansätzen gut unterstützt. Diese Ansätze funktionieren in einem komplexen System, das neben Software auch aus elektrischen, elektronischen oder mechanischen Anteilen besteht, allerdings nur bedingt. An dieser Stelle setzt Systems-Engineering an, das in einem interdisziplinären Ansatz sowohl das gewünschte Produkt als Gesamtsystem als auch alle Bereiche in seinem Lebenszyklus betrachtet.

Diese Broschüre gibt Ihnen einen Überblick über spezielle Herausforderungen im Requirements-Engineering bei der Entwicklung großer und komplexer Systeme. Doch erst die enge Verknüpfung mit den nachfolgenden Entwicklungstätigkeiten, insbesondere mit einer Systemarchitektur, führt zu einer effizienten Systementwicklung. Mehr und detailliertere Informationen bieten wir Ihnen in Form eines Systems-Engineering-Trainings oder natürlich auch in einer persönlichen Beratung an.

Die hier vorgestellten Ansätze und Methoden spiegeln das Wissen von vielen SOPHISTen wider. Somit muss an dieser Stelle allen SOPHISTen gedankt werden, die ihr Wissen aus zahlreichen Projekten in einen Topf geworfen haben und den Inhalt im Rahmen von heißen Diskussionen kräftig umgerührt haben. Nur dadurch konnten wir das Beste daraus abschöpfen und Ihnen hier vorstellen.

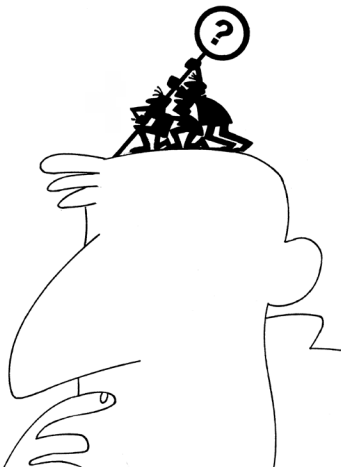
Besonderer Dank gilt zudem:

- Dr. Stefan Queins für die Erstellung der Inhalte
- Chris Rupp und Dominik Häußer für das inhaltliche Review
- Alexander Holz und David Nawzad für das Design und Layout

1.1 Motivation

In den letzten Jahren gewinnt der Begriff des Systems-Engineerings immer mehr an Bedeutung. Begründet wird dies durch eine steigende Erwartungshaltung von Nutzern gegenüber technischen Systemen. Wurde z. B. eine Waschmaschine früher mit einem Drehknopf und mehreren Tasten bedient, so besitzen heute moderne Waschmaschinen einen Touchscreen und eine WiFi-Verbindung. Aber auch ein steigender Qualitätsanspruch führt zu immer komplexeren Systemen, da immer kompliziertere Algorithmen, ausgefeilte Elektronik und immer höherwertig anmutende Mechanik zum Einsatz kommen muss. Zusammenfassend führt diese steigende Erwartungshaltung zu den Smart-Eco-Systems, die uns Menschen in der bestmöglichen Art, die zurzeit vorstellbar ist, unterstützen sollen.

Die steigende Komplexität der Systeme spiegelt sich natürlich in der wachsenden Komplexität einzelner Komponenten wider, aus denen sich ein System zusammensetzt. Damit eng verknüpft ist der Komplexitätszuwachs bei den Abhängigkeiten und Schnittstellen zwischen diesen Komponenten oder sogar zwischen ganzen Systemen (Stichwort Digitalisierte Fabrik). Beides führte in den letzten Jahren zu der Erkenntnis, dass die Entwicklung solch komplexer Systeme mehr als die Entwicklung der einzelnen Komponenten umfassen muss.



Heutzutage wird unter einem komplexen technischen System ein Zusammenbau aus mehreren Komponenten verstanden, die aus verschiedenen Gewerken stammen. So kann sich ein solches System aus elektrischer und elektronischer Hardware, aus mechanischen Komponenten, aus Software und, je nach Anwendungsgebiet, weiteren Gewerken (Optik, Hydraulik, etc.) zusammensetzen. Eine Herausforderung des Systems-Engineerings ist somit, die frühere multidisziplinäre Entwicklung hin zu einer interdisziplinären Entwicklung zu vereinen.

Diese Erkenntnis führt zu einer weiteren Herausforderung des Systems-Engineerings: Der gesamte Lebenszyklus eines Systems (und nicht nur der operative Einsatz) muss bei der Entwicklung eines Systems mit betrachtet werden. Bewegen wir uns z. B. auf der Seite eines Zulieferers, so sollte die Entwicklungsabteilung schon in einer Angebotsphase koordiniert integriert werden. Des Weiteren müssen Anforderungen an das System aus Sicht der Fertigung, Verpackung, Transport, Pflege etc. mit berücksichtigt werden.

Diese neuen (und zum Teil auch alten) Herausforderungen führen zwangsläufig dazu, über eine Entwicklungsdisziplin nachzudenken, deren Tätigkeiten oberhalb der Prozesse der einzelnen Gewerke liegt, diese miteinander koordiniert und so die korrekte

Entwicklung der einzelnen Bestandteile sicherstellt, damit deren Zusammenbau als gesamtes System die Wünsche der Anwender erfüllt. Zudem besteht die Hoffnung, die Effizienz der Entwicklung durch das Systems-Engineering zu steigern. Hier soll gerade die Steigerung des Aufwands in den frühen Phasen der Entwicklung helfen. Damit sollen die folgenden Punkte für die späteren Phasen der Entwicklung ausgeschlossen bzw. abgeschwächt werden:

- Überraschungen durch falsche Risikoeinschätzung.
- Änderungen aufgrund von Fehlern bzw. falschen Entscheidungen in den vorderen Phasen.

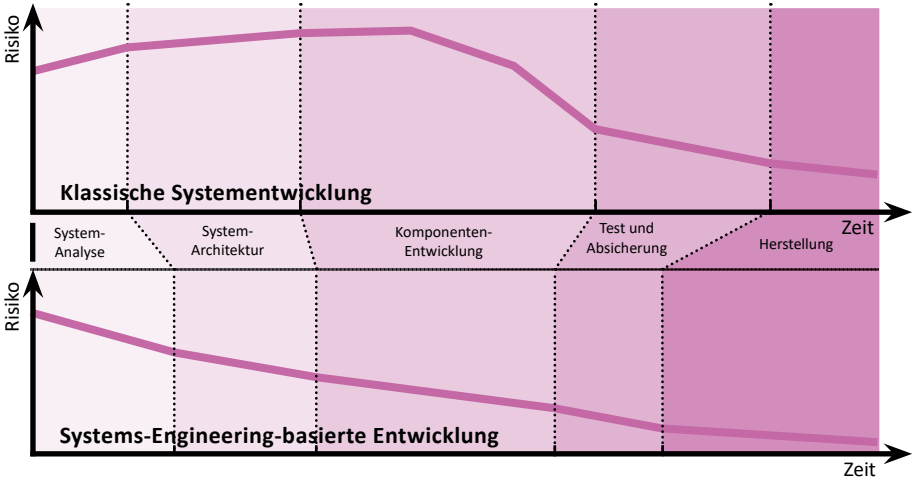


Abbildung 1.1: Kosten und Risiken vor und nach Einführung von Systems-Engineering

Abbildung 1.1 zeigt, angelehnt an [Incose15], die benötigte Entwicklungszeit und den qualitativen Verlauf der Risikoentwicklung ohne und mit Verwendung von Systems-Engineering-Ansätzen. Man sieht, dass bei deren Verwendung das Risiko früh abnimmt und dass sich die Gesamtentwicklungszeit verkürzt. Der Grund hierfür ist die Verlängerung der frühen Phasen. In wie weit sich diese Angaben in der Realität bestätigen, lässt sich besonders im Vorfeld einer Entwicklung nur sehr schwer abschätzen. Ein paar Ideen zu möglichen Metriken zur Messung des Effizienzgewinns finden sich in Kapitel 5.

1.2 Der Begriff System

Der Begriff System stammt aus dem Griechischen und kann mit „aus mehreren Einzelteilen zusammengesetztes Ganzes“ übersetzt werden. Im allgemeinen Sprachgebrauch wird der Begriff „System“ sehr breit verwendet. Beispiele hierfür sind:

- das Zusammenspiel mehrerer Menschen
- eine Firma mit ihren Mitarbeitern und Gütern
- das Zusammenwirken von Hochs und Tiefs (ein Wettersystem)
- eine Sonne mit ihren Planeten
- ein Dolby Surround System
- ein Auto

Bei allen Verwendungen des Begriffs System sind zwei Charakteristika zu finden: Zum einen besteht ein System immer aus mehreren kleineren Teilen und zum anderen würde das System ohne das Zusammenspiel dieser Teile nicht funktionieren.

Dies sind auch die beiden entscheidenden Eigenschaften von technischen Systemen (im Folgenden kurz „System“).

Definition System:

Systeme können in einzelne Komponenten zerlegt werden, die das Systemverhalten durch ihre eigenen Eigenschaften und durch ihr Zusammenspiel erreichen.

Auch bei diesen technischen Systemen ist die Bandbreite beliebig. So können sehr komplexe als auch relativ einfach aufgebaute Anwendungen als System bezeichnet werden. Auffällig ist jedoch, dass den Systemen eine beliebig tiefe Hierarchie zu Grunde liegt. Das heißt, Systeme können in Bestandteile zerlegt werden, die wiederum in kleinere Teile zerlegt werden können. Diese Hierarchisierung macht auch sehr komplizierte Systeme beherrschbar.

In dieser Broschüre betrachten wir als Beispiel ein System, das ein Wohnhaus bestmöglich automatisiert. Wir bezeichnen es als Smart-Home-System, kurz SHS (siehe Abbildung 1.2).

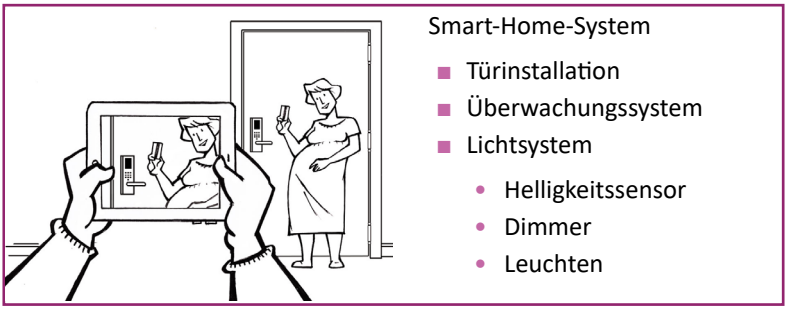


Abbildung 1.2: Verschiedene Systeme in einem Smart-Home-System

Wie man in dem Beispiel sehen kann, taucht der Begriff „System“ im Namen der Strukturelemente an mehreren Stellen auf unterschiedlichen Hierarchieebenen auf.

Ein Grund hierfür ist die Fokussierung auf unterschiedliche Ebenen. Die Bewohner des Smart-Home sehen als „System“ die gesamte Unterstützung, da dies für sie das Produkt ist, für das sie einem Hersteller Geld bezahlt haben. Für den Hersteller ist das SHS das zu entwickelnde System. Das Lichtsystem wäre ein Teil, also ein Subsystem, dieses SHS. Für die Organisationseinheit, die sich mit der Entwicklung des Lichtsystems beschäftigt, ist dieses Subsystem jedoch der Betrachtungsgegenstand auf ihrer obersten Ebene, und wird demnach in dieser Organisationseinheit als „System“ betrachtet. Das SHS wäre dann für das Lichtsystem das sogenannte „Supersystem“. Aber auch das SHS hat ein Supersystem. Es ist in das Haus eingebettet, das wiederum in die entsprechende Straße (oder Baugebiet) eingebettet ist. So können Sie für fast jedes System ein Supersystem definieren.

Der Betrachtungsgegenstand, der dem „System“ entspricht, muss demnach immer relativ zu der Organisationseinheit festgelegt werden, die den Betrachtungsgegenstand verantwortet.

2. Überblick über die Tätigkeiten

Die Haupttätigkeiten im Bereich des Systems-Engineerings spielen sich in der Entwicklung ab, wobei der Systems-Engineer die Einflüsse der weiteren, am Produktlebenszyklus beteiligten Disziplinen mit berücksichtigen muss. In diesen Disziplinen wird er in seiner Rolle jedoch keine Tätigkeiten verantworten, wobei er mit seinem übergreifenden Wissen über das Produkt in diesen Disziplinen unterstützend tätig werden kann. In dieser Broschüre werden wir die Tätigkeiten einer Systementwicklung anhand eines vereinfachten V-Modells vorstellen. Dabei werden wir uns auf den linken, den konstruierenden Ast des V-Modells konzentrieren.

Die Entwicklungstätigkeiten werden im Weiteren aus didaktischen Gründen in eine logische Reihenfolge gebracht. Dies soll aber nicht implizieren, dass ein solches wasserfallartiges Vorgehen durchgeführt werden muss. Ab einer gewissen Ebene wird es möglich sein, mehr iterativ-inkrementell oder sogar agil (besonders in der Softwareentwicklung) vorzugehen.

Die folgende Abbildung gibt nur einen Überblick über die wichtigsten Tätigkeiten im Systems-Engineering. In Kapitel 5.1 werden weitere Tätigkeiten eingeführt, die die folgende Abbildung vervollständigen.

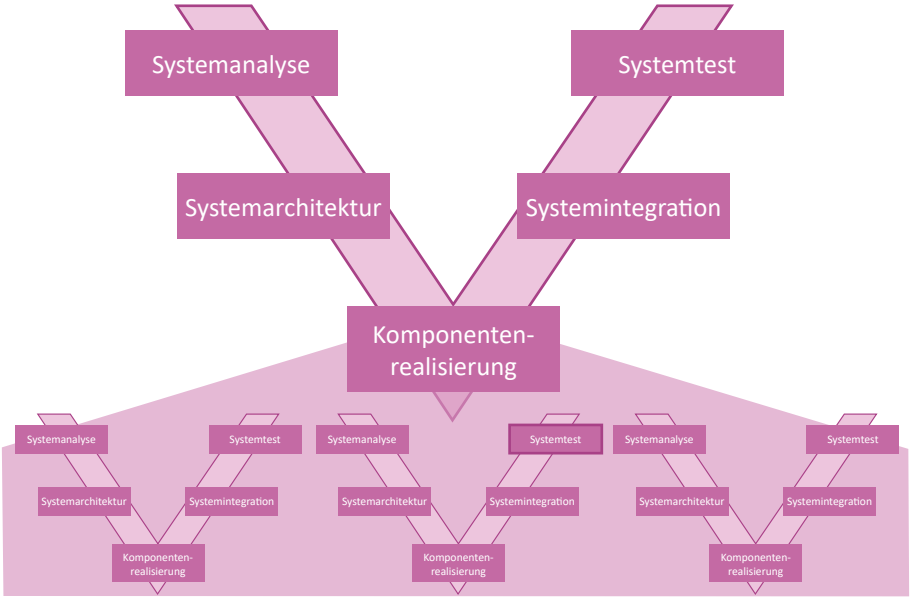


Abbildung 2.1: Grobe Einordnung der SE-Tätigkeiten

Laut Abbildung 2.1 werden auf dem linken Ast des V's zunächst eine Analyse und dann eine Architektur für das betrachtete System durchgeführt. Daraus folgen dann Komponenten, für die dann wiederum jeweils ein komplettes V durchlaufen wird.

Im besten Fall erhält der Systems-Engineer für die oberste Ebene die getesteten Komponenten zurück, die er dann zu dem gewünschten System integrieren kann. Den Abschluss der Systementwicklung bildet dann der Systemtest, oft auch als Qualifikations- oder Abnahmetest bezeichnet.

Im Allgemeinen muss das Systems-Engineering mehr als zwei Ebenen betrachten, da nicht immer aus der ersten Zerlegung die gewerksspezifischen Komponenten entstehen und damit die unterste Ebene in der Systembetrachtung erreicht ist. Die Realisierung der in der ersten Zerlegung gefundenen Komponenten würde dann wiederum mit Systems-Engineering-Methoden durchgeführt werden.

Aus dieser Betrachtung ergibt sich für den linken Ast des V-Modells eine Abfolge von Analyse- und Architekturschritten, bis eine Ebene von Komponenten erreicht wird, die

- entweder gewerksspezifisch ist und der entsprechenden Entwicklungsabteilung übergeben wird
- oder als komplexere Komponente einer anderen Organisationseinheit bzw. einem Zulieferer übergeben wird.

Abbildung 2.2 stellt diesen Zusammenhang für ein System über drei Ebenen graphisch dar.

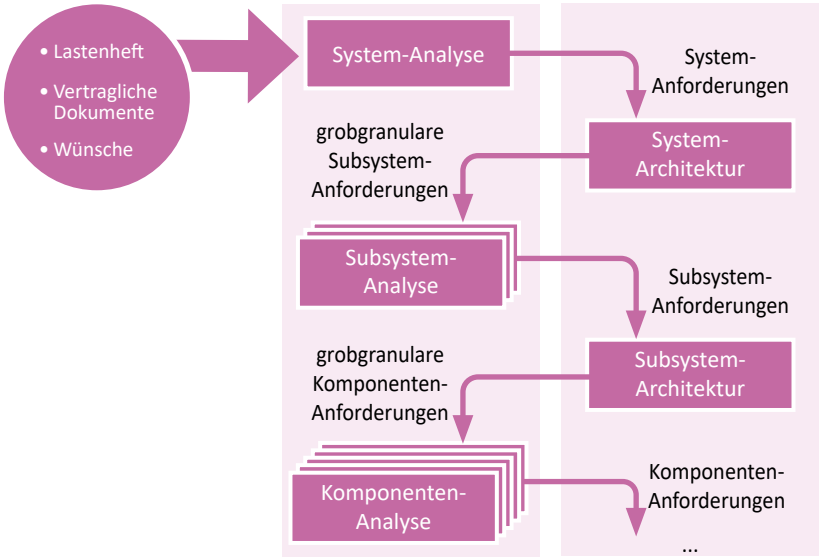


Abbildung 2.2: Abfolge von Analyse und Architektur auf mehreren Ebenen

Die Eingabe in den Prozess bilden Anforderungen, die von unterschiedlichen Anforderungsquellen, z. B. den Stakeholdern, geliefert werden können. Befinden wir uns auf Seiten eines Auftraggebers, z. B. eines OEMs (Original Equipment Manufacturer), so werden diese meist von einem Produktmanagement als Eingabe in den Entwick-

lungsprozess erstellt. Auf Seiten eines Zulieferers werden die Anforderungen von dem Auftraggeber vorgegeben. In beiden Fällen ist es üblich, diese Anforderungen der Entwicklung in Form eines Lastenhefts zur Verfügung zu stellen. Unabhängig davon, wie formal das Lastenheft gestaltet ist, kann nicht davon ausgegangen werden, dass dieses Lastenheft Anforderungen in der benötigten Qualität beinhaltet. Dies ist ein Grund, eine Analyse der gegebenen Anforderungen vorzunehmen. Auf Seiten eines Zulieferers kann ein weiterer Grund sein, die Anforderungen an das zu erstellende System mit Systemen zu vergleichen, die in früheren Projekten entwickelt wurden und so als Grundlage für das neue System dienen können.

Die Ausgabe der Analysephase sind belastbare Anforderungen an das System aus Sicht der Entwicklung. Sie werden im Allgemeinen in Form eines Pflichtenhefts notiert.

Mit diesem Pflichtenheft kann nun der erste Architekturschritt durchgeführt werden. Hierbei werden die Subsysteme identifiziert und die Schnittstellen zwischen ihnen festgelegt. Diese Schnittstellen folgen aus der dritten Aufgabe in der Architektur, der Verteilung der Systemanforderungen auf die Subsysteme. Es entstehen neue Anforderungen an die Subsysteme, die wiederum als Lastenheftanforderungen für die Subsysteme angesehen werden können. Sie bilden nun die Eingabe in den Analyseschritt auf der Subsystemebene, wobei dieselben Ziele wie auf Systemebene verfolgt werden. Der nachfolgende Architekturschritt führt zu Anforderungen an die Komponenten, die zur Entwicklung an die entsprechenden Organisationseinheiten übergeben werden können.

Obwohl der Systems-Engineer die Komponentenentwicklung nicht mehr verantwortet, so ist er daran zumindest begleitend beteiligt. Er beobachtet und sammelt Informationen, um zu entscheiden, ob sich eine der folgenden Änderungen ergeben:

- An den Anforderungen auf Systemebene: Änderungen in einer Komponente führen dazu, dass die anfangs gestellten Systemanforderungen nicht realisiert werden können. Die geänderten Anforderungen müssen abgeklärt, neu auf die Komponenten verteilt und die Schnittstellen angepasst werden.
- An den Anforderungen an einzelne Komponenten: Zwar meldet eine Komponente den Bedarf an Änderungen an ihren Anforderungen, diese können jedoch dadurch aufgefangen werden, dass die betroffene Systemanforderung anders auf die Komponenten aufgeteilt wird und bei Bedarf die Schnittstellen angepasst werden.

Das in Abbildung 2.2 dargestellte Wechselspiel zwischen Analyse und Architektur wird auch in dem sogenannten Twin-Peaks-Modell betrachtet [Nuseibeh01]. Wir stellen Ihnen dieses in dem folgenden Video vor. Eine weitere Beschreibung des Twin-Peaks-Modells und der Umgang mit Schnittstellen auf den verschiedenen Ebenen finden Sie in [Rupp20, Kapitel 23].

3. Systemanalyse

In diesem Kapitel werden wir einige Techniken einführen, die in dem ersten Schritt der Entwicklung, der Systemanalyse, eingesetzt werden. Durch ihn wird sichergestellt, dass die Anforderungen den Wünschen der Stakeholder entsprechen. Somit wird die Grundlage für die weitere Entwicklung des Systems gelegt.

3.1 Überblick

Definition Systemanalyse:

Die Systemanalyse hat zur Aufgabe, die von außen an das System gestellten Anforderungen in Anforderungen zu überführen, die den Bedürfnissen der weiteren Entwicklung genügen.

Die in der Systemanalyse entstehenden Anforderungen (im Folgenden auch „Systemanforderungen“) sollten dabei zumindest den folgenden Qualitätskriterien genügen:

- **Vollständigkeit**: Auf einer hohen, abstrakten Ebene müssen alle Anforderungen erfasst sein. Dabei müssen diese dann soweit detailliert sein, dass sie die Herausforderungen, die auf die weitere Entwicklung zukommen, beschreiben. Wird ein agileres Vorgehen angestrebt, so müssen zumindest die architekturrelevanten Anforderungen detailliert genug beschrieben sein.
- **Eindeutigkeit**: Da das Systems-Engineering im Allgemeinen eine verteilte Entwicklung bedeutet, sollten alle entstehenden Artefakte eindeutig dokumentiert werden. Wir schlagen u. a. die Verwendung der UML [RuQu12] oder der SysML [SysML] vor, da sie mit ihrer definierten Syntax und Semantik die notwendigen Voraussetzungen mitbringen.
- **Korrektheit**: Entgegen der klassischen Definition von Korrektheit meinen wir mit Korrektheit in dieser Broschüre an erster Stelle, dass sich die Anforderungen auf den richtigen Betrachtungsgegenstand beziehen müssen. Auch wenn der Anforderungsgeber eine Anforderung zum Beispiel an eine bestimmte Komponente gerichtet hat, so muss doch untersucht werden, ob nicht auch andere Komponenten von dieser Anforderung betroffen sind.

Anforderungen an ein System und dessen Entwicklung lassen sich in verschiedene Arten einteilen, die gerade in der Dokumentation unterschiedlich behandelt werden:

- Funktionale Anforderungen
- Qualitätsanforderungen
- Anforderungen an Daten
- Technische Randbedingungen
- Rechtlich-vertragliche Anforderungen

Für die drei konstruktiven Tätigkeiten (Ermittlung, Analyse und Dokumentation von Anforderungen), die der Systems-Engineer in dieser Phase durchführen muss, schlagen wir ein an die Use-Case-Analyse angelehntes Vorgehen vor (siehe auch [Cockburn00] und Abschnitt 3.3). Zunächst werden die abstraktesten Funktionen des Systems gesucht und dann sukzessive genauer untersucht. Dadurch entsteht eine Hierarchie von funktionalen Anforderungen, die um die Qualitätsanforderungen ergänzt werden, welche die jeweilige Funktion genauer beschreiben. Die anderen, oben beschriebenen Typen von Anforderungen werden in diesem Ansatz parallel betrachtet.

3.2 Anforderungen ermitteln

Bei der Ermittlung der Anforderungen kommt auf den Systems-Engineer eine besondere Herausforderung zu, da er die Schnittstelle zu den Stakeholdern des betrachteten Systems darstellt (vgl. Abbildung 2.2). Die Stakeholder können aus anderen Organisationen oder einer Fachbereichsabteilung kommen, die nicht immer die Sicht der Entwicklung auf das System haben.

Beim Ermitteln der Stakeholder-Anforderungen wird der Systems-Engineer durch zahlreiche Ermittlungstechniken unterstützt. Die Auswahl der geeigneten Technik hängt hierbei maßgeblich von gegebenen Randbedingungen (z. B. Verfügbarkeit oder inhaltliches Wissen der Stakeholder) und von der Art der zu ermittelnden Anforderungen ab. Angelehnt an [Sauerwein00] kann man dazu drei Arten von Anforderungen unterscheiden:

- **Basisfaktoren**: Was wird von dem System stillschweigend vorausgesetzt? Dies betrifft vor allem Funktionen, die von einem Vorgängersystem zur Verfügung gestellt wurden und deswegen häufig gar nicht mehr explizit angesprochen werden.
- **Leistungsfaktoren**: Was fordern die Stakeholder explizit von dem System? Diese Anforderungen bilden häufig den Auslöser für eine neue Entwicklung.
- **Begeisterungsfaktoren**: Was kann das System noch tun, woran der Stakeholder noch gar nicht gedacht hat. Seien Sie an der Stelle kreativ und suchen Sie Anforderungen, die Ihr neues System von den Marktbegleitern abhebt.

Manchmal stehen einem Systems-Engineer als Eingabe auch nur Ziele zur Verfügung, die sich an dem bestehenden Markt orientieren. Aus diesen Zielen müssen nun die Anforderungen an das System hergeleitet werden. Hierbei hilft die QFD-Methode (Quality-Function-Deployment), die systematisch die Ziele in Systemanforderungen überführt, eine Priorisierung durchführt und die Auswirkungen der neuen Anforderungen betrachtet (siehe [Akao92])

Als zweites Hilfsmittel werden hier Regeln zur Untersuchung von natürlichsprachlichen Beschreibungen genutzt (das REgelwerk). Da die Anforderungen aus einem Produktmanagement oder von einem externen Kunden meist in natürlicher Sprache dokumentiert sind, bietet diese Art der Untersuchung ein großes Potenzial, um auf sogenannte Defekte in den gegebenen Anforderungen zu kommen [Rupp20, Kapitel 9].

Die dritte hier genannte Unterstützung wird durch eine geregelte Art der Dokumentation gebildet, durch die Defizite im Wissen des Systemanalytikers bewusst werden. Abschnitt 3.4 wird darauf eingehen.

Neben diesen drei Arten Anforderungen zu ermitteln, kommen in der Systemanalyse auch einige der aus dem klassischen Requirements-Engineering bekannten Ermittlungstechniken [Rupp20, Kapitel 8] zum Einsatz. Gerade die dokumentbasierten Techniken und die Beobachtungstechniken versprechen einen Wissensgewinn für den Systemanalytiker.

3.3 Anforderungen analysieren

Nachdem die Anforderungen Ihrer Stakeholder (im Folgenden Ursprungsanforderungen) ermittelt wurden, müssen aus diesen nun Anforderungen hergeleitet werden, die gut genug sind, um als Basis für Entwicklung und Test zu dienen.

Der Erfahrung nach besteht ein häufiges Manko in den Ursprungsanforderungen darin, dass sie mehr fordern als das, was Ihr Entwicklungsgegenstand, Ihr System, zu leisten vermag. Deswegen sollten Sie besonderes Augenmerk bei der Analyse der Ursprungsanforderungen darauf legen, was Sie daraus für Ihr System herleiten. Die abgeleiteten Anforderungen bezeichnen wir im Folgenden als Systemanforderungen.

Diese Systemanforderungen können sich von ihren Ursprungsanforderungen unterscheiden, weil Sie

- Freiheiten in der Interpretation in den Ursprungsanforderungen ausgenutzt haben,
- weitergehende Festlegungen getroffen haben,
- Anpassungen in den Anforderungen vornehmen mussten oder
- fehlende Anforderungen ergänzen mussten.

Diese (und noch viele andere) Gründe führen zu der Notwendigkeit, die gefundenen Systemanforderungen von den Stakeholdern prüfen zu lassen und bei Bedarf Unstimmigkeiten zu beheben. Für diese beiden Tätigkeiten verweisen wir auf [Rupp20, Kapitel 14, 15].

Die primäre Herleitung der Systemanforderungen durch eine Analyse der Ursprungsanforderungen kann durch die Einordnung der Systemanforderungen in eine gut strukturierte Anforderungsspezifikation unterstützt werden. Bei einer solchen Spezifikation können wir, bis auf ein paar Sonderfälle, davon ausgehen, dass jede Systemanforderung eine andere Anforderung verfeinert. Eine Ausnahme bildet dabei die abstrakteste Ebene von Systemanforderungen.

Diese Anforderungen stehen nebeneinander und bilden die Basis für die verfeinernden Anforderungen. Somit ergeben sich für die Strukturierung der Anforderungen viele Bäume (mathematisch als „Wald“ bezeichnet), wobei die Wurzeln durch die abstraktesten Anforderungen gebildet werden. Die Blätter repräsentieren die Anforderungen, die nicht mehr verfeinert werden.

Bei einem Use-Case-basierten Ansatz repräsentieren die Wurzeln der Anforderungsbäume dabei entweder

- die Use-Cases des Systems oder
- die Kategorien der nicht-funktionalen Anforderungen, die nicht den funktionalen Anforderungen zugeordnet werden können.

Ein unvollständiges Beispiel dazu ist in der folgenden Abbildung gegeben.

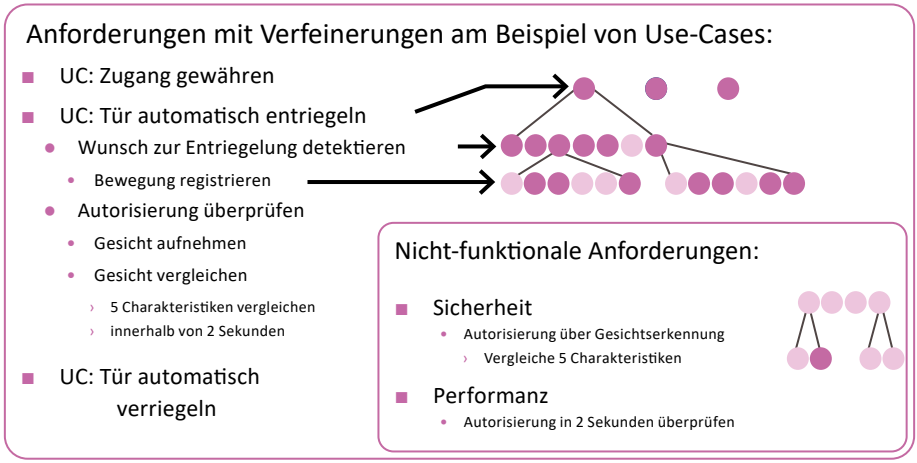


Abbildung 3.1: Einordnung von Systemanforderungen

3.4 Anforderungen dokumentieren

Bei der Dokumentation der Systemanforderungen gehen wir in dieser Broschüre von einer Mischung zwischen natürlichsprachlichen Anforderungen und einer modellbasierten Darstellung in SysML aus. Im Regelfall reicht für die modellbasierte Dokumentation eine solche semiformale Notation aus. In sehr sicherheitskritischen Anwendungsbereichen (Militärischer Bereich, Teile eines Fahrzeugs), in denen z. B. ausschließlich zertifizierter Programmcode eingesetzt werden darf, können auch formalere Dokumentationstechniken wie temporale Logik oder Prädikatenlogik [Barwise05] eingesetzt werden.

Klassischerweise wird die Analyse mit einer Systemabgrenzung begonnen. Das hier vorgestellte weitere Vorgehen in der Analyse richtet sich nach der im vorigen Abschnitt eingeführten Use-Case-basierten Analyse. Dadurch können wir die in Use-Case- und Aktivitätsdiagrammen dokumentierten funktionalen Anforderungen als „führend“ ansehen.

Die natürlichsprachlichen Anforderungen zur Präzisierung der so notierten funktionalen Anforderungen werden in das entstehende SysML-Modell integriert. Ergänzt werden diese Anforderungen durch die Modellierung eines Informationsmodells, in dem die Anforderungen an die Daten beschrieben werden. Abgerundet wird das Bild durch die Einführung von Zustandsautomaten, um entweder die wichtigen Systemzustände oder Zustände wichtiger Entitäten aus unserem System darzustellen.

Dieses Vorgehen ist mittlerweile zum Standard bei der Dokumentation von Anforderungen geworden. In [ALM16] werden auch die Notationselemente vorgestellt, die man typischerweise bei der modellbasierten Dokumentation von funktionalen Anforderungen einsetzt. Dabei geht man davon aus, dass alle in einem Modell notierten Informationen als Anforderungen gelten und nicht zusätzlich als natürlichsprachliche Anforderungen dokumentiert werden müssen. Die Dokumentation der weiteren Anforderungstypen (siehe Abschnitt 3.1) wird im Regelfall natürlichsprachlich und nicht im Modell durchgeführt. Dabei können Schablonen zur Formulierung guter natürlichsprachlicher Anforderungen verwendet werden. Da diese jedoch nicht spezifisch für das Systems-Engineering sind, sondern für die Formulierung von Anforderungen im Allgemeinen gültig sind, werden wir diese hier nicht weiter vorstellen sondern verweisen auf [Rupp20, Kapitel 19].

Wir stellen Ihnen im Folgenden kurz einige Diagrammtypen vor, die bei einem Use-Case-basierten Ansatz zum Einsatz kommen können. Abweichungen und Ergänzungen (z.B. die Modellierung von Zustandsautomaten) können je nach Anwendungsdomäne natürlich immer sinnvoll sein. Bezüglich der hier dargestellten Beispiele gilt, dass sie keinen Anspruch auf Vollständigkeit erheben.

Block-Definitions -Diagramm für den Systemkontext

Definition Technischer Kontext:

Der technische Kontext zeigt die mit dem betrachteten System in Beziehung stehenden Nachbarsysteme und Benutzer.

Diese Einordnung wird im Allgemeinen entweder aus einer darüber liegenden Architektur oder von den Stakeholdern vorgegeben. Es ist eine sehr einfache Sicht auf das System, in ihr können aber gut sowohl die Nachbarsysteme und Anwender des betrachteten Systems als auch dessen Ein- und Ausgabedaten dargestellt werden. Im Rahmen der Analyse von technischen Systemen wird häufig auf ein eigenes Kontextdiagramm in der Analyse verzichtet. Es wird vielmehr ein Diagramm genutzt, dass der Systemarchitektur zugeordnet wird. Das „Black-Box-Diagramm“ (siehe Abbildung 4.1, Abschnitt 4.1) beinhaltet die oben beschriebenen Informationen, jedoch in einer sehr präzisen Art und Weise.

Use-Case-Diagramm für Systemprozesse

Aus der Literatur [Cockburn00] leitet sich die folgende Definition für einen Use-Case ab, wobei mit dem Begriff Akteur ein Benutzer des Systems oder ein Nachbarsystem gemeint ist:

> **Definition Use-Case:**
>
> Ein Use-Case ist eine typische Interaktion eines Akteurs mit dem System mit einem fachlichen Wert.

Aus dieser Definition folgen zwei wichtige Kriterien für Use-Cases, aus denen sich die weitere Vorgehensweise ableitet.

- Einem Use-Case liegt ein **Ablauf** zugrunde. Deswegen werden wir die Use-Cases mit Ablaufdiagrammen genauer beschreiben.
- Ein Use-Case dient einem **fachlichen Ziel**. Die geforderte Funktionalität folgt aus einem übergeordneten Prozess. Dies kann ein Geschäfts- oder Behandlungsprozess sein, den ein Anwender unter anderem mit dem betrachteten System durchführt. Oder es ist ein Prozess, der von dem Supersystem durchgeführt wird und zu dem das betrachtete System nur einen Teil zuliefert.

Als Ergebnis der Herleitung von Use-Cases wie in Abschnitt 3.3 entsteht ein Use-Case-Diagramm mit den für das System relevanten Systemprozessen.

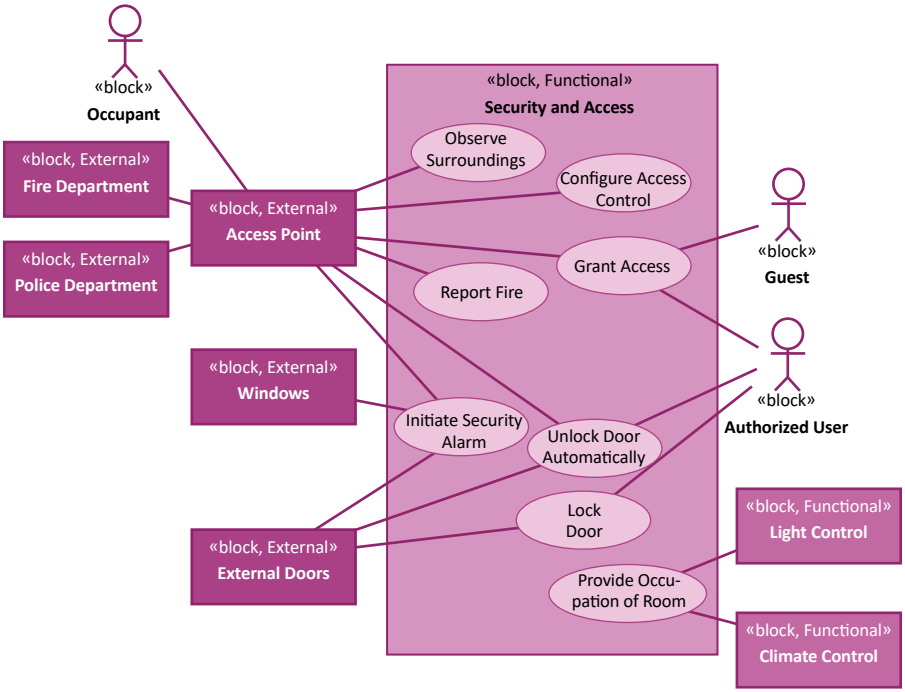


Abbildung 3.3: Use-Case-Diagramm mit den Systemprozessen des SHS

Bitte beachten Sie, dass in Abbildung 3.3 nur ein Teil der Systemprozesse des SHS dargestellt ist. In der Abbildung betrachten wir nur den Teil, der sich mit der Sicherheit und dem Zugang des Hauses beschäftigt. Für andere Bereiche der Unterstützung wie z.B. Komfort, Licht- oder Klimasteuerung existieren weitere Use-Case-Diagramme.

Aktivitätsdiagramm als Ablaufbeschreibung

Definition Aktivitätsdiagramm:

Aktivitätsdiagramme bringen die Benutzereingaben und Systemaktionen für einen Systemprozess in einen Ablauf.

Bei der Erstellung der Aktivitätsdiagramme können sehr komplexe Abläufe entstehen. Versuchen Sie, die einzelnen Aktionen in Ihren Diagrammen hierarchisch zu gliedern, um auf einer Ebene in einem Diagramm die Übersichtlichkeit zu gewährleisten. Bezüglich der in der Systemanalyse angestrebten Detaillierung für die funktionalen Anforderungen gilt hier:

- Die Ursprungsanforderungen sollten sinnvoll den Schritten im Ablauf zugeordnet werden können.
- Alle Anforderungen, die Sie als Requirements-Engineer aus der fachlichen Domäne heraus der weiteren Entwicklung vorgeben möchten, sollten ebenfalls sinnvoll einem Schritt im Ablauf zugeordnet werden können.

Damit kann eine Vielzahl von Aktivitätsdiagrammen entstehen, die entweder direkt einem Systemprozess oder als verfeinerter Ablauf einer Aktion in einem solchen Diagramm zugeordnet sind.

Abbildung 3.4 zeigt den Ablauf des Systemprozesses „Unlock Door Automatically“, wobei zwei Aktionen („Compare Face“, „Lock Door“) durch detailliertere Aktivitätsdiagramme verfeinert werden.

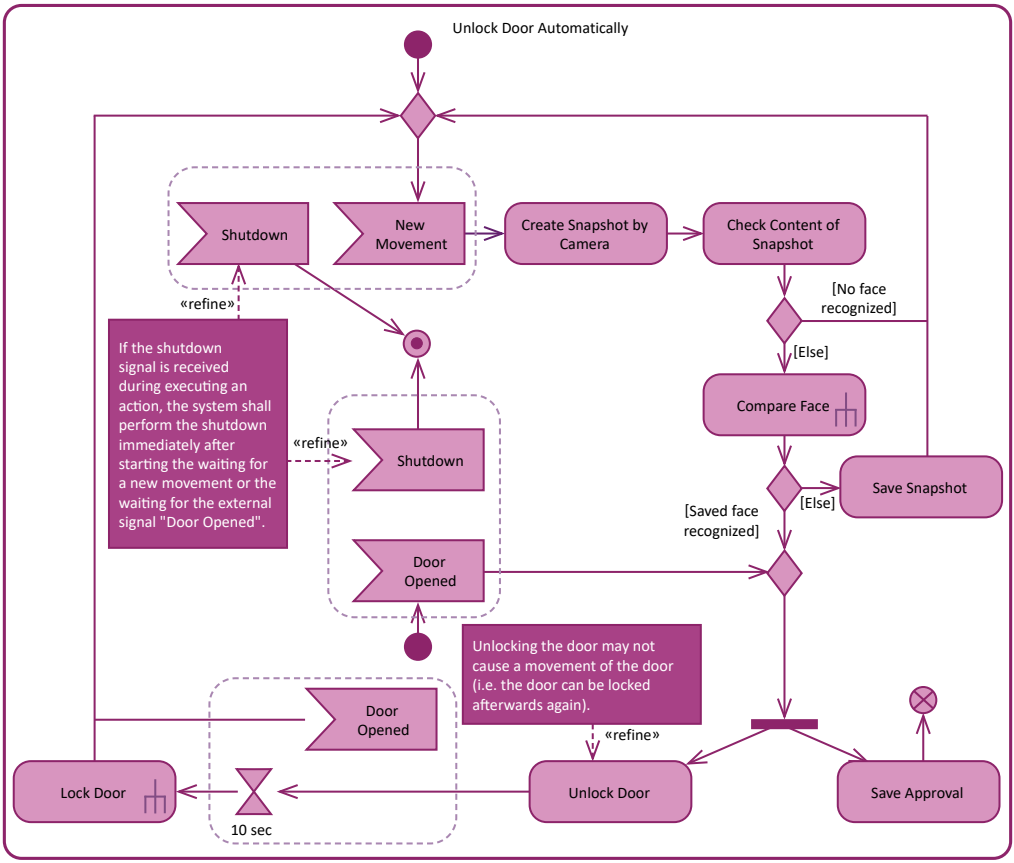


Abbildung 3.4: Aktivitätsdiagramm für einen Systemprozess

Zusätzlich zu den Abläufen sind natürlichsprachliche Anforderungen dargestellt, die Modellelemente in den Diagrammen z. B. durch Qualitätsanforderungen weiter beschreiben können.

Einige der möglichen Abläufe sind zur besseren Nachvollziehbarkeit in dem folgenden Verweis animiert dargestellt.

Block-Definitions-Diagramm als Informationsmodell

Definition Informationsmodell:

In einem Informationsmodell werden die fachlich relevanten Begriffe zusammen mit ihren fachlich relevanten Daten definiert.

Beide Anteile des Informationsmodells stellen wiederum Anforderungen dar. In Anforderungen, in denen z. B. der Begriff „Room Climate“ auftaucht, müsste dieser durch die Betrachtung der auswählbaren Optionen präzisiert werden. Dies kann dadurch erspart werden, dass diese speziellen Optionen im Informationsmodell modelliert werden. Somit kann ein Informationsmodell Anforderungen enthalten und gleichzeitig als andere Darstellungsform eines Glossars angesehen werden.

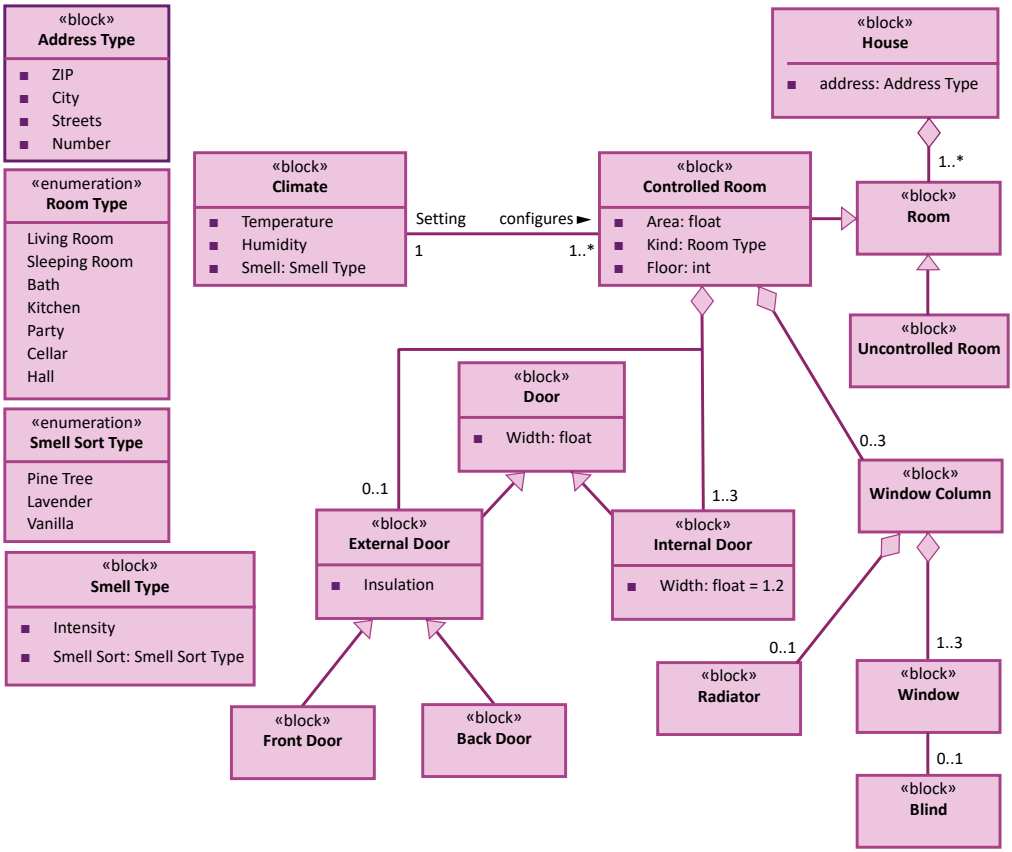


Abbildung 3.6: Informationsmodell des SHS

4. Systemarchitektur

Die Systemarchitektur bildet den ersten Schritt der Realisierung und ist somit die erste konsumierende Disziplin der zuvor definierten Anforderungen.

Definition Kernaufgaben der Systemarchitektur:

Unter dem Begriff der Systemarchitektur werden die folgenden Kernaufgaben zusammen gefasst:

- Die Bestandteile des Systems zu identifizieren bis auf die kleinste Ebene, die für die Betrachtung im Rahmen des Systems-Engineerings Relevanz hat
- Für diese Bestandteile die Anforderungen aus den Systemanforderungen abzuleiten
- Die Schnittstellen zwischen den Bestandteilen festzulegen

Ähnlich wie in der Systemanalyse können wir auch die in der Systemarchitektur durchgeführten Aktivitäten drei groben Tätigkeiten zuordnen: dem Finden, dem Dokumentieren und dem Überprüfen der Systemarchitektur.

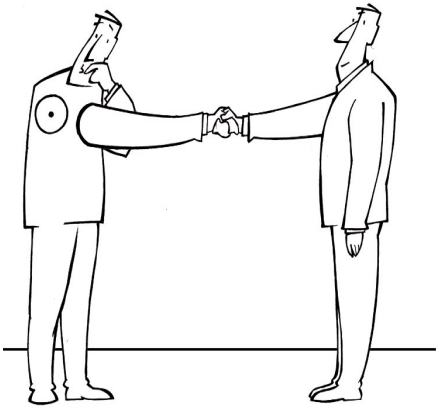
Das Finden einer geeigneten Architektur ist im Allgemeinen ein sehr kreativer Prozess, der sehr von der Erfahrung der beteiligten Entwickler abhängt und oftmals auf alte Produkte mit ihren Architekturen aufsetzt. Unterstützt wird diese Tätigkeit durch das Bilden verschiedener Sichten auf das System. Da diese Tätigkeiten immer sehr produktspezifisch sind, werden wir darauf im Rahmen dieser Broschüre nicht weiter eingehen.

Zum Überprüfen der getroffenen Entscheidungen existieren einige Ansätze, die in die folgenden drei Gruppen eingeteilt werden können:

- Ein **Review** mit einem Verantwortlichen der gefundenen Komponenten sollte auf jeden Fall durchgeführt werden. Es stellt sicher, dass die Kommunikation der Entscheidungen zu den Entwicklern, die auf diesen Entscheidungen ihre Entwicklungen aufbauen, stattfindet.
- Eine **Bewertung** der aktuellen Architektur kann u. A. mit Hilfe der QFD ([Akao92]) durchgeführt werden, um z. B. die Stabilität Ihrer Architektur gegenüber Änderungen der Marktanforderungen zu überprüfen. Um spezielle Aspekte wie z. B. das Datenaufkommen an Schnittstellen zu untersuchen, können auch Techniken aus der Softwarearchitektur (z. B. statische und dynamische Kopplung) verwendet werden [Starke14]. Auch eine FMEA kann als überprüfende Technik angesehen werden, da sie eine bestehende Architektur auf Schwachstellen durch die Betrachtung der Fehlerauswirkungen überprüft [Pfeufer14].
- **Formale Ansätze** dienen zur Überprüfung, ob die gewählte Systemarchitektur spezielle Aspekte aus den Anforderungen erfüllt [Pflüger14]. Diese Aspekte

beziehen sich hauptsächlich auf die Qualitätsanforderungen wie Bandbreite eines Busses, Stromaufnahme des Systems oder auch die benötigte Rechenleistungen der Prozessoren.

Für die Dokumentation der Systemarchitektur werden in dieser Broschüre verschiedene Sichten definiert, die jeweils einen Ausschnitt der in der Architektur relevanten Informationen bzw. Ergebnisse darstellen. Es wird zunächst das Kontextdiagramm aus Abschnitt 3.4.1 in eine formalere Darstellung, die **Black-Box-Sicht**, überführt (Abschnitt 4.1). Für die Darstellung der Zerlegung des Systems führen wir die **Physikalische Sicht** ein (Abschnitt 4.2), in der wir das System in seine Komponenten eindeutig zerlegen. Eine weitere wichtige Sicht ist die **Kollaborationssicht** (Abschnitt 4.3). Sie hilft Ihnen bei der Identifikation der Komponenten, die an der Realisierung der Systemanforderungen, zum Beispiel an einem Systemprozess, beteiligt sind. Die daraus folgenden Anforderungen an die Komponenten und die Verfolgbarkeit zu den Systemanforderungen können Sie detailliert in einer **Zuweisungssicht** (Abschnitt 4.4) darstellen.



Prinzipiell richtet sich die Wahl der benötigten Sichten nach Ihrem System und dem von Ihnen gewählten Vorgehen. Deshalb können die hier angesprochenen Sichten nur als Vorschlag verstanden werden. Eine andere Art der Definition der Sichten findet sich in [Pohl et al.12]. Hier wird vor die Definition der physikalischen Sicht die Definition einer logischen Sicht gestellt. Dies hilft bei der Betrachtung sehr großer, komplexer System, die aus einer Vielzahl von Komponenten bestehen, oder auch bei der Betrachtung von Systemen, die zwar die immer wiederkehrende, gleiche Grundstruktur besitzen, deren Teile jedoch unterschiedlich auf unterschiedliche physikalische Einheiten (z. B. Steuergeräte in einem Fahrzeug) verteilt werden müssen.

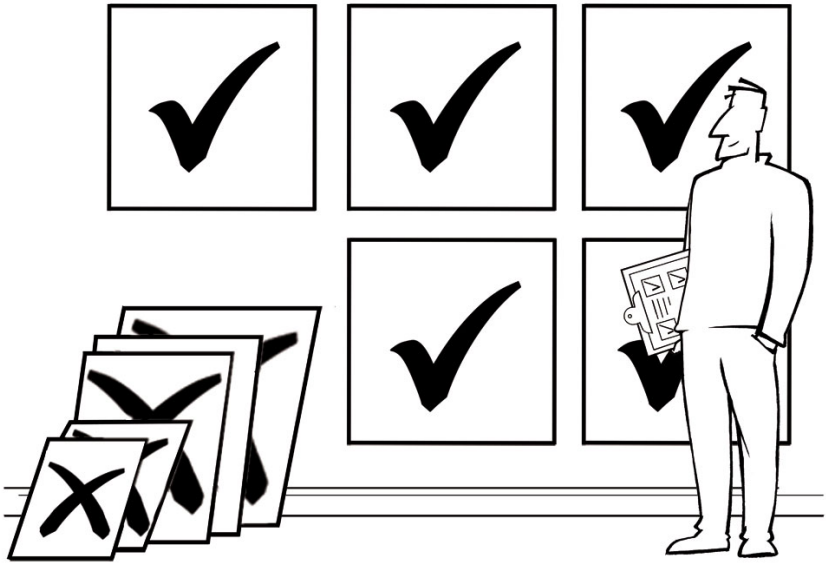
4.1 Black-Box-Sicht

Definition Black-Box-Sicht:

Die Black-Box-Sicht stellt das System in seinem technischen Kontext zusammen mit seinen Schnittstellen dar.

Die benötigten Informationen für dieses Diagramm sind im Wesentlichen die Schnittstellen des Systems. Sie können in zwei Gruppen unterteilt werden:

- Schnittstellen, die für die Entwicklung in dem Lastenheft oder aus einer darüber liegenden Systemarchitektur vorgegeben sind.
- Schnittstellen, die während der Entwicklung festgelegt werden.



Die Schnittstellen der ersten Kategorie können direkt zu Beginn der Entwicklung modelliert werden. Das so entstandene Diagramm sollte während der gesamten Entwicklung immer wieder angepasst werden, um Änderungen direkt zum Auftraggeber oder zu dem Systemarchitekten des Supersystems kommunizieren zu können.

Als Ergebnis liegt ein Block-Definition-Diagramm wie in Abbildung 4.1 vor.

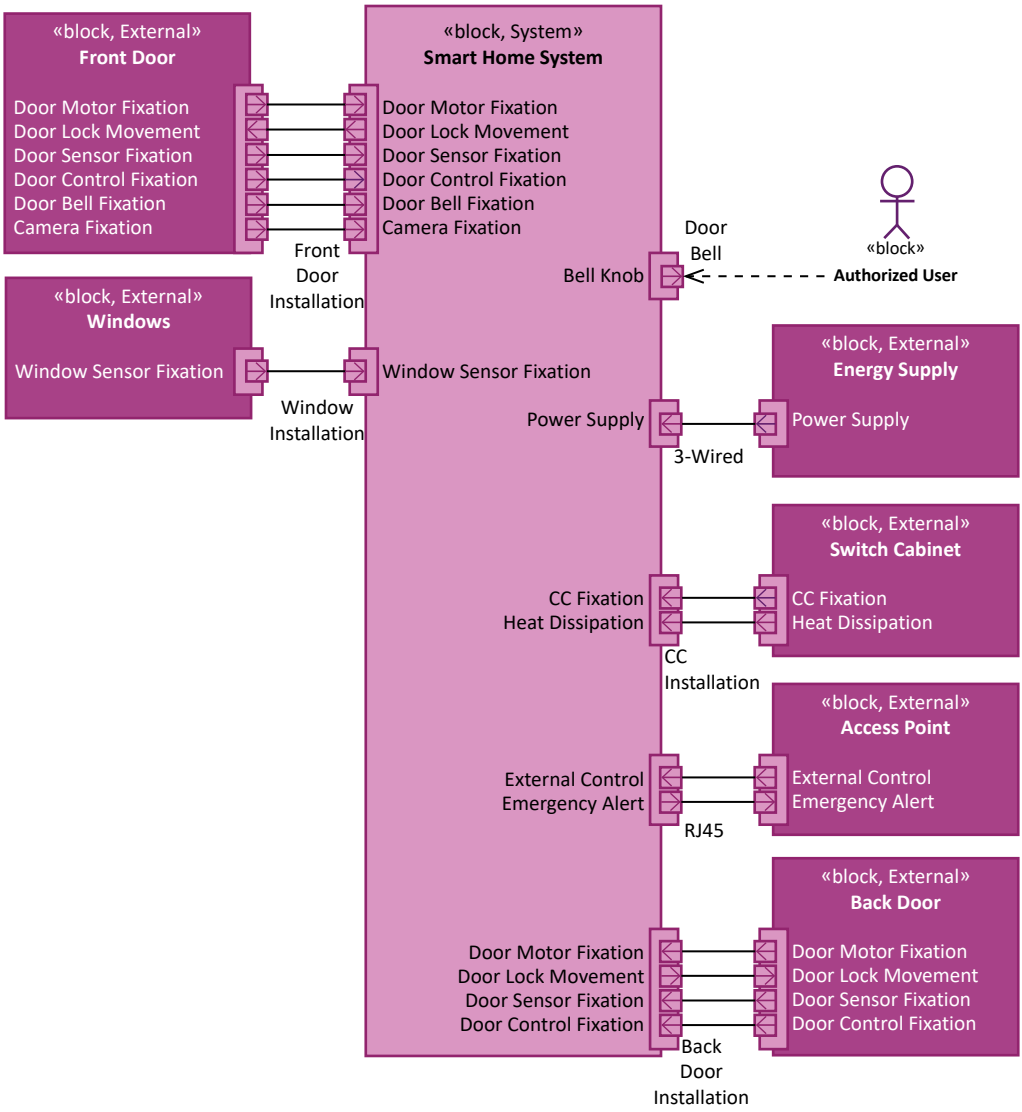
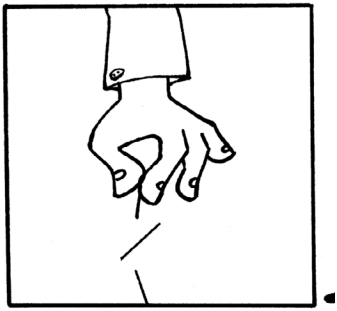


Abbildung 4.1: Black-Box-Sicht des SHS

4.2 Physikalische Sicht

Definition Physikalische Sicht:

In der physikalischen Sicht werden die Bestandteile eines Systems inklusive ihrer Schnittstellen und Anforderungen dargestellt.



Das Ziel dieser Sicht ist es, die Zerlegung des Systems bis hinunter zu den kleinsten, in der Systemarchitektur betrachteten Komponenten in einer Baumstruktur zu beschreiben. Damit ist die Einordnung einer Komponente eindeutig in eine, eventuell mehrstufige, Hierarchie gegeben. Dadurch bietet sich diese Sicht auch für die Strukturierung aller Dokumentationen von der Beschreibung der Systemanforderungen über die Systemarchitektur bis hin zu den Anforderungen an die Komponenten an (siehe Abschnitt 5.3).

In einem Block-Definitions-Diagramm (Abbildung 4.2) werden die Bestandteile des SHS über mehrere Ebenen hinweg definiert.

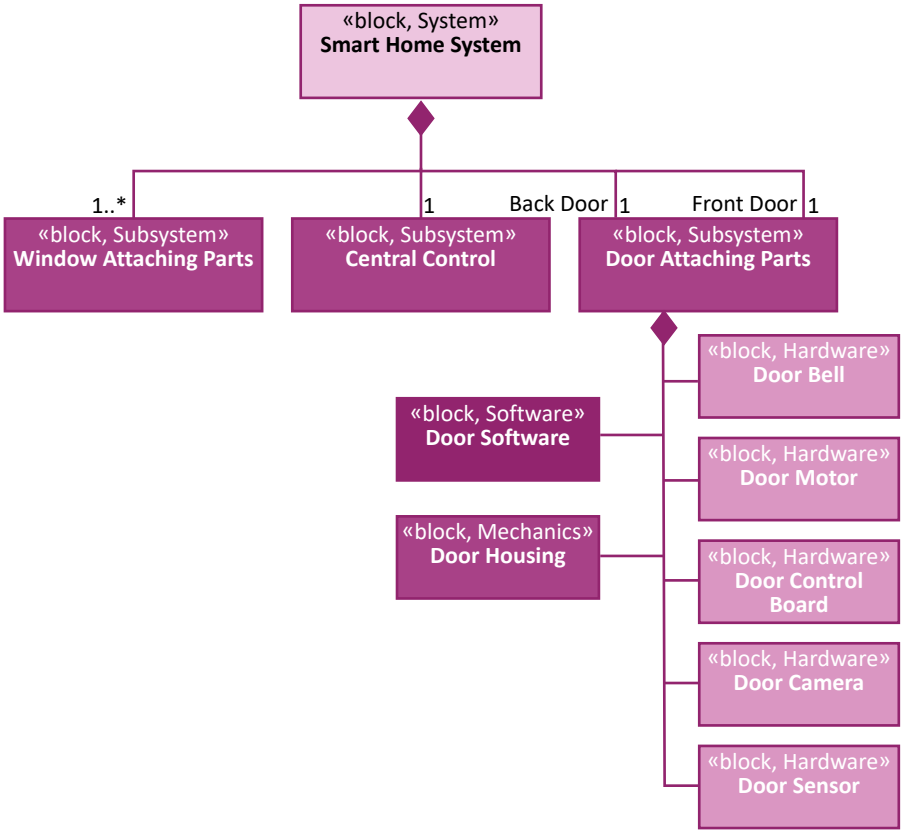


Abbildung 4.2: Block-Definitions-Diagramm des SHS

Dieses Diagramm bildet die Grundlage für das wesentlich interessantere Diagramm, das Internal-Block-Diagramm, das die folgenden, zusätzlichen Informationen beinhaltet:

- Interne Schnittstellen der Subsysteme untereinander
- Verantwortlichkeiten für die Realisierung der externen Systemschnittstellen

Abbildung 4.3 zeigt auszugsweise beide Informationen für die erste Ebene der Zerlegung des SHS. Entsprechende Diagramme müssen für die weitere Zerlegung der Subsysteme erzeugt werden.

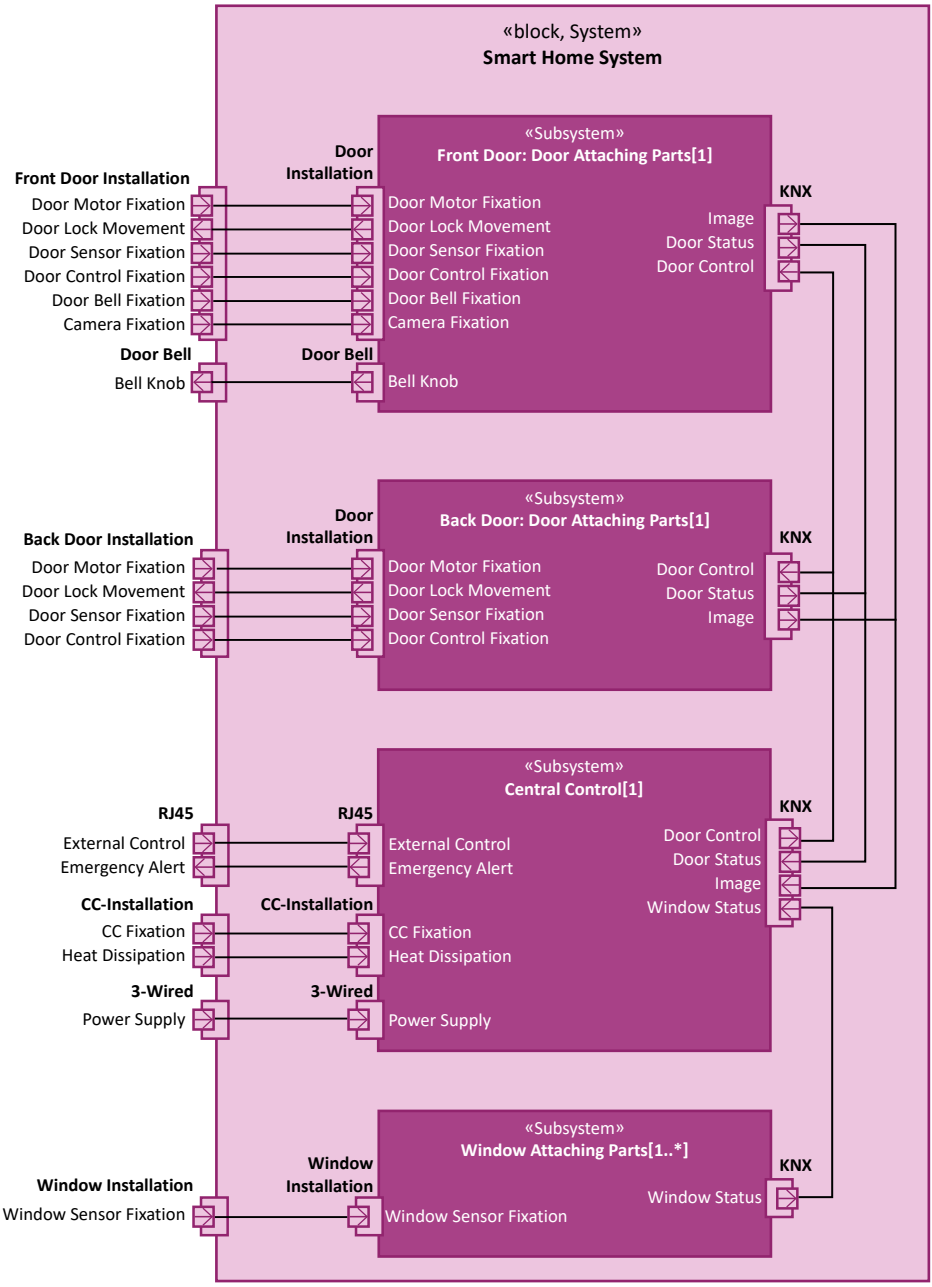


Abbildung 4.3: Internes Block-Diagramm des SHS

4.3 Kollaborationssicht

Definition Kollaborationssicht:

Die Kollaborationssicht stellt die Komponenten des Systems mit ihren Schnittstellen dar, die bei der Realisierung einer Funktion zusammenarbeiten müssen.

Dieser Inhalt dieser Sicht ist dazu prädestiniert, eine zentrale Aufgabe der Architektur zu unterstützen: die Zerlegung der geforderten Systemanforderungen in Anforderungen an die beteiligten Komponenten. Dabei können bei Bedarf ebenfalls die einer Systemfunktion zugeordneten nicht-funktionalen Anforderungen zerlegt und den Komponenten zugeordnet werden.

Bei der Erstellung dieser Sicht werden Sie die Komponenten und ihre Schnittstellen verwenden können, die bereits in der physikalischen Sicht definiert wurden. Sollten diese nicht ausreichend sein, so können Sie in diesem Schritt natürlich neue Komponenten und neue Schnittstellen definieren. Diese sollten Sie allerdings ebenfalls in der physikalischen Sicht hinzufügen, um so die dort gewünschte Vollständigkeit zu erreichen.

Welche Systemfunktionen Sie auf welchem Abstraktionsgrad in der Kollaborationssicht betrachten, hängt von der Komplexität der Systemfunktionen und damit der Anzahl der beteiligten Komponenten ab.

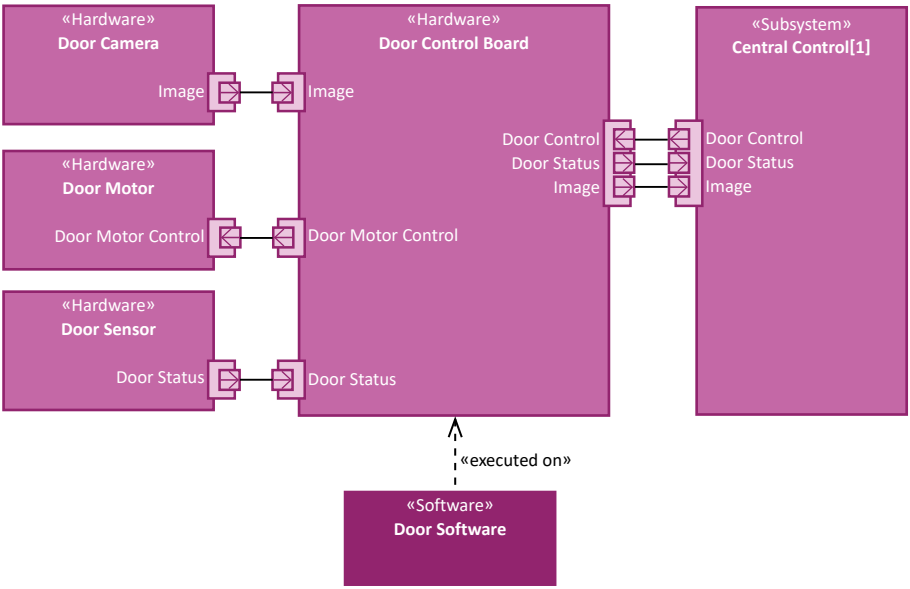


Abbildung 4.4: Statische Kollaboration von Komponenten

4.4 Zuweisungssicht

Definition Zuweisungssicht:

In der Zuweisungssicht werden die aus den Systemanforderungen abgeleiteten Anforderungen einer Komponente zugewiesen und die Verbindung zu den Systemanforderungen dargestellt.

Während in der zuvor beschriebenen Kollaborationssicht der Fokus auf die Realisierung einer Systemanforderung bzw. –funktion gelegt wurde, werden in dieser Sicht die daraus resultierenden Anforderungen komponentenspezifisch dargestellt. Sie erhalten so einen Überblick über alle Anforderungen, die Sie einer Komponente zugewiesen haben. Hierdurch können Sie Widersprüche oder Dopplungen in den Anforderungen erkennen und erhalten so eine hohe Qualität an Vorgaben an die jeweilige Komponente.

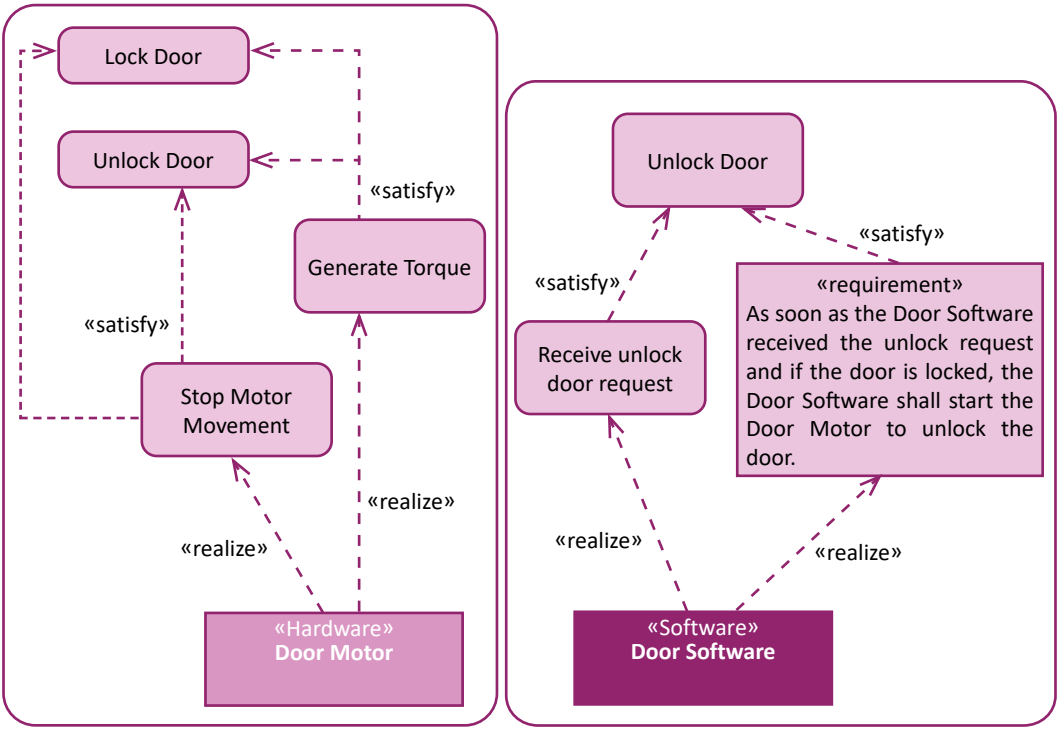


Abbildung 4.5: Darstellung der Komponentenanforderungen

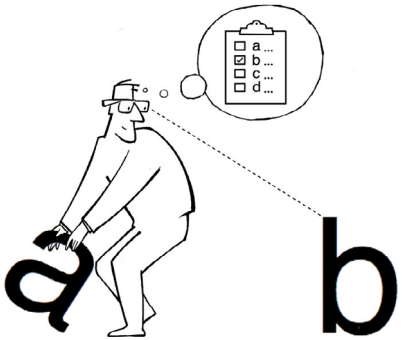
Die in Abbildung 4.5 aufgezeigte Verfolgbarkeit zu den Systemanforderungen unterstützt zwei wichtige Tätigkeiten. Zum einen wird der dargestellte Architekturschritt an sich unterstützt, indem Sie überprüfen können, ob Sie alle (wichtigen oder kritischen) Anforderungen des Systems auf die Komponenten verteilt haben. Zum anderen werden Sie in der späteren Entwicklung diese Verfolgbarkeit bei einer Impact-Analyse ausnutzen können: Falls sich eine Systemanforderung ändert, können Sie die Auswirkungen auf die daraus abgeleiteten Komponentenanforderungen analysieren.

5. Weitere Aspekte des Systems-Engineerings

5.1 Weitere Tätigkeiten in der Entwicklung

Wir haben in dieser Broschüre mehr Wert auf den linken, den konstruierenden Ast des Vs gelegt, da hier die Weichen für eine erfolgreiche Systementwicklung gestellt werden. Nichtsdestotrotz ist der Systems-Engineer auch an weiteren Tätigkeiten im Systementwicklungsprozess beteiligt.

Die erste Aufgabe nach der Entwicklung der Komponenten besteht in der Integration dieser Komponenten zu den Subsystemen bzw. zu dem System. Hierbei muss sichergestellt werden, dass die entsprechenden Betrachtungsgegenstände rechtzeitig für die nachfolgenden Tests in der benötigten Qualität zur Verfügung gestellt werden. Die eigentliche Integration wird häufig von den einzelnen Gewerken durchgeführt. So werden zum Beispiel die Software-Entwickler für die Integration ihrer Software mit der entsprechenden Hardware sorgen, und die Mechanik-Abteilung wird diese Hardware in ein Gehäuse einfügen.



Auch bei den Tätigkeiten, die sich mit dem Testen der Entwicklungsgegenstände beschäftigen, ist der Systems-Engineer nur indirekt beteiligt. Er muss jedoch die Qualität der folgenden Ergebnisse aus diesen Tätigkeiten sicherstellen:

- Testkonzept: Wird der Entwicklungsgegenstand durch das im Testkonzept definierte Vorgehen ausreichend getestet?
- Testfälle: Erlauben die definierten Testfälle, die im Testkonzept definierten Ziele zu erreichen?
- Testdurchführung: Wurden alle Tests entsprechend erfolgreich durchgeführt?

Der Systems-Engineer sollte aber auch die für das Systems-Engineerings eingesetzten Aufwände im Auge behalten und dem Gewinn, den es für den gesamten Entwicklungsprozess bringt, gegenüberstellen. Dazu erscheint eine Metrik über die Änderungsanträge (Change Requests) am vielversprechendsten. Verfolgen Sie, auch über vergleichbare Projekte hinweg, ob sich die Anzahl der Änderungen verringern. Hierbei sollten Sie zwischen zwei Arten von Verursachern für die Änderungen unterscheiden:

- Änderungen durch die Stakeholder
- Änderungen durch die Realisierung

Die erste Ursache kann nicht vollständig durch eine verbesserte Analysephase ausgeschlossen werden, da sich Änderungswünsche bei den Stakeholdern immer ergeben können. Demnach sollte durch das Systems-Engineering gerade die zweite Ursache von Änderungen minimiert werden. Damit haben Sie gezeigt, dass die frühe und komponentenübergreifende Sichtweise auf das System einen Vorteil für die Entwicklung bringt.

5.2 Einordnung von Systems-Engineering in den Lebenszyklus des Systems

Bisher haben wir über die Beteiligung des Systems-Engineerings an einem Entwicklungsprozess gesprochen, ohne diesen jedoch in einen übergeordneten Prozess einzuordnen, der alle Phasen im Lebenszyklus eines Systems beschreibt. Solche Prozesse sind im Allgemeinen sehr unternehmensspezifisch ausgestaltet, ähneln sich jedoch auf einer abstrakten Ebene. In den frühen Phasen ist aber immer wieder eine Unterscheidung zwischen den Prozessen bei einem OEM und einem Zulieferer zu erkennen.

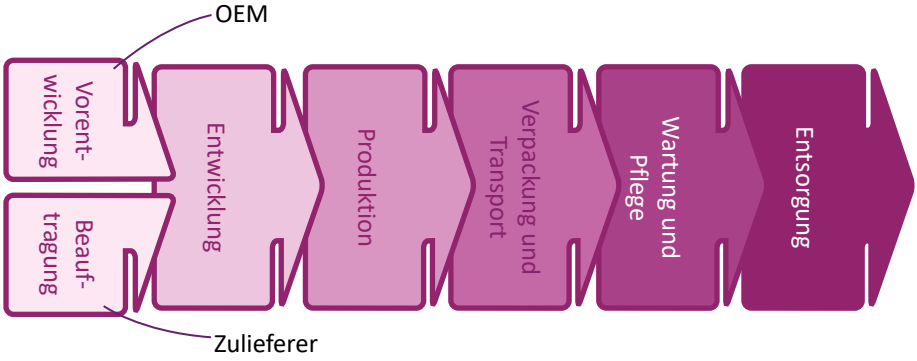


Abbildung 5.1: Abstrakter Lebenszyklus eines Systems

Bei vielen OEMs wird zunächst eine Vorentwicklung durchgeführt, während bei einem Zulieferer am Anfang eines Projekts die Beauftragung, also die Angebotsphase steht. Bezüglich der Beteiligung des Systems-Engineers an diesen beiden Phasen gilt das Folgende:

Vorentwicklung: Hier ist die Aufgabe des Systems-Engineers, aus den Wünschen z. B. eines Produktmanagers die Anforderungen soweit zu analysieren, bis die kritischen Themen identifiziert werden können. Für diese Themen wird dann eine detailliertere Analyse durchgeführt. Sie dient als Startpunkt für eine vorläufige Systemarchitektur,

um die an den kritischen Themen beteiligten Komponenten zu identifizieren und erste Anforderungen an diese zu stellen.

Bei diesem Schritt sollten auf jeden Fall die Komponentenentwickler beteiligt sein. Nach der Entwicklung der Komponenten wird der Systems-Engineer die Konformität der entstandenen Komponenten bzw. ihres Zusammenbaus zu den Anforderungen überprüfen. Falls in Vorentwicklung für unterschiedliche Themen verschiedene Konzepte getrennt voneinander entstanden sind, ist es zusätzlich notwendig, die Möglichkeit zur Integration dieser Konzepte zu bewerten.

Beauftragung: In dieser Phase wird im Allgemeinen der linke Ast des Vs durchlaufen, um ein abschätzbares Realisierungskonzept zu erhalten. Dabei müssen aber nicht alle im Prozess geforderten Artefakte erzeugt werden. Gleichzeitig wird das Lastenheft kommentiert, um zusammen mit dem Realisierungskonzept als Grundlage für ein Angebot zu dienen. Der Systems-Engineer sollte neben den Komponentenentwicklern an allen Schritten beteiligt sein, da er sich verantwortlich für alle Artefakte fühlen sollte, die aus Sicht der Entwicklung für ein Angebot geliefert werden müssen. Es können sein:

- Kommentiertes Lastenheft mit Anpassungen und Annahmen
- Realisierungskonzept und evtl. Alternativen
- Testkonzept
- Bill of Material
- Risikoliste

Die Beteiligung des Systems-Engineers an der Entwicklung haben wir in den vorigen Kapiteln beschrieben. Für die hinteren drei Phasen im Lebenszyklus gilt, dass der Systems-Engineer die aus diesen Phasen folgenden Anforderungen an das System mit berücksichtigen muss.

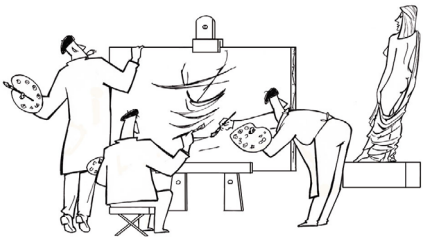
Zusätzlich wird er, gerade aus der Wartung und Pflege, mit Änderungswünschen konfrontiert werden. Seine Aufgabe ist es, diese Wünsche entgegenzunehmen, ihre Auswirkungen zu analysieren und die daraus folgenden Änderungen in den Anforderungen an die Komponenten zu definieren. Dabei hilft ihm die im folgenden Abschnitt eingeführte Produktbeschreibung.

5.3 Die weichen Herausforderungen

Wenn Sie diese Broschüre bis hierhin aufmerksam gelesen haben, kann Ihnen der Gedanke kommen, dass Systems-Engineering eine Disziplin ist, die nur von Personen durchgeführt werden kann, die Spezialisten in allen betroffenen Disziplinen und Gewerken sind. Das wäre natürlich wünschenswert, ist jedoch nicht realistisch.

Zum Einen haben wir in dieser Broschüre die Aufgaben der Rolle Systems-Engineer beschrieben. Diese Rolle kann in einem Projekt natürlich von mehreren Personen wahrgenommen werden. Wahrscheinlich ist dies auch schon in Ihren Projekten der Fall, ohne dass die Rolle explizit benannt ist, und die Tätigkeiten werden von bekannten Rollen (z. B. von einem Hardware-Entwickler) durchgeführt. Ein definierter Systems-Engineering-Prozess mit der entsprechenden Rolle soll dafür sorgen, dass die benötigten Tätigkeiten aus Systemsicht bewusst und zum richtigen Zeitpunkt unter Einbeziehung der betroffenen Personen durchgeführt werden.

Damit muss ein Systems-Engineer nicht das Spezialwissen aus den verschiedenen Disziplinen besitzen. Er muss jedoch genügend Wissen haben, um Entscheidungen von den richtigen Personen einzufordern, sie in Zusammenhang zu anderen Entscheidungen zu setzen und sie bei Bedarf zu weiteren Beteiligten zu kommunizieren.



Damit können die Fähigkeiten eines guten Systems-Engineers in drei Ebenen eingeteilt werden.

- Inhaltlich: Ein Systems-Engineer sollte Wissen in der Anwendungsdomäne besitzen.
- Methodisch: Er sollte die Methoden und Techniken, die in den verschiedenen Tätigkeiten zum Einsatz kommen, kennen und können.
- Persönlich: Da Systems-Engineering viel mit Kommunikation zu tun hat, sollte er teamorientiert und konfliktbereit sein. Weiterhin benötigt er Abstraktionsvermögen und sollte Vertrauen zu Entscheidungen von Dritten haben.

Falls Sie solche Personen in Ihrem Unternehmen kennen, haben Sie gute Kandidaten für die Rolle des Systems-Engineers gefunden. Falls nicht, so können Sie diese Rolle auch mit mehreren Personen, die unterschiedliche Skills besitzen, besetzen. So kann zum Beispiel ein interner Mitarbeiter mit dem benötigten inhaltlichen Wissen durch eine Person von außen, die das methodische Wissen beisteuern kann, unterstützt werden.

Die Rolle des Systems-Engineer und den entsprechenden Prozess einzuführen, ist aber eine Aufgabe, die nicht unterschätzt werden darf. Ein Grund hierfür kann die Tatsache sein, dass viel wertvolles Wissen zentralisiert und dokumentiert wird. Dies kann auf Widerstand stoßen, da die bisherigen Wissensträger das Gefühl bekommen

könnten, entbehrlich zu werden. Ein weiterer Grund kann sein, dass Aufwand in die frühen Phasen eines Projekts investiert wird, der sich erst in späteren Phasen bezahlt macht.

Desweiteren gelten für die Einführung eines Systems-Engineering-Prozesses die gleichen Aussagen (und Herausforderungen) wie bei jedem Einführungsprozess. Ein Beispiel ist in [Rupp20, Kapitel 26] gegeben.

Quellenverzeichnis

[ALM16] www.ireb.org/de/downloads/, Handbuch zum CPRE Advanced Level Modeling, letzter Zugriff: Juni 2020

[Akao92] Akao, Y.: QFD – Quality Function Deployment, Verlag Moderne Industrie, Landsberg 1992

[Barwise05] Barwise, J.: Sprache, Beweis und Logik. Band I: Aussagen- und Prädikatenlogik, Mentis Verlag, 2005

[Cockburn00] Cockburn, A.: Writing Effective Use Cases, 1. Auflage, Addison-Wesley Professional, 2000

[Incose15] Incose: Systems Engineering Handbook, A Guide for System Life Cycle Processes and Activities 4. Auflage, Wiley John and Sons, 2015

[Nuseibeh01] Nuseibeh, B.: Weaving the Software Development Process Between Requirements and Architectures (2001), URL: https://www.semanticscholar.org/, letzter Zugriff: Januar 2020

[Pfeufer14] Pfeufer, H.: FMEA – Fehler-Möglichkeits- und Einfluss-Analyse, Hanser Verlag, 2014

[Pflüger14] Pflüger, A.: Modellgetriebene Validierung von System-Architekturen gegen architekturrelevante Anforderungen: Ein Ansatz zur Validierung mit Hilfe von Simulationen, University of Bamberg Press, 2014

[Pohl et al.12] Pohl, K.: Harald Hönninger, Reinhold Achatz, Manfred Broy: Model-Based Engineering of Embedded Systems, Springer Verlag, 2012

[Robertson12] Robertson, S.: Robertson J.: Mastering the Requirements Process. Getting Requirements Right. 2nd Edition, Pearson Verlag, 2012

[Rupp20] Rupp, C.: Requirements-Engineering und –Management, Das Handbuch für Anforderungen in jeder Situation. 7. Auflage. Hanser Verlag, 2020.

[RuQu12] Rupp, C.: Stefan Queins: UML Glasklar – Praxiswissen für die UML-Modellierung, 4. Auflage, Hanser Verlag, 2012

[Starke14] Starke, G.: Effektive Softwarearchitekturen – Ein praktischer Leitfaden, 6. Auflage, Hanser Verlag, 2014

[Sauerwein00] Sauerwein, E.: Das Kano-Modell der Kundenzufriedenheit. 1. Auflage, Innsbruck, 2000.

[SysML] www.omgsysml.org, Offizielle Homepage der OMG zur SysML, letzter Zugriff: Juni 2020

SOPHIST Eigenproduktionen

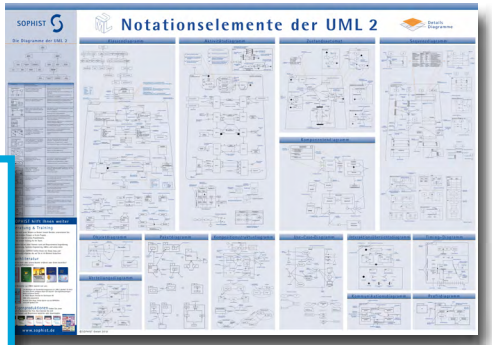
Wissensträger mal anders!

Kostenfrei!

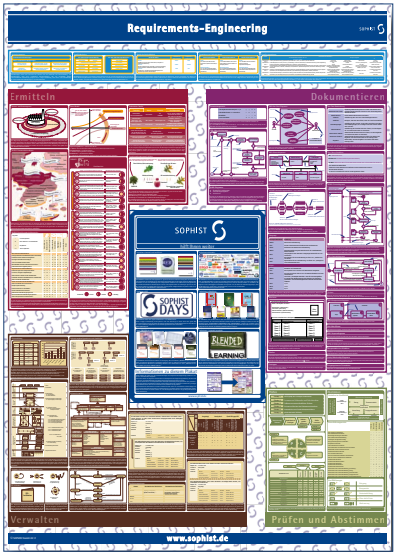
WISSEN for free

Poster/Plakate

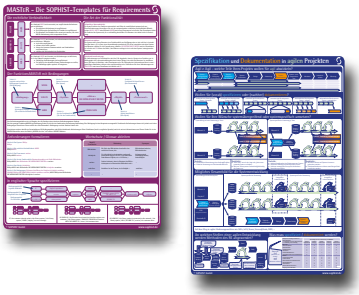
Das SOPHIST UML-Plakat



Das SOPHIST RE-Plakat ...



... und seine „Kerne“



WISSEN for free

Broschüren

www.sophist.de/wissen-for-free

SOPHIST

Kompetenz und Fachwissen

par excellence

Methodenerfinder

Speaker

Buchautoren

Berater Coach Trainer

Das bieten wir Ihnen an:

Wir unterstützen Sie kompetent, tatkräftig und zielführend sowohl bei der Anpassung Ihrer Entwicklungsprozesse und -methoden als auch bei der Durchführung Ihres Projekts.

Zu unseren Kunden gehören viele weltbekannte Unternehmen. Die Vielzahl an positiven Meinungen und Projektberichten unserer zufriedenen Kunden spricht für sich. Werfen Sie doch einen Blick auf **www.sophist.de/referenzen**

Unsere Leistungen:

- Verbesserungspotenziale unter Berücksichtigung der Randbedingungen in Ihrer Organisation identifizieren, ausschöpfen und einführen
- Anforderungen und Architekturen angemessen erheben, analysieren sowie vermitteln und dokumentieren
- Unterwegs in einfachen Software-Anwendungen bis hin zu komplexen Systemen
- agil und angepasst zu arbeiten

All das und noch viele weitere Themen aus der Welt des Requirements- und Systems-Engineerings bieten wir Ihnen in Form

Wie können wir Ihnen helfen?

Gerne arbeiten wir mit Ihnen ein Konzept aus, um Sie bestmöglich in Ihrem Vorhaben zu unterstützen.

Kontaktieren Sie uns unverbindlich:

+49 (0) 911 40 900 - 0

heureka@sophist.de

SOPHIST Trainings

UPDATE

Was ist daran neu?

Alle Trainings von SOPHIST sind nach neuestem Stand der Wissensvermittlung konzipiert. Unsere Trainer folgen unter anderem den Grundsätzen der Methode „... from the Back of the Room“, um Trainingsinhalte nachhaltig zu vermitteln.

Eine aktivierende Lernumgebung – mehr Bewegung, weniger Text, mehr Interaktion mit den Teilnehmern und überraschende Übungskonzepte – sorgt für Spaß und Effizienz beim Lernen.

Ihre Vorteile?

Sie profitieren nicht nur von dem Know-How der Methodenführer, sondern auch von einer didaktischen Umsetzung, die ihre Spuren hinterlässt.

Neugierig geworden?

Kontaktieren Sie uns unverbindlich:

+49 (0) 911 40 900 - 0

heureka@sophist.de

Online/Remote Trainings

Ihre SOPHIST Weiterbildung - an nahezu jedem Ort der Welt

SOPHIST Online/Remote Trainings sind speziell für die Vermittlung von Wissen und Können über das Internet entwickelt worden. Die besondere und sorgfältig durchdachte Ausarbeitung – inhaltlich und didaktisch – sowie eine Teilnehmerzahl von maximal 12 Personen gewährleistet einen perfekten Online-Wissenstransfer.

Für **Einzelpersonen** und **kleinere Teams** eignen sich unsere „Offenen Trainings“ perfekt. Und durch den modularen Aufbau, der Kombination verschiedener Schwerpunkte und der Möglichkeit der individuellen Anpassung, eignen sich Remote/Online Trainings auch perfekt für firmeninterne Weiterbildungen **kompletter Teams.**

Natürlich gibt es auch unsere bekannten CPRE-Zertifizierungstrainings in diesem Format.

... egal wo

Systems-Engineering

SOPHIST beschäftigt sich seit über 20 Jahren mit Anforderungen, also der Ein- und Durchführung von gutem Requirements-Engineering. Über die Jahre hinweg haben wir darin viele Erfahrungen aus unterschiedlichsten Anwendungsgebieten gesammelt. Dabei haben wir festgestellt, dass neben der Ermittlung und Dokumentation von Anforderungen, die sich an den Betrieb des betrachteten Produkts richten, die Anforderungen aus den anderen Lebenszyklen des Produkts immer mehr in den Fokus rücken.

Eine adäquate Betrachtung der Anforderungen wird in der reinen Softwareentwicklung mit agilen Ansätzen gut unterstützt. Diese Ansätze funktionieren in einem komplexen System, das neben Software auch aus elektrischen, elektronischen oder mechanischen Anteilen besteht, allerdings nur bedingt. An dieser Stelle setzt **Systems-Engineering** an, das in einem interdisziplinären Ansatz sowohl das gewünschte Produkt als Gesamtsystem, als auch alle Bereiche in seinem Lebenszyklus betrachtet. Darüber hinaus ist die Architektur zur Aufteilung des Produkts in seine Komponenten ein integraler Bestandteil des Systems-Engineerings.

Die Investition von Aufwänden in die frühen Phasen der Systementwicklung hilft Ihnen, Risiken zu minimieren und späteren Überraschungen vorzubeugen.