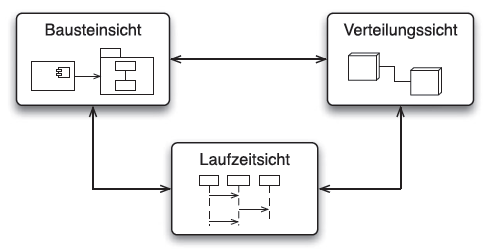
Softwarearchitekten können:

* Softwarearchitekturen auf Basis bekannter funktionaler und Qualitätsanforderungen für nicht sicherheits- oder unternehmenskritische Softwaresysteme entwerfen und angemessen kommunizieren und dokumentieren
* Strukturentscheidungen hinsichtlich Systemzerlegung und Bausteinstruktur treffen, dabei Abhängigkeiten zwischen Bausteinen festlegen
* gegenseitige Abhängigkeiten und Abwägungen bezüglich Entwurfsentscheidungen erkennen und begründen
* Begriffe Blackbox und Whitebox erklären und zielgerichtet anwenden
* schrittweise Verfeinerung und Spezifikation von Bausteinen durchführen
* Architektursichten entwerfen, insbesondere Baustein-, Laufzeit- und Verteilungssicht
* die aus diesen Entscheidungen resultierenden Konsequenzen auf den Quellcode erklären
* fachliche und technische Bestandteile in Architekturen trennen und diese Trennung begründen
* Risiken von Entwurfsentscheidungen identifizieren.

# Verschiedene Sichten

Wir empfehlen Ihnen für Software- und IT-Systeme folgende drei Sichten



* Bausteinsicht: das Kernstück der Software-Architektur, entspricht dem Grundrissplan eines Hauses. Sie zeigt die statische Struktur des Systems, seinen Aufbau aus Softwarebausteinen sowie deren Beziehungen und Schnittstellen untereinander. Jeder dieser Bausteine wird letztlich durch eine Menge (selbst erstelltem oder zugekauftem) Quellcode implementiert. Beispiele für solche Bausteine sind einzelne Java-, C#- oder Objective-C-Klassen, Package-Konstrukte von Programmiesprachen, Datenbank- oder Shell-Skripte, Konfigurationen einer Standardsoftware oder ähnliche Artefakte. Die Bausteinsichtklärt, wie das System aus Einzelteilen konstruiert ist, welche Eigenschaften diese Einzelteile haben und welcher Baustein welche fachlichen und technischen Eigenschaften des Systems bereitstellt.
* Die Verteilungssicht zeigt die Verteilung von Systembestandteilen auf Hard- und Softwareumgebungen. Diese Sicht klärt, welche Teile des Systems auf welchen Rechnern, an welchen geographischen Standorten oder in welchen Umgebungen ablaufen können, wenn es in einer konkreten technischen Infrastruktur installiert wird. Sollten Sie ein Softwaresystem entwickeln, das nur auf einem einzelnen Rechner installiert und betrieben wird, so kann diese Sicht entfallen, denn sie bringt keinen wesentlichen Erkenntnisgewinn. Bei komplexeren Konfigurationen, Multiprozessorsystemen oder verteilten Systemen klärt diese Sicht die Frage, welcher Teil der Software auf welcher Hardware oder an welchem Standort ablaufen.
* Schließlich zeigt die L aufzeitsicht das Verhalten des Systems beziehungsweise einzelner Bausteine zur Laufzeit. Genauer gesagt zeigt die Laufzeitsicht, wie die Bausteine des Systems zur Laufzeit miteinander bzw. mit den Nachbarsystemen interagieren. Sie klärt, wie das System funktioniert, wie wesentliche, beispielhafte Abläufe aussehen und wie die Bausteine des Systems die wesentlichen Funktionen oder Anwendungsfälle erledigen.

Quelle Software Architektur kompakt, 2. Auflage Seite 30

Eigene Bemerkung: Zusätzlich wird die Kontextsicht empfohlen. Aber die Kontextsicht kann man als eine BlackBox für die alleroberste Ebene betrachten.

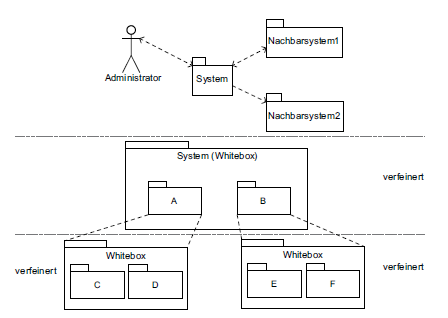


Abbildung Von Kontextsicht auf Bausteinsicht

Quelle Basiswissen für Softwarearchitekten 4. Auflage Kapitel 4.3.9

# BlackBox und WhiteBox

## Definition Baustein

Unter dem Begriff „Bausteine“ fasse ich sämtliche Software- oder Implementierungskomponenten zusammen. Bausteine repräsentieren (existierenden oder geplanten) Quellcode in verschiedenen Detaillierungsgraden. Dazu gehören Klassen, Prozeduren, Programme, Pakete, Komponenten oder Subsysteme. Das untere Bild illustriert das in Form eines einfachen Metamodells.

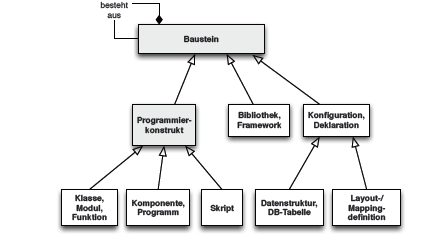


Abbildung Begriff des Bausteins

Quelle: Effektive Softwarearchitekturen, 8. Auflage, Kapitel 5.6

## Definition BlackBox und WhiteBox

Blackboxes sind ausschließlich durch ihre externen Schnittstellen und ihre Funktionalität beschrieben. Sie folgen dem Geheimnisprinzip: Ihr gesamtes Innenleben und somit ihre innere Komplexität bleiben verborgen.

Whiteboxes sind *geöffnete* Blackboxes: Sie zeigen deren innere Struktur und Arbeitsweise. Whiteboxes bestehen ihrerseits wiederum aus einer Anzahl von Blackboxes. Den Zusammenhang zwischen Black- und Whitebox zeigt die untere Abbildung

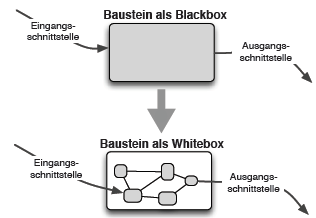


Abbildung Zusammenhang zwischen BlackBox und WhiteBox

Quelle: Effektive Softwarearchitekturen, 8. Auflage, Kapitel 5.6

## Verfeinerung der Bausteinsicht mit Hilfe von BlackBox und WhiteBox

Insbesondere im Zusammenspiel zwischen Blackbox und Whitebox wird die hierarchische (De-)Komposition einer Architektur und ihrer Bausteine deutlich. Wie die untere Abbildung zeigt, kann die Blackbox-Sicht eines Bausteins A in einer darunter liegenden Whitebox-Sicht hierarchisch dekomponiert werden. In dieser Whitebox- Sicht zerfällt der Baustein A in seiner Konfiguration in die Bestandteile B1,

B2 und B3. Dabei ist zu beachten, dass die Bestandteile b1, b2 und b3 keine Bausteine sind.

Sondern es sind Platzhalter (in der UML auch Parts genannt), die eine Instanz eines Bausteins nutzen. Wir nennen diese Platzhalter auch Bausteininstanzen oder, falls es nicht weiter von Belang ist, auch nur einfach Bausteine. Auch die Subbausteininstanzen b1 bis b3 sind Platzhalter für Bausteininstanzen in der

Konfiguration des Bausteins A. Jedoch kann nicht jede beliebige Bausteininstanz an den Platzhalter b1, b2 oder b3 gesetzt werden, da die Platzhalter typisiert sind.

Platzhalter sind vergleichbar mit Variablen: Sie haben einen Wert (= Bausteininstanz) und einen Typ (= Baustein). Somit kann an den Platzhalter »b1: Baustein B1« nur eine Instanz des Bausteins B1 gesetzt werden.

Da diese Bausteininstanz in der Konfiguration den Typ des Bausteins festlegt, gibt es für diesen Baustein wiederum eine Blackbox-Sicht.

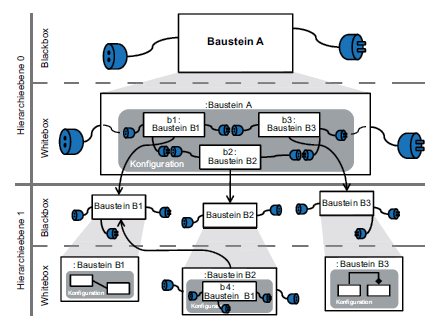


Abbildung Zusammenhang zischen BlackBoxes, WhiteBoxes und Schnittstellen

Die Darstellung zeigt aber nur eine Hierarchie im Sinne der Beschreibung der Architektur. Somit können Instanzen eines Bausteins in mehreren Konfigurationen anderer Bausteine, zu sehen in deren Whitebox-Sichten, verwendet werden. die unter Umständen auf unterschiedlichen Hierarchieebenen anzuordnen sind.

In der obigen Abbildung taucht der Baustein B1 als Bausteininstanz in der Konfiguration der Whitebox-Sicht sowohl als Subbausteininstanz von A wie auch als Subbausteininstanz von B2 auf. B1 könnte ein XML-Parser-Baustein sein, der in den verschiedensten Bausteinen auf den unterschiedlichsten Ebenen verwendet wird.

Dabei ist zu beachten, dass diese hierarchische Zerlegung nicht nur für Bausteine, sondern auch für ihre Schnittstellen gilt. Das heißt, besitzt ein Baustein eine Schnittstelle zu einem anderen Baustein in der Greybox, so existiert auch eine entsprechende Schnittstelle auf der Blackbox- und Whitebox-Ebene und

muss demgemäß implementiert werden, z.B. dadurch, dass die Schnittstelle auf einen Subbaustein delegiert wird.

Quelle Basiswissen für Softwarearchitekten, 4. Auflage, Kapitel 2.3.2

# Trennung Fachlichkeit und Technik

Aus unserer Sicht bildet die explizite Trennung fachlicher und technischer Teile von Software-Architekturen eine wesentliche Grundlage für die langfristige Qualität von Systemen. Daher sollten Sie grundsätzlich versuchen, die fachlichen Bausteine Ihrer Systeme von den notwendigen technischen Infrastrukturbausteinen zu trennen. Jeder Baustein Ihres Systems sollte dabei eine eindeutige Verantwortung

für genau einen dieser beiden Aspekte tragen – entweder fachlich oder technisch, möglichst nie in Mischformen.

Der Vorteil dieser strikten Trennung liegt in der besseren Modularisierung und Wartbarkeit, meistens verbunden mit besserer Verständlichkeit. Außerdem können auf Basis eines rein fachlichen Modells

auch Domänen- und Softwareexperten besser miteinander reden.

Die Anforderungen an fachliche und technische Teile ändern sich unabhängig voneinander – ein weiterer Grund, diese beiden getrennt voneinander zu entwickeln und zu pflegen. Sie sollten bei rein technologisch begründeten Änderungswünschen davon ausgehen, dass der f fachliche Kern Ihrer Architektur stabil und unverändert bleiben kann.

Quelle Softwarearchitektur kompakt, 2. Auflage, Seite 42