

# Bausteinsicht



Stand: April 2020

© Diese Unterlagen sind urheberrechtlich geschützt von Dr. Peter Hruschka und Dr. Gernot Starke.  
Jede Verwendung außerhalb der engen Grenzen des Urheberrechts ist ohne schriftliche Zustimmung der Autoren unzulässig und strafbar.  
Dies gilt insbesondere für Vervielfältigungen, Übersetzungen sowie Speicherung und Verarbeiten in elektronischen Systemen.

[www.arc42.de](http://www.arc42.de)

A blue curved line, similar to the one in the logo, located in the bottom right corner of the page.

# Ziele und Inhalt



Sie lernen:

- Begriff „Architekturbaustein“
  - UML-Ausdrucksmittel für Bausteine
- Top-Down *Darstellung* von Bausteinen
- Black- und Whitebox
  - mit Templates zu ihrer Beschreibung
- Entwurf und Beschreibung von Schnittstellen

Sie üben:

- Entwurf von Bausteinen
- (Top-Down) Darstellung von Bausteinen

Lernziele gemäß iSAQB CPSA-F:

- LZ 2-2: Softwarearchitekturen entwerfen (R1)
- LZ 2-7: Abhängigkeiten von Bausteinen planen (R1)
- LZ 2-9: Schnittstellen entwerfen und festlegen (R1-R3)
- LZ 3-2: Softwarearchitekturen beschreiben und kommunizieren (R1)
- LZ 3-3: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2)
- LZ 3-4: Architektursichten erläutern und anwenden (R1)
- LZ 3-7: Schnittstellen beschreiben (R1)
- LZ 3-8: Architekturentscheidungen erläutern und dokumentieren (R2)
- LZ 5-1: Bezug von Anforderungen und Randbedingungen zu Lösung erfassen (R3)

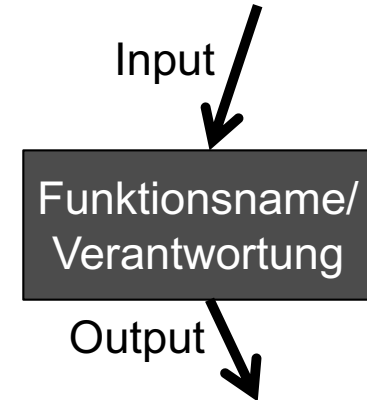
# Bausteine

A word cloud centered around the main title 'Bausteine'. The words are in various shades of blue, purple, and pink, with different font sizes and orientations. The words include: 'Konfiguration', 'Buildingblock', 'Quellcode', 'Programm', 'Modul', 'Funktion', 'Package', 'Framework', 'Datei', 'Shellscript', 'Komponente', 'Block', 'Methode', 'Routine', 'Subsystem', 'File', 'Klasse', 'Paket', 'Executable', 'Skript', and 'Library'. The word 'Bausteine' is the largest and most prominent, written in a bold, blue, 3D-style font.

# Black- und Whitebox

## Blackbox:

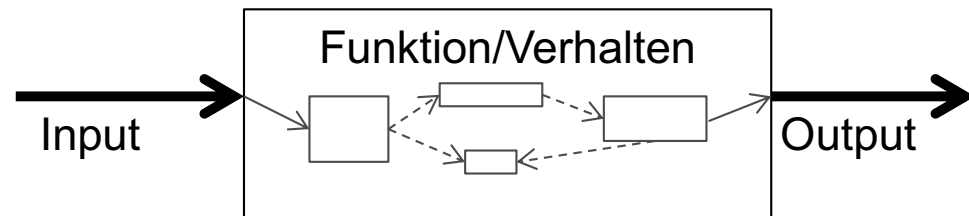
- *Geschlossenes System mit (i.d.R definiertem) Ein-/Ausgabeverhalten ohne Beachtung des inneren Aufbaus*
- Erfüllt:
  - **Geheimnis-Prinzip (Information Hiding\*)**  
(„WIE“ bleibt verborgen)
  - Kapselung des Innenlebens



\*) David Parnas, 1972

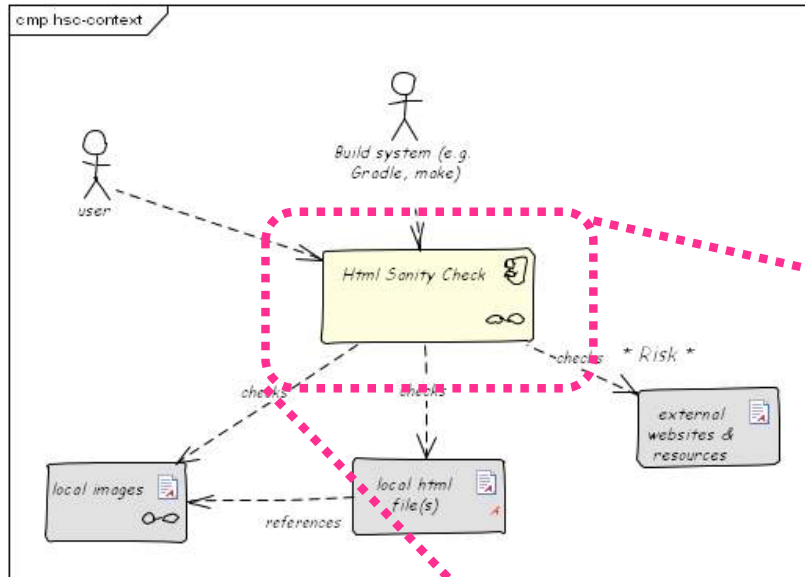
## Whitebox:

- *Geschlossenes System mit (i.d.R definiertem) Ein-/Ausgabeverhalten und bekanntem oder vorgegebenen innerem Aufbau*





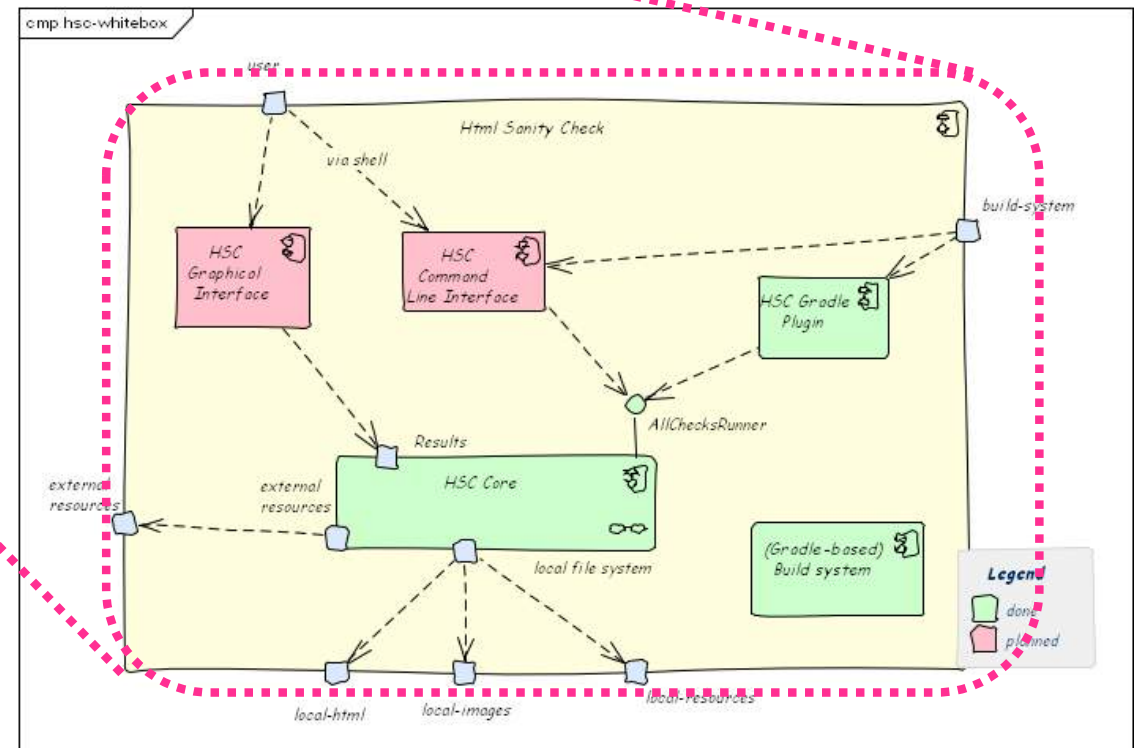
## Kontext (System als Blackbox)



Neighbor	Description
user	documents software with toolchain that generates html. Wants to ensure that links within this html are valid.
build system	
local html files	HtmlSC reads and parses local html files and performs sanity checks within those.
local image files	HtmlSC checks if linked images exist as (local) files.
external web resources	HtmlSC can be configured to optionally check for the existence of external web resources. Due to the nature of web systems, this check might need a significant amount of time and might yield invalid results due to network and latency issues.

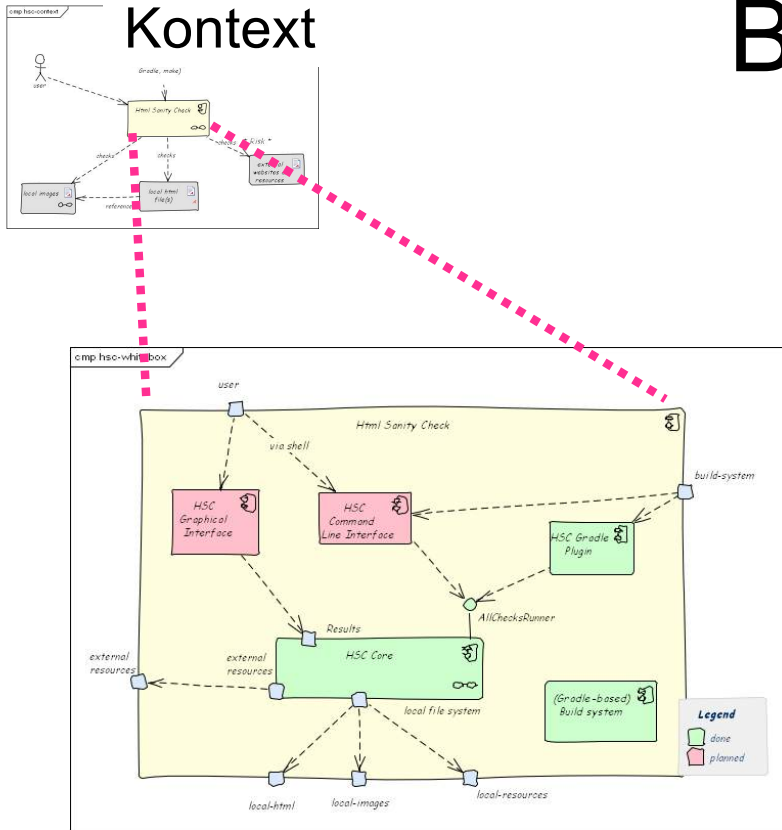
## Beispiel (1)

### Top-Level-Zerlegung (System als Whitebox)



# Beispiel (2)

## Kontext



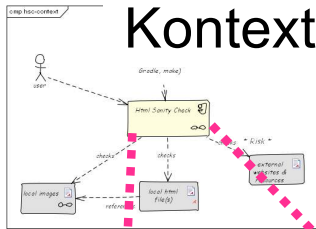
## Rationale

We used functional decomposition to separate responsibilities:

- CheckerCore shall encapsulate checking logic and Html parsing/processing.
- all kinds of outputs (console, html-file, graphical) shall be handled in a separate component (Reporter)
- Implementation of Gradle specific stuff shall be encapsulated.

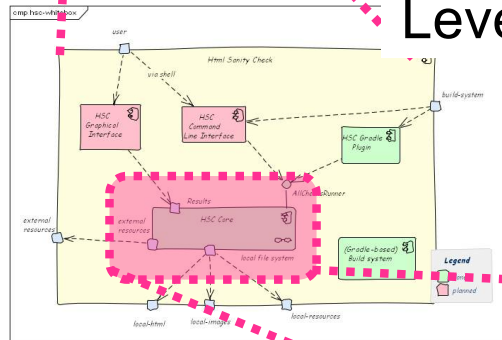
<u>HSC Core</u>	hsc_core: html parsing and sanity checking, file handling
HSC Gradle Plugin	integrates the Gradle build tool with HtmlSC, enabling arbitrary gradle builds to use HtmlSC functionality.
HSC Command Line Interface	(not documented)
HSC Graphical Interface	(planned, not implemented)
Reporter	outputs the collected checking results to configurable destinations e.g., stdout or a Html file.

## Kontext

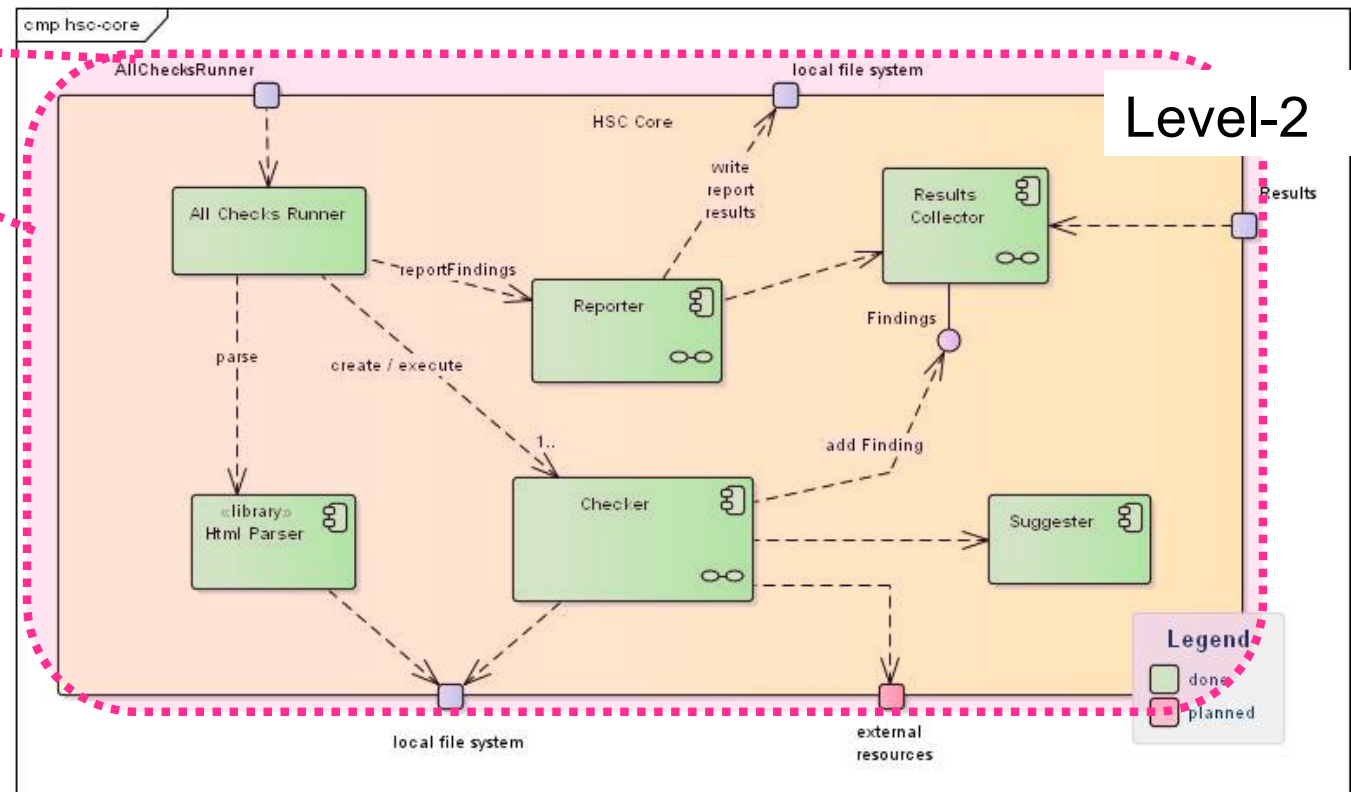


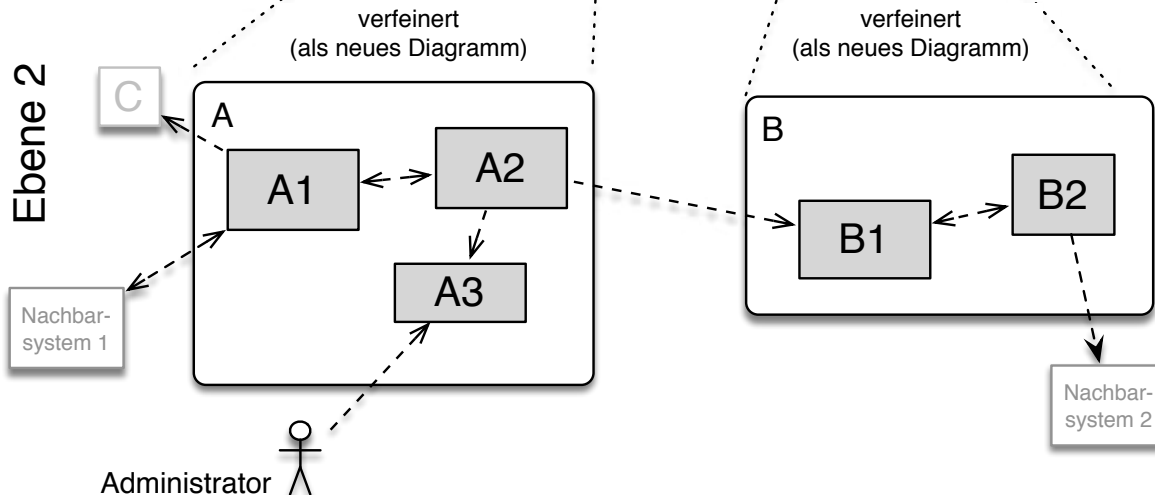
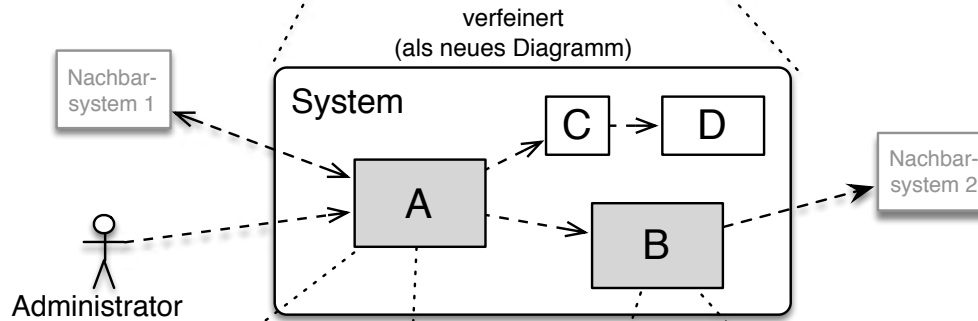
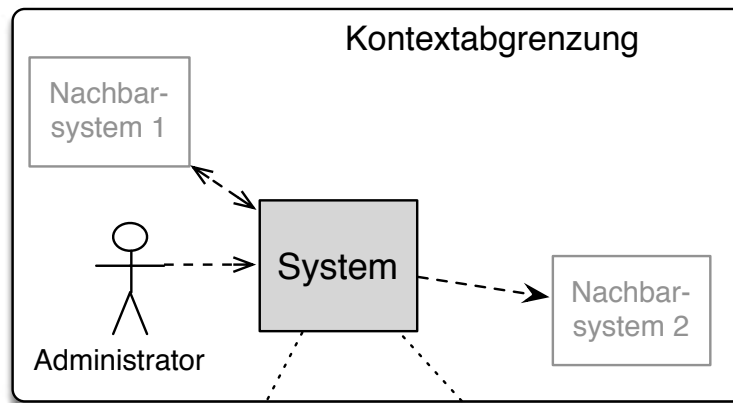
# Beispiel (3)

## Level-1



## Level-2





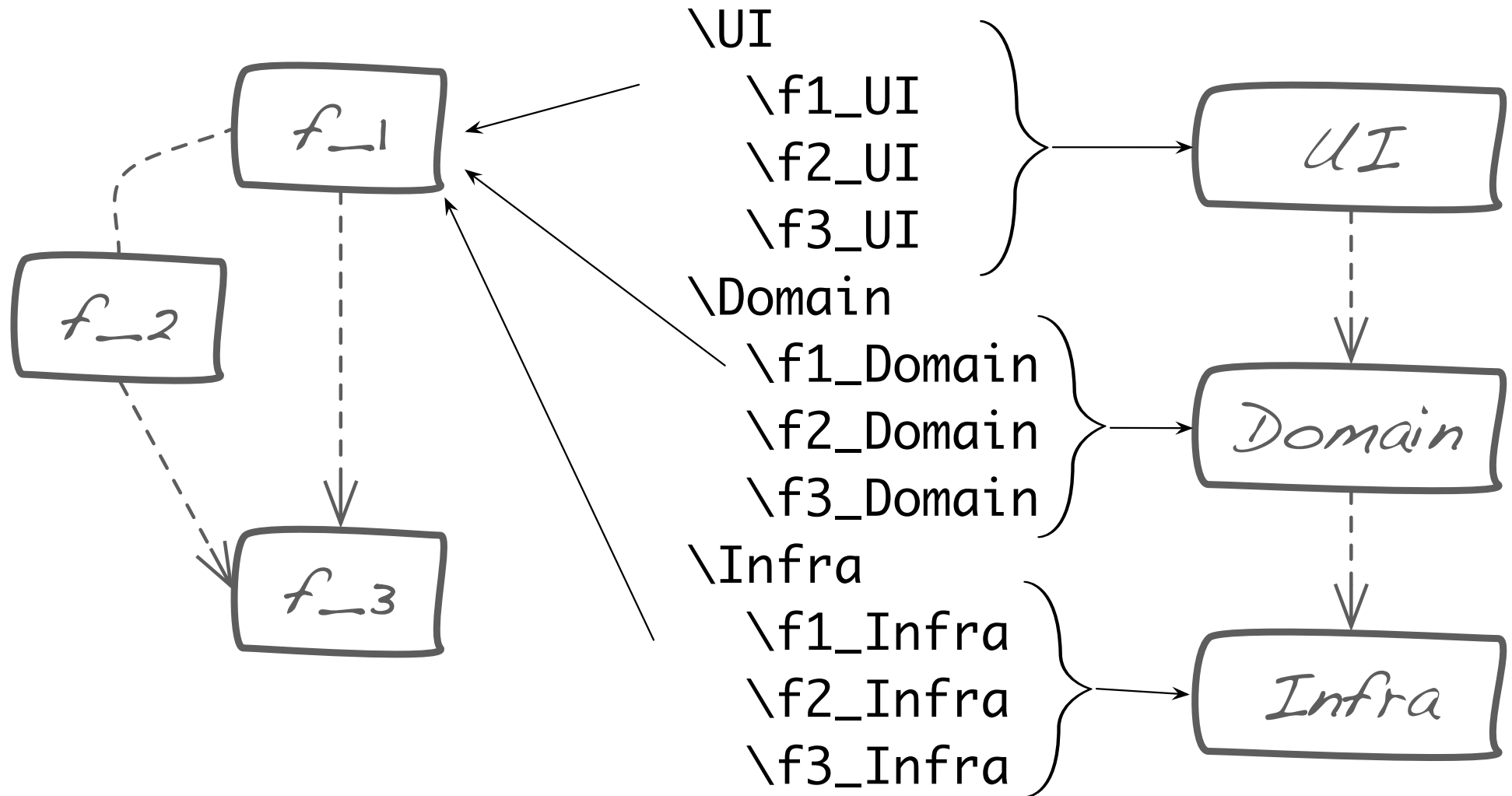
Das Ziel:

Bausteine des Systems  
als Hierarchie darstellen:

- vom Gesamtsystem  
als Black Box
- bis zu **kleineren Bausteinen**
  - in angemessenem  
Detailgrad
  - **mit deren  
Abhängigkeiten**
- Bausteine nur verfeinern,  
wenn mehr Details bekannt  
sein müssen

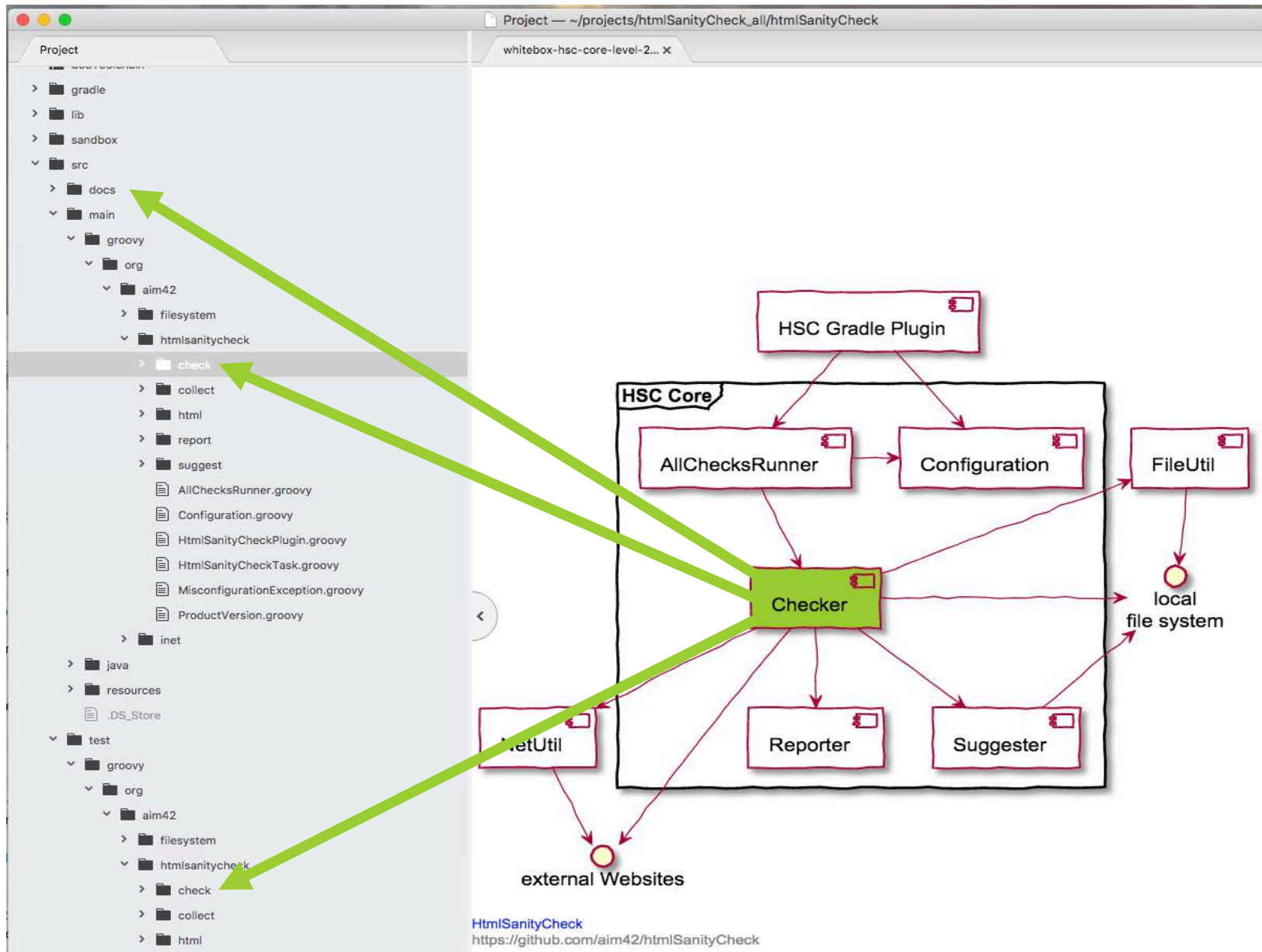


# Vorsicht: Quellcode und Bausteine...

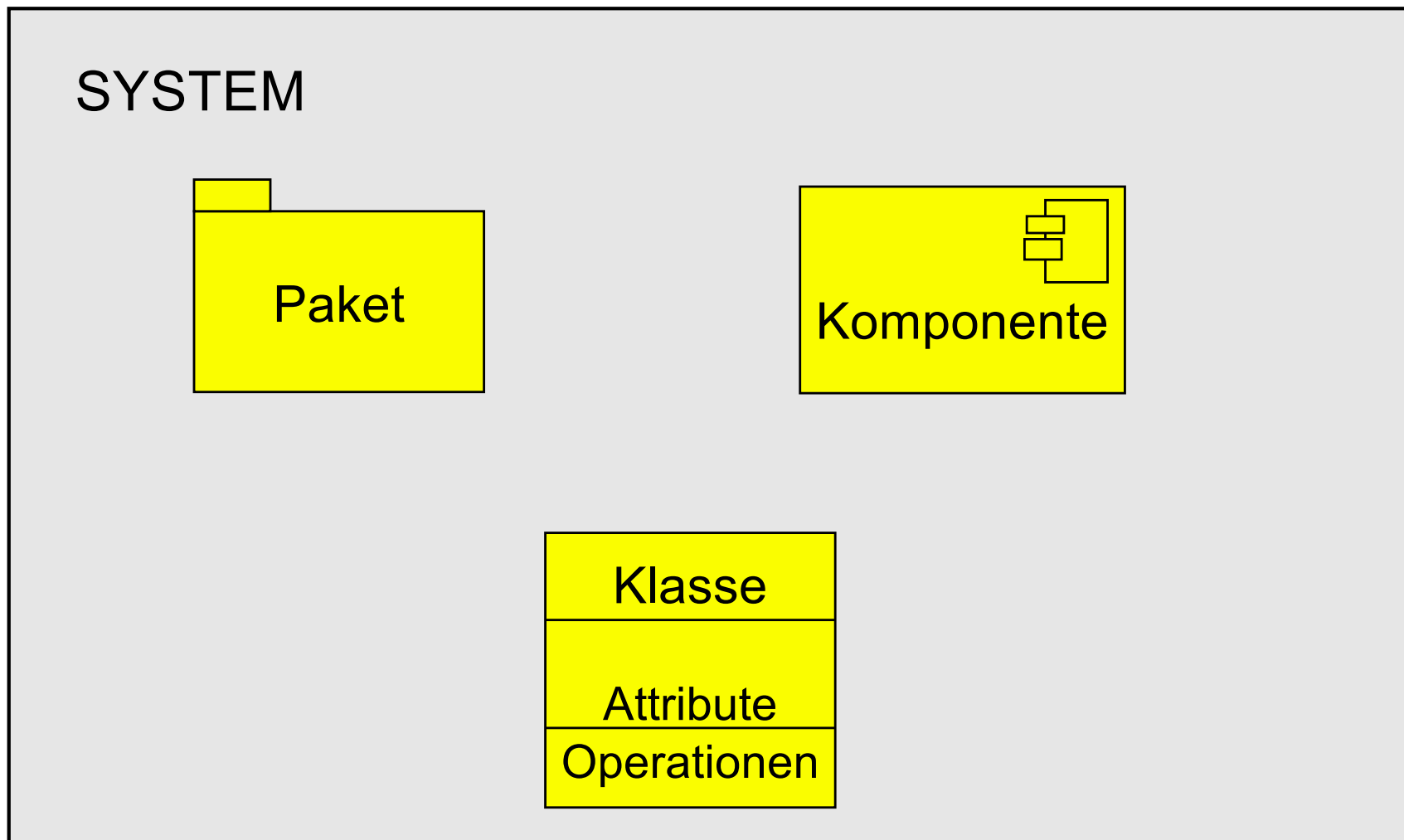


Zuordnung ist Entwurfsentscheidung!

# Quellcode und Bausteine (2)...

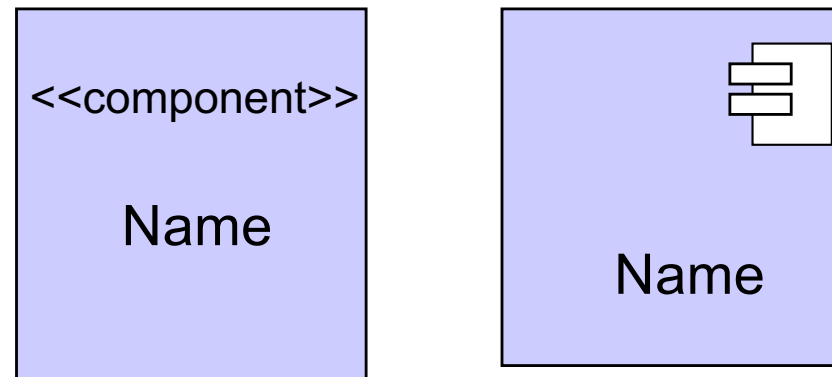


# Die UML Begriffe für unterschiedlich große Bausteine



# Komponenten

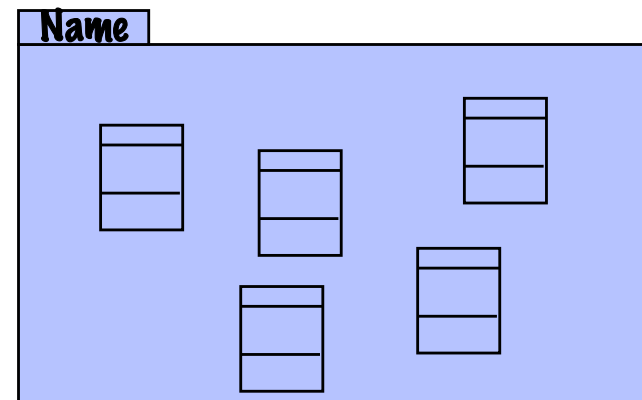
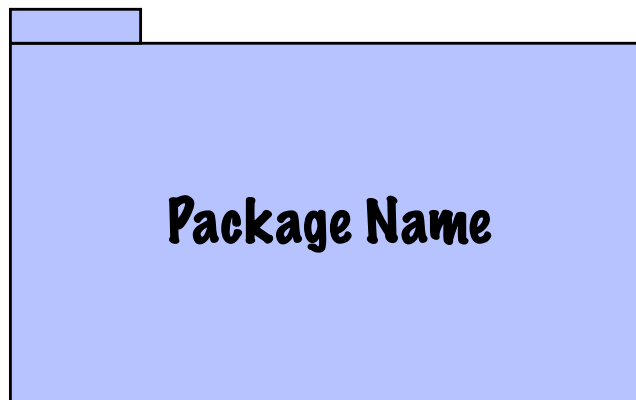
- Ein Teil des Systems mit wohldefinierten Schnittstellen, der seinen Inhalt abkapselt, und der in seiner Umgebung ersetzbar ist. (dessen Manifestation ...ersetzbar ist)
- Eine Komponente definiert ihr Verhalten über die (angebotenen und geforderten) Schnittstellen; kann ersetzt werden durch “typgleiche” Komponente, d.h. eine Komponente mit “typverträglicher” Schnittstelle.



- Nutzen Sie – wenn möglich – immer Komponenten für größere Bausteine mit sauberen Schnittstellen

# Pakete

- Ein Paket (Package) ist ein allgemein verfügbarer Mechanismus der UML, um Elemente zu gruppieren.
- Das Paketsymbol: ein Mappe (mit “Reiter”)



- Tipp: Nutzen Sie Pakete für (nur) logische Gruppierungen ohne Schnittstellen (Beispiel: 3 Teams im Großraumbüro)



# größere Einheiten: wofür?

- Abstraktion

- Komplexität verringern

In der UML durch  
Pakete oder durch  
Komponenten

- Regelung der Sichtbarkeit (Information Hiding)

- Seiteneffekte bei Änderungen reduzieren
  - Anzahl der erreichbaren Objekte begrenzen

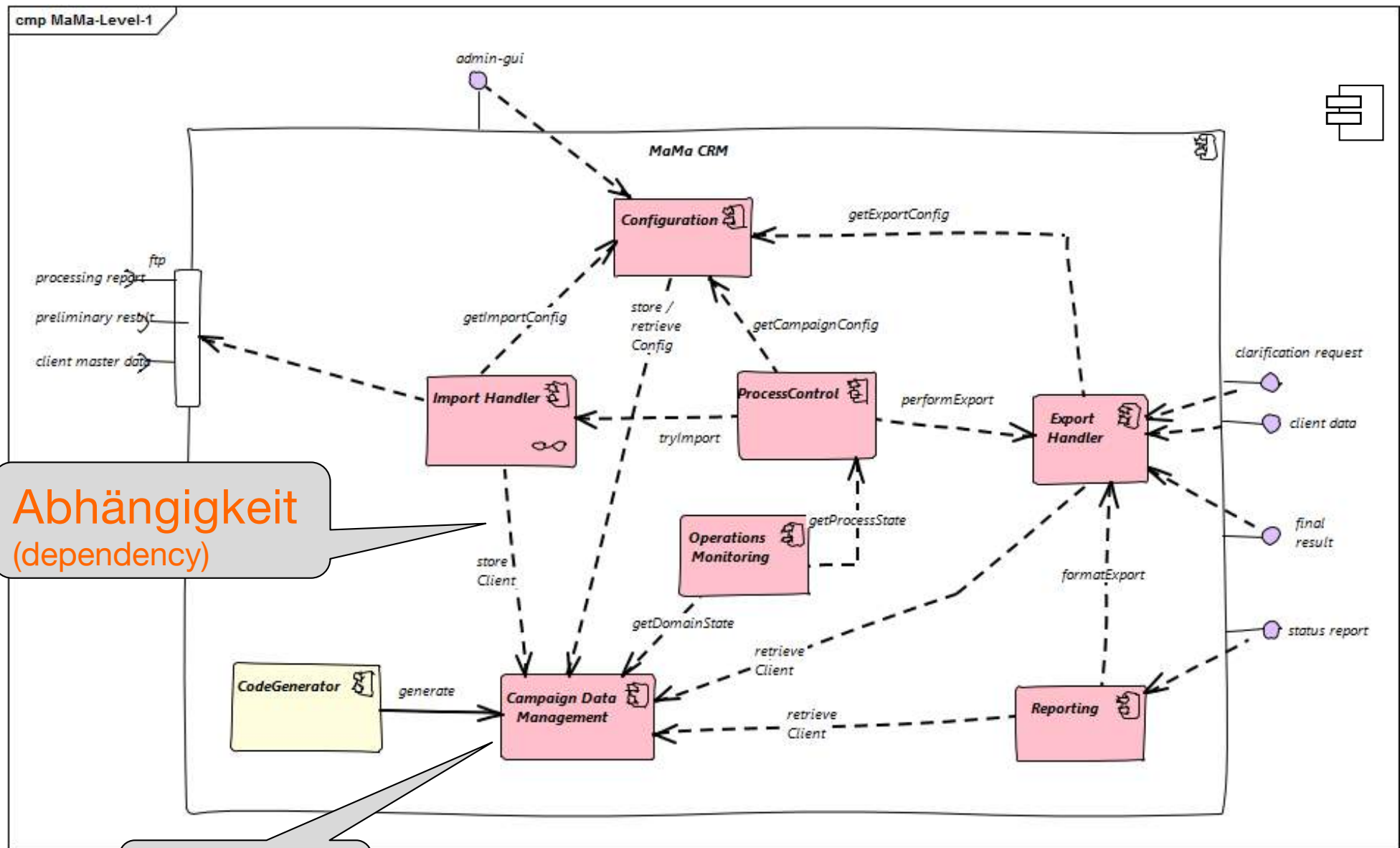
In der UML durch  
Komponenten

# Abstraktion schafft Übersicht

- Eine Operation besteht aus einigen Anweisungen
  - eine Zehnerpotenz gewonnen
- Eine Klasse besteht aus einigen Daten, Operationen und Verhalten
  - eine Zehnerpotenz gewonnen
- Eine Komponenten besteht aus mehreren Klassen
  - wieder eine Zehnerpotenz gewonnen
- Eine Komponente besteht aus anderen Komponenten
  - beliebig viele Zehnerpotenzen gewonnen

Statt über 1.000.000 Zeilen Code lieber über 100.000 Operationen sprechen oder 10.000 Klassen oder 1.000 Komponenten oder 100 große Komponenten oder 10 Subsysteme!

# Komponenten-Diagramm



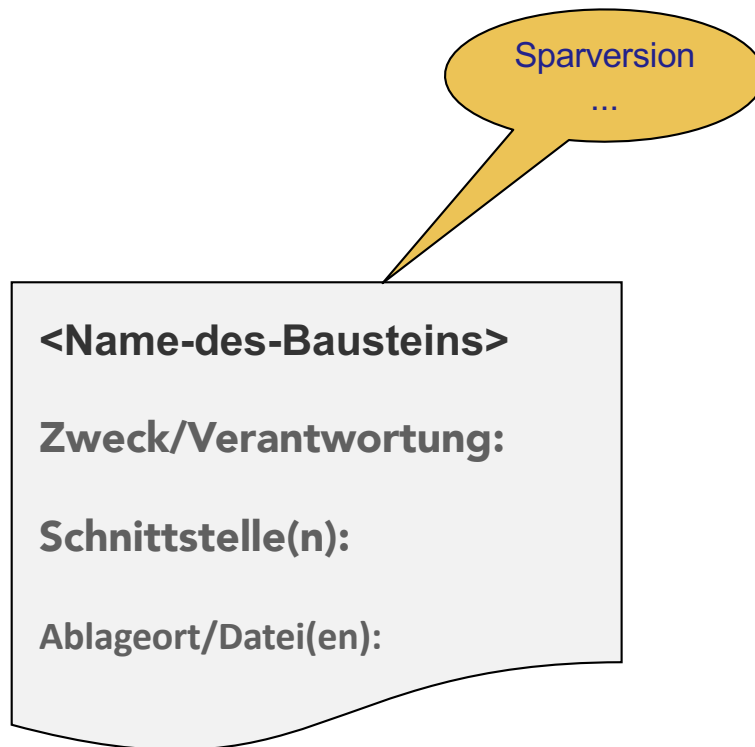
Abhängigkeit  
(dependency)

Baustein

# Blackbox-Template

## Zweck/Verantwortung

Beschreiben Sie aus der Sicht eines Nutzers oder Klienten dieses Bausteins, welche Aufgabe dieser Baustein übernimmt beziehungsweise welche Verantwortung er im Rahmen des Gesamtsystems wahrnimmt.




**<Name-des-Bausteins>**

**Zweck/Verantwortung:**

**Schnittstelle(n):**

**Ablageort/Datei(en):**

Sparversion  
...



**<Name-des-Bausteins>**

**Zweck/Verantwortung:**

**Schnittstelle(n):**  
inkl. Leistungsmerkmale

**Ablageort/Datei:**

**Erfüllte Anforderungen:**

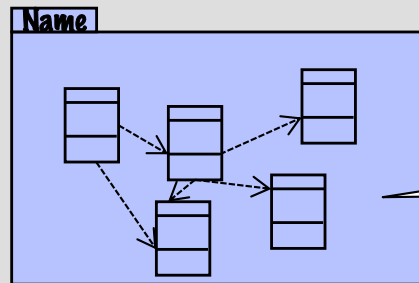
**Sonstige Infos:**

**Offene Punkte:**

# Whitebox-Template

<Name-des-Bausteins>

Übersicht:



**Diagramm mit  
feineren  
Bausteinen und  
deren  
Abhängigkeiten**

**Entscheidungen, Gründe & Alternativen**

.....  
.....

**Motivation für  
Zerlegung**

**Blackbox-Beschreibung der lokalen Bausteine**

Lokaler Baustein 1

Lokaler Baustein 2

.....

Lokaler Baustein n

„Innereien“ =  
Spezifikation enthaltenen  
Black-Boxes

**(interessante) Beziehungen zwischen lokalen Bausteinen**

.....  
.....

... und deren  
Zusammenhang  
(„benutzt“, „erbt von“,  
„instanziert“)

**Offene Punkte:** .....

.....



# Was macht gute Schnittstellen aus?

- Einfach zu benutzen
  - Einfach zu erlernen (intuitiv)
- Vollständig aus Benutzersicht
  - Deckt alle funktionalen Anforderungen ab
  - Erfüllt Qualitätsanforderungen
- Angemessene Dokumentation
- Client-Code leicht zu verstehen
- Neue Versionen brechen keinen bestehenden (Client-) Code



# (Ein) Ratschlag

## Robustheitsprinzip

(aka **Postel's Law**):

„be **conservative** in what you do,  
be **liberal** in what you accept from  
others“

„sei streng bei dem was du tust und offen  
bei dem was du von anderen akzeptierst“

– RFC 761

a general social code for software:  
“be strict in your own behavior, but  
tolerant of harmless quirks in  
others.”

## Vorteile:

- ▶ Möglicherweise höhere Robustheit
- ▶ (Kleine) Fehler anderer Bausteine führen nicht zu Abbruch

## Nachteile:

- ▶ Probleme werden u.U. von verschiedenen Bausteinen unterschiedlich behandelt
- ▶ Verletzt Separation of Concern und Kohäsion
- ▶ Verminderte Datenintegrität
- ▶ (Erheblicher) Mehraufwand

<https://devopedia.org/postel-s-law>

# Aufgaben bei Schnittstellenentwurf

## ■ **Formate**

- Syntax
- Namen + Datentypen
- Gültigkeits-/Validierungsregeln

## ■ **Semantik**

- Übertragungsweg und –protokoll
  - Technisches (Träger-)Protokoll
  - Ablauf der Interaktion / Handshake
- Vermittlung (zwischen Nehmer und Erbringer)
  - OO-Laufzeitsystem (z.B. Java-VM)
  - RPC
  - Broker

## ■ **Fehlerbehandlung**

## ■ **Versionierung / Evolution**

- Binär kompatibel
- Source kompatibel
- abwärtskompatibel

## ■ **Qualitätsanforderungen**

- Vertraulichkeit
- Berechtigung
- Nachweisbarkeit
- Durchsatz
- Robustheit / Verfügbarkeit

# Vorsicht: externe Schnittstellen...

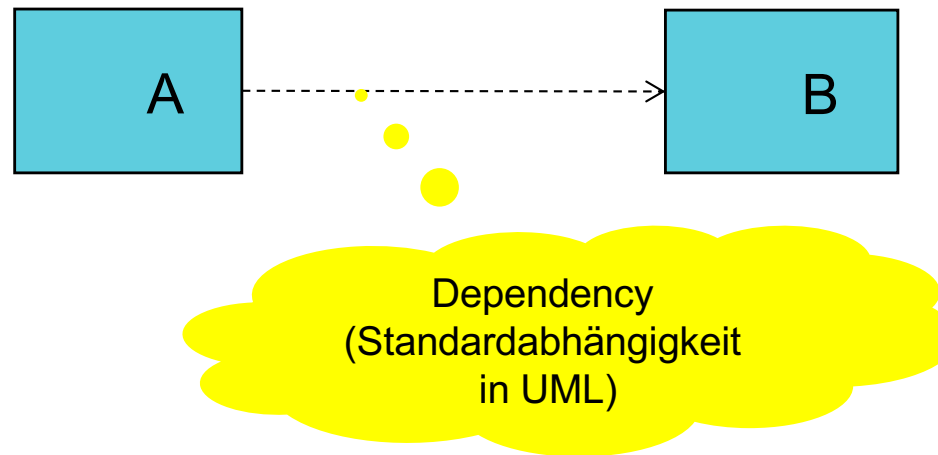
Relevante Informationen für (externe) Schnittstellen (Auszug):

- Formate / Typen der übertragenen Daten
- Gültigkeitsregeln für übertragene Daten (Validierungsregeln)
- Angebotene / genutzte Dienste (Services) / Funktionen
- Übertragungswege
  - Methoden/Funktionsaufrufe, Dateiübertragung, Datenträger- oder Materialaustausch
- Protokolle (im Sinne zeitlicher Abfolge / Handshakes)
- Organisatorische Regeln bei Nutzung
  - Begleitende Verträge
  - Bezahlung
- Verhalten im Fehlerfall
- Qualitätsanforderungen (QoS)
  - Zeitliche Verfügbarkeit
    - z.B. nur Wochentags, nur abends, nur nach Ereignis <x>
  - Notwendige Berechtigung zur Nutzung
    - Art der Überprüfung
  - erlaubte oder geforderte Mengengerüste

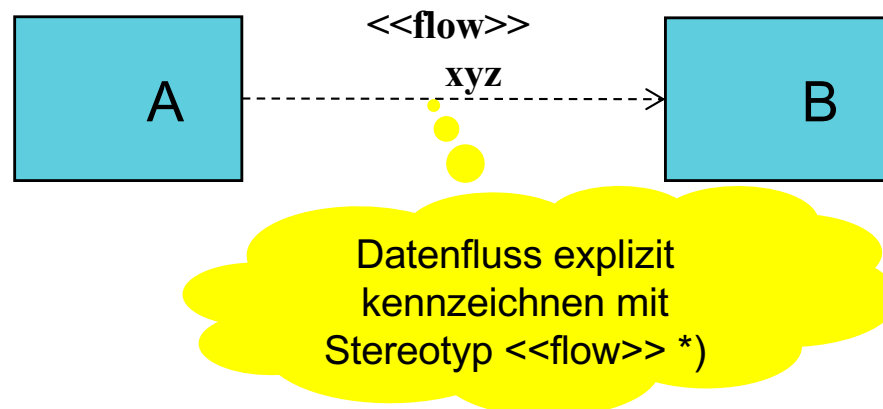
Schwierig!

# Zusammenarbeit von Bausteinen

- A benutzt B



- A schickt Daten an B

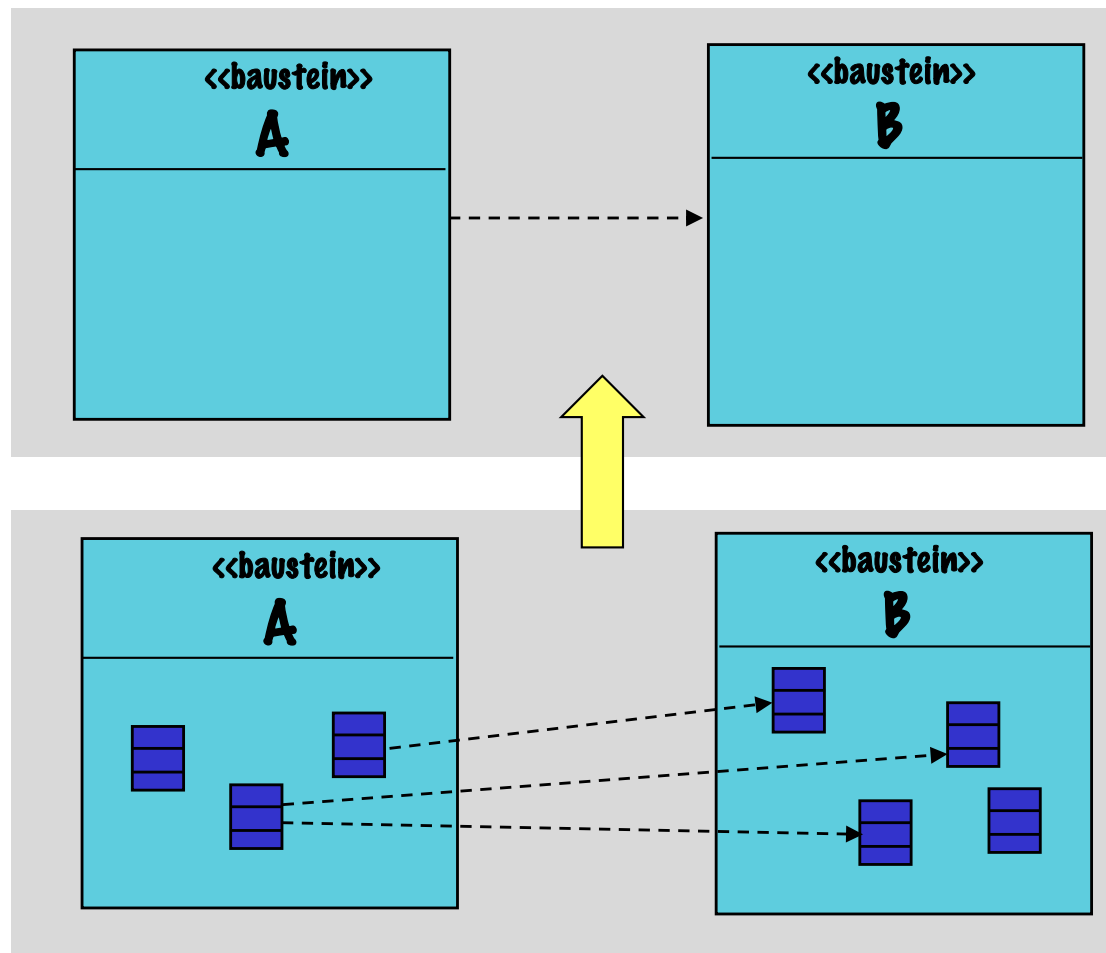


\*) oder Hausregel einführen: `—>` = Datenfluss  
Bausteinsicht 23



# Delegation zwischen größeren Bausteinen

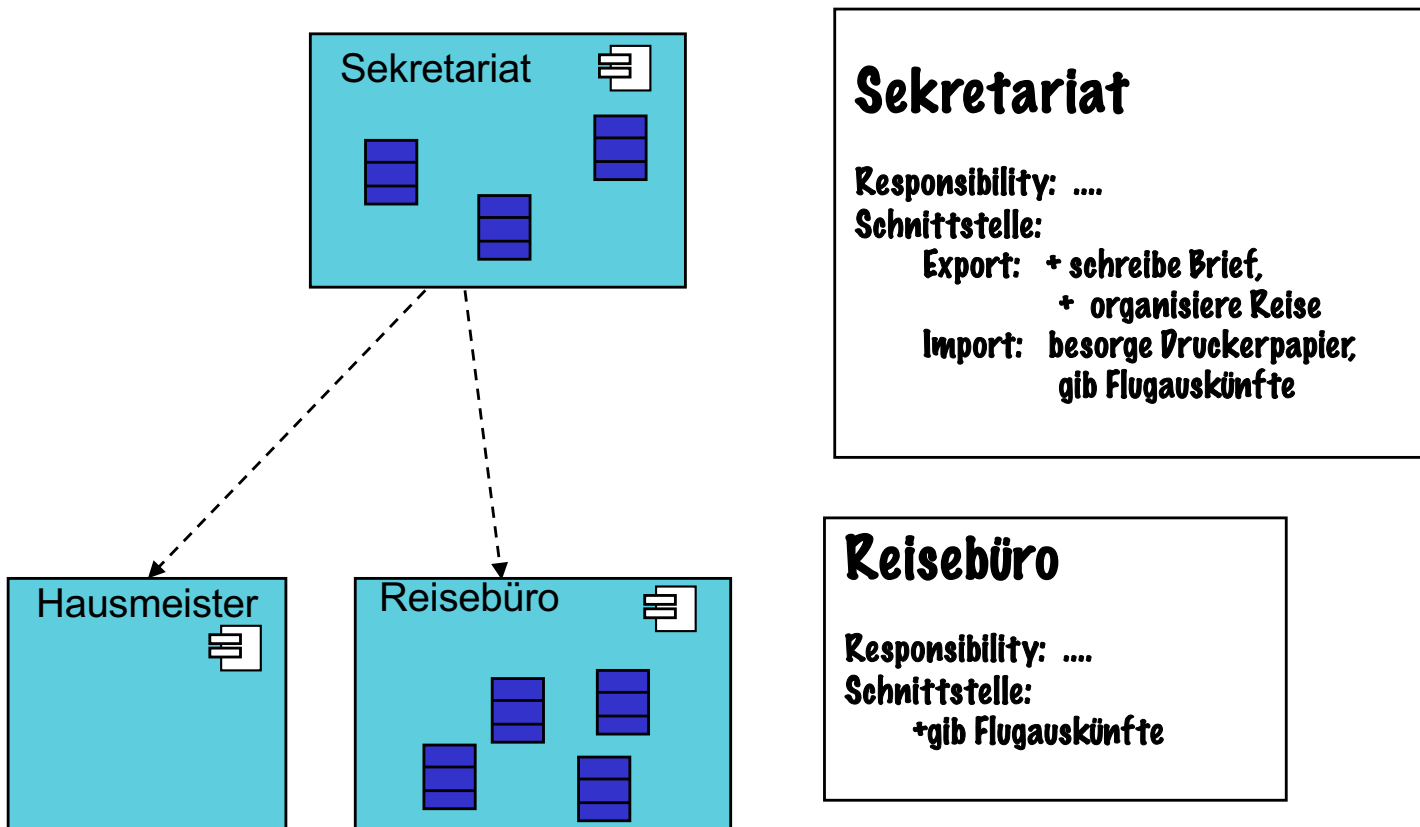
- Auch große Bausteine können andere Bausteine benutzen
  - Definition: Ein Baustein A benutzt einen anderen Baustein B, wenn es Bausteine in A gibt, die Nachrichten an Bausteine in B schicken.



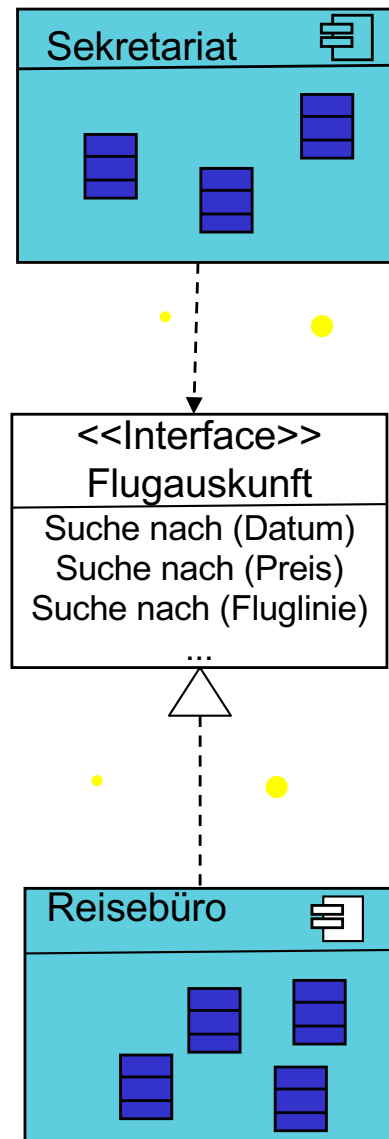
# Schnittstellenbeschreibung im Blackbox-Template

Bausteine sollten explizit zwei Arten von Schnittstellen festlegen:

1. was sie anderen bieten (Export Interface, Provided Interface) und
2. was sie von anderen erwarten (Import Interface, Required Interface)



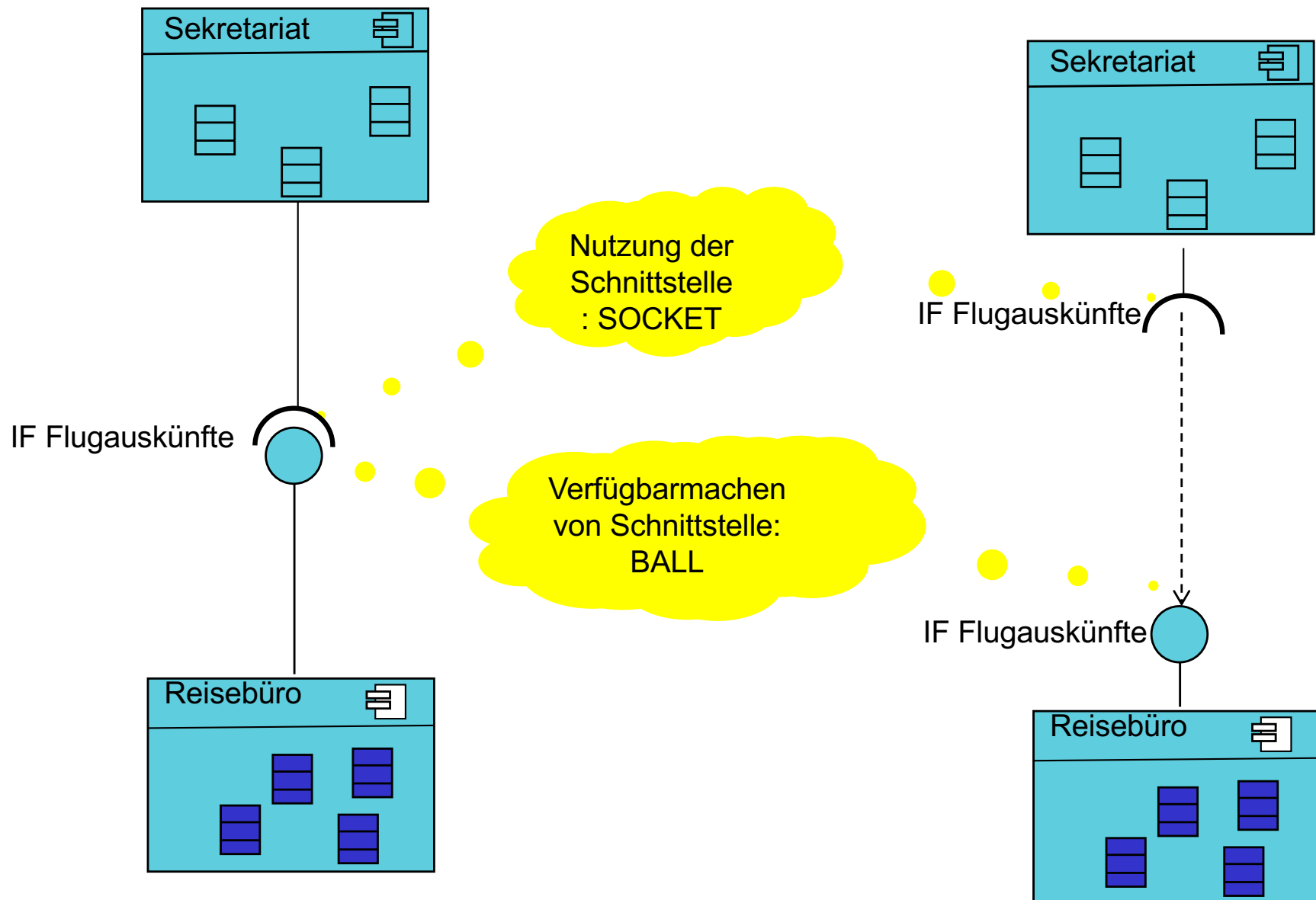
# Explizite Darstellung von Schnittstellen (1)



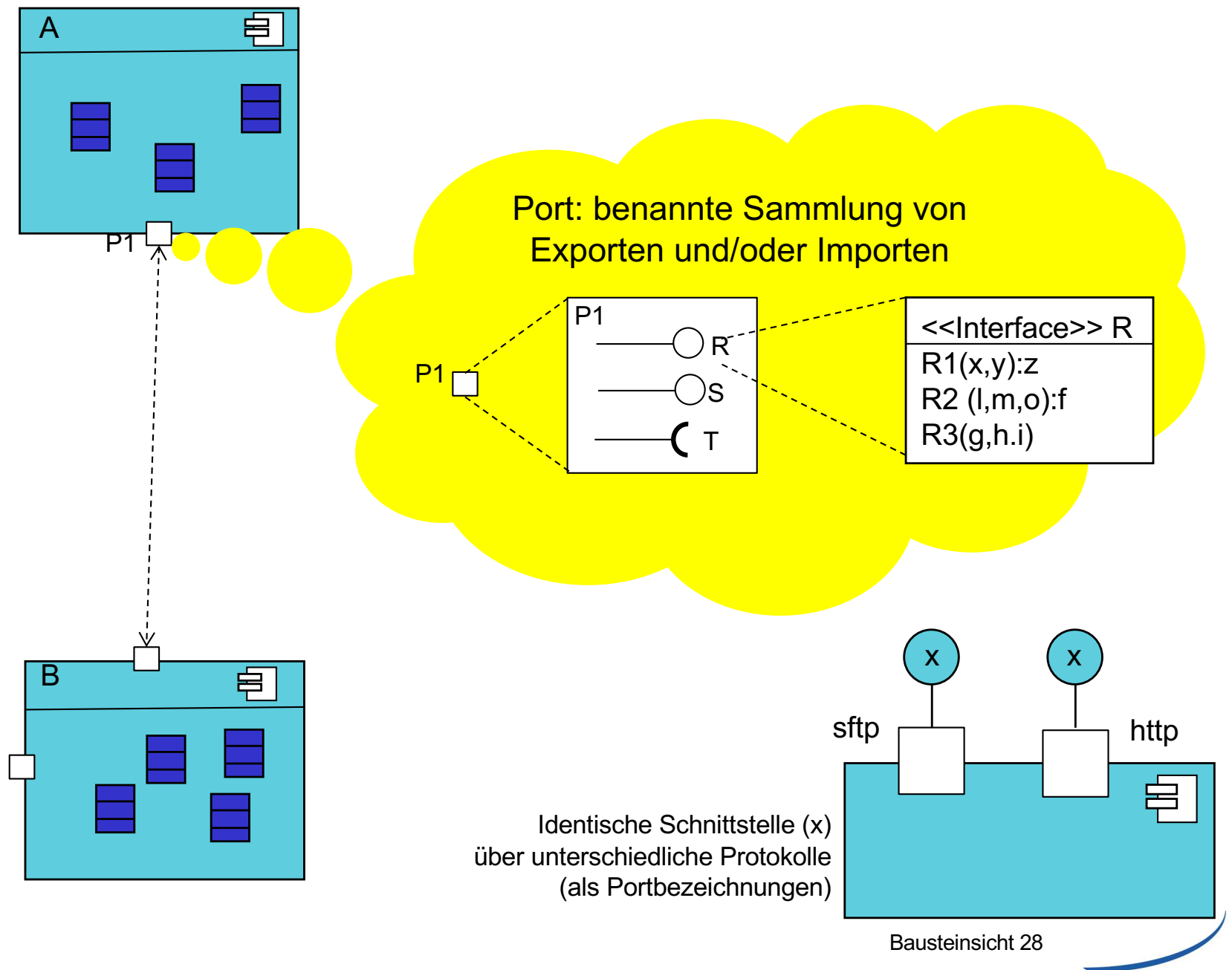
Nutzung der  
Schnittstelle

Implementierung  
der Schnittstelle

# Explizite Darstellung von Schnittstellen (2)



# Explizite Darstellung von Schnittstellen (3)





# Schnittstellenbeschreibung

## Außerschnittstellen:

Siehe  
Kontext-  
abgrenzung

- Eigenständige Dokumente
- Beachte Verantwortlichkeit (organisatorisch & technisch)

## Interne Schnittstellen:

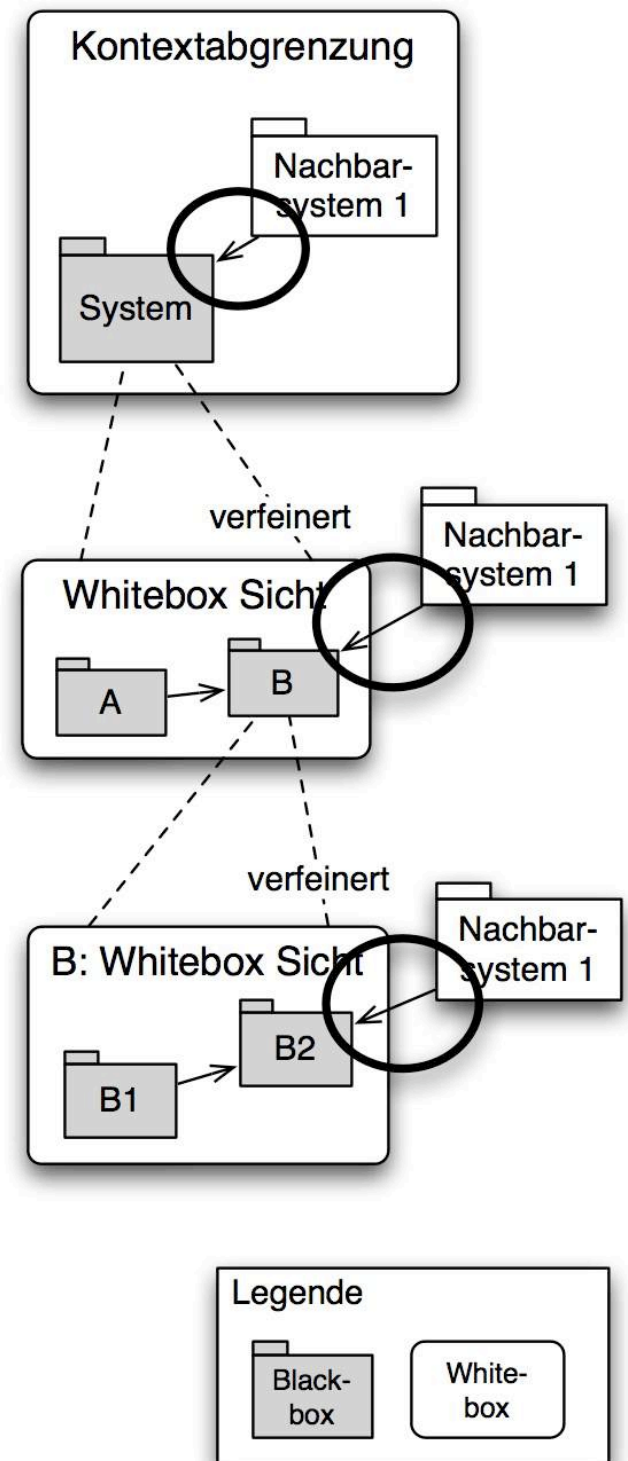
- Eher beim „Anbieter“ (Provider) beschreiben, oder
- ... als separate Beschreibung herauslösen  
(falls z.B. mehrere Anbieter derselben Schnittstelle)

Beschreibung durch:

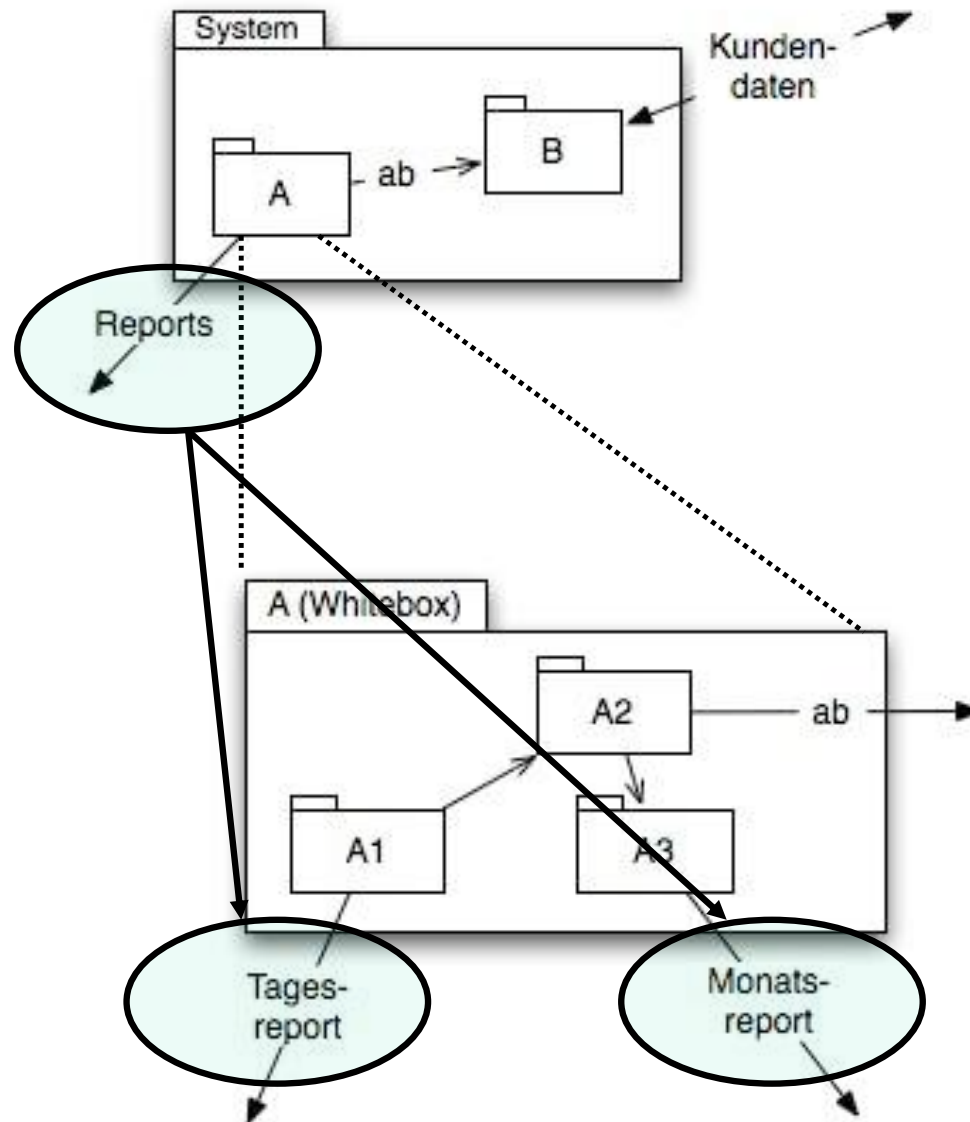
- Beispielhafte Nutzung (etwa: Unit-Tests)
- Schnittstellen-Template

# Wo beschreiben Sie Schnittstellen?

- Sie haben die Wahl...
  - Externe Schnittstellen: Eigene Dokumente
  - Interne Schnittstellen: Möglichst weit „oben“
  - „Oben“ grob (z.B: Datenfluss), „weiter unten“ präziser (komplette Signatur)
  
- Aber: (möglichst) redundanzfrei!



# Schnittstellen und Verfeinerung



# Links zu Schnittstellenentwurf

- Tech-Talk von Joshua Bloch zu „Good API Design“  
<https://www.youtube.com/watch?v=aAb7hSCtvGw>
- Kai Spichale: API-Design (dpunkt Verlag, 2016)
- J. Blanchette (Nokia/Trolltech) zu API-Design:  
<https://people.mpi-inf.mpg.de/~jblanche/api-design.pdf>
- Jaroslav Tulach (Architekt/Designer der Netbeans-IDE)  
<http://apidesign.org>
- Martin Reddy: API-Design for C++  
<http://apibook.com>



# Wohin mit den Ergebnissen?

## 1. Einführung und Ziele

- 1.1 Aufgabenstellung
- 1.2 Qualitätsziele
- 1.3 Stakeholder

## 2. Randbedingungen

- 2.1 Technische Randbedingungen
- 2.2 Organisatorische Randbedingungen
- 2.3 Konventionen

## 3. Kontextabgrenzung

### 3.1 Fachlicher Kontext

- 3.2 Technischer- oder Verteilungskontext

## 4. Lösungsstrategie

## 5. Bausteinsicht

- 5.1 Ebene 1
- 5.2 Ebene 2
- ....

## 6. Laufzeitsicht

- 6.1 Laufzeitszenario 1
- 6.2 Laufzeitszenario 2
- ....

## 7. Verteilungssicht

- 7.1 Infrastruktur Ebene 1
- 7.2 Infrastruktur Ebene 2
- ....

## 8. Konzepte

- 8.1 Fachliche Struktur und Modelle
- 8.2 Typische Muster und Strukturen
- 8.3 Persistenz
- 8.4 Benutzeroberfläche
- ....

## 9. Entwurfsentscheidungen

- 9.1 Entwurfsentscheidung 1
- 9.2 Entwurfsentscheidung 2
- ....

## 10. Qualitätsszenarien

- 10.1 Qualitätsbaum
- 10.2 Qualitäts-/Bewertungsszenarien

## 11. Risiken

## 12. Glossar

# Zusammenfassung



Finden von Bausteinen ist ein **zentrales Entwurfsproblem**

- Viele Ansätze zur **Zerlegung** von Problem in „Lösungsbausteine“:
  - fachlich/technische Trennung
  - Patterns
  - funktionale, räumliche, organisatorische...
- Zusammenhang zu „technischen Konzepten“

Beschreiben Sie Bausteine

- als **Hierarchie** (Detaillierung / Präzisierung / Abstraktion)
- mittels Black- und Whitebox-Templates
  - Diagramme zur Darstellung von (Whitebox-) Strukturen

entwerfen Sie  
möglichst  
iterativ!