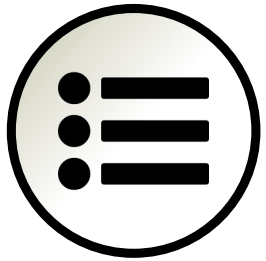
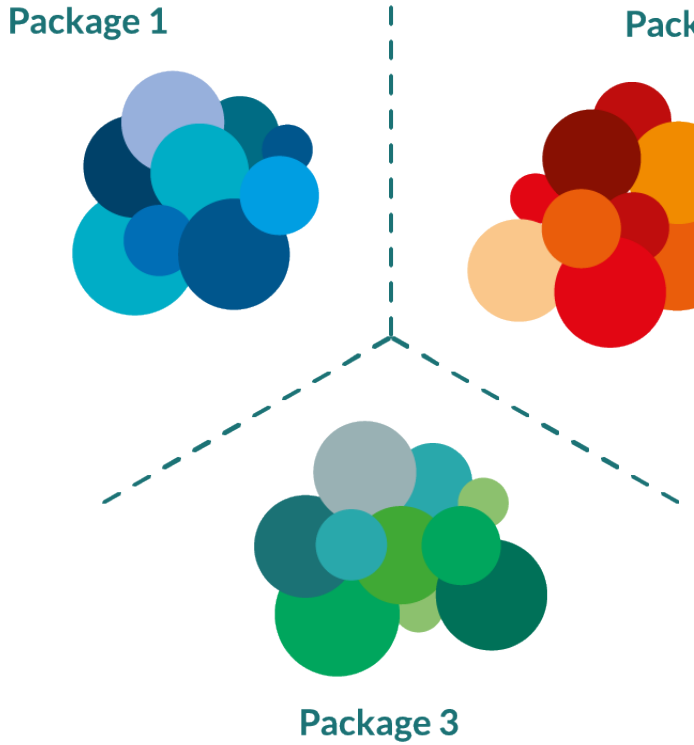


Die 5C-Design Prinzipien für einen nachhaltigen Softwareentwurf

SOLID von Robert C. Martin

- SOLID ist nicht vollständig. Es fehlt z.B. das Prinzip des Information- Hiding und damit eine der wichtigsten Handlungsmaximen beim Modulentwurf.
- SOLID hat den Fokus zu sehr auf die damals sehr populäre objektorientierte Programmierung (OOP).
- Das O (für Open-Closed) und das D (für Dependency-Inversion) in SOLID interpretieren wir heute anders als damals.
- Mit dem L (für Liksovsches-Substitutionsprinzip) und dem I (für Interface-Segregation) aus SOLID wird das Thema effizientes Schnittstellendesign doppelt, aber trotzdem nicht vollständig oder gut priorisiert, adressiert.

1. Effizient abtrennen (Cut)

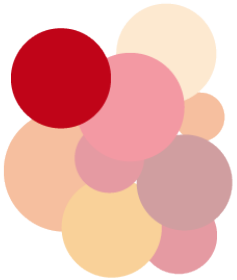


Je unabhängiger ein Baustein ist, desto einfacher wird er zu handhaben sein. Schneide Bausteine so, dass sie sich möglichst gut voneinander abgrenzen.

CUT

- Ein Baustein erfüllt nur eine konkrete Aufgabe,
  womit es nur einen Grund gibt, diesen zu ändern.
- Der innere Zusammenhalt (Kohäsion) ist möglichst groß, was zu einer geringeren Kopplung (siehe Schritt Connect) führt.
- Siehe auch:
  - Separation-of-Concerns
  - Single-Responsibility
  - Single-Level-of-Abstraction

2. Interna verbergen (Conceal)



geschützt

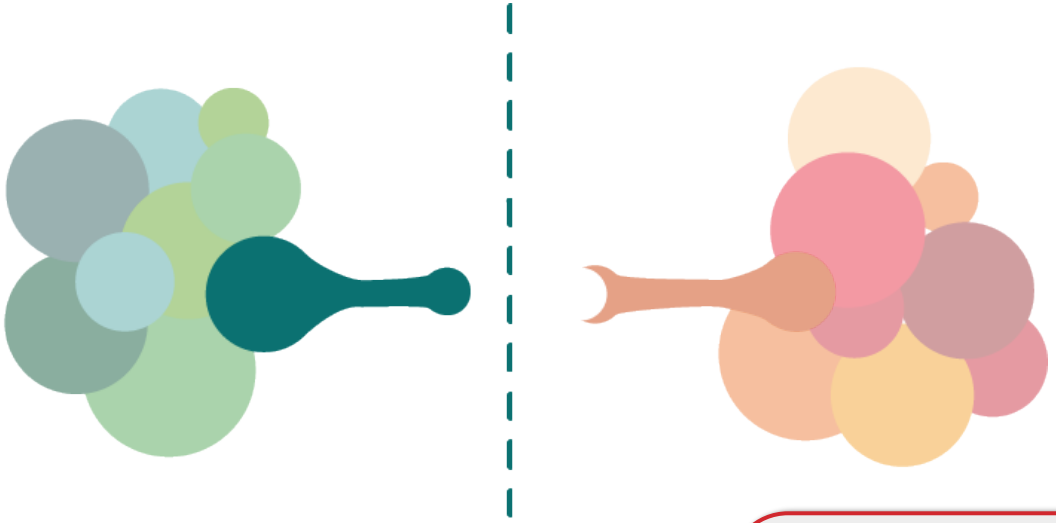
öffentlich

Verbirg so viel der internen Struktur eines Bausteins und der Art der Umsetzung vor der Außenwelt wie möglich.

CONCEAL

- Das Innenleben eines Bausteins kann ohne Abstimmungsaufwände, und auch ohne Gefahr von unerwünschten Seiteneffekten angepasst werden.
- Siehe auch:
  - Information-Hiding-Principle
  - Law-of-Demeter / Principle-of-Least-Knowledge

3. Schnittstellen festlegen (Contract)

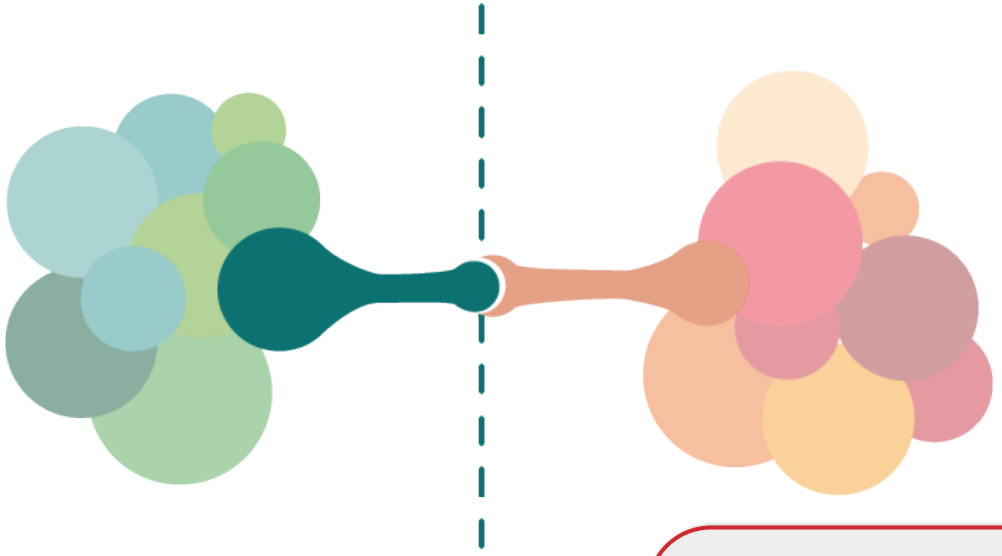


Entwerfe Schnittstellen so, dass eine möglichst reibungslose Interaktion zwischen dem Baustein und seinen Consumern möglich ist.

CONTRACT

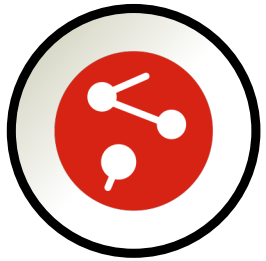
- Die Schnittstelle ist für den Consumer einfach zu verstehen und anzuwenden.
- Eine missbräuchliche Verwendung ist kaum möglich.
- Sie ist ausreichend konkret spezifiziert und dokumentiert.
- Ihr Entwurf ist schmal und dem jeweiligen Anwendungsfall angemessen.
- Siehe auch:
  - Interface-Segregation
  - Liskovsches-Substitutionsprinzip
  - Postel´s-Law / Robustheitsgrundsatz
  - Open-Closed-Principle

4. Verbinden (Connect)



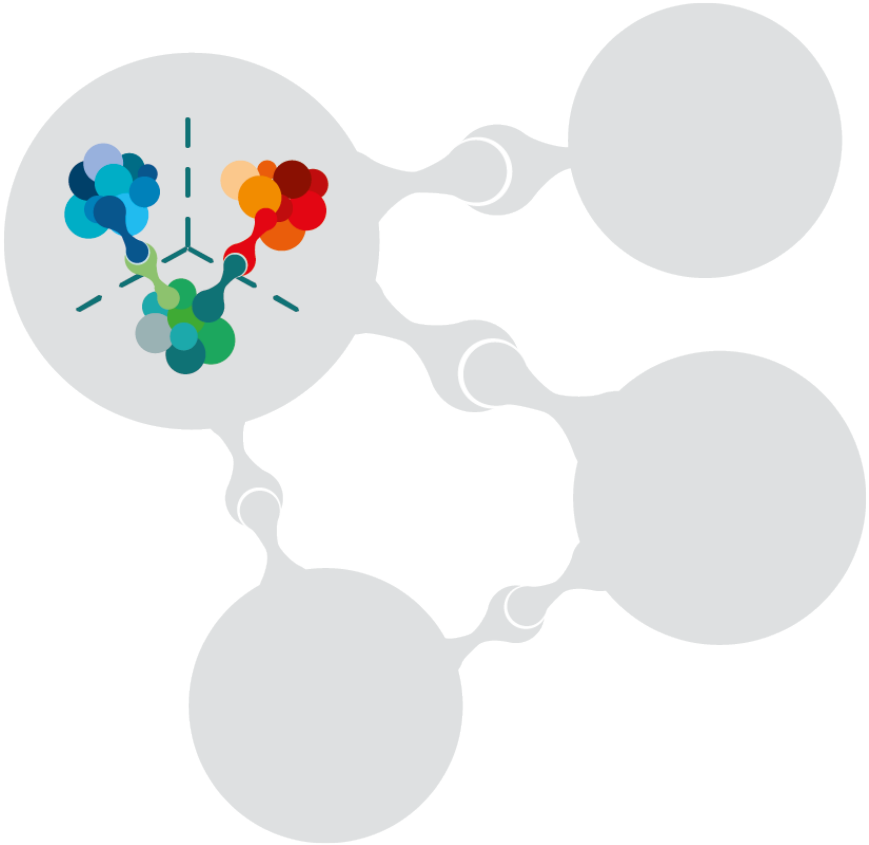
Durch Verwendung einer Schnittstelle eines anderen Bausteins kommt es immer zu Abhängigkeiten. Plane explizit zwischen welchen Bausteinen es welche Art von Abhängigkeit geben soll.

CONNECT



- Es herrscht lose Kopplung und es gibt wenig einschränkende Auswirkungen der Verbindungen.
  - Daten und Schnittstellenformate
  - Zeit
  - Laufzeitumgebung
  - Technologie
- Risiken und Probleme pflanzen sich nicht durch kaskadierende Abhängigkeiten fort.
- Es herrscht Ausgewogenheit, da es keine zentralen Bausteine mit sehr vielen Abhängigkeiten gibt.
- Siehe auch:
  - Dependency-Inversion

5. Hierarchien bauen (Construct)

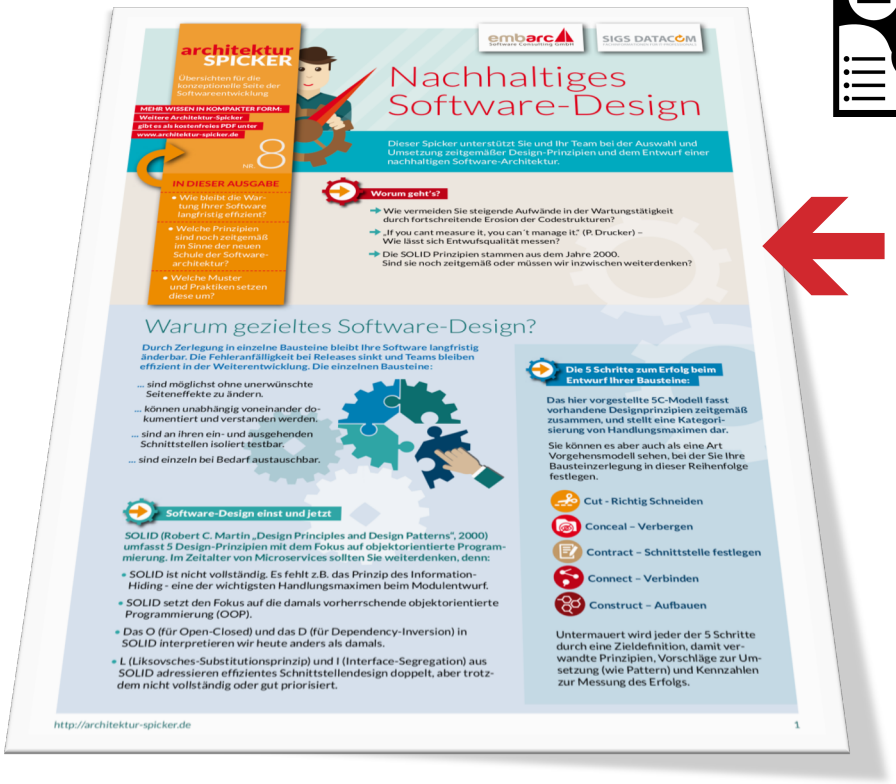


Eine Bausteinstruktur kann auf einer Ebene selbst wieder unübersichtlich werden. Baue Systeme höherer Komplexität durch Zusammenfassen von Bausteinen einer Ebene zu einem neuen Baustein einer nächsthöheren Ebene. Dabei sind weiterhin dieselben Handlungsmaximen anzuwenden.

CONSTRUCT

- Komplexere Systemlandschaften werden dadurch wartbarer.
- Eine aus dem Ufer geratene Komplexität kann den Unternehmenserfolg nicht gefährden.
- Verringerung von Abstimmungsaufwänden und Bürokratie.
- Siehe auch:
  - Divide-et-Impera / Teile-und-Herrsche
  - Islands-of-Change
  - Top-Down und Bottom-up-Entwurf

Spicken erlaubt!



Unsere Architektur-Spicker beleuchten die konzeptionelle Seite der Softwareentwicklung.

Spicker #8:
„Nachhaltiges Software-Design“

- Wie bleibt die Wartung Ihrer Software langfristig effizient?
- Welche Prinzipien sind noch zeitgemäß im Sinne der neuen Schule der Softwarearchitektur?
- Welche Muster und Praktiken setzen diese um?

PDF, 4 Seiten
Kostenloser Download.

➔ embarc.de/architektur-spicker/