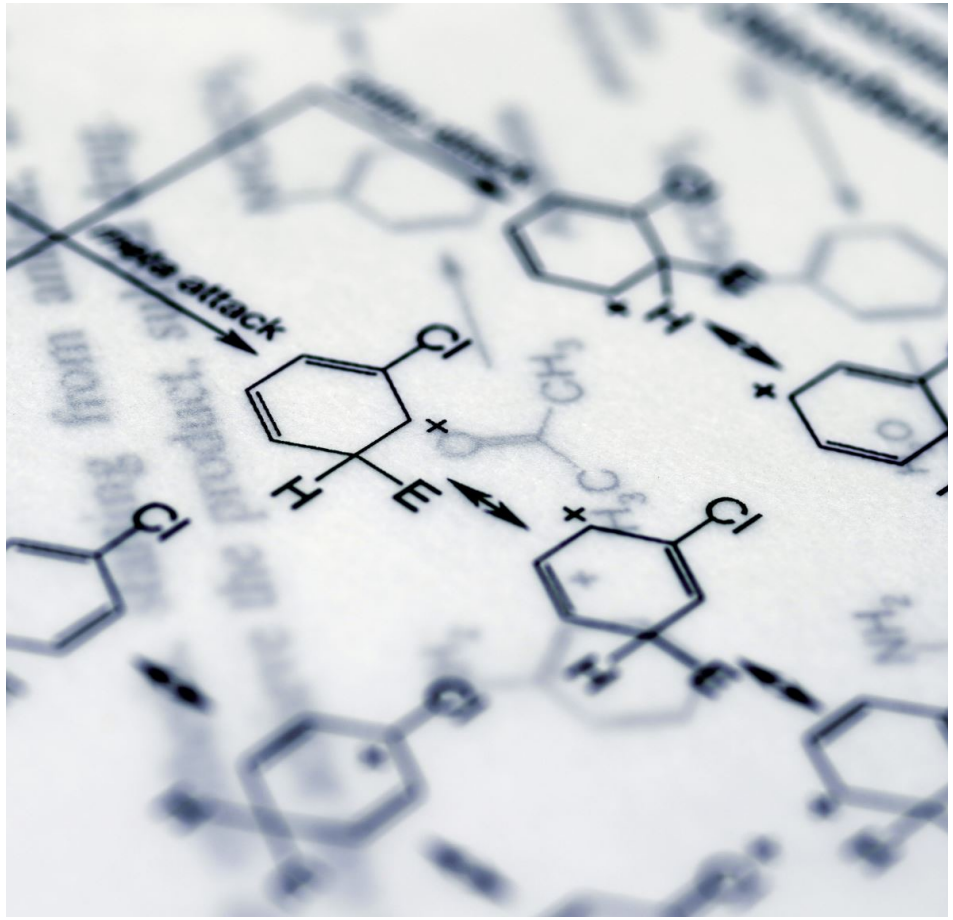
# Themenbereiche der Vorlesung




Themenbereiche

**A – Grundlegende Konzepte**

B – Dokumentation und Kommunikation

C – Entwurf von Softwarearchitekturen

D – Architekturmuster

E – Qualität von Softwarearchitekturen

# Grundlegende Konzepte

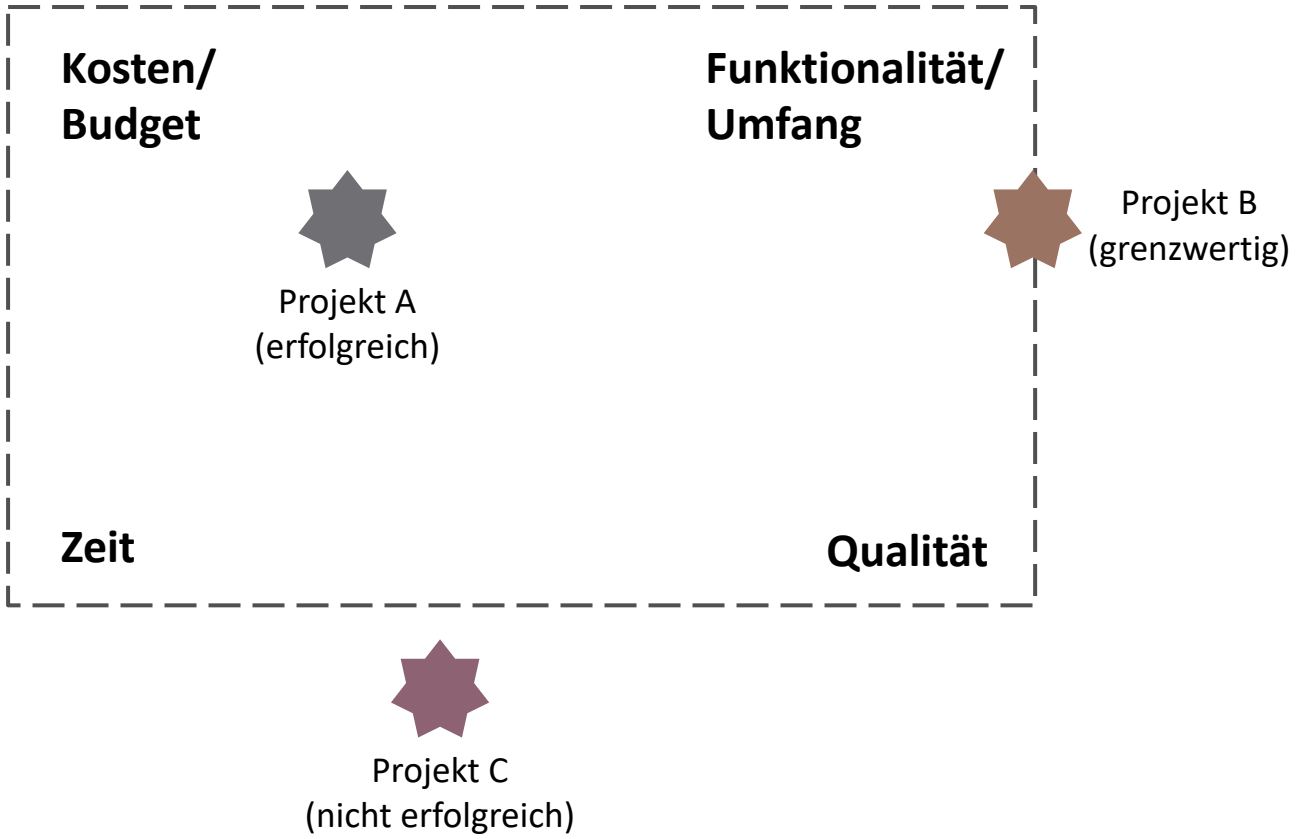
Themenbereich A

# Inhalte

- Bedeutung von Softwarearchitektur
- Definition Softwarearchitektur & Schnittstelle
- Komponenten
- Beispiel-Implementierung einer Komponente

# Bedeutung von Softwarearchitektur

# Das magische Viereck erfolgreicher Softwareprojekte




Softwaresysteme sind ein zentrales Rückgrat unserer Gesellschaft!

Die Fähigkeit IT-Projekte durchzuführen muss verbessert werden!

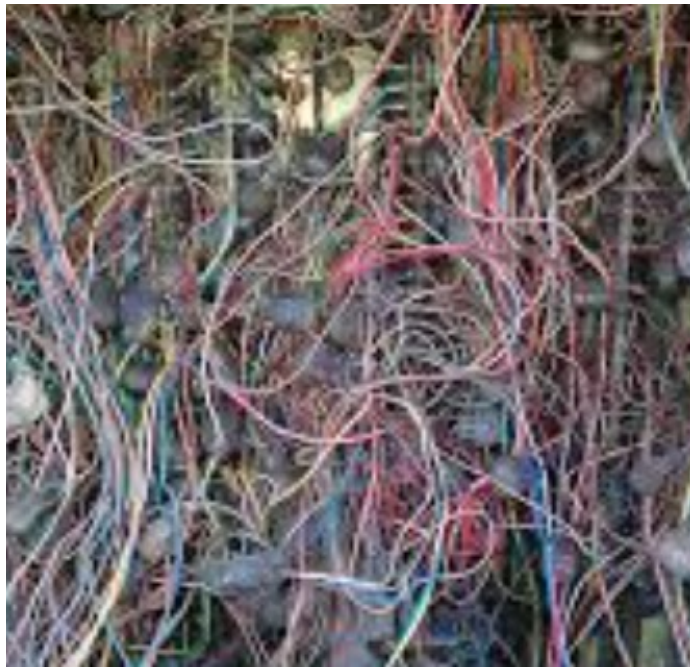
Software Engineering / Softwarearchitektur

# Warum sind Kenntnisse über Softwarearchitektur wichtig?

Gerade heutzutage sind Kenntnisse in Softwarearchitektur wichtig!

- Komplexität von Softwareprojekten
- Skalierbarkeit und Leistung
- Sicherheit und Datenschutz
- Technologische Vielfalt
- Agilität und kontinuierliche Bereitstellung

# Big Ball of Mud



- Big Ball of Mud (BBOM)
  - Begriff, um eine unstrukturierte und chaotische Softwarearchitektur zu beschreiben
- Typische Merkmale sind
  - Mangel an Struktur
  - Schlechte Modularisierung
  - Spaghetti-Code
  - Technologische Vielfalt
  - Wachsende Komplexität

**Die Entstehung eines BBOM muss unter allen Umständen vermieden werden!**

**Softwarearchitektur-Arbeit ist hier von entscheidender Bedeutung!**

# Technische Schulden

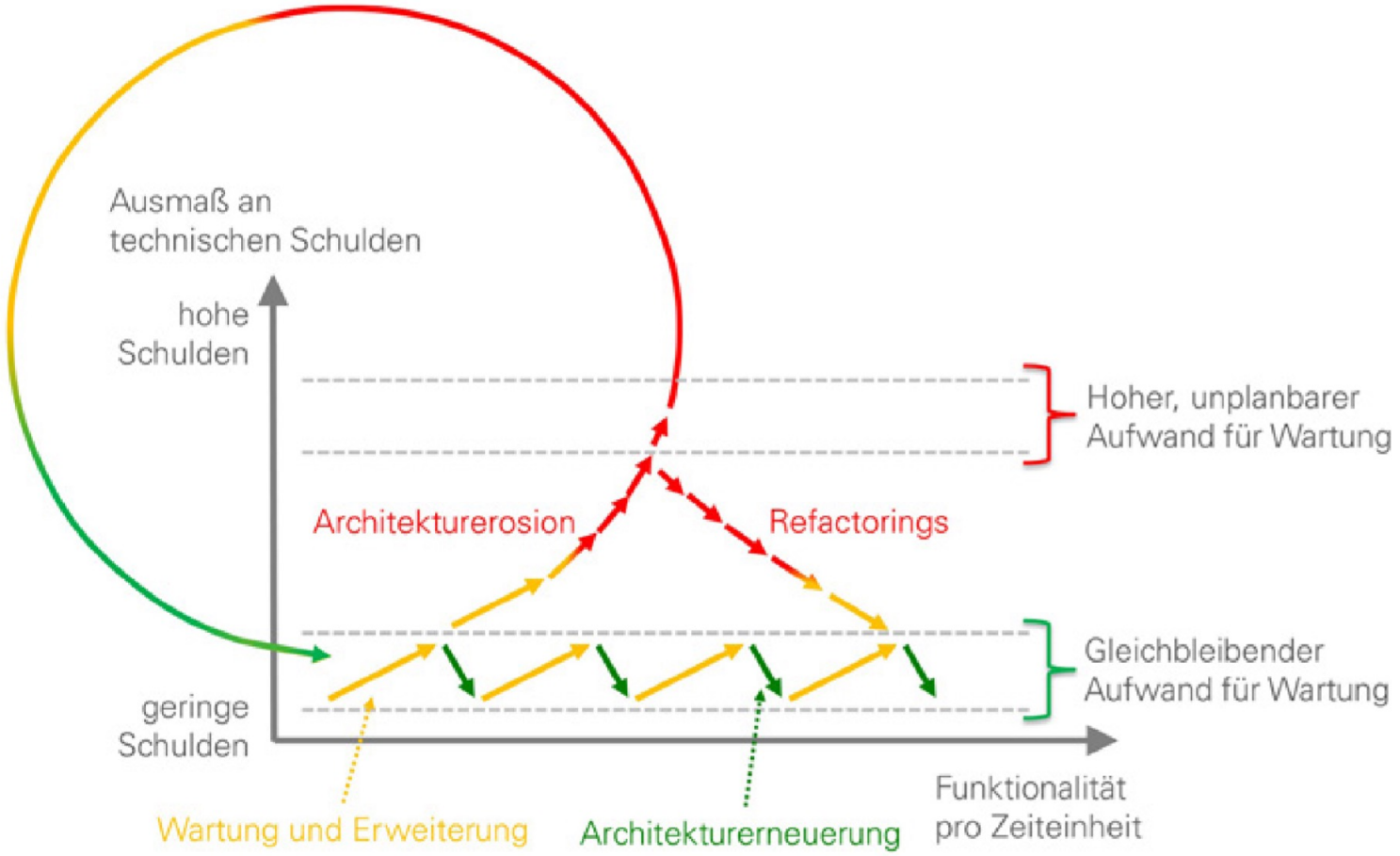
- Entstehen, wenn Teams bewusst Entscheidungen treffen, die kurzfristig Vorteile bringen, aber langfristig Auswirkungen auf die Qualität haben.
- Ähnlich wie finanzielle Schulden müssen auch technische Schulden irgendwann „zurückgezahlt“ werden durch Verbesserung des Source Codes.

# Ursachen und Arten technischer Schulden

- Schnelle Lösungen
- Veraltete Technologien
- Fehlende Tests
- Schlechte Codequalität
- Nicht behandelte Probleme

Rückzahlung technischer Schulden erfordert Zeit, Ressourcen und Engagement des Entwicklungsteams.

# Entwicklung und Effekt von technischen Schulden – Qualitätsmerkmal Wartbarkeit




*Quelle: Dr. Carola Lilienthal*

# Arten von IT-Architekten & Aufgaben


Typologie abgeleitet u.a. aus: Gellweiler, C. (2020). Types of IT architects: A content analysis on tasks and skills. Journal of Theoretical and Applied Electronic Commerce Research, 15(2), 15–37.

# Typische Aufgaben von Softwarearchitekten

- Architektonische Planung
- Technologieauswahl
- Code-Reviews und Best Practices
- Skalierung und Performance
- Risikomanagement
- Kommunikation und Zusammenarbeit
- Wartbarkeit und Erweiterbarkeit
- Qualitätssicherung

# Wann ist eine Softwarearchitektur erfolgreich?

**Der Erfolg einer Softwarearchitektur bemisst sich am Erfolg der Software!**

- Beitrag zur Werterbringung leisten!

- **Der Erfolg bemisst sich NICHT an der**
  - Qualität der Diagramme
  - Anzahl der Diagramme
  - Komplexität der Diagramme
  - Anzahl der verwendeten Architekturmuster
  - ...

# Notwendige Fähigkeiten von IT-Architekten

| Kompetenz-Kategorien | Alle Architekten (112) [%] | Enterprise Architekt (54) [%] | Solution Architekt (43) [%] | Software Architekt (15) [%] |
|---|---|---|---|---|
| Sozial | 82,1 | 74,1 | 90,7 | 86,7 |
| Technisch | 89,3 | 79,6 | 97,7 | 93,3 |
| Methoden | 91,1 | 90,7 | 88,4 | 93,3 |
| Standards / Frameworks | 28,6 | 38,9 | 20,9 | 20,0 |
| Business | 39,3 | 51,9 | 34,9 | 20,0 |
| Zertifikate | 8,9 | 11,1 | 7,0 | 6,7 |
| Rechtlich/ Regulatorisch | 6,3 | 11,1 | 2,3 | 0,0 |

Anzahl Stellenausschreibungen

Prozent der Stellenausschreibungen, welche die jeweilige Kompetenz-Kategorie voraussetzen.

Angelehnt an Tabelle 4 aus: Gellweiler, C. (2020). Types of IT architects: A content analysis on tasks and skills. Journal of Theoretical and Applied Electronic Commerce Research, 15(2), 15–37.

# Definition Softwarearchitektur & Schnittstelle

# Softwarearchitektur nach ISO/IEC/IEEE 42020:2019

> Fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes.
>
> [ISO/IEC/IEEE 42020:2019]

„Grundlegende Konzepte oder Eigenschaften einer Entität in ihrer Umgebung und leitende Prinzipien für die Realisierung und Entwicklung dieser Entität und der damit verbundenen Lebenszyklusprozesse.“

# Softwarearchitektur nach ISO/IEC/IEEE 42020:2019

> Fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes.
>
> [ISO/IEC/IEEE 42020:2019]

> **Fundamental concepts or properties**
>
> Are usually intended to be embodied in the entity‘s components, the relationships between components, and the relationships between the entity and its environment.
>
> [ISO/IEC/IEEE 42020:2019]

„Sie sollen in der Regel durch die Komponenten der Entität, die Beziehungen zwischen den Komponenten und durch die Beziehungen zwischen der Entität und ihrer Umgebung verkörpert werden.“

# Softwarearchitektur nach ISO/IEC/IEEE 42020:2019

> Fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes.
>
> [ISO/IEC/IEEE 42020:2019]

> **architecture entity (Architektureinheit)**
>
> thing being considered, described, discussed, studied or otherwise addressed during the architecting effort
>
> [ISO/IEC/IEEE 42020:2019]

**Beispiele**

- Unternehmen
- Organisationen
- **Systeme (insbesondere Softwaresysteme)**
- **Subsysteme**

# Zentrale Aspekte der Definition von Softwarearchitektur

- Fundamentale Konzepte und Eigenschaften eines Systems
  - Die Komponenten des Systems
  - Die Beziehungen/Schnittstellen zwischen den Komponenten
  - Die Beziehungen zwischen dem System und seiner Umgebung

# Definition: Schnittstelle

Wir verwenden folgende Definition:

> Eine Schnittstelle repräsentiert einen **wohldefinierten Zugangspunkt zum System oder zu dessen Komponenten**. Dabei beschreibt eine Schnittstelle die **Eigenschaften dieses Zugangspunkts**, wie z.B. Attribute, Daten und Funktionen.

Quelle: Kleuker, S. (2018). *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten* (4. Aufl.). Wiesbaden: Springer Vieweg. http://doi.org/10.1007/978-3-658-19969-2

# Angebotene und benötigte Schnittstellen

- Angebotene Schnittstelle
  - Wird von einer Komponente bereitgestellt
  - Definiert, welche Dienste oder Funktionen verfügbar sind
  - Andere Komponenten können diese Schnittstelle nutzen, um Dienste in Anspruch zu nehmen
- Angeforderte Schnittstelle
  - Wird von einer Komponente benötigt
  - Definiert Abhängigkeiten und Anforderungen an andere Komponenten
  - Komponente kann nur dann funktionieren, wenn alle angeforderten Schnittstellen von anderen Komponenten bereitgestellt werden

# Verantwortung für Schnittstellen

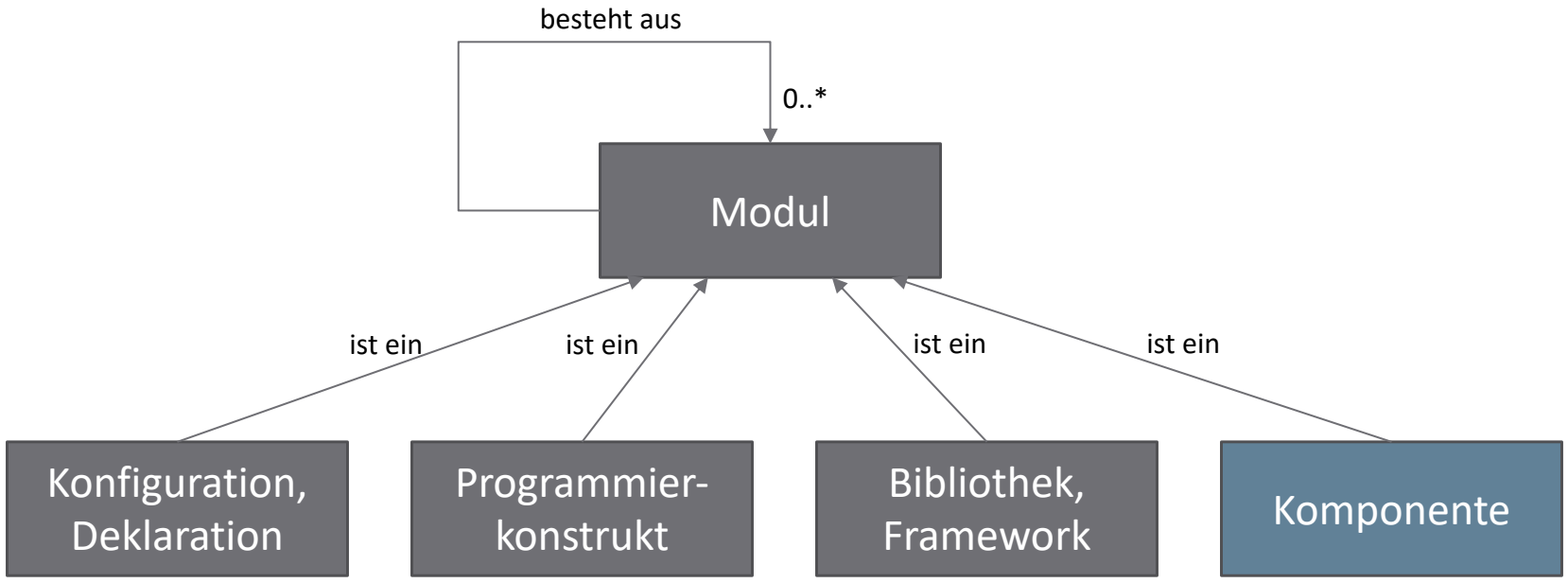
- Die Verantwortung für Schnittstellen kann unterschiedlich verteilt sein
  - **Normungsorganisationen**
    - Für allgemein akzeptierte Schnittstellen
    - Meist standardisiert und definiert durch Organisationen wie zum Beispiel ISO, IEEE, etc.
    - Beispiele für Schnittstellen: USB, HDMI, TCP/IP
    - Normungsorganisationen definieren Schnittstellen; Implementierung und Wartung werden meist durch Drittanbieter vorgenommen
  - **Schnittstellenanbeiter**
    - Schnittstellenanbieter kann verantwortlich für die Definition, Entwicklung, Wartung und Dokumentation der Schnittstelle sein (ist meistens der Fall)
    - Beispiel: Google Maps API
      - Google ist Anbieter und übernimmt auch die Verantwortung für die Definition, Entwicklung, Wartung und Dokumentation dieser Schnittstelle

# Verantwortung für Schnittstellen

- Die Verantwortung für Schnittstellen kann unterschiedlich verteilt sein
  - **Schnittstellenverwender**
    - Es kann vorkommen, dass ein Verwender der Schnittstelle gleichzeitig die Gesamtverantwortung oder Teile der Verantwortlichkeiten für angebotene Schnittstellen übernimmt.
    - Beispiel: Entwicklung einer maßgeschneiderten API durch ein Softwareentwicklungsunternehmen für ein E-Commerce-Unternehmen
      - E-Commerce-Unternehmen
        - Ist durch E-Commerce-Unternehmen beauftragtes Unternehmen
        - Trägt inhaltliche Verantwortung für die Schnittstellen
        - Definiert Anforderungen an die Schnittstellen
      - Softwareentwicklungsunternehmen
        - Implementiert die Schnittstellen, ist ggf. zusätzlich verantwortlich für Dokumentation, Wartung und weiterer Aspekte

# Komponenten

# Modul vs. Komponente




- Datenstruktur
- DB-Tabelle
- O/R-Mapping

- Klasse
- Funktion
- Skript

- React.js
- Flutter
- JUnit
- Bootstrap
- Unity

*Siehe Kerneigenschaften einer Komponente auf den folgenden Folien*

In der Praxis werden diese Begriffe je nach Kontext häufig unterschiedlich interpretiert.

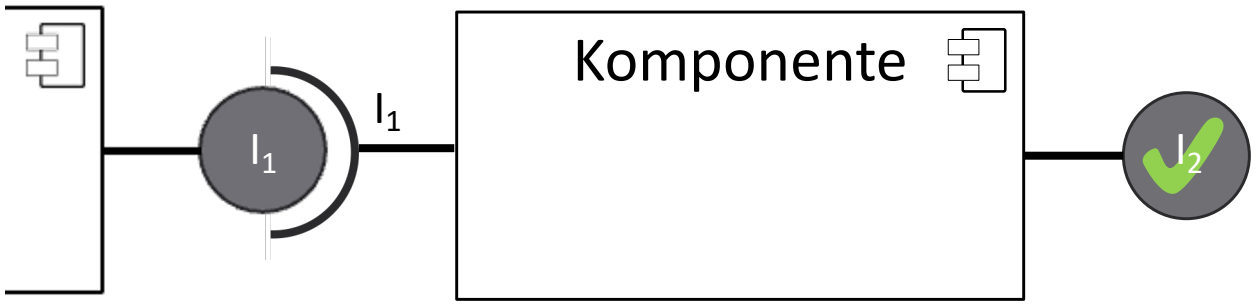
# Kerneigenschaften einer Komponente

Export und Import
von Schnittstellen

Kapselung und
Austauschbarkeit

Konfigurierbarkeit

# Export und Import von Schnittstellen



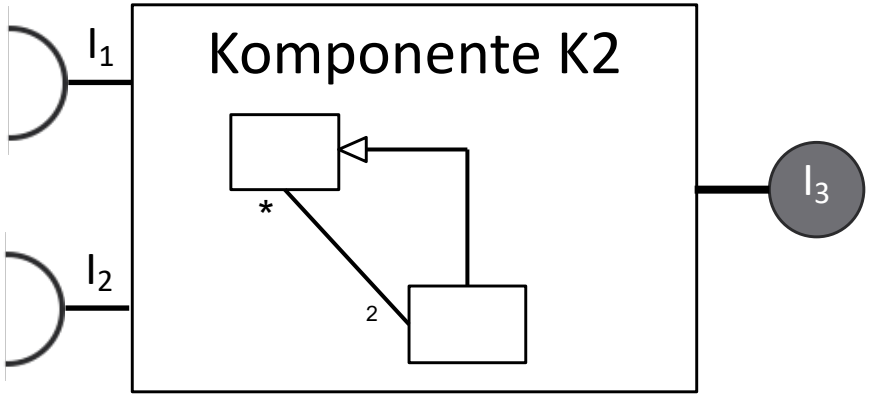

**Importierte Schnittstellen**

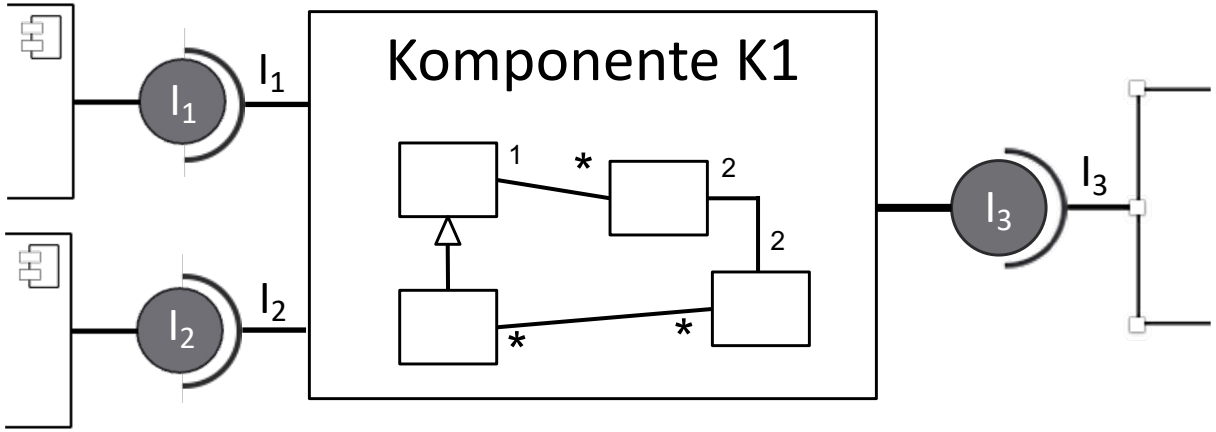
**Exportierte Schnittstellen**

- Eine Komponente bietet Schnittstellen an, die sie im Sinne eines Vertrages garantiert.
- Diese Garantie gilt unter der Bedingung, dass die von ihr benötigten Schnittstellen im Rahmen einer entsprechenden Konfiguration bereitgestellt werden.

# Austauschbarkeit und Kapselung

- Über die Schnittstellen kapselt eine Komponente die Implementierung
- Solange alle Schnittstellenverträge erfüllt werden, können Komponenten ersetzt werden.



# Konfigurierbarkeit

**Arten von Konfigurierbarkeit**

- Konfiguration von Eigenschaften und Verhalten
  - Allgemeine Einstellungen, Aussehen, Verhalten, …
- Komposition von Komponenten
  - Verbindungen zwischen Komponenten zur Erstellung eines vollständigen Anwendungssystems
- Dynamische vs. statische Konfiguration

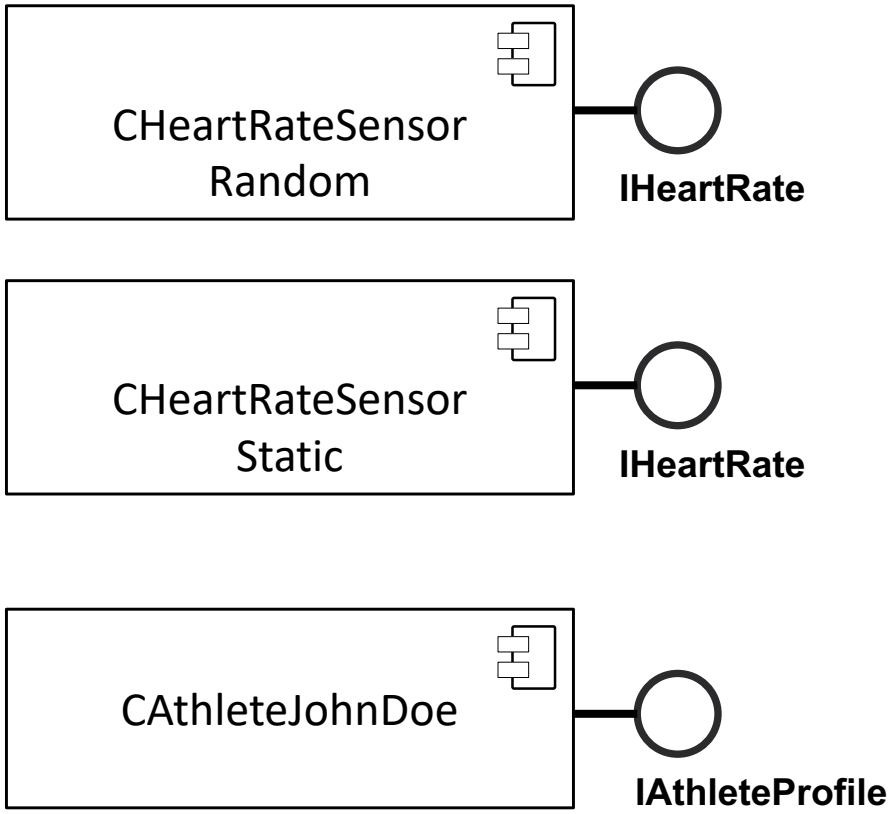
- Voraussetzung von Konfigurierbarkeit: Effektives Konfigurationsmanagement
  - Konfigurationsdateien
  - Versionierung
  - Fehlermanagement

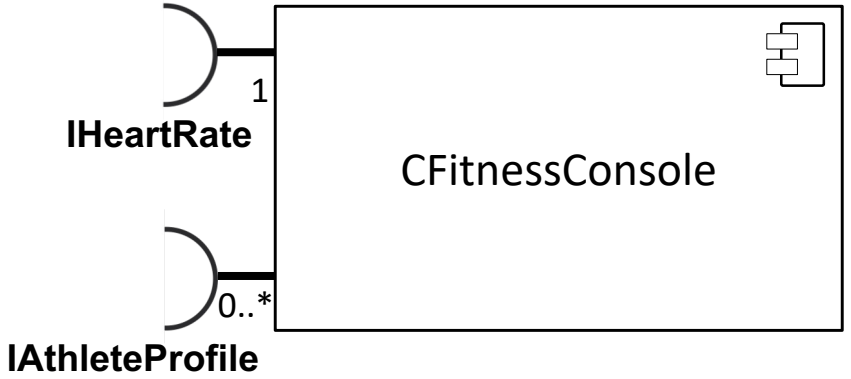
# Beispiel-Implementierung einer Komponente

# Informationen zum Beispiel

- Das folgende Beispiel zeigt eine Möglichkeit, grundlegende Konzepte Komponenten-basierter Anwendungen zu programmieren.
- Es wurde absichtlich auf die Verwendung von Frameworks verzichtet.
- Beispiel aus der Fitness-Domäne, aber keine echte Funktionalität
  - Stattdessen Fokus auf Codestruktur und Abhängigkeiten
- Die Anwendung gibt einen simulierten Puls eines Sportlers sowie dessen Namen auf der Konsole aus.

# Komponenten der Beispielanwendung

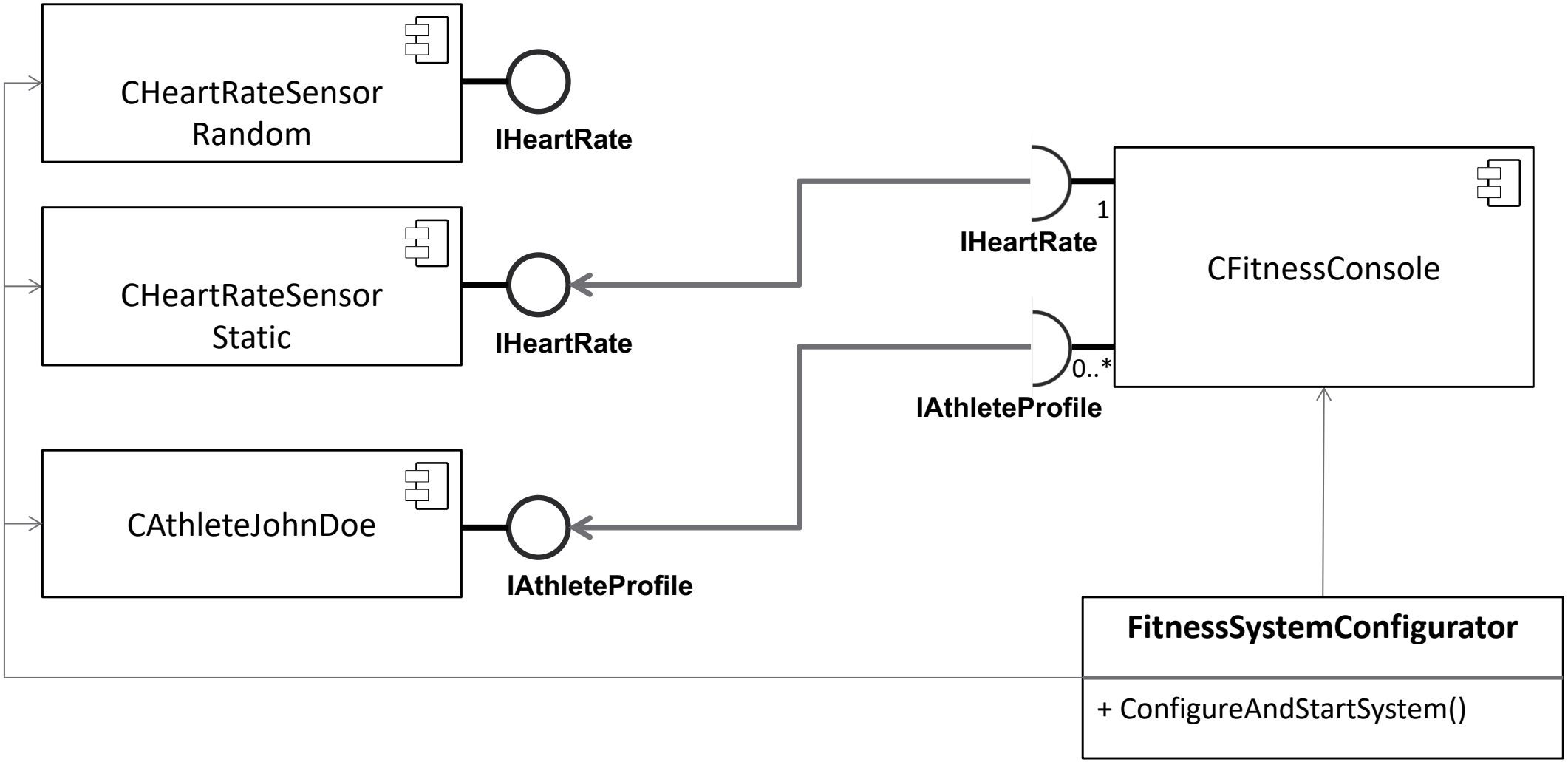


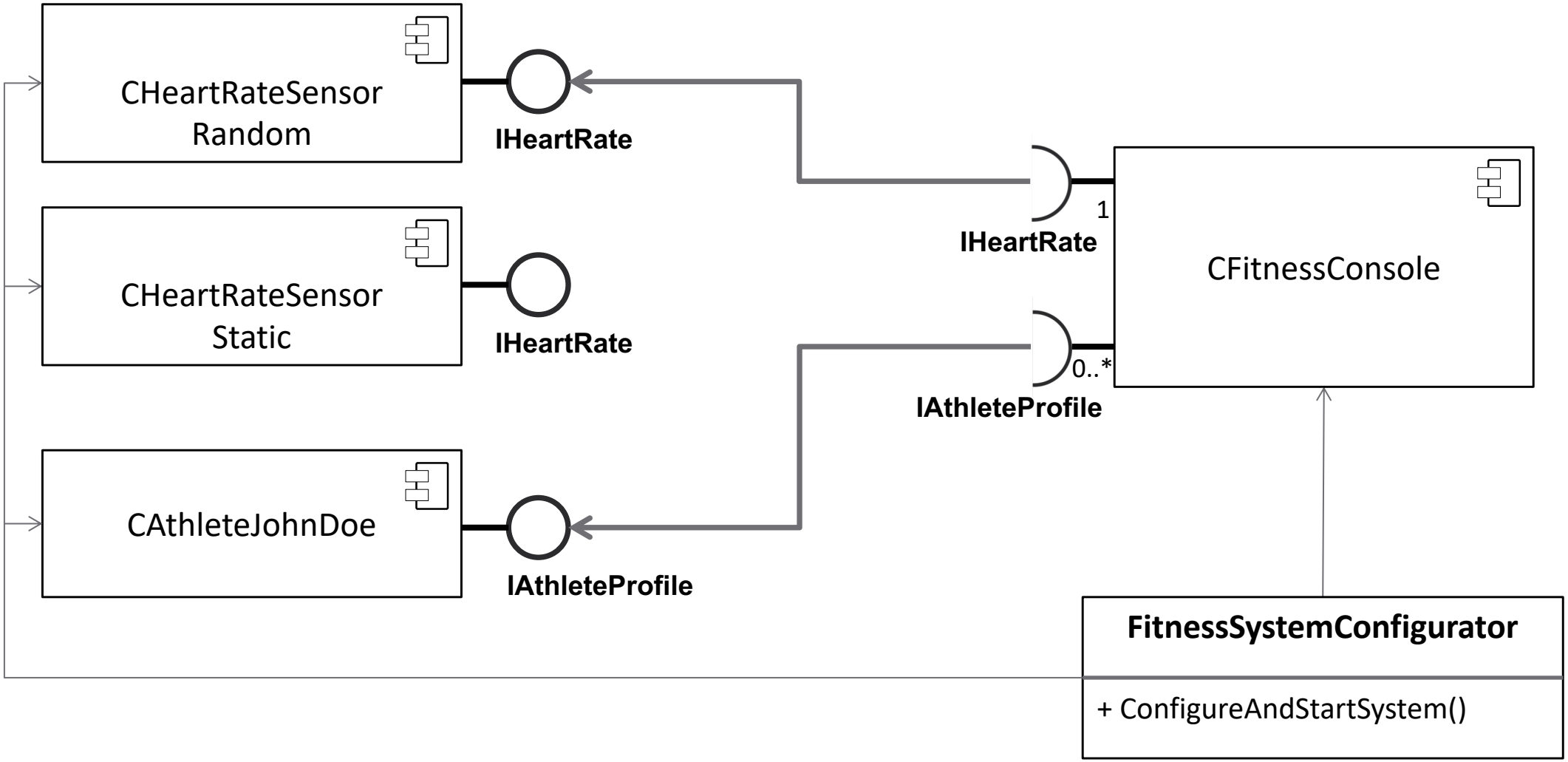

**FitnessSystemConfigurator**

+ ConfigureAndStartSystem()

# Beispielkonfiguration 1

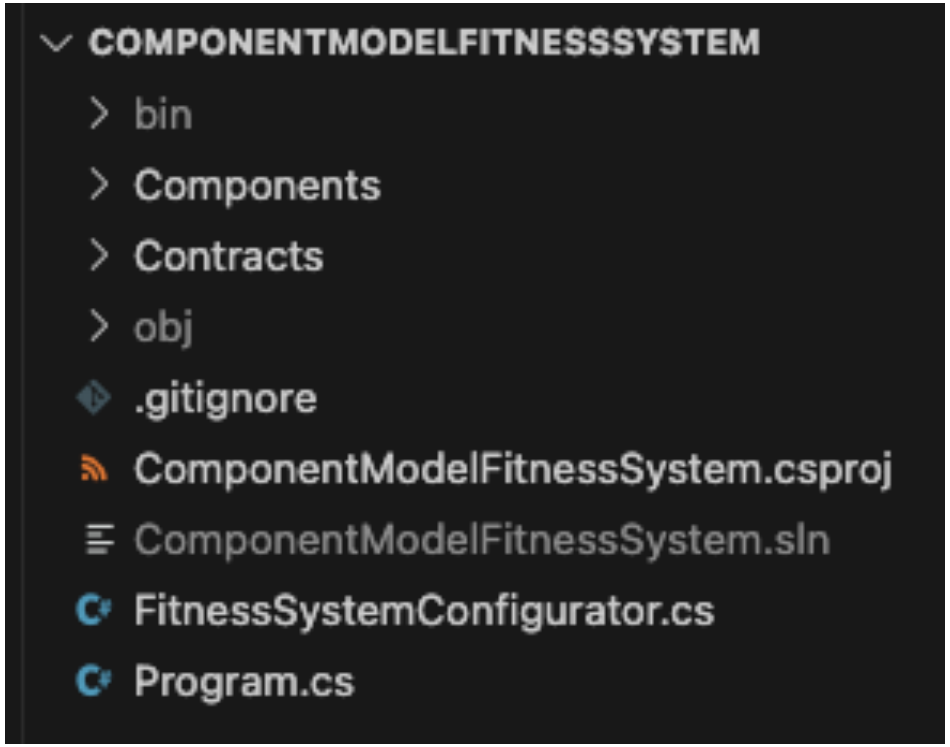


# Beispielkonfiguration 2



# Beispiel-Implementierung in C# Ordner-Struktur

**Components**
- Enthält alle Komponenten des Systems

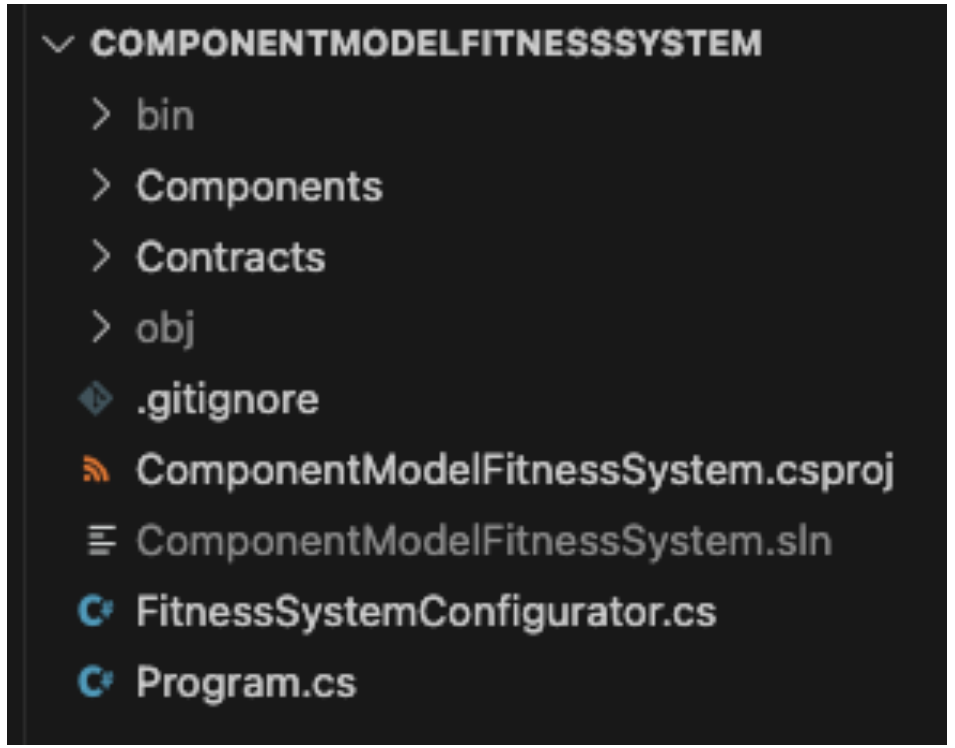



**Contracts**
- Enthält
  - Schnittstellen
  - Data Transfer Objects (Klassen, die als Parameter in den Schnittstellen verwendet werden)

**FitnessSystemConfigurator.cs**
- Verantwortlich für die Konfiguration des Gesamtsystems (insbesondere für die Verknüpfung zwischen Komponenten)

# Beispiel-Implementierung in C# Abhängigkeiten

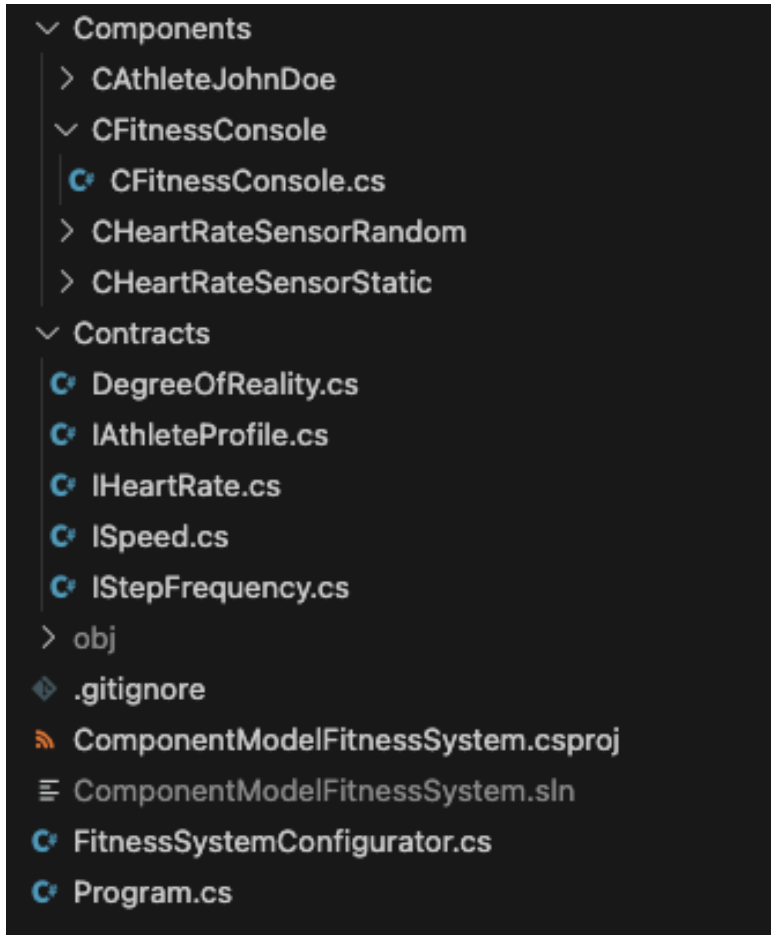
Konfigurator ist einzige Klasse, die Abhängigkeiten zu allen anderen Assets besitzt.




- Komponenten sind von Schnittstellen abhängig, nicht umgekehrt!
- Komponenten haben keine direkten Abhängigkeiten untereinander

Program „kennt“ nur den Konfigurator

# Beispiel-Implementierung in C# Komponenten




Komponenten (hier 4 Komponenten)

- Jede Komponente enthält eine Klasse, die die Komponenten repräsentiert (z.B. die Klasse CFitnessConsole.cs für die Komponente CFitnessConsole)
- Jede Komponente kann zusätzlich zu dieser Klasse zahlreiche weitere Klassen und Dateien enthalten (dies ist in realen Anwendungen meist immer der Fall)

# Beispiel-Implementierung in C# Schnittstellen & DTOs

```
3       public interface IAthleteProfile
4       {
            2 Verweise
5           public string GetFirstName();
6
            2 Verweise
7           public string GetLastName();
8
            1 Verweis
9           public DateOnly GetBirthday();
10
            1 Verweis
11          public double GetWeight();
12      }
13  }
```

```
3       public interface IHeartRate
4       {
            4 Verweise
5           public int GetCurrentHeartRate();
6       }
7   }
```

```
3       public enum DegreeOfReality {
            0 Verweise
4           HIGH,
            1 Verweis
5           LOW
6       }
7   }
```

# Beispiel-Implementierung in C# Komponenten

```
4   public class CAthleteJohnDoe : IAthleteProfile
5   {
6
        1 Verweis
7       public DateOnly GetBirthday()
8       {
9           return new DateOnly(1983, 8, 28);
10      }
11
        2 Verweise
12      public string GetFirstName()
13      {
14          return "John";
15      }
16
        2 Verweise
17      public string GetLastName()
18      {
19          return "Doe";
20      }
21
        1 Verweis
22      public double GetWeight()
23      {
24          return 86.5;
25      }
26  }
```

```
3   public class CHeartRateSensorRandom : IHeartRate
4   {
        1 Verweis
5       private readonly Random rand = new();
6
        1 Verweis
7       public CHeartRateSensorRandom(DegreeOfReality degreeOfReality)
8       {
9
10      }
11
        3 Verweise
12      public int GetCurrentHeartRate()
13      {
14          return rand.Next(30, 200);
15      }
16  }
```

```
3   public class CHeartRateSensorStatic : IHeartRate
4   {
        3 Verweise
5       public int GetCurrentHeartRate()
6       {
7           return 120;
8       }
9   }
```

# Beispiel-Implementierung in C# Komponenten

```
3   public class CFitnessConsole {
4
        3 Verweise
5       private IHeartRate HeartRateSensor { get; set; }
6
        4 Verweise
7       public IAthleteProfile? AthleteProfile {private get; set;}
8
        1 Verweis
9       public CFitnessConsole(IHeartRate heartRateSensor)
10      {
11          this.HeartRateSensor = heartRateSensor;
12      }
13
        1 Verweis
14      public void StartConsole() {
15          while(true) {
16              this.UpdateAndShowValues();
17              Thread.Sleep(1000);
18          }
19      }
20
        1 Verweis
21      private void UpdateAndShowValues() {
22          string statusMessage;
23
24          if (AthleteProfile == null)
25          {
26              statusMessage = $"HR: {HeartRateSensor.GetCurrentHeartRate()}";
27          } else
28          {
29              statusMessage = $"{AthleteProfile.GetFirstName()} {AthleteProfile.GetLastName()}: HR: {HeartRateSensor.GetCurrentHeartRate()}";
30          }
31
32          Console.WriteLine(statusMessage);
33      }
34  }
```

# Beispiel-Implementierung in C# Konfigurator & Program

```
3   public class FitnessSystemConfigurator {
        1 Verweis
4       public void ConfigureAndStartSystem() {
5           IHeartRate heartRateSensorStatic = new CHeartRateSensorStatic();
6           IHeartRate heartRateSensorRandom = new CHeartRateSensorRandom(DegreeOfReality.LOW);
7           IAthleteProfile athleteProfileJohnDoe = new CAthleteJohnDoe();
8
9           var fitnessConsole = new CFitnessConsole(heartRateSensorRandom);
10          fitnessConsole.AthleteProfile = athleteProfileJohnDoe;
11
12          fitnessConsole.StartConsole();
13      }
14  }
```

Dependency Injection

```
3   class Program
4   {
        0 Verweise
5       static void Main(string[] args)
6       {
7           new FitnessSystemConfigurator().ConfigureAndStartSystem();
8       }
9   }
```