Markus Schlegel, Active Group, 2024-11-06

Decoupled by Default

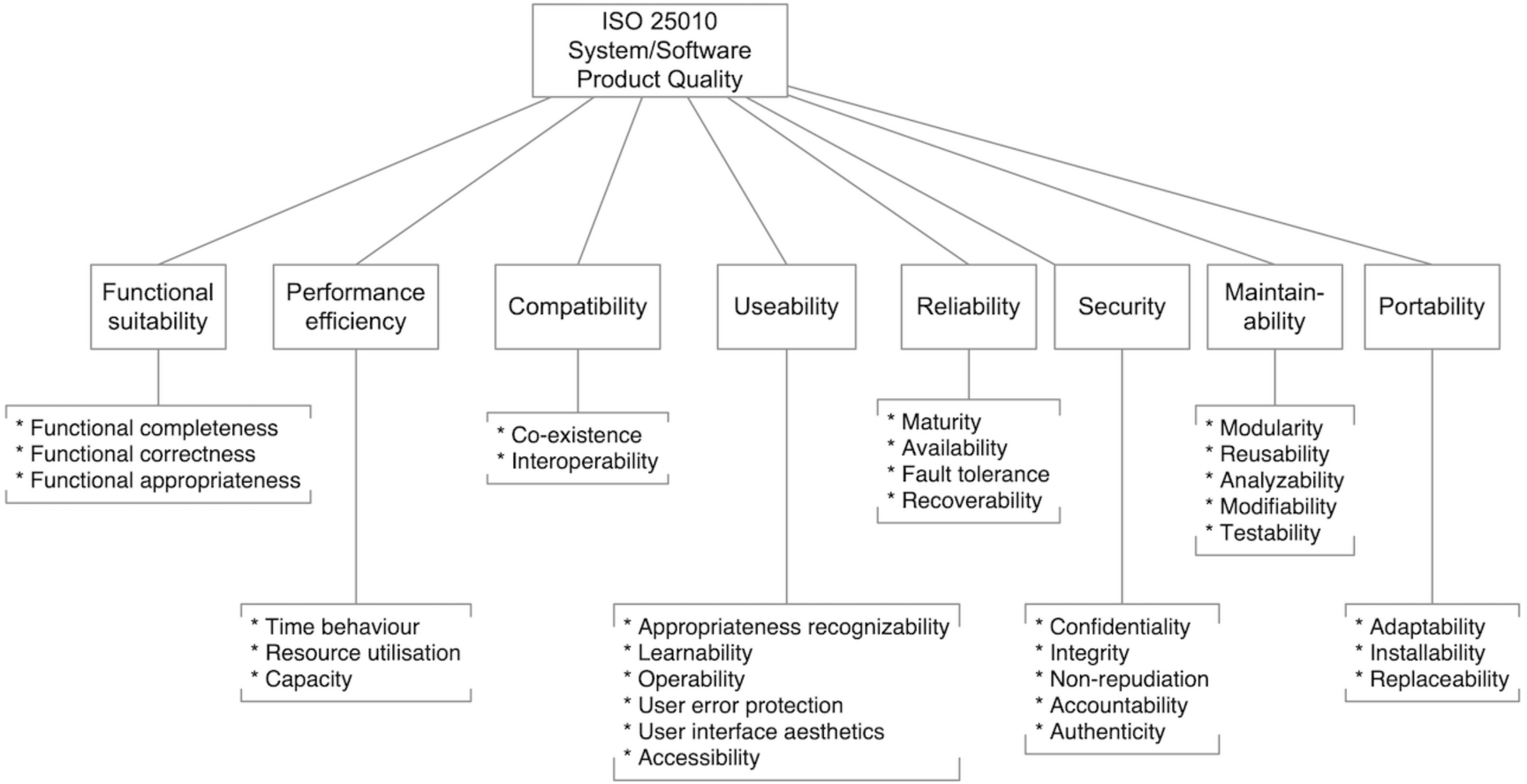
Funktionale Softwarearchitektur

Funktionale Softwarearchitektur

"Functional programming in the large"

Functional Software Architecture refers to methods of construction and structure of large and long-lived software projects that are implemented in functional languages and released to real users, typically in industry.

Warum Wartbarkeit?




Ohne Wartbarkeit ist alles doof.
Funktionalität: doof
Zuverlässigkeit: doof
Sicherheit doof
Compatibility: doof
Portability doof
Krabbelkäfer: doof
Performance doof
Zum Ausfüllen und Verschenken für deinen Lieblingsarchitekten
riva

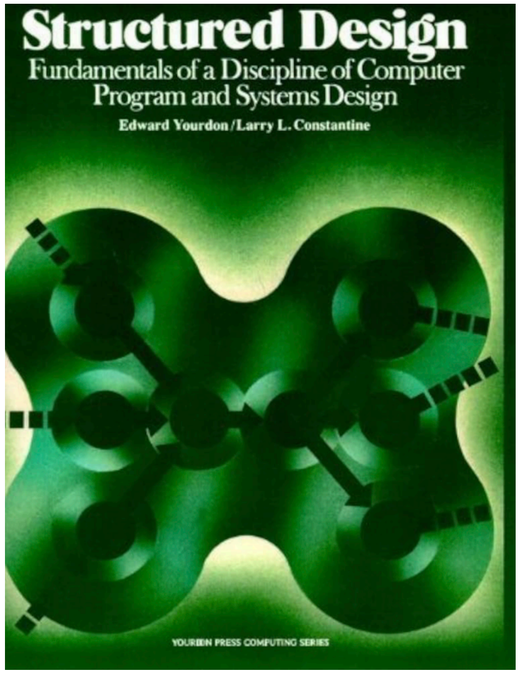
Decoupled (Coupling)

Dt. Kopplung

Kopplung ist das Maß der Abhängigkeiten zwischen zwei Modulen.

“Je mehr wir über Modul B wissen müssen, um Modul A zu verstehen, desto enger gebunden ist A an B.”

– Yourdon & Constantine, Structured Design

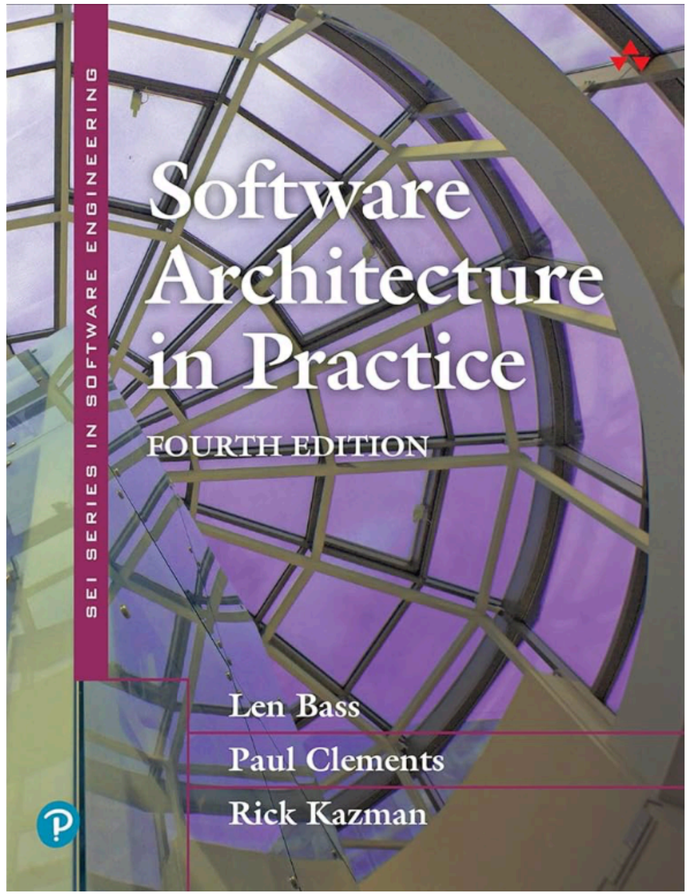


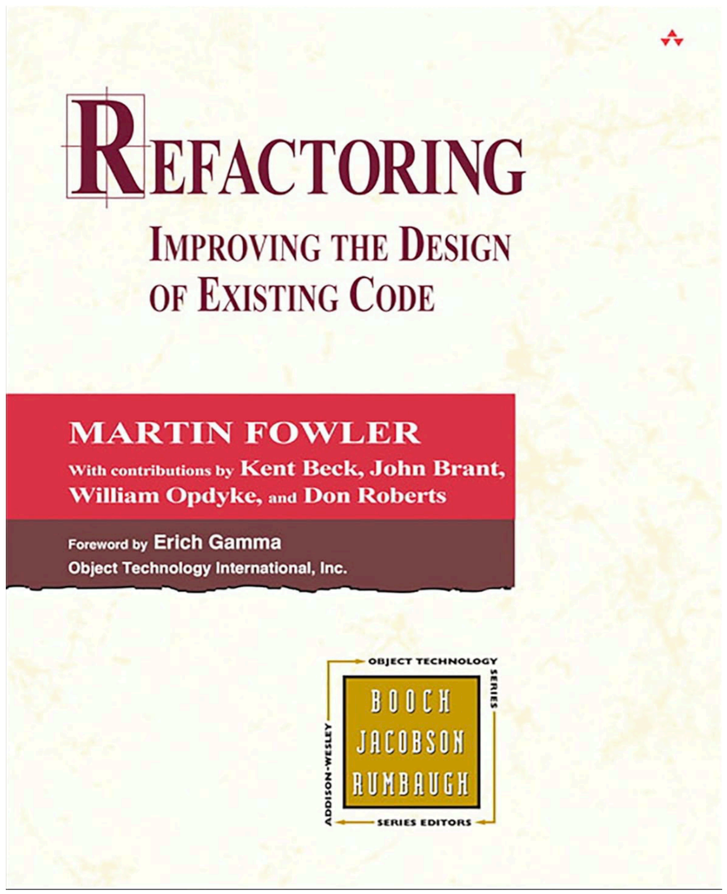
Decoupled (Coupling)

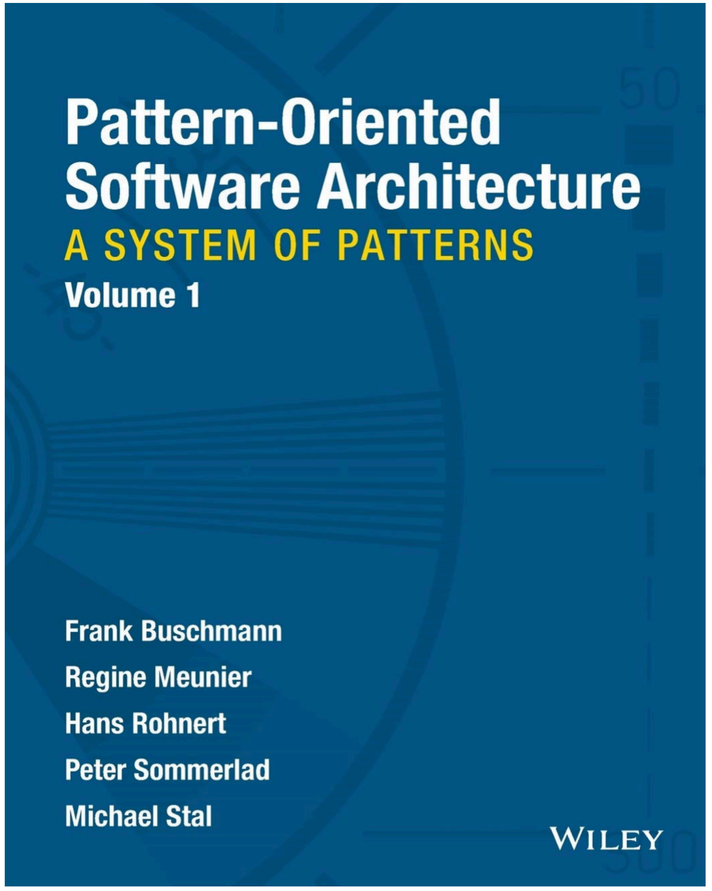
Dt. Kopplung

Kopplung ist damit das entscheidende Maß für die Wartbarkeit (Formbarkeit) eines Softwaresystems.

Decoupled **by Default**







Arten von Kopplung

- Common Coupling: geteilter Zustand
- External Coupling: Kopplung an die Plattform
- Content Coupling: Verletzung des Information-Hiding-Prinzips
- …

Common Coupling

Programmieren mit veränderlichem Zustand

```java
public class Person {
    private String name;
    private String address;
    ...
    public void setAddress(String newAddress) {
        this.address = newAddress;
    }
}

public class A {
    private Person myPerson;
    ...
    public void receivePerson(Person p) {
        myPerson = p;
    }
}

public class B {
    private Person myPerson;
    ...
    public void transferPerson(A other) {
        // Transfer to other department
        other.receivePerson(myPerson);
    }

    public void backToTheOffice() {
        myPerson.setAddress("Seattle");
    }
}
```

Imperativ: Data

Funktional: Data - Mutation

Imperativ: Data

Funktional: Data - Mutation

Während *Daten* in der Umgangssprache Gegebenheiten, Tatsachen oder Ereignisse sind, sind *Daten* in der Fachsprache Zeichen, die eine Information darstellen.

~~Imperativ: Data~~

~~Funktional: Data - Mutation~~

Während *Daten* in der Umgangssprache Gegebenheiten, Tatsachen oder Ereignisse sind, sind *Daten* in der Fachsprache Zeichen, die eine Information darstellen.

Imperativ: Data + Place

Funktional: Data

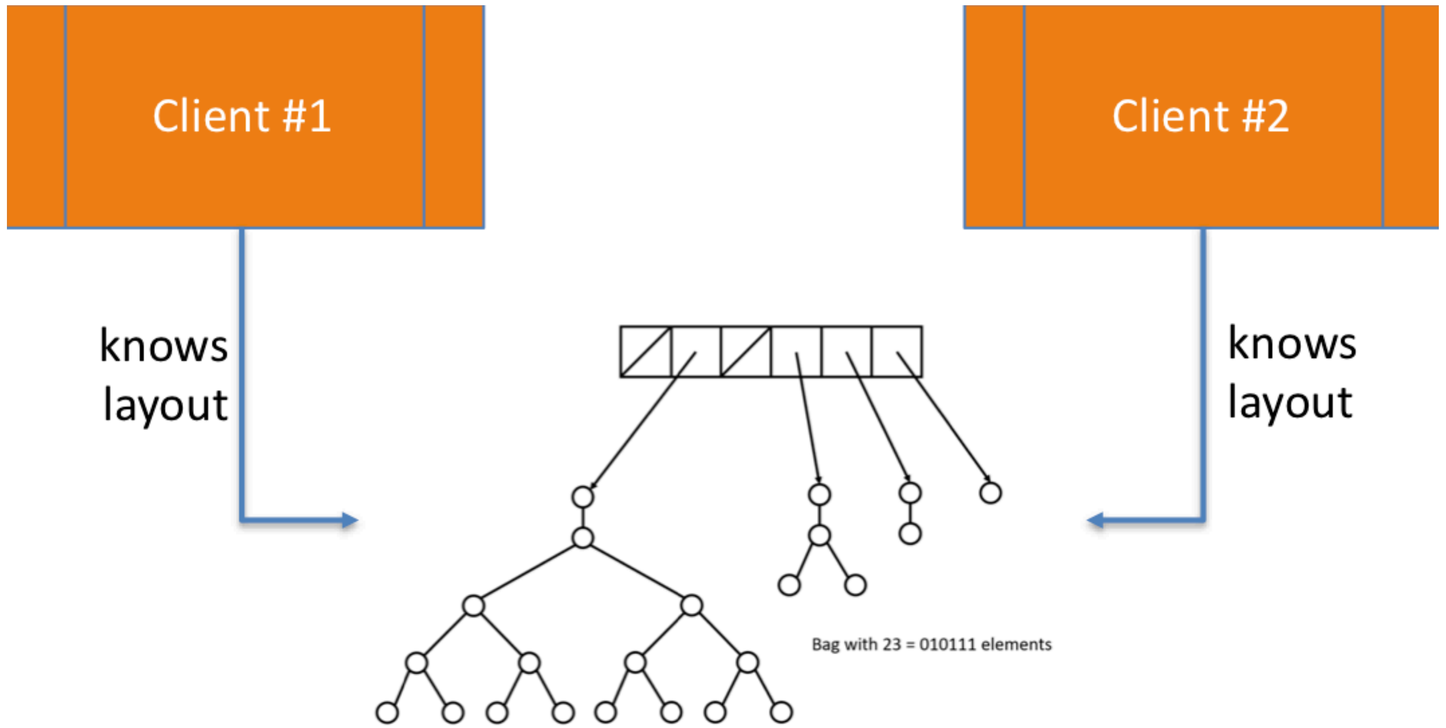
External Coupling

Umgang mit Effekten

- Übung für die Zuhörer

Content Coupling

Verletzung des Information-Hiding-Prinzips



Zeitreihen

```
object TimeSeriesServiceLegacy {
  def getTimeSeriesData(from: Time, to: Time): List[(Time, Double)] = ???
}
```

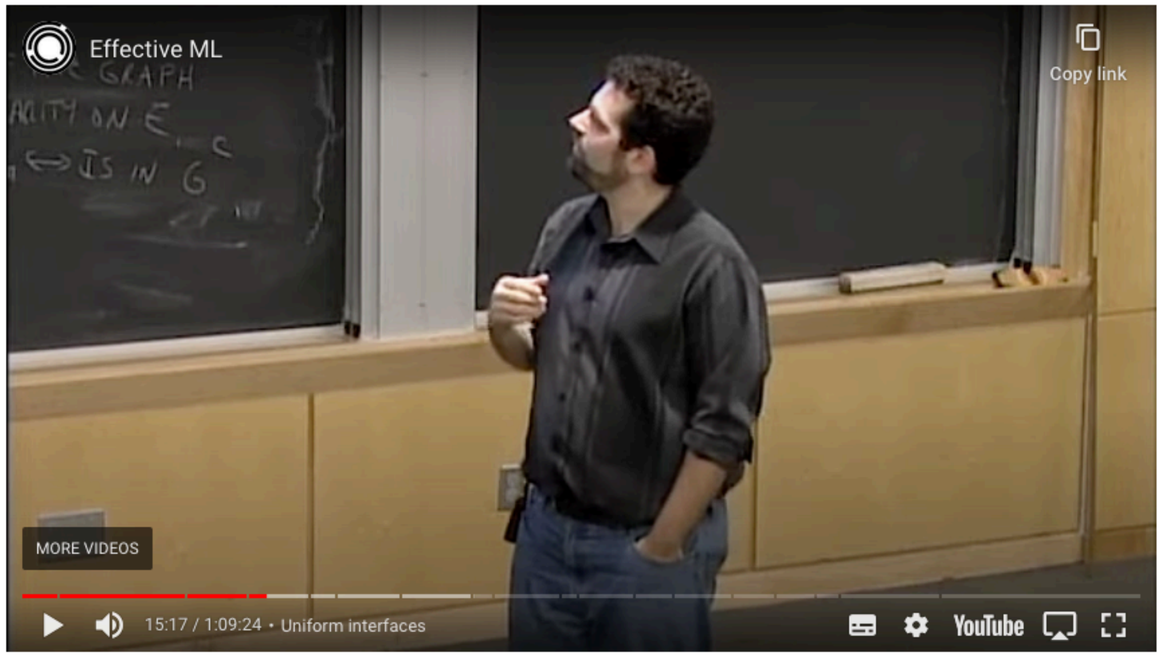
Make Illegal States Unrepresentable

Effective ML video

AUG 21, 2010 | 1 MIN READ

By: Yaron Minsky

A while back I mentioned that I'd given a guest lecture at classes at Harvard and Northeastern, and that the Harvard class had been taped. I finally got and uploaded that video:



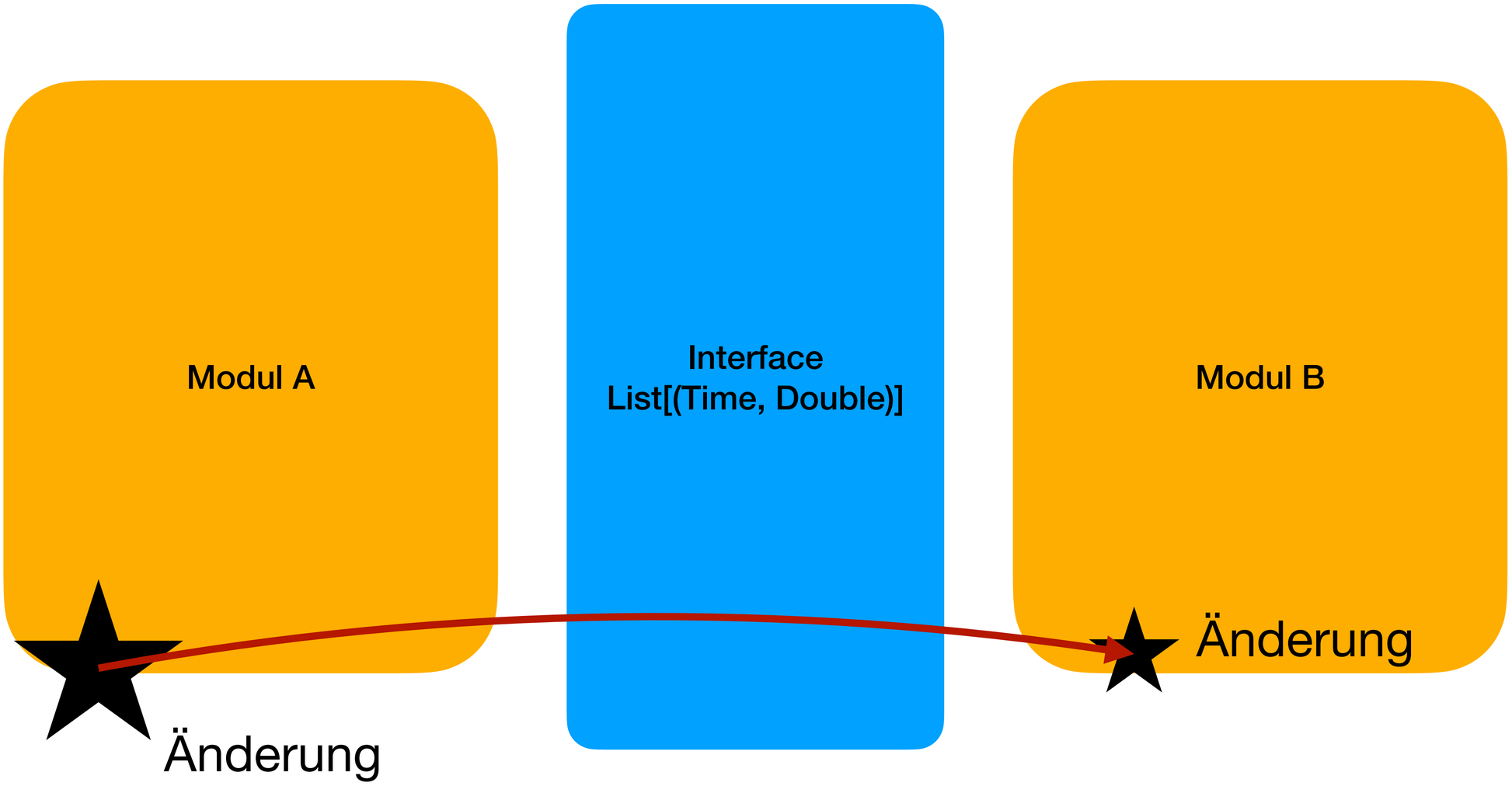
Make Illegal States Unrepresentable

Aber wie?

- Flache Produkttypen mit nullables zu Summentypen machen
- Smart Constructors
- Parsing mit nominell frischen Typen (“Parse, don’t validate”)
- Assoziative Maps nutzen
- Funktionen nutzen
- …

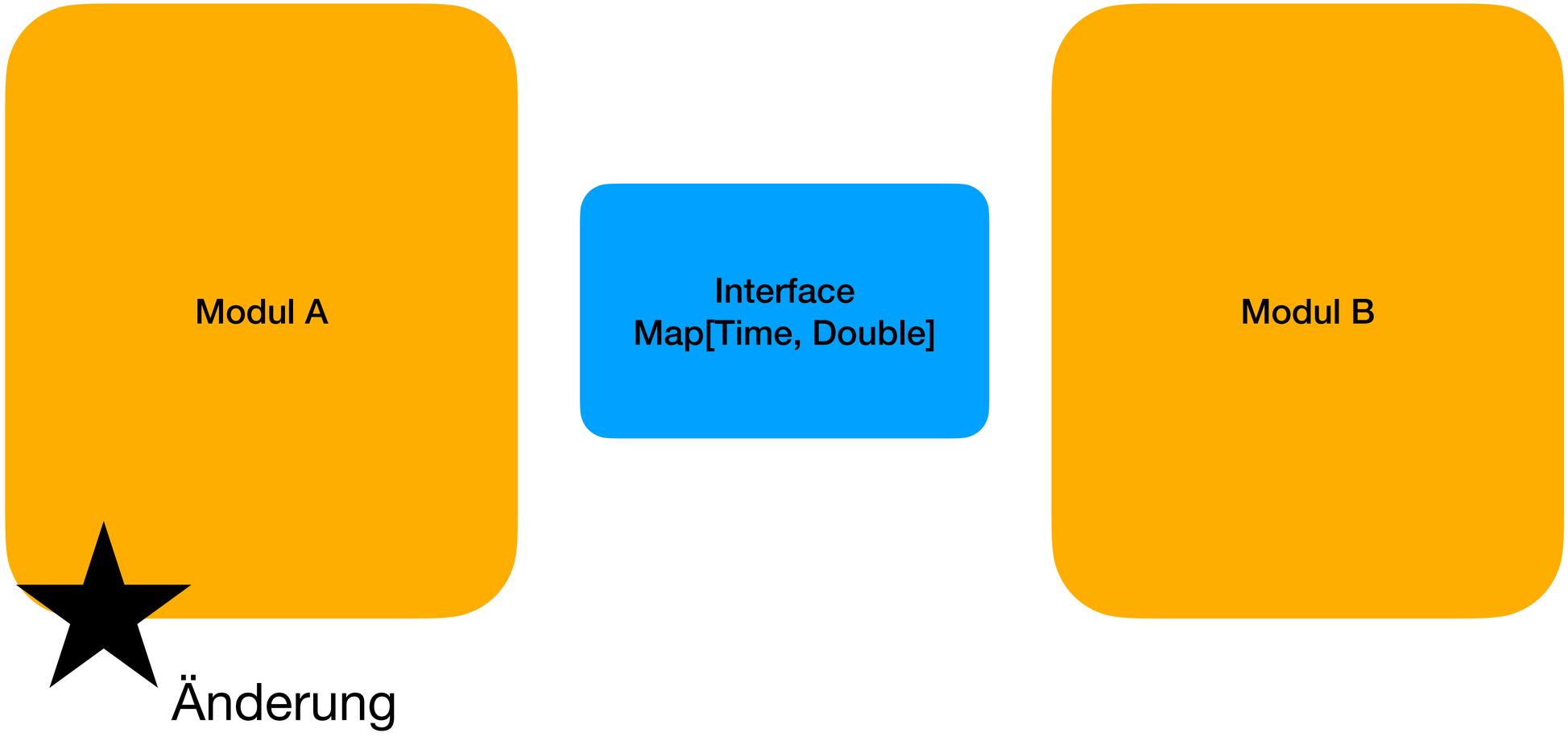
```scala
object TimeSeriesServiceAntiCorruption {
  private def parseList(lst: List[(Time, Double)], acc: Map[Time, Double]): Map[Time, Double] =
    lst match {
      case Nil => acc
      case (t, x) :: rest => parseList(rest, acc + (t -> x))
    }

  def getTimeSeriesData(from: Time, to: Time): Map[Time, Double] =
    parseList(TimeSeriesServiceLegacy.getTimeSeriesData(from, to), Map.empty)
}
```

Zu offenes Interface



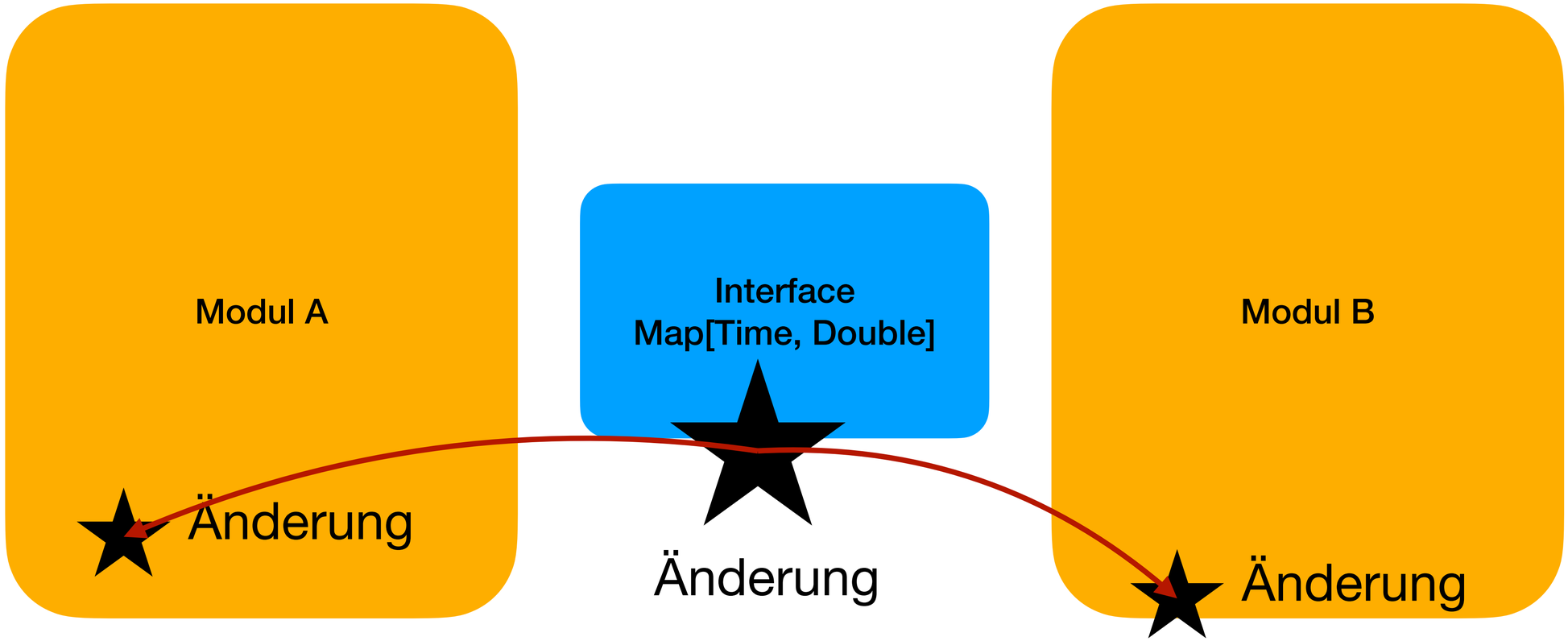
Zu beschränktes Interface

(Make Illegal States Unrepresentable)



Zu beschränktes Interface

(Make Illegal States Unrepresentable)



Make Legal States Representable

Denotational Design

Denotational design with type class morphisms (extended version)

Conal Elliott

LambdaPix

conal@conal.net

Abstract

Type classes provide a mechanism for varied implementations of standard interfaces. Many of these interfaces are founded in mathematical tradition and so have regularity not only of *types* but also of *properties* (laws) that must hold. Types and properties give strong guidance to the library implementor, while leaving freedom as well. Some of this remaining freedom is in *how* the implementation works, and some is in *what* it accomplishes.

To give additional guidance to the *what*, without impinging on the *how*, this paper proposes a principle of *type class morphisms* (TCMs), which further refines the compositional style of denotational semantics. The TCM idea is simply that *the instance's meaning follows the meaning's instance*. This principle determines the

22

little. It is form without *essence*. Users of this abstraction care about what *functionality* these names have. They care what the names *mean*.

Which leads to the question: what do we mean by "mean"? What is the *essence* of a type, enlivening its form? One useful answer is given by denotational semantics. The meaning of a program data type is a type of mathematical object (e.g., numbers, sets, functions, tuples). The meaning of each operation is defined as a function from the meanings of its arguments to the meaning of its result. For instance, the meaning the type $Map\ k\ v$ could be partial functions from k to v. In this model, *empty* is the completely undefined function, *insert* extends a partial function, and *lookup* is just function application, yielding $\perp$ where undefined. The meaning of the

Time Series = Map = Partial + Function ?

Time Series = ~~Map = Partial~~ + Function

```scala
sealed trait TS[A] extends Function1[Time, A]
```

Zusammenfassung

Decoupled by Default

- Wartbarkeit/Formbarkeit ist der Enabler für alle weiteren Softwarequalitäten
- Kopplung ist das wichtigste Maß für Wartbarkeit
- Funktionale Architekturen sind per Werkseinstellung wartbarer als klassische Architekturen
- Mit Denotational Design entstehen ganz beiläufig lose gekoppelte Systeme
- Mehr gibt’s im iSAQB-Modul FUNAR: https://active-group.de/schulung/