Was ist Softwarearchitektur?

Ein Beispiel und vier Architekturen

Burkhardt Renz

Fachbereich MNI
Technische Hochschule Mittelhessen

Wintersemester 2020/21

„Architektur ist die Kombination von
utilitas, *firmitas* und *venustas*.“

frei nach Vitruvius (Römischer Architekt 90-20 v. Chr.)

Softwarearchitektur will erreichen, dass das Anwendungssystem die Anforderungen erfüllt (utilitas), robust ist gegenüber Änderungen (firmitas) und eine gewisse Schönheit hat (venustas).

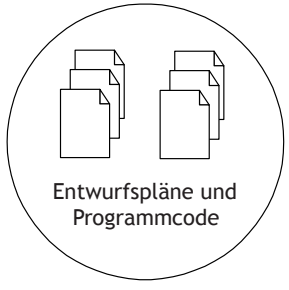
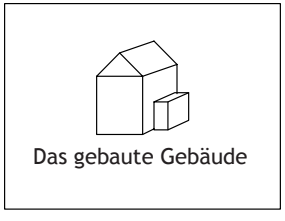
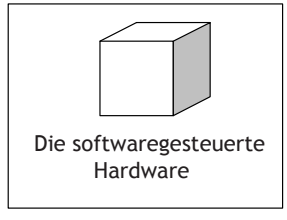
–

Fragt sich nur: wie?!

Übersicht

- Besonderheit von Softwaresystemen
- Ein Beispiel – vier Architekturen
- Diskussion der Beispiele
- Was ist Softwarearchitektur?

Eine (plakative) Gegenüberstellung

Gegenständliche Systeme	Softwarebestimmte Systeme
	
	
	

nach S. Wendt, HPI

Ergebnis der Gegenüberstellung

Gegenständliches System

- Grobstruktur mühelos sichtbar
- Feinstruktur nur mit großem Aufwand sichtbar zu machen

Softwaresystem

- Grobstruktur nur mit besonderer Anstrengung sichtbar zu machen
- Feinstruktur mühelos sichtbar

Was folgt daraus?

☞ Wir müssen die Struktur, den Aufbau von Software verständlich machen.
☞ Softwarearchitektur

Übersicht

- Besonderheit von Softwaresystemen
- Ein Beispiel – vier Architekturen
  - Spezifikation
  - Objektorientierte Organisation
  - Funktionaler Stil
  - Pipes & Filters
  - Ereignisgesteuertes System
- Diskussion der Beispiele
- Was ist Softwarearchitektur?

Ein Beispiel – vier Architekturen

Spezifikation

Spezifikation von Keywords in Context (KWIC)

The KWIC (Key Word in Context) index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circulary shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

— *David Parnas*

Beispiel für KWIC

Input

```
Software Architecture in Practice
Bringing Design to Software
Problem Frames
```

Output

```
Architecture in Practice Software
Bringing Design to Software
Design to Software Bringing
Frames Problem
in Practice Software Architecture
Practice Software Architecture in
Problem Frames
Software Architecture in Practice
Software Bringing Design to
to Software Bringing Design
```

Diskussion

Wie kann man ein Programm strukturieren, das diese Spezifikation erfüllt?

Offensichtlich muss man

- Die Daten lesen
- Jede Zeile shiften um die Ergebniszeilen zu erhalten
- Alle Ergebniszeilen alphabetisch sortieren
- Das Ergebnis ausgeben

Ad-hoc-Lösung

Demo

- kwicah

Struktur der Ad-hoc-Lösung

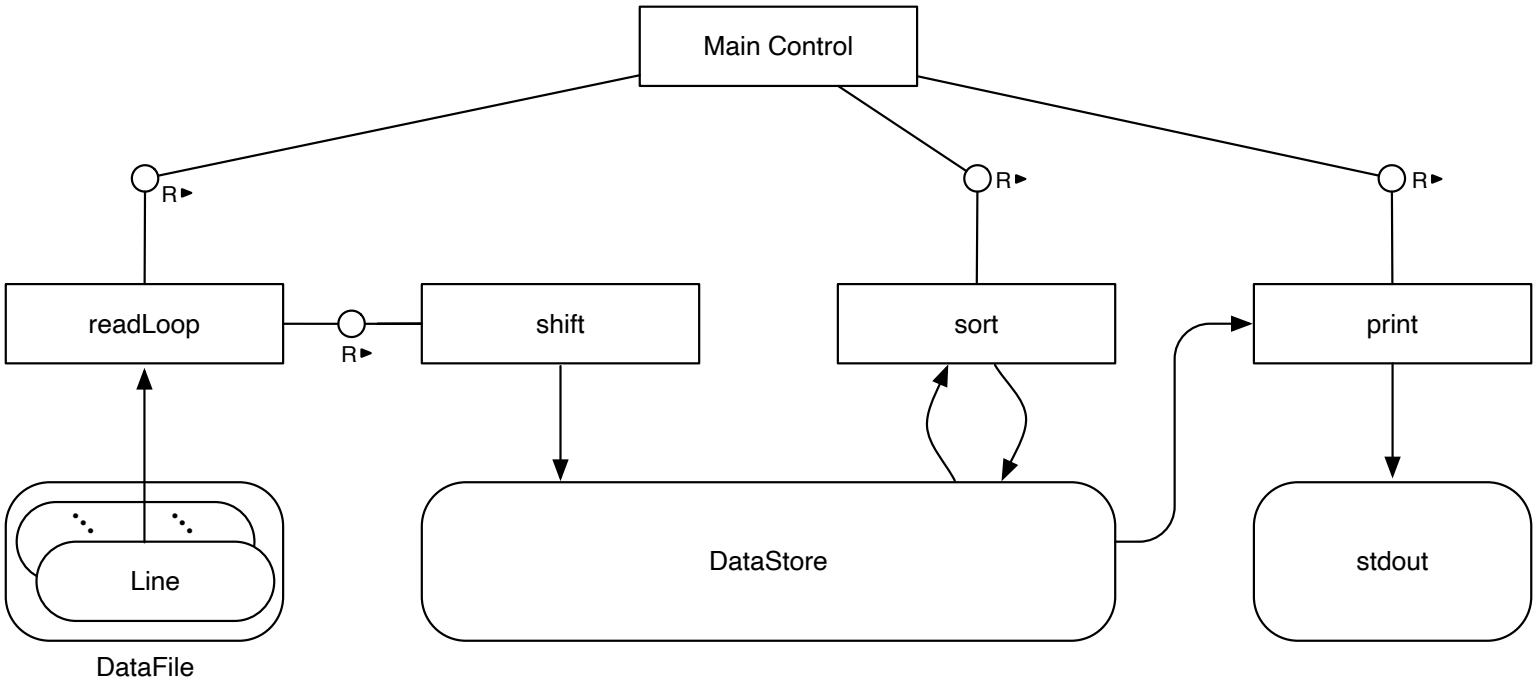


Abbildung: Ad-hoc-Lösung für kwic

Ein Beispiel – vier Architekturen

Objektorientierte Organisation

Objektorientierte Organisation

Konzepte

- Kapselung von Daten
- Methoden zur Manipulation der Daten

Demo

- kwic00

Objektorientierte Organisation - Struktur

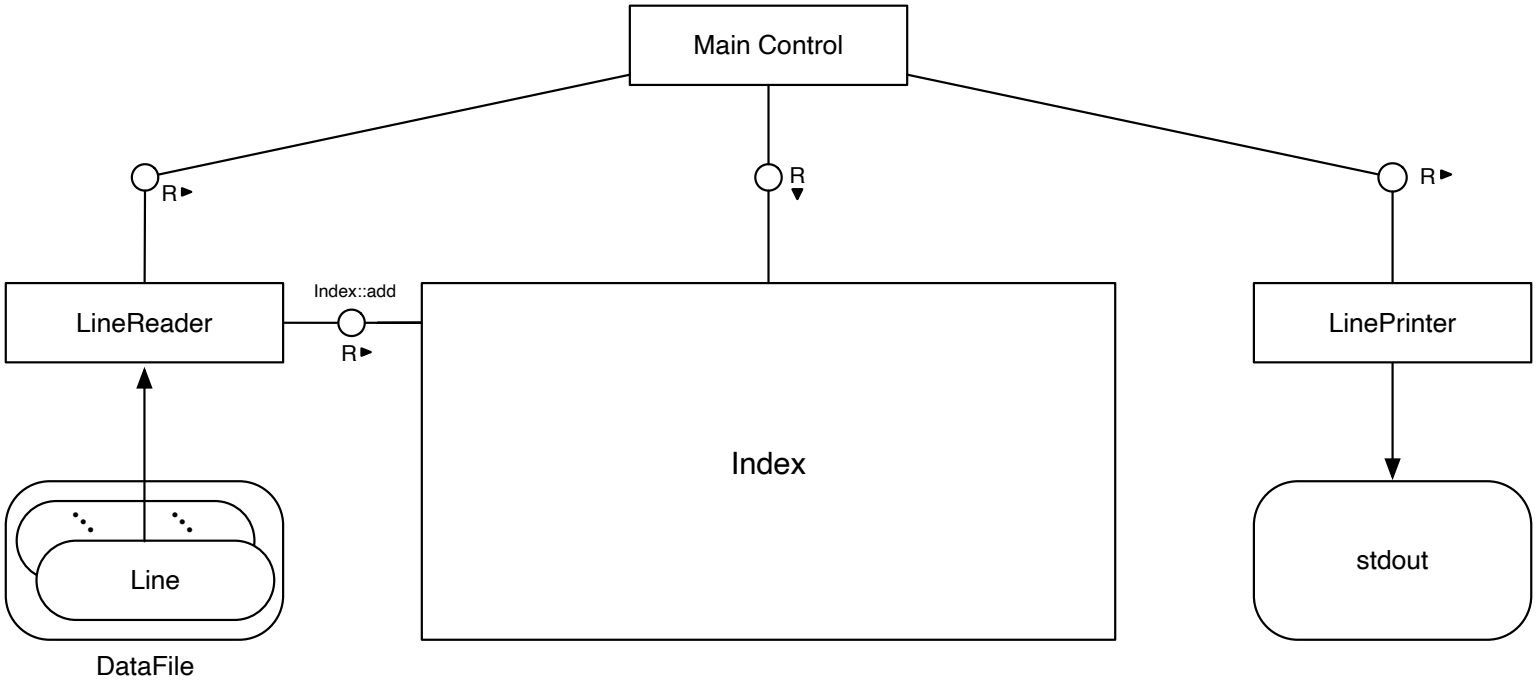


Abbildung: Index kapselt die Daten

Interne Struktur von Index

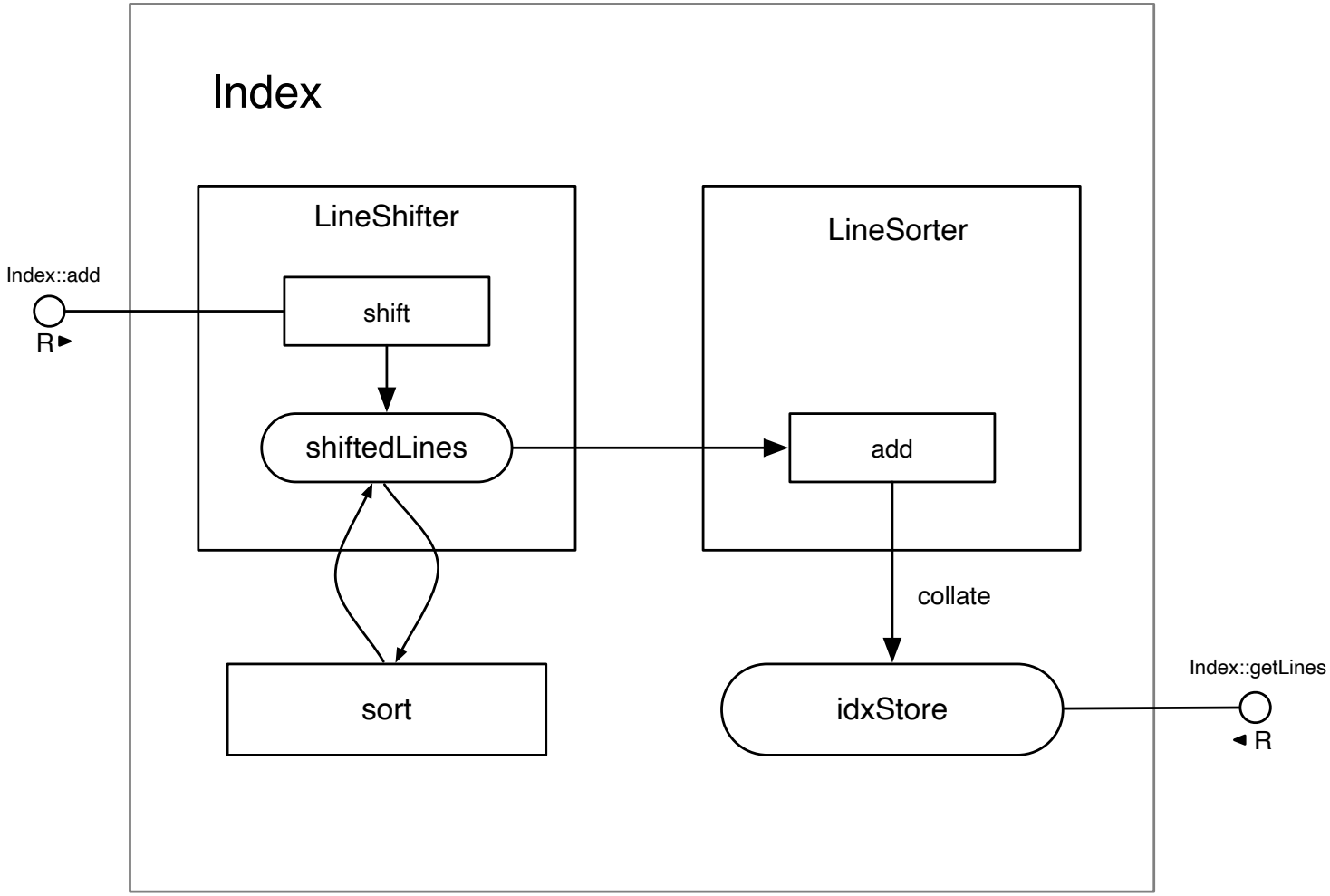


Abbildung: Die interne Struktur von Index

Objektorientierte Organisation

Charakteristik

- Kontrollfluss durch zentrale Koordination
- Prozedurale Abstraktion durch Kapselung der Zugriffsmethoden
- Datenkapselung
- Algorithmen und Datenrepräsentation der einzelnen Klassen können geändert werden, ohne die anderen zu beeinflussen.
- Änderbarkeit durch Interface/Implementierung
- Wiederverwendbarkeit durch „Dependency Injection“

Ein Beispiel – vier Architekturen

Funktionaler Stil

Funktionaler Stil

Konzept

- Reine Funktionen (ohne Seiteneffekte) verändern Input zu Output
- Solche Funktionen werden zusammengeschaltet

Demo

- kwicfn0 in der rein funktionalen Sprache Clojure

Funktionaler Stil - Struktur

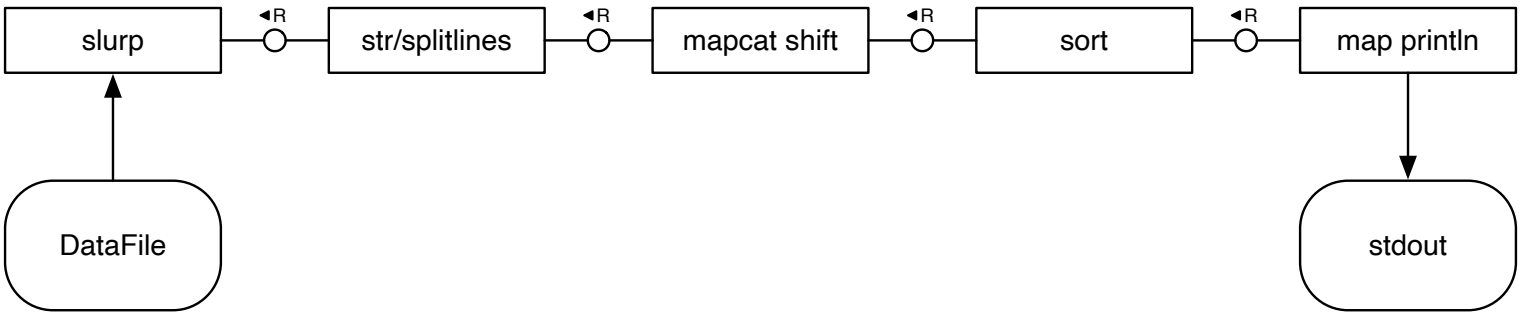


Abbildung: kwic in Clojure

Funktionaler Stil in Java

Demo

- kwicfn1 mit Java Streams
- kwicfn2 mit Observables/Subscriber in RxJava

Funktionaler Stil

Charakteristik

- Kontrollfluss durch Komposition von Funktionen
- Interne Verarbeitung durch reine Funktionen, Seiteneffekte minimiert
- Prozedurale Abstraktion durch Funktionen
- Datenfluss folgt Kontrollfluss (oder umgekehrt)
- Algorithmen der einzelnen Funktionen können geändert werden, ohne die anderen zu beeinflussen.
- Sehr unterschiedliches internes Verhalten je nach Plattform

Ein Beispiel – vier Architekturen

Pipes & Filters

Pipes & Filters

Konzept

- Filter, die Input zu Output verändern
- Mechanismus durch Kanäle (Queues/Pipes) die Filter hintereinander zu schalten

Demo

- Filter als Prozesse – kwicpf1
- Filter als Threads – kwicpf2

Pipes & Filters – Struktur

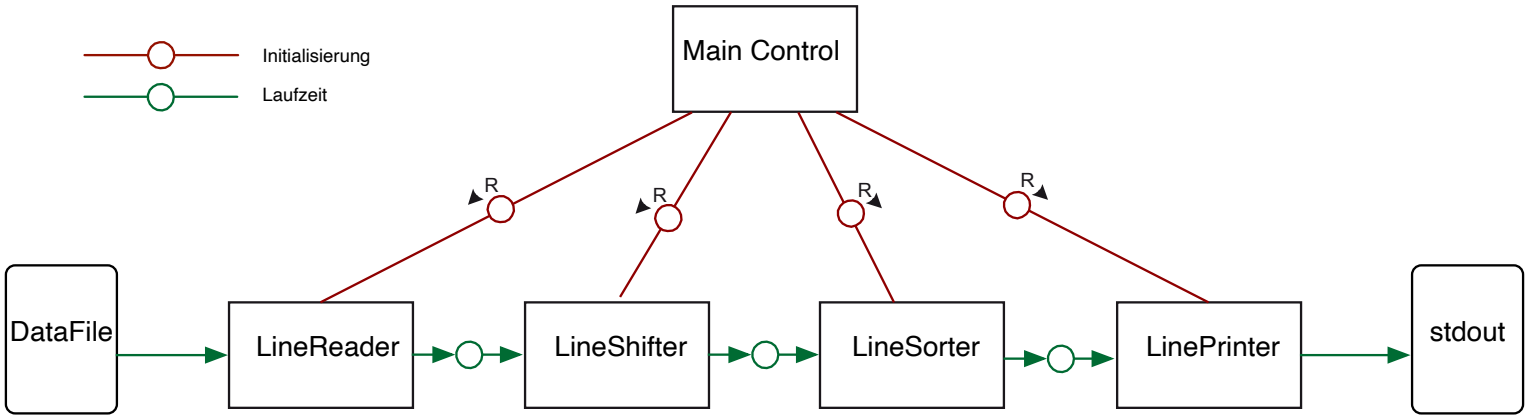


Abbildung: Pipes & Filters

Charakteristik

- Die Steuerung ist verteilt: jeder Filter arbeitet, sobald Daten an seiner Eingabe anliegen.
- Die Filter sind voneinander unabhängig, sie kommunizieren nur über die Pipes zwischen ihnen.
- Datenfluss und Kontrollfluss fallen zusammen

Ein Beispiel – vier Architekturen

Ereignisgesteuertes System

Ereignisgesteuertes System mit indirektem Aufruf

Konzept

- Voneinander unabhängige Akteure, die sich nicht kennen
- Akteure reagieren auf Ereignisse/Nachrichten
- Infrastruktur zur Verteilung der Nachrichten

Demo

- kwiceb

Ereignisgesteuertes System - Struktur

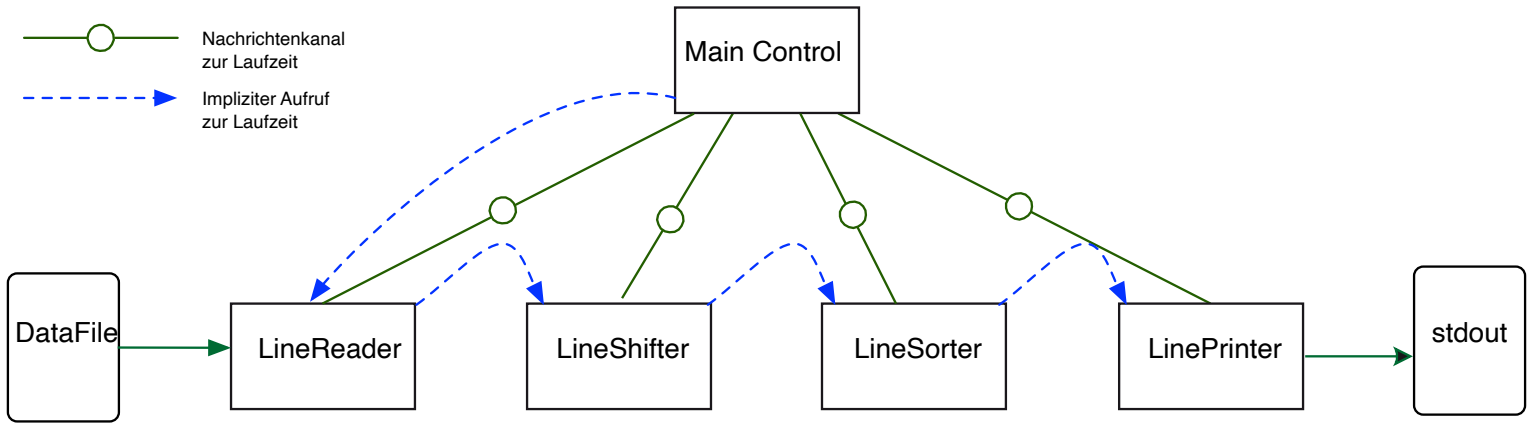


Abbildung: Ereignisgesteuertes System mit indirekten Aufrufen

Charakteristik

- Keine zentrale Steuerung.
- Ereignisse führen zu Aktionen, die ihrerseits Ereignisse erzeugen.
- Veränderung der Daten löst Ereignisse aus.
- Protokoll erforderlich
- Modell „aktiver Daten“

Übersicht

- Besonderheit von Softwaresystemen
- Ein Beispiel – vier Architekturen
- Diskussion der Beispiele
  - Architektur und Codestruktur
  - Architektur und Qualitätsmerkmale
  - Fazit
- Was ist Softwarearchitektur?

Diskussion der Beispiele

Architektur und Codestruktur

Laufzeitstruktur von KWIC in Variante 3

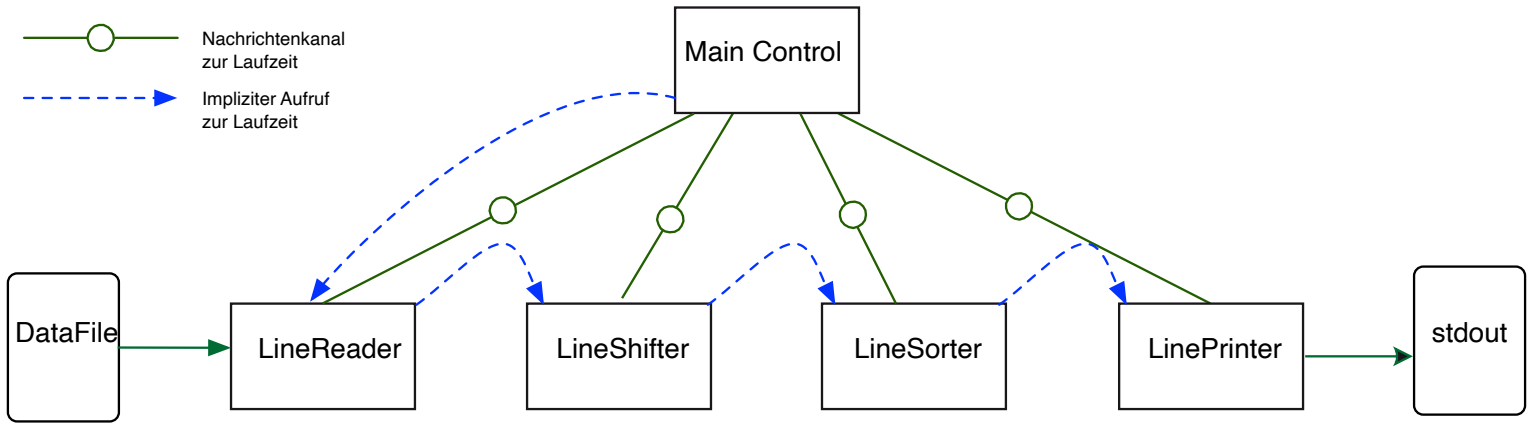


Abbildung: Ereignisgesteuertes System mit indirekten Aufrufen

Codestruktur von KWIC in Variante 3

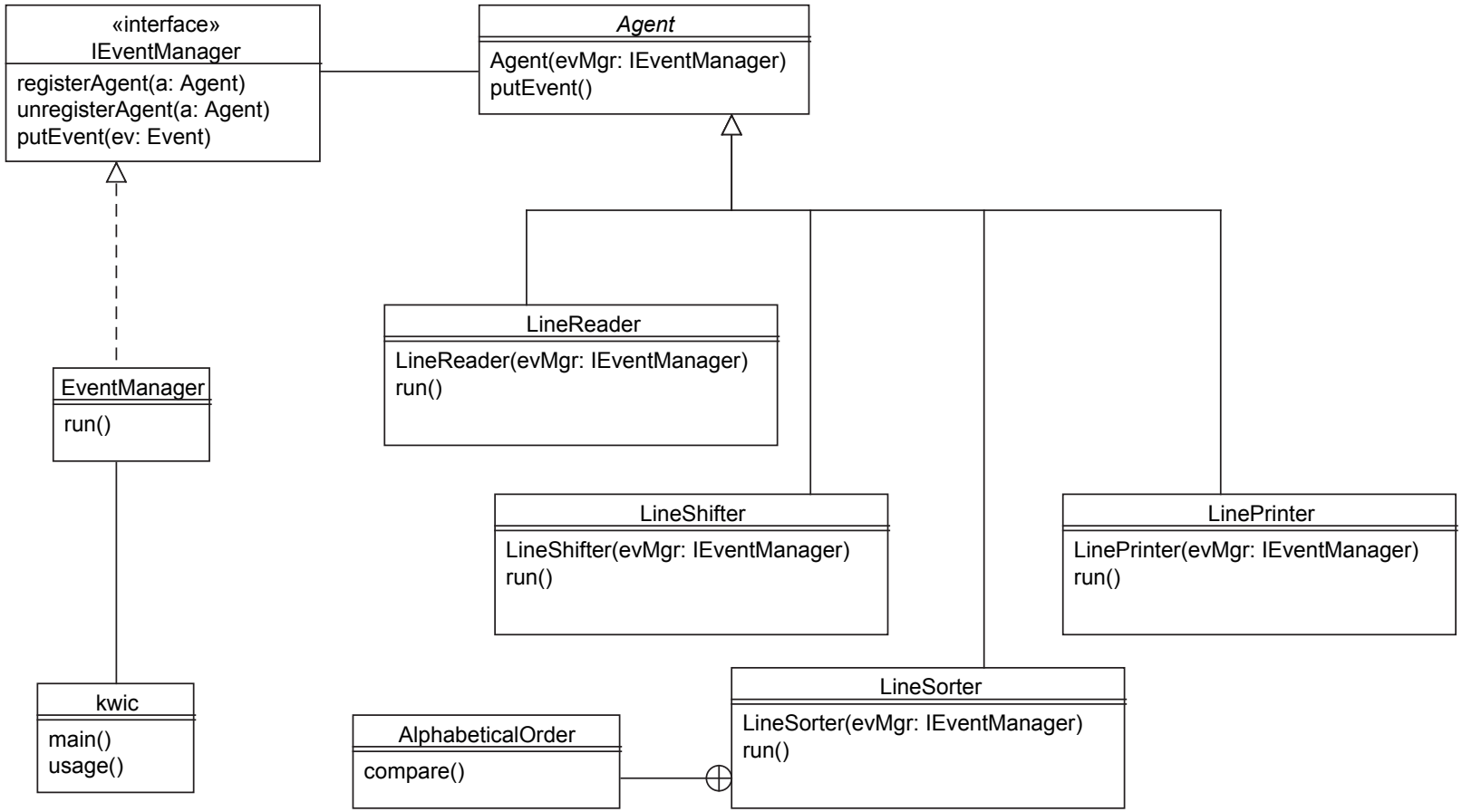


Abbildung: Codestruktur des ereignisgesteuerten Systems

Architektur und Codestruktur

Resultat

In der Codestruktur ist der konzeptionelle Grundgedanke der Architektur des ereignisgesteuerten Systems nur schwer erkennbar.

Diskussion der Beispiele

Architektur und Qualitätsmerkmale

Einige Qualitätsmerkmale - beispielhaft

1. Erweiterbarkeit
   Nicht-Berücksichtigung bestimmter Wörter (wie „der, die, das“) für den Index.
2. Leistungsfähigkeit 1
   Sehr große Datenmengen
3. Leistungsfähigkeit 2
   Geschwindigkeit
4. Testbarkeit
   Wie leicht ist das Programm zu überprüfen?
   Wie leicht sind fehlerhafte Stellen zu finden?

Erweiterbarkeit

- Objektorientierte Organisation:
  Neue Variante von LineShifter implementieren und injizieren
  oder Interne Struktur von Index erweitern
- Funktionaler Stil:
  Weitere Funktion, die mit Stopworten beginnende Zeilen
  herausfiltert
- Pipes & Filters:
  Zusätzlicher Filter
- Ereignisgesteuertes System:
  LineShifter durch erweiterten Akteur ersetzen oder
  Zusätzlichen Akteur in das Protokoll einbauen

Leistungsfähigkeit 1 (Datenmenge)

- Objektorientierte Organisation:
  Eingabedatei kann zeilenweise verarbeitet werden, aber: Sortierung!
- Funktionaler Stil:
  Lazy Sequences/Streams und Observables sehr gut für die Verarbeitung der Eingabedatei geeignet, aber: Sortierung!
- Pipes & Filters:
  LineReader kann zeilenweise lesen, LineShifter kann zeilenweise shiften, aber: Sortierung!
- Ereignisgesteuertes System:
  Auch hier: Sortierung!

Externes Sortieren

- Alle Varianten benötigen bei Ergebnisdaten, die die Größe des Hauptspeichers überschreiten, Methoden des externen Sortierens.
- Möglichkeit 1: Verwendung eines Datenbankmanagementsystems, etwa H2 oder SQLite
- Möglichkeit 2: Eigene Implementierung eines MergeSort-Algorithmus

Leistungsfähigkeit 2 (Geschwindigkeit)

- Objektorientierte Organisation:
  Gut, weil kaum Duplikation von Daten und weil MergeSort verwendet wird
- Funktionaler Stil:
  Reine Funktionen haben Schnittstellen zur Folge, die Objekte duplizieren
- Pipes & Filters:
  Da alle Filter dasselbe Datenformat erwarten, müssen Daten oft transformiert werden
- Ereignisgesteuertes System:
  Overhead durch Nachrichtenverkehr

Leistungsfähigkeit 2 (Parallelisierbarkeit?)

- Wenn die Eingabe parallel verarbeitet wird, benötigen alle Varianten für das Erstellen des Ergebnisses einen Mechanismus, der die sortierten Resultate der parallel arbeitenden Prozesse zusammenführt.
- Wie aufwändig ist dies in dem jeweiligen Stil?
- Würde sich für die Fragestellung *mapreduce*[1] anbieten?

[1] Jeffrey Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters* `http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf` - Eine Architektur geboren aus der Hardwareinfrastruktur von Datacenters mit Tausenden Linux-Boxen

Leistungsfähigkeit 2 (Parallelisierung)

- Objektorientierte Organisation:
  Objektorientierung bietet von Hause aus keine naheliegende Lösung dafür
- Funktionaler Stil:
  Viele Funktionen (wie z.B. *map*) kann man leicht parallel auf mehrere Threads laufen lassen, das Sortieren erfordert jedoch den join der Threads - schwieriger
- Pipes & Filters:
  Gut geeignet, man braucht Filter, die die Ergebnisse verschiedener Prozesse zusammenfassen
- Ereignisgesteuertes System:
  Wird gerne auch als verteiltes System betrieben (MessageBroker) und kann dadurch gut für parallele Verarbeitung verwendet werden, zusätzlicher Akteur, der Resultate zusammenführt

Testbarkeit

- Objektorientierte Organisation:
  Hängt von der Komplexität der Klassen ab. Zustandsbehaftete Objekte oft schwer testbar.
- Funktionaler Stil:
  Reine Funktionen gut testbar. Komposition leicht überblickbar. Lazy Streams können täuschen.
- Pipes & Filters:
  Filter gut testbar. Kontrollfluss = Datenfluss, d.h. leicht überblickbar
- Ereignisgesteuertes System:
  Einzelne Akteure leicht testbar. Kontrollfluss schwer überprüfbar.

Diskussion der Beispiele

Fazit

Fazit

1. Architektur $\perp$ Funktionalität
2. Architektur $\neq$ Codestruktur
3. Architektur $\Rightarrow$ Qualitätsmerkmale

Übersicht

- Besonderheit von Softwaresystemen
- Ein Beispiel – vier Architekturen
- Diskussion der Beispiele
- Was ist Softwarearchitektur?
  - Definition
  - Wozu Softwarearchitektur?
  - Literatur & Links

Was ist Softwarearchitektur?

Definition

Was ist Softwarearchitektur? – Definition

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

— *Len Bass et al.*

Elemente der Definition

Elemente der Softwarearchitektur

- Elemente, *Komponenten* – definieren den Ort von Berechnungen oder auch Speicher, z.B. Filter, Datenbanken, Objekte, Server, Klienten usw.
- Beziehungen, *Konnektoren* – definieren die Interaktion der Komponenten, z.B. Funktionsaufruf, Pipe, Event usw.
- Eigenschaften – definieren Vorgaben und Beschränkungen für Komponenten, z.B. Schnittstellen, Qualitätsmerkmale usw.

Diskussion

- Abstraktion: *Wesentliche* Elemente und ihre Beziehungen
- Struktur: Zerlegung und Zusammenbau
- Struktur*en*: Verschiedene Sichten auf das Wesentliche des Systems
- Beziehungen: Organisierende Prinzipien des Zusammenwirken der Elemente
- Plan: *Bauplan* im Unterschied zu Dokumentation und Projektplan

Was ist Softwarearchitektur? – Kritik der Definition

The architecture of a software system defines that system in terms of components and of interactions among those components. In addition to specifying the structure and topology of the system, the architecture shows the intended correspondence between the system requirements and elements of the constructed system.

— *Mary Shaw et al.*

What is surprising about the Shaw et al. model [in diesem Geiste ist auch die Definition von Len Bass] is that, rather than defining the software's architecture as existing within the software, it is defining a description of the software's architecture as if that were the architecture

— *Roy Thomas Fielding*

Welche Strukturen?

Software ist die Beschreibung des gewollten Verhaltens einer universellen, abstrakten Maschine (Computer).

Beschreibung – Beschriebenes – Umgebung

1. Struktur der Beschreibung: Codestruktur
2. Struktur des Beschriebenen: konzeptionelle Struktur des (laufenden) Systems
3. Struktur der Umgebung: Infrastruktur

Was ist Softwarearchitektur?

Wozu Softwarearchitektur?

Architektonische Themen – Auswahl

- Regeln für „Querschnittsthemen“
  z.B. Fehlerbehandlung, Verteilung...
- Richtlinien für Bauplan
- Fachliche Architektur
  z.B. Strukturen des Anwendungsgebiets
- Qualitätsanforderungen und Strategien, sie zu erreichen
  z.B. Erweiterbarkeit, Anpassbarkeit des Systems
- Infrastruktur, eingesetzte Technik
  z.B. Frameworks, Sprachen, Datenbanken...

Die Abgrenzung der architekturell relevanten Strukturen ist die zentrale Aufgabe des Architekten und kann nicht nach formalen Kriterien vollzogen werden

— *Peter Tabeling*

Aufgaben und Ziele

- Mittel der Kommunikation und Diskussion aller Interessengruppen (*Stakeholder*)
- Festlegung der ersten und grundlegenden Entscheidungen für Design und Implementierung (*Constraints*)
- Grundlage für das Erreichen der funktionalen und qualitativen Merkmale des zu konstruierenden Systems
- Architektur fungiert als „mentaler Prototyp“
- Architektur(-stile, -muster) sind wiederverwendbar, auf andere Systeme übertragbar

Erwartungen an eine Softwarearchitektur

- Klare Entwurfsabsichten, damit die intendierte Architektur erhalten und verständlich bleibt
- Designzentren festlegen
- Basis für Design, damit die wesentlichen Fragen gestellt werden
- Verbesserung der Änderbarkeit, indem die Designzentren erkennbar bleiben und erhalten werden

Grundlegende Entscheidungen in der Architektur

Funktionalität Fähigkeit des Systems, die intendierte Aufgabe zu erfüllen (kann durch verschiedene Architekturen erreicht werden)

Qualitätsmerkmale Insbesondere nicht-funktionale Merkmale, die in hohem Maße von der Architektur beeinflusst werden

Organisation Architektur bestimmt auch die Organisation des Entwicklungsprozesses

Eckpfeiler Architektur legt den Rahmen fest, in dem sich Design und Implementierung bewegen (z.B. Framework)

Zusammenfassung: Aufgaben von Softwarearchitektur

1. Verstehbare Darstellung der zentralen Entwurfsentscheidungen
2. Wiederverwendbarkeit von Komponenten eines Systems
3. Grundlegender Bauplan für die Konstruktion eines Softwaresystems
4. Basis für kontrolllierte Evolution eines Systems oder einer Produktlinie
5. Grundlage für die Analyse eines (evtl noch gar nicht gebauten) Softwaresystems – z.B. in Bezug auf Qualitätsmerkmale
6. Mittel des Managements der Entwicklung und Weiterentwicklung

Fazit: Was ist Softwarearchitektur?

To be architectural is to be the most abstract depiction of the system that enables reasoning about critical requirements and constrains all subsequent refinements.

— *Mark Klein*

A mental prototype is a model of the system which outlines a potential system solution. It describes functionality, but leaves open most of the implementation details.

— *Andreas Knöpfel, Bernhard Gröne, Peter Tabeling*

Was ist Softwarearchitektur?

Literatur & Links

Literatur


Mary Shaw, David Garlan
Software Architecture: Perspectives on an Emerging Discipline
Upper Saddle River, NJ: Prentice-Hall, 1996.


Len Bass, Paul Clements, Rick Kazman
Software Architecture in Practice
Boston: Addison-Wesley, Third Edition 2012.


Jan Bosch
Design and Use of Software Architectures
Harlow, UK: Addison-Wesley, 2000.


Christine Hofmeister, Robert Nord, Dili Soni
Applied Software Architecture
Reading, MA: Addison-Wesley, 2000.

Literatur


Ian Gorton
Essential Software Architecture
Berlin: Springer, Second Edition 2011.


Peter Tabeling
Softwaresysteme und ihre Modellierung
Berlin: Springer, 2006.


Richard N. Taylor, Nenad Medvidović, Eric M. Dashofy
Software Architecture: Foundations, Theory, and Practice
John Wiley & Sons, 2010.


George Fairbanks
Just Enough Software Architecture: A Risk-Driven Approach
Boulder CO: Marshall & Brainerd, 2010.

Links

- Software Engineering Institute
  Software Architecture
  http://www.sei.cmu.edu/architecture/
- Michael Stal, Stefan Tilkov, Markus Völter, Christian Weyer
  SoftwareArchitekTOUR - Podcast für den professionellen Softwarearchitekten
  http://www.heise.de/developer/podcast/