Software-Technologie: Stand der Kunst und Herausforderungen

O.Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Pree

Software & Systems Research Group
Universität Salzburg
cs.uni-salzburg.at

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

UNIVERSITÄT SALZBURG

Kontext

- Das Phänomen Software
- Wie kann Software ingenieurmäßig entwickelt werden?
- Softwaretechnik – Quo vadis?

© W. Pree

Das Phänomen Software

© W. Pree

Die Universalmaschine Computer macht Software allgegenwärtig

Flugzeug-/Raketensteuerungen

ca. 70 Prozessoren im Auto

© W. Pree

Qualität ist wesentlich schlechter als bei anderen Produkten

- **Softwarefehler/-mängel** mit drastischen Auswirkungen:
  - Y2K, €-Umstellung
  - fehlerhafte Finanztransaktionen
  - Abstürze (zB Ariane: $ 800 Mio.)
  - . . .

© W. Pree

UNIVERSITÄT SALZBURG

Was ist besonders an Software?

© W. Pree

Die Probleme bei der Herstellung von Software resultieren aus der Komplexität der zu realisierenden Produkte

- **Spezifikation der Anforderungen** ← Prototyping
- **Beherrschung der Komplexität** ← Programmiermodelle
- **Wiederverwendung/PlugIns, Änderbarkeit und Erweiterbarkeit** ← Entwurfsmuster, Frameworks
- **Automatisierung im Herstellungsprozeß**
- **Portabilität**
- Dokumentation
- **Produktergonomie (Mensch-Computer-Schnittstelle)** ← Psychologie (zB Piaget)
- Projektorganisation u. -kontrolle
- Qualitätssicherung und -bewertung
- Personenunabhängigkeit
- Kostenabschätzung

© W. Pree

Beispiel:
Problem der exakten Spezifikation

© W. Pree

Eine exakte Spezifikation ist oft unpraktikabel

geg.: $n \geq 3$,

$L: N_n \rightarrow N$

ges.: Ein Programm P, sodass

$a: N_3 \xrightarrow{inj} N_n$, sodass

$1 \leq i \leq 3$ $\quad$ $j \in N_n \setminus \cup \{ a_k \}$ $\quad$ $L(a_i) \geq L(a_J)$

$1 \leq k \leq j$

© W. Pree

UNIVERSITÄT SALZBURG

... im Vergleich zur nicht exakten verbalen Spezifikation

Gegeben ist eine Liste mit mindestens drei positiven Zahlen.

Gesucht ist ein Programm P, das die Indizes der drei größten Elemente der Liste liefert.

© W. Pree

Meisterung der Komplexität

© W. Pree

Bei klassischen Ingenieurdisziplinen gilt:

- Schlechte Qualität läßt sich kaum verbergen
  - Tür zu einem Raum geht nicht gut auf
  - unnötige Schnörksel fallen auf
    → „5. Rad am Wagen“
- Die Ressourcen sind beschränkt
  - ingenieurmäßiges Herangehen bedeutet, unter den gegebenen Rahmenbedingungen zu optimieren

© W. Pree

Bei Software hingegen ist schlechte Qualität nicht unmittelbar sichtbar

- schlechte Strukturierung
  - „Spaghetti"-Programmcode: Radwechsel => Motor funktioniert nicht mehr
  - replizierter Programmcode
- kaum Wiederverwendung
  - das Rad wird immer neu erfunden

© W. Pree

Ingenieurmäßiges Vorgehen scheint sich nicht auszuzahlen

- Hardware-Ressourcen werden nach *Moore's Law* potenter; der gedankenlose Umgang damit führt zu
  - unnötiger Komplexität
  - nicht mehr verstehbaren Artefakten

OberonOS (ETH ZH)
30.000 Zeilen
Programmcode

4,1 cm

27,5 m

Windows XP:
20,000.000 (!!) Zeilen
Programmcode

© W. Pree

Wie kann Software ingenieurmäßig entwickelt werden?

© W. Pree

von Einzelteilen zu Komponenten

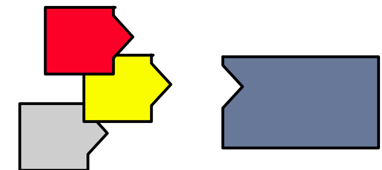
50er Jahre

- Maschinen-/Assembler-Programme: auf bestimmten Prozessor zugeschnitten

60er/70er Jahre

- höhere Programmiersprachen (wie Pascal, C)
- **Anweisungen können zu Funktionen/Prozeduren zusammengefasst werden**

  → Einzelteile, Schrauben, etc.



80er/90er Jahre

- **Funktionen/Prozeduren werden zu Modulen zusammengefasst** (Modula, Oberon, C++, Java, C#)

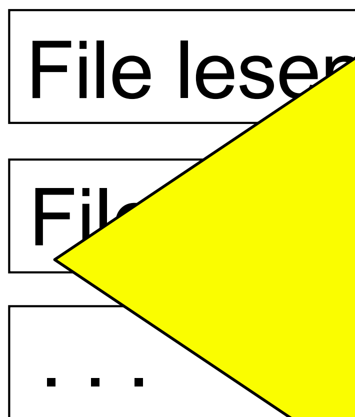
  → Software-Komponenten

© W. Pree

UNIVERSITÄT SALZBURG

Beispiel: Komponente File-Handler

einfache Schnittstelle



versteckte Implementierungsdetails:

Zugriff auf Festplatte

Aufsplitten des Inhalts eines Files

etc.

© W. Pree

UNIVERSITÄT SALZBURG

Architektur-Patterns

Software-Patterns

© W. Pree

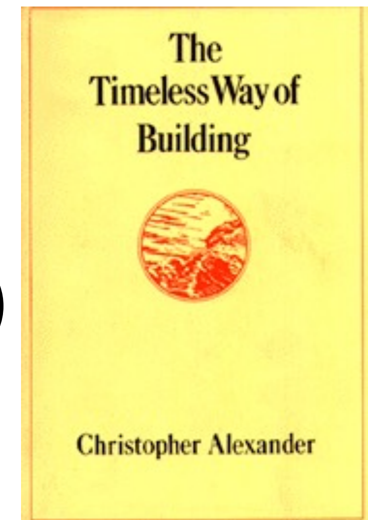
The Timeless Way of Building

Christopher Alexander, Professor of Architecture, Univ. of California, Berkeley:

1979 erschienene Bücher:

The Timeless Way of Building

A Patttern Language (253 Patterns)

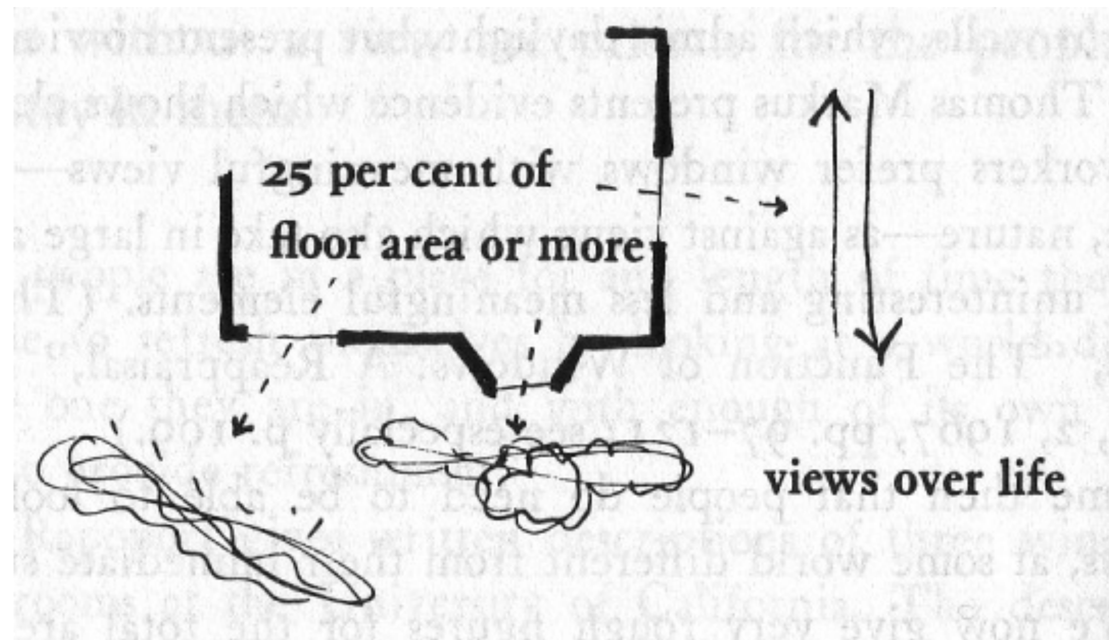


Quality without a name

1991 von der Software-Community entdeckt

© W. Pree

Beispiel: Windows Overlooking Life



© W. Pree
UNIVERSITÄT SALZBURG

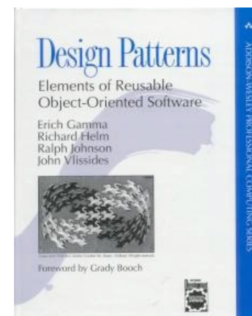
Beispiele für Software Patterns

© W. Pree

Wie können SW-PlugIn-Architekturen geschaffen werden?

Beschrieben in Architektur-Handbüchern (1995):

- E. Gamma, R. Helm, R. Johnson, J. Vlissides:
  Design Patterns: Elements of Reusable Software



- W. Pree:
  Design Patterns for Object-Oriented Software Development

© W. Pree
UNIVERSITÄT SALZBURG

Was sind PlugIn-Architekturen?

- Küchenmaschine: durch Einstecken einer Komponente wird das vorhandene **„PlugIn-System“** zum fertigen Mixer oder Fleischwolf

- neue Automodelle gleichen meist „im Kern“ (Chassis, Getriebe, Motorpalette) den Vorgängermodellen

© W. Pree

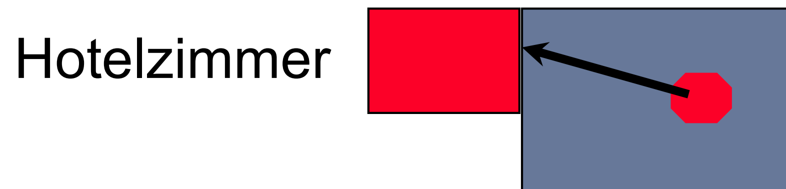
SW-Beispiele

- Einwegsoftware:
  - Hotelreservierungssystem
  - Autovermietungssystem
  - Schiverleihsystem
  - Motorradverleihsystem
  - etc.
- PlugIn-Architektur:
  - Reservierungssystem (Mietgegenstand)

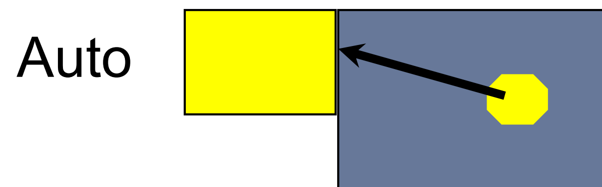
© W. Pree

Einwegsoftware

Abhängigkeit zwischen den Komponenten ist im Programmquelltext:



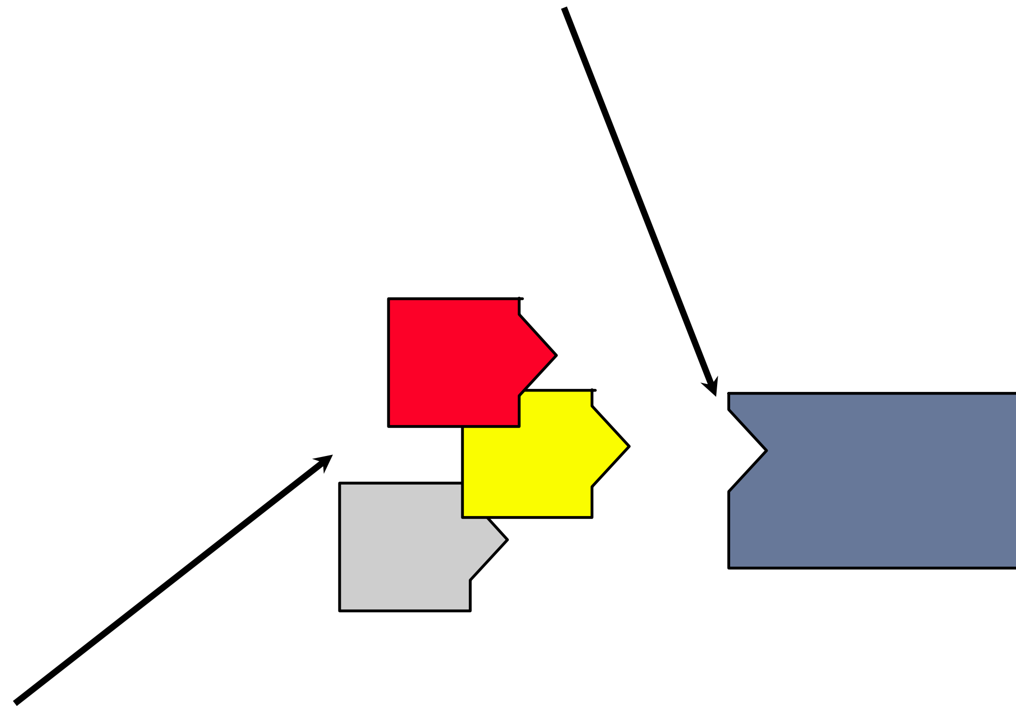
Kopplung mit einer anderen Komponente erfordert Änderungen:



© W. Pree

Pattern: PlugIn-Architekturen erfordern die Definition von „Steckern“

Stecker „Mietgegenstand“



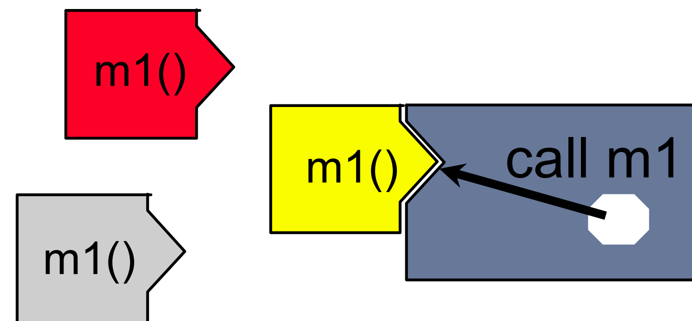
Stecker-kompatible

Komponenten

© W. Pree

UNIVERSITÄT SALZBURG

sogenannte dynamische Bindung von Aufrufen macht Änderungen im Source-Code obsolet



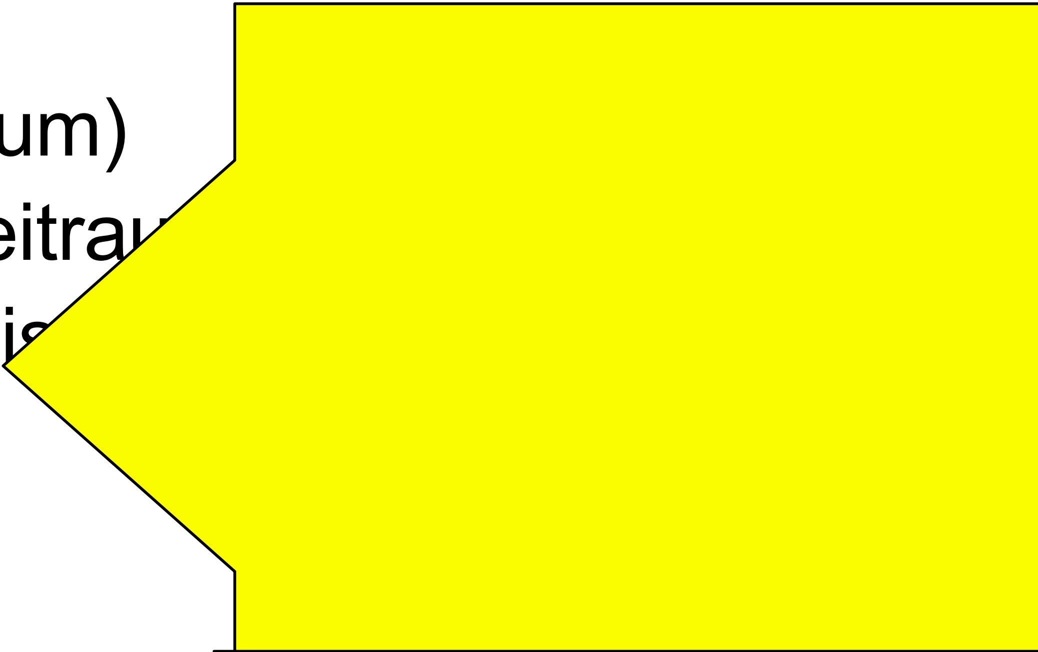
© W. Pree

UNIVERSITÄT SALZBURG

„Stecker“ Mietgegenstand

Definiert allgemeine, abstrakte Eigenschaften:

- istFrei(Zeitraum)
- reserviere(Zeitra
- berechnePrei
- etc.



© W. Pree

UNIVERSITÄT SALZBURG

control 1

Control

moveByVec(vec: 3DVector)
. . .

segments 1..N

FlightSegment

startPos: Position

h flyIt()
calcNextPos(actPos, speed, stepSize): Position
h *calcLength()*
h *calcReqTime(speed)*
h *isValidPattern(...): bool*
h setStartPos(pos: Position)
getActualLength()
getActualTime()

navigation

FlightPattern

parentPattern: FlightPattern

t flyIt()
t calcLength()
t calcReqTime(speed)
t isValidPattern(...): bool
t setStartPos(pos: Position)
addSeg(...)
removeSeg(...)

1..N

for each sgmt in segments:
sgmt.flyIt()

Line

line: 3DVector

calcNextPos(actPos, speed, stepSize): Position
calcLength()
calcReqTime(speed)
isValidPattern(...): bool
setLine(l: Vector)

Circle

... // instance variables for
// storing the plane and
// circle-specific data

calcNextPos(actPos, speed, stepSize): Position
calcLength()
calcReqTime(speed)
isValidPattern(...): bool
setDirection(...)
... // methods for defining
// the plane in which the
// circle is located, and
// the radius of the circle

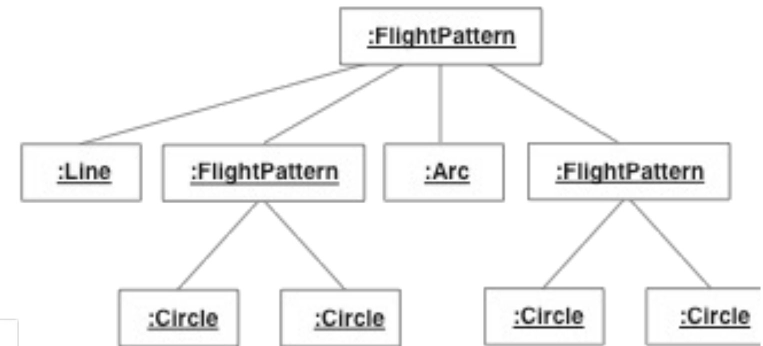
Arc

arcLength: float

calcNextPos(actPos, speed, stepSize): Position
calcLength()
calcReqTime(speed)
isValidPattern(...): bool
setLength(len: float)

1

Navigation



PlugIn-Architektur für Satellitensteuerungen

in Kooperation mit der European Space Agency (ESA): 1998 – 2002

© W. Pree

Automatische Generierung von Software aus „Bauplänen“ (Modellen)

© W. Pree

Compiler: DIE Erfolgsgeschichte der Softwaretechnik

- Programmtext (Pascal, C#, xUML)
  → ausführbares (Maschinen-)Programm

  a+b als Maschinenprogramm:

010 100	hole die Zahl vom Speicherplatz 100(a) in einen Puffer
015 101	addiere dazu den Inhalt von Speicherplatz 101(b)
011 102	speichere das Ergebnis in Zelle 102(a+b)

- für das Zeitverhalten von Embedded Systems noch ausständig
- für andere Anwendungsbereiche denkbar?

© W. Pree

Beispiel: Helicopter Control System (I)

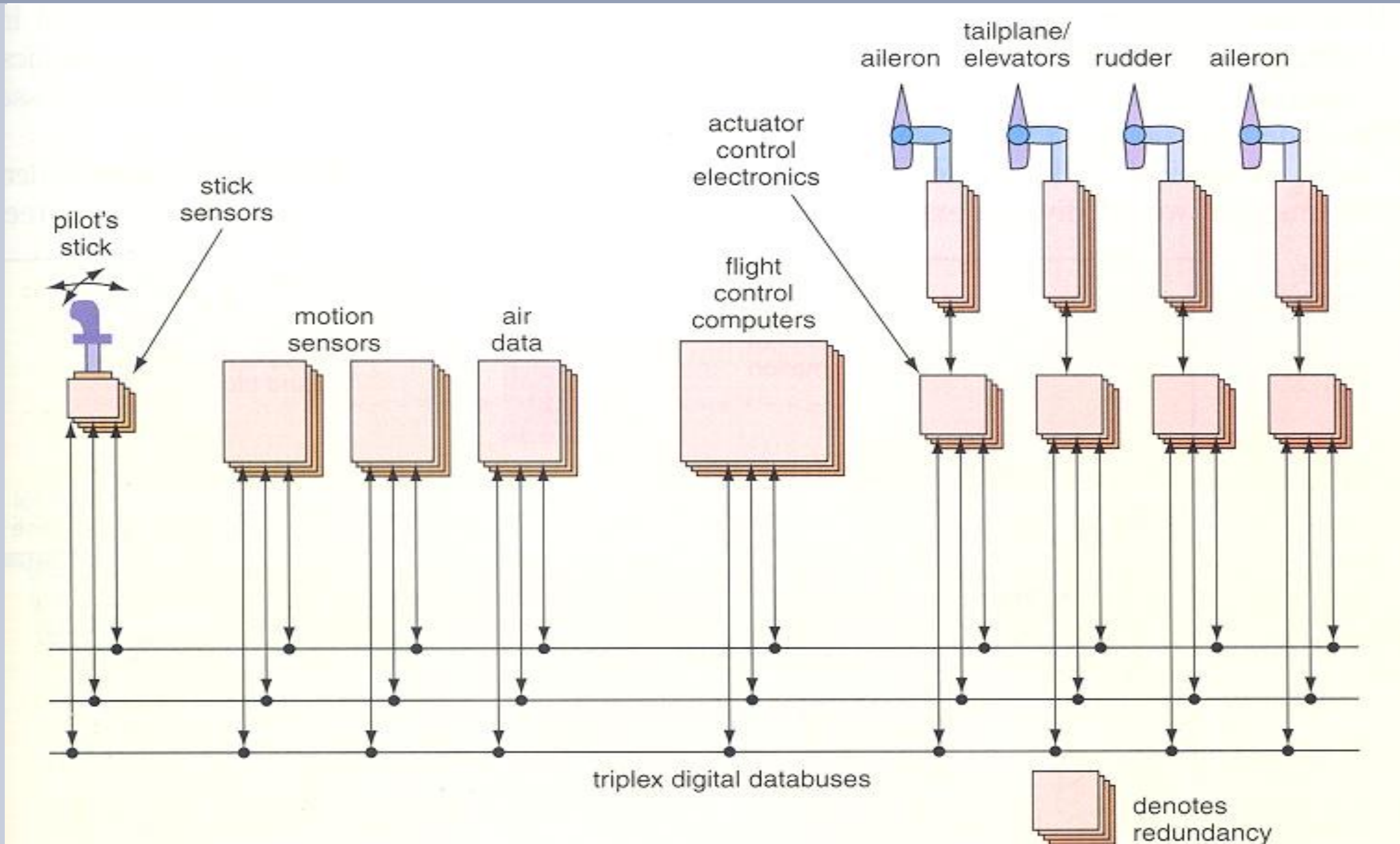
Henzinger, Kirsch, Pree, Sanvido (UC Berkeley)

Schaufelberger, Wirth (ETH Zürich)

© W. Pree

UNIVERSITÄT SALZBURG

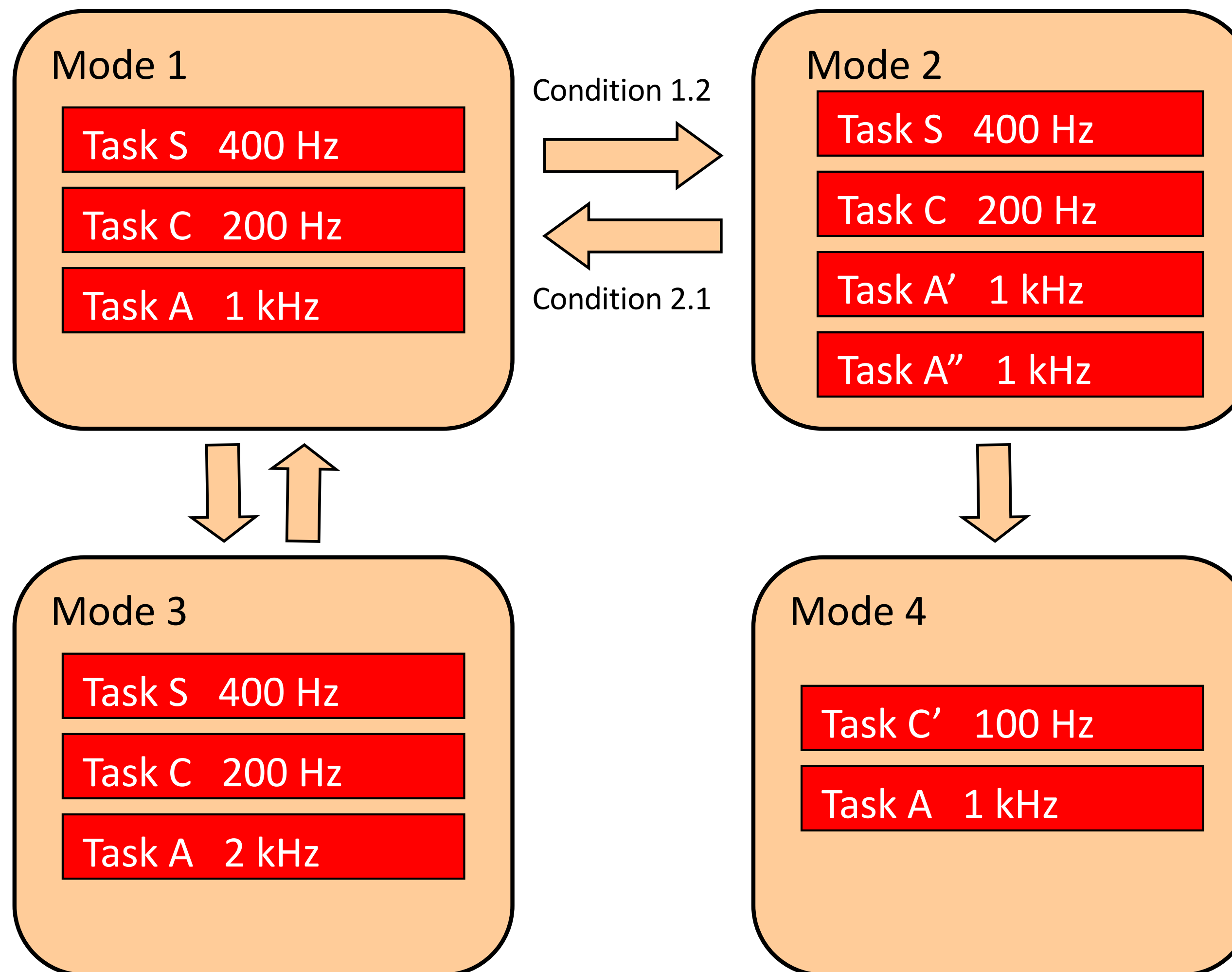
Beispiel: Helicopter Control System (II)



© W. Pree

UNIVERSITÄT SALZBURG

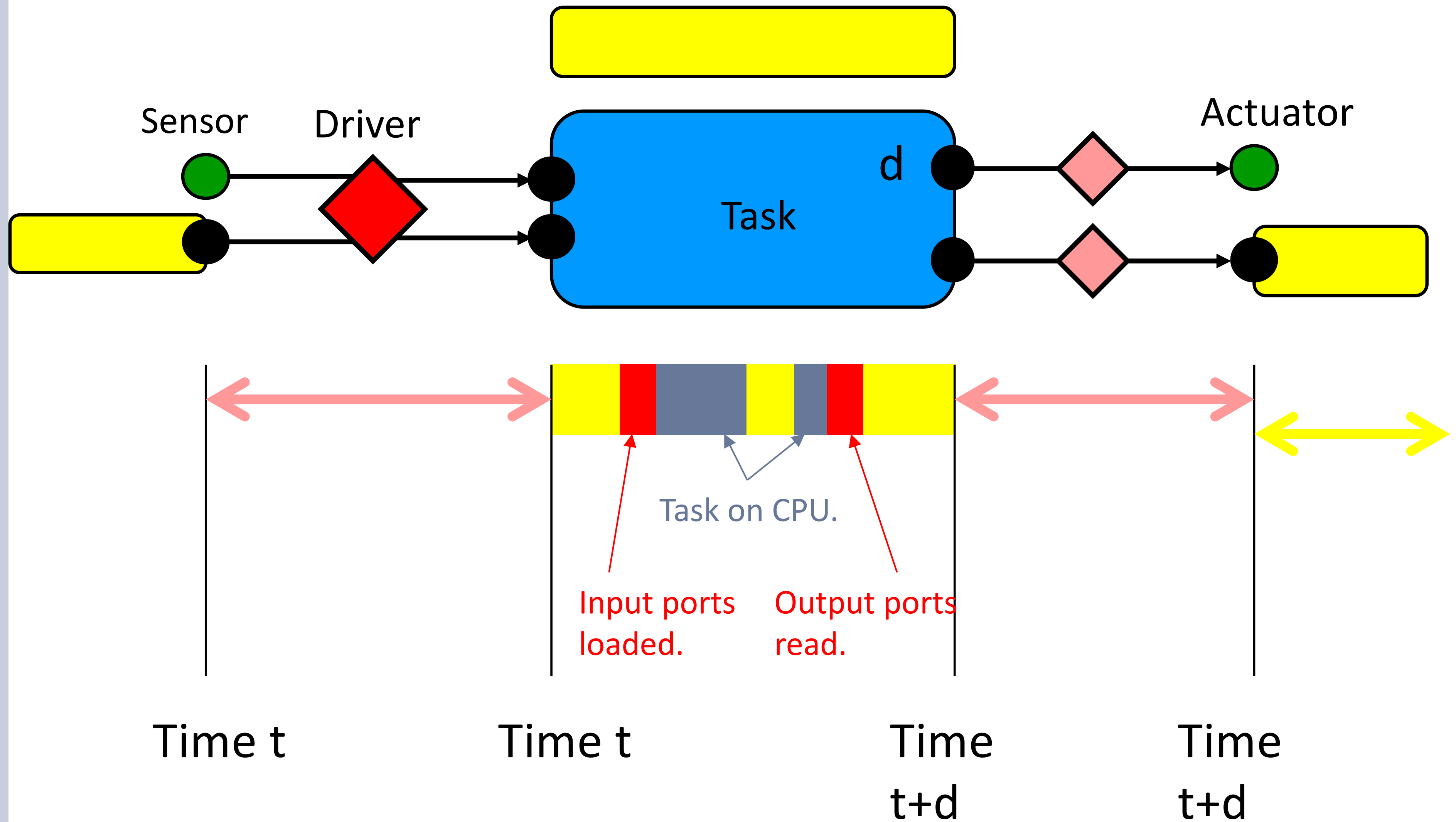
Giotto als „höhere Programmiersprache“ = Bauplan = Modell für Zeitaspekte



=> spezif. HW-Plattform wird irrelevant

© W. Pree

Generierung der Software + deren Verteilung auf Prozessoren erfolgt automatisch



© W. Pree
UNIVERSITÄT SALZBURG

Softwaretechnik – Quo vadis?

© W. Pree

UNIVERSITÄT SALZBURG

- kostenintensive Wartung von Software, die 20-30 Jahre alt ist
- ingenieurmäßige Herangehensweise wird sich zumindest in Teilbereichen etablieren, zB bei sicherheitskritischen Systemen

© W. Pree

- simple, mechanische Weltsicht schwer skalierbar

- Vorbild biologische Systeme
  → Internet wuchs um den Faktor 100 Mio.

© W. Pree

UNIVERSITÄT SALZBURG

Softwaretechnik – Quo vadis?

© W. Pree
UNIVERSITÄT SALZBURG

Vielen Dank für Ihre Aufmerksamkeit!

© W. Pree