UNIVERSITÄT
DUISBURG
ESSEN

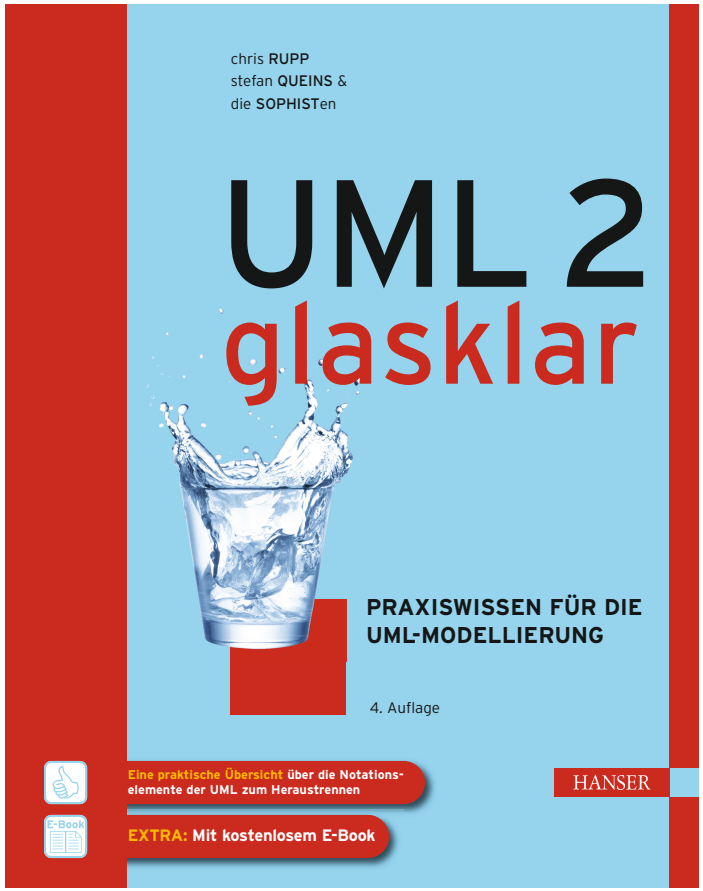
Offen im Denken

Modellierung

Prof. Janis Voigtländer ■ Folienversion: 22.10.2025, 14:51:05 +00:00

Literatur

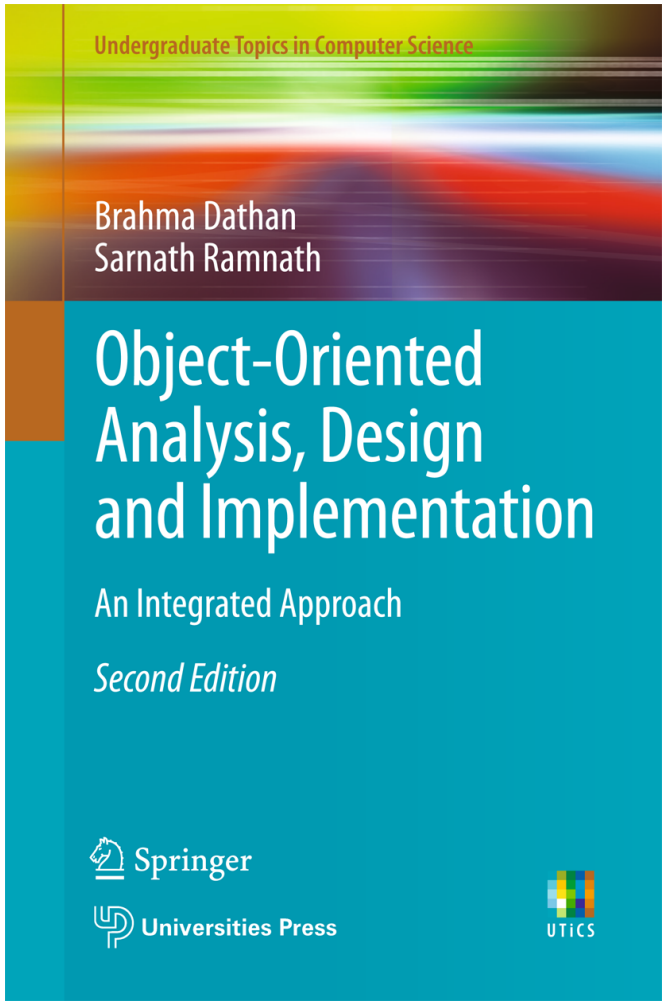
Chris Rupp, Stefan Queins.
UML 2 glasklar.
Hanser Fachbuch, 2012



Buch ist in der Bibliothek verfügbar.

Literatur

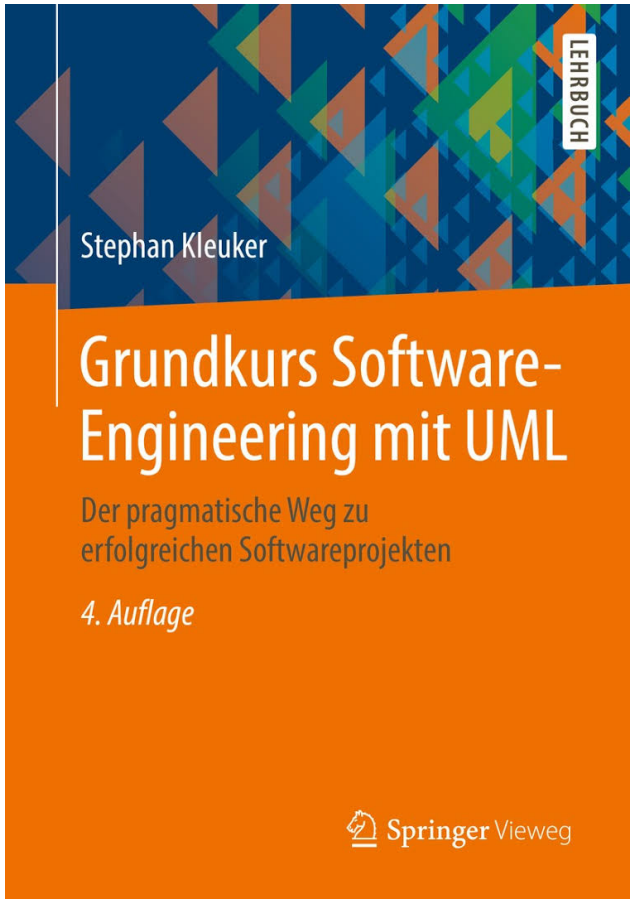
Brahma Dathan,
Sarnath Ramnath.
Object-Oriented Analysis, Design and Implementation – An Integrated Approach.
Springer, 2015



Buch ist in der Bibliothek verfügbar.

Literatur

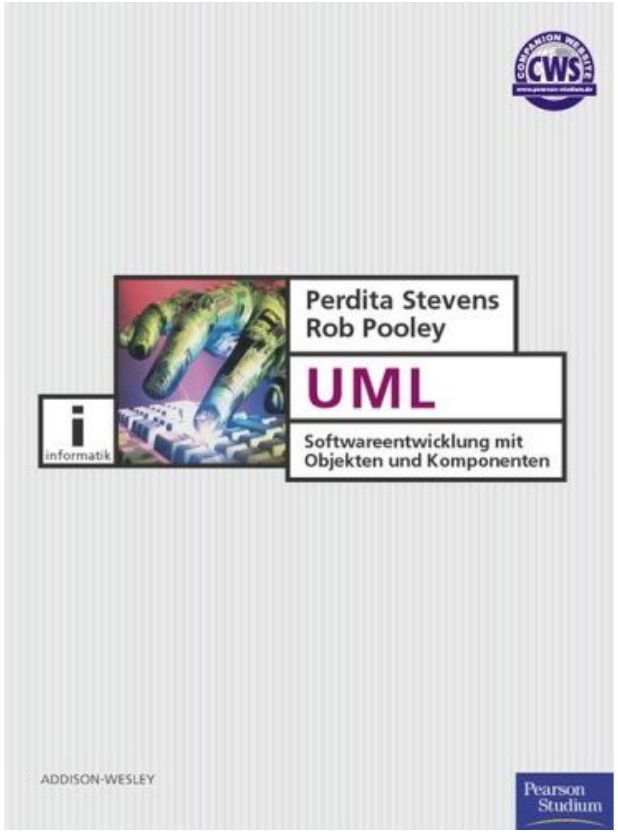
Stephan Kleuker.
Grundkurs Software-Engineering mit UML.
Springer, 2018



https://dx.doi.org/10.1007/978-3-658-19969-2
(elektronische Version über den Uni-Account)

Literatur

Perdita Stevens, Rob Pooley.
UML – Softwareentwicklung mit Objekten und Komponenten.
Pearson, 2001



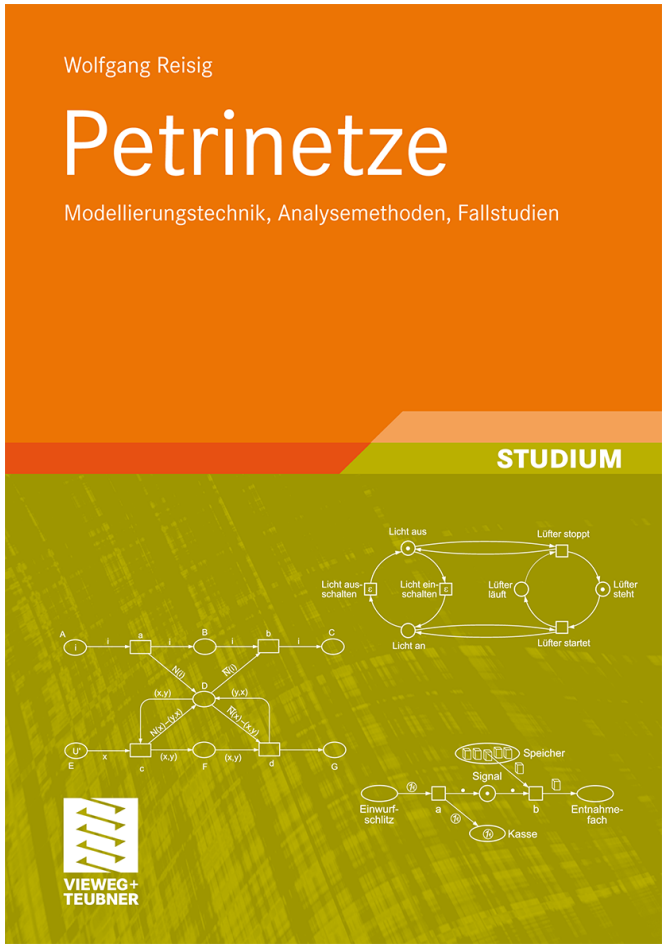
Das englische Original ist in der Bibliothek verfügbar.

Literatur

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Wolfgang Reisig.
Petrinetze –
Modellierungstechnik,
Analysemethoden, Fallstudien.
Vieweg+Teubner, 2010



https://dx.doi.org/10.1007/978-3-8348-9708-4
(elektronische Version über den Uni-Account)

Literatur

Lutz Priese, Harro Wimmel.
Petri-Netze.
Springer, 2008

https://dx.doi.org/10.1007/978-3-540-76971-2
(elektronische Version über den Uni-Account)

Literatur

Tadao Murata.
Petri Nets: Properties, Analysis and Applications.
Proc. of the IEEE, 77(4), pages 541–580, 1989

Petri Nets: Properties, Analysis and Applications

TADAO MURATA, FELLOW, IEEE

Invited Paper

This is an invited tutorial-review paper on Petri nets—a graphical and mathematical modeling tool. Petri nets are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.

The paper starts with a brief review of the history and the application areas considered in the literature. It then proceeds with introductory modeling examples, behavioral and structural properties, three methods of analysis, subclasses of Petri nets and their analysis. In particular, one section is devoted to marked graphs—the concurrent system model most amenable to analysis. In addition, the paper presents introductory discussions on stochastic nets with their application to performance modeling, and on high-level nets with their application to logic programming. Also included are recent results on reachability criteria. Suggestions are provided for further reading on many subject areas of Petri nets.

I. INTRODUCTION

Petri nets are a graphical and mathematical modeling tool applicable to many systems. They are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Petri nets can be used by both practitioners and theoreticians. Thus, they provide a powerful medium of communication between them: practitioners can learn from theoreticians how to make their models more methodical, and theoreticians can learn from practitioners how to make their models more realistic.

Historically speaking, the concept of the Petri net has its origin in Carl Adam Petri's dissertation [1], submitted in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany. The dissertation was prepared while C. A. Petri worked as a scientist at the University of Bonn. Petri's work [1], [2] came to the attention of A. W. Holt, who later led the Information System Theory Project of Applied Data Research, Inc., in the United States. The early developments and applications of Petri nets (or their predecessors) are found in the reports [3]–[8] associated with this project, and in the Record [9] of the 1970 Project MAC Conference on Concurrent Systems and Parallel Computation. From 1970 to 1973, the Computation Structure Group at MIT was most active in conducting Petri-net related research, and produced many reports and theses on Petri nets. In July 1975, there was a conference on Petri Nets and Related Methods at MIT, but no conference proceedings were published. Most of the Petri-net related papers written in English before 1980 are listed in the annotated bibliography of the first book [10] on Petri nets. More recent papers up until 1984 and those works done in Germany and other European countries are annotated in the appendix of another book [11]. Three tutorial articles [12]–[14] provide a complimental, easy-to-read introduction to Petri nets.

Since the late 1970's, the Europeans have been very active in organizing workshops and publishing conference proceedings on Petri nets. In October 1979, about 135 researchers mostly from European countries assembled in Hamburg, West Germany, for a two-week advanced course on General Net Theory of Processes and Systems. The 17 lectures given in this course were published in its proceedings [15], which is currently out of print. The second advanced course was held in Bad Honnef, West Germany, in September 1986. The proceedings [16], [17] of this course contain 34 articles, including two recent articles by C. A. Petri; one [18] is concerned with his axioms of concurrency theory and the other [19] with his suggestions for further research. The first European Workshop on Applications and Theory of Petri Nets was held in 1980 at Strasbourg, France. Since then, this series of workshops has been held every year at different locations in Europe: 1981, Bad Honnef, West Germany; 1982, Varenna, Italy; 1983, Toulouse, France; 1984, Aarhus, Denmark; 1985, Espoo, Finland; 1986, Oxford, England;

Manuscript received May 28, 1988; revised November 4, 1988. This work was supported by the National Science Foundation under Grant DMC-8518588.

The author is with the Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60680, USA.

IEEE Log Number 8826798.

https://dx.doi.org/10.1109/5.24143
(elektronische Version über den Uni-Account)

Literatur

David Harel.
Statecharts: A visual formalism for complex systems.
Science of Computer Programming, 8,
pages 231–274, 1987

Science of Computer Programming 8 (1987) 231-274
North-Holland

STATECHARTS: A VISUAL FORMALISM FOR COMPLEX SYSTEMS*

David HAREL
Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel

Communicated by A. Pnueli
Received December 1984
Revised July 1986

Abstract. We present a broad extension of the conventional formalism of state machines and state diagrams, that is relevant to the specification and design of complex discrete-event systems, such as multi-computer real-time systems, communication protocols and digital control units. Our diagrams, which we call statecharts, extend conventional state-transition diagrams with essentially three elements, dealing, respectively, with the notions of hierarchy, concurrency and communication. These transform the language of state diagrams into a highly structured and economical description language. Statecharts are thus compact and expressive—small diagrams can express complex behavior—as well as compositional and modular. When coupled with the capabilities of computerized graphics, statecharts enable viewing the description at different levels of detail, and make even very large specifications manageable and comprehensible. In fact, we intend to demonstrate here that statecharts counter many of the objections raised against conventional state diagrams, and thus appear to render specification by diagrams an attractive and plausible approach. Statecharts can be used either as a stand-alone behavioral description or as part of a more general design methodology that deals also with the system's other aspects, such as functional decomposition and data-flow specification. We also discuss some practical experience that was gained over the last three years in applying the statechart formalism to the specification of a particularly complex system.

1. Introduction

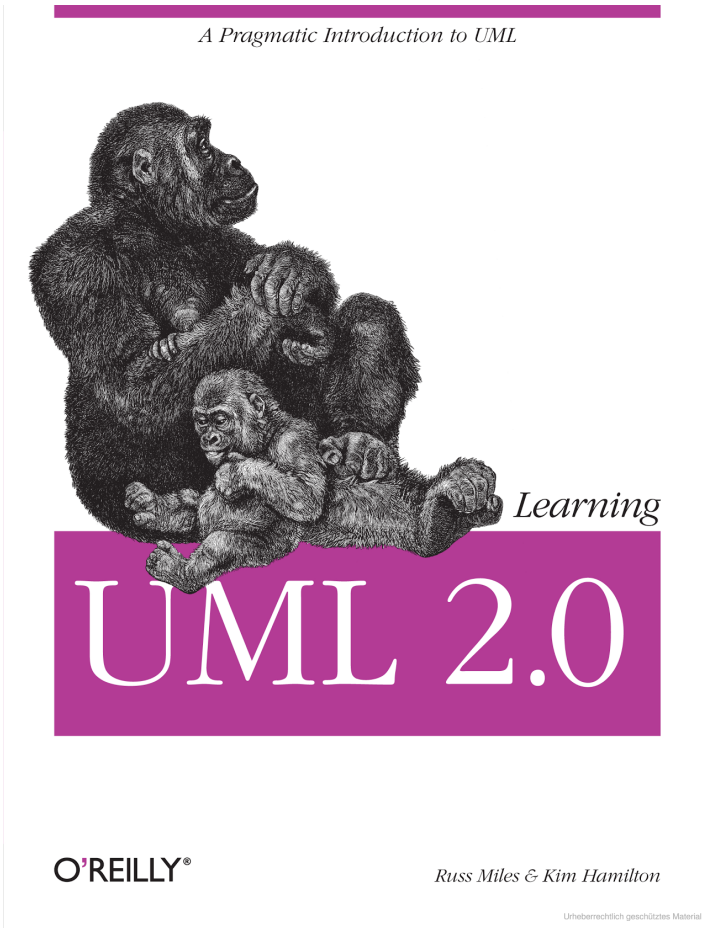
The literature on software and systems engineering is almost unanimous in recognizing the existence of a major problem in the specification and design of large and complex *reactive systems*. A reactive system (see [14]), in contrast with a *transformational system*, is characterized by being, to a large extent, event-driven, continuously having to react to external and internal stimuli. Examples include telephones, automobiles, communication networks, computer operating systems, missile and avionics systems, and the man-machine interface of many kinds of ordinary software. The problem is rooted in the difficulty of describing reactive behavior in ways that are clear and realistic, and at the same time formal and

* The initial part of this research was carried out while the author was consulting for the Research and Development Division of the Israel Aircraft Industries (IAI), Lod, Israel. Later stages were supported in part by grants from IAI and AD CAD, Ltd.

https://dx.doi.org/10.1016/0167-6423(87)90035-9

Literatur

Russ Miles, Kim Hamilton.
Learning UML 2.0.
O'Reilly, 2006



Buch ist in der Bibliothek verfügbar.

Einführung in die Modellierung

Was ist Modellierung?

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Modell

Ein Modell ist eine Repräsentation eines Systems von Objekten, Beziehungen und/oder Abläufen. Ein Modell vereinfacht und abstrahiert dabei im Allgemeinen das repräsentierte System.

System

Der Begriff System wird hier sehr allgemein verwendet. Er kann

- einen Teil der Realität oder ein (noch) nicht bestehendes Gebilde;
- etwas Gegenständliches oder Virtuelles

bezeichnen.

Was ist Modellierung?

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Modellierung

Modellierung ist der Prozess, bei dem ein Modell eines Systems erstellt wird.

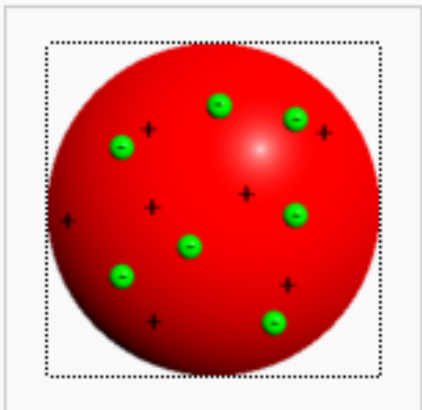
Warum wird überhaupt modelliert?

⇝ Um ein System zu entwerfen, besser zu verstehen, zu visualisieren, zu simulieren, . . .

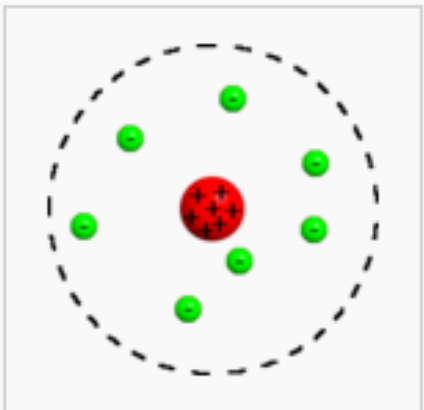
Um etwas konkreter zu werden, betrachten wir kurz klassische Beispiele aus verschiedenen Disziplinen (Physik, Biologie, Klimaforschung).

Atommodelle

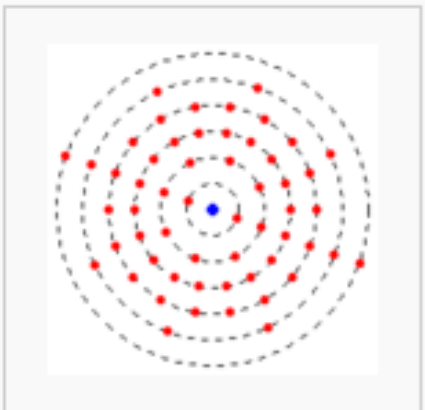
Atome bestehen aus Protonen, Neutronen und Elektronen.
Wie diese Teilchen zusammenwirken, wird in Atommodellen beschrieben, die im Laufe der Zeit immer wieder verändert wurden (obwohl sich die Natur von Atomen ja nicht verändert hat).



Thomsonsches Atommodell



Rutherfordsches Atommodell

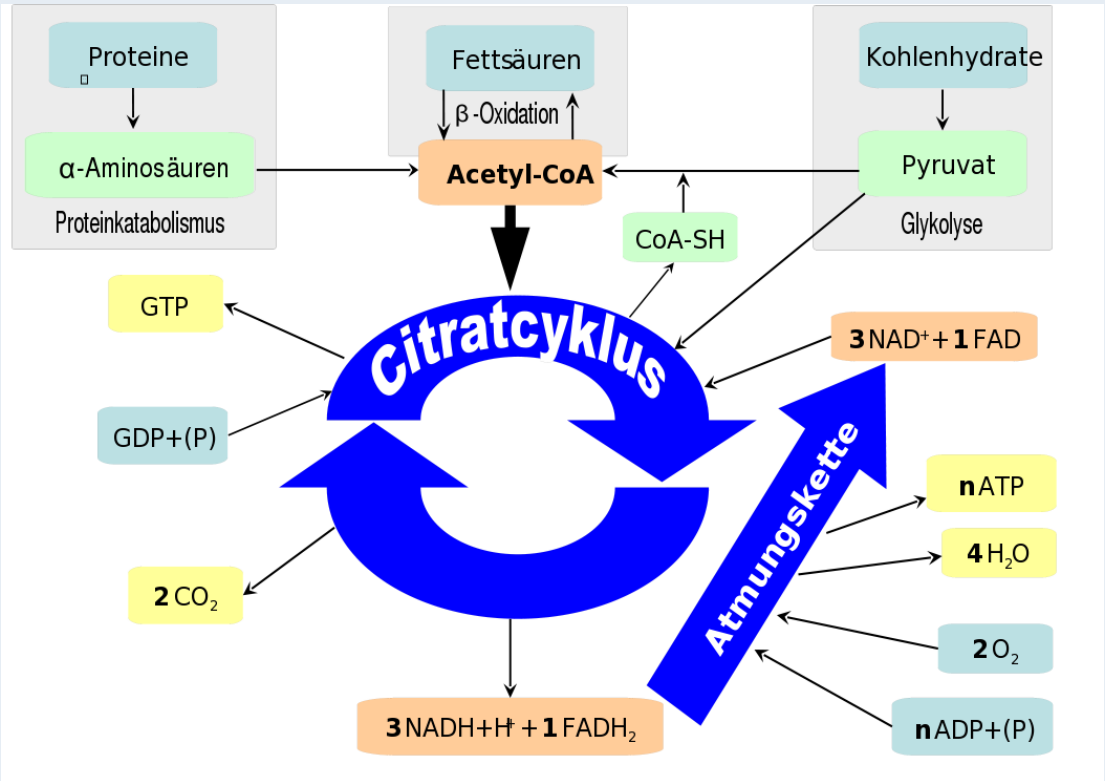


Bohrsches Atommodell

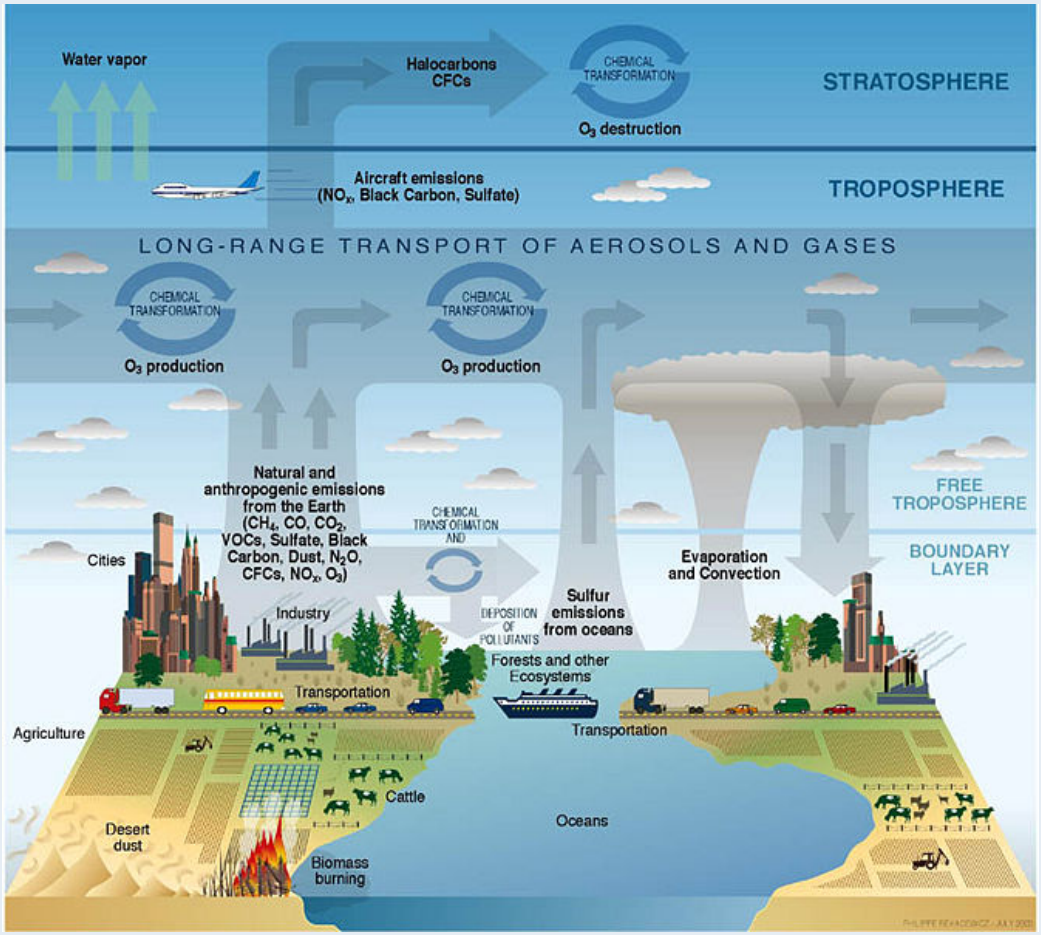
Orbitalmodell

Zitronensäurezyklus

Der Zitronensäurezyklus oder Citratzyklus modelliert den Abbau organischer Stoffe im Körper.



Modell des Transports von Gasen in der Atmosphäre



Arten von Modellen

visuell vs. textuell

Nicht alle Modelle sind visuell bzw. grafisch. Auch mit textuellen Beschreibungen und Formeln kann modelliert werden (siehe beispielsweise mathematische Modelle, Logik, Algebra).

Dennoch werden häufig grafische Darstellungen benutzt, auch aus didaktischen Gründen und um sich besser über die Modelle verständigen zu können.

Arten von Modellen

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

qualitativ vs. quantitativ

- qualitative Modelle: Welche Objekte gibt es? Welche Merkmale und Beziehungen zueinander haben sie? Was passiert? Warum passiert es? In welcher Reihenfolge geschehen die Ereignisse? Was sind die kausalen Zusammenhänge? Welche Phänomene treten auf?
- quantitative Modelle: Wieviele Objekte gibt es? In welchem Mengenverhältnis zueinander treten verschiedene Arten von Objekten auf? Wie lange dauert ein Vorgang? Wie wahrscheinlich ist ein bestimmtes Ereignis?

Arten von Modellen

black box vs. white box (oder glass box)

- black box: Nur das von außen beobachtbare Verhalten wird beschrieben.
- white box: Es wird auch beschrieben, wie das von außen beobachtbare Verhalten im „Inneren“ des Systems erzeugt wird.

Arten von Modellen

statisch vs. dynamisch

- Ein statisches Modell beschreibt den Aufbau oder einen bestimmten Zustand des Systems.
- Ein dynamisches Modell beschreibt hingegen auch, wie das System sich entwickelt (ein oder mehrere mögliche Abläufe oder sogar das gesamte Systemverhalten).

Arten von Modellen

formal vs. semi-formal vs. nicht-formal

Je nach Exaktheit der Modelle erhalten wir:

- formale Modelle, die vollkommen exakt in ihren Aussagen sind (vor allem mathematische Modelle)
- semi-formale Modelle, die teilweise exakt sind, jedoch nicht alles vollständig spezifizieren
- nicht-formale Modelle, die als grobe Richtlinie dienen können, jedoch eher vage Aussagen machen

Wozu ist Modellierung gut?

Wozu werden in der Informatik Modelle benötigt?

Je komplexer ein Informatik-System sein wird, desto wichtiger ist es, einen Plan zu haben, während es konstruiert wird.

Dies führt idealerweise zu:

- Vermeidung von Fehlern
- besserer Qualität
- niedrigeren Kosten
- besserer Dokumentation und Wiederverwendbarkeit

Modellierung ist in der Informatik weniger „explizit“ verbreitet als in den (anderen) Ingenieurwissenschaften, aber ebenso relevant.

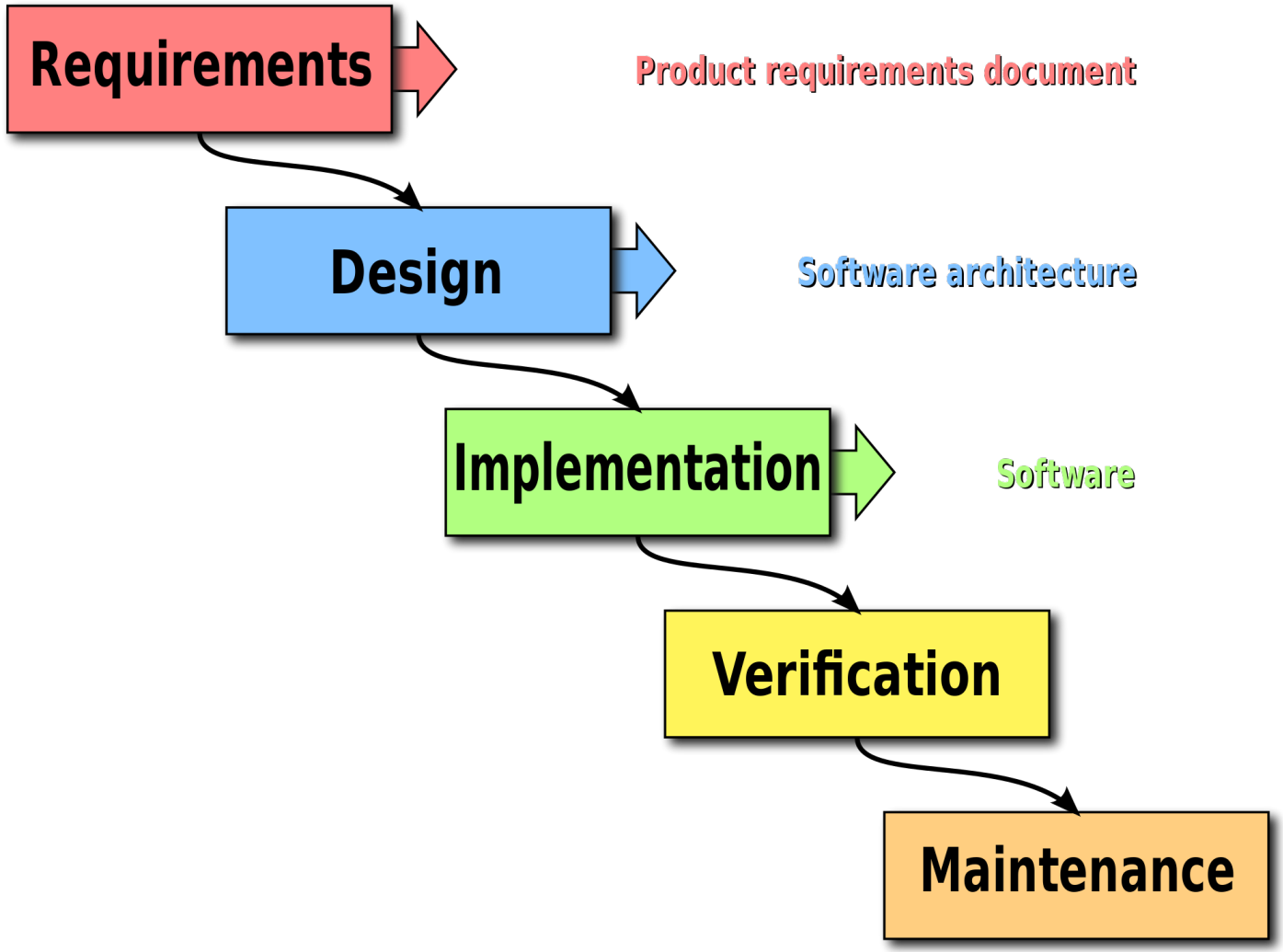
Wozu ist Modellierung gut?

Besonderheiten von Software beeinflussen die Rolle von Modellen:

- Immaterialität; daher gar nicht so leicht, etwa festzustellen, „wie viel“ eines Modells schon umgesetzt wurde
- Einzigartigkeit jedes Softwareprojekts, daher Modelle nicht in der Rolle von Vorlagen zum Zweck mehrfacher Realisierung

Wer modelliert, und wann?

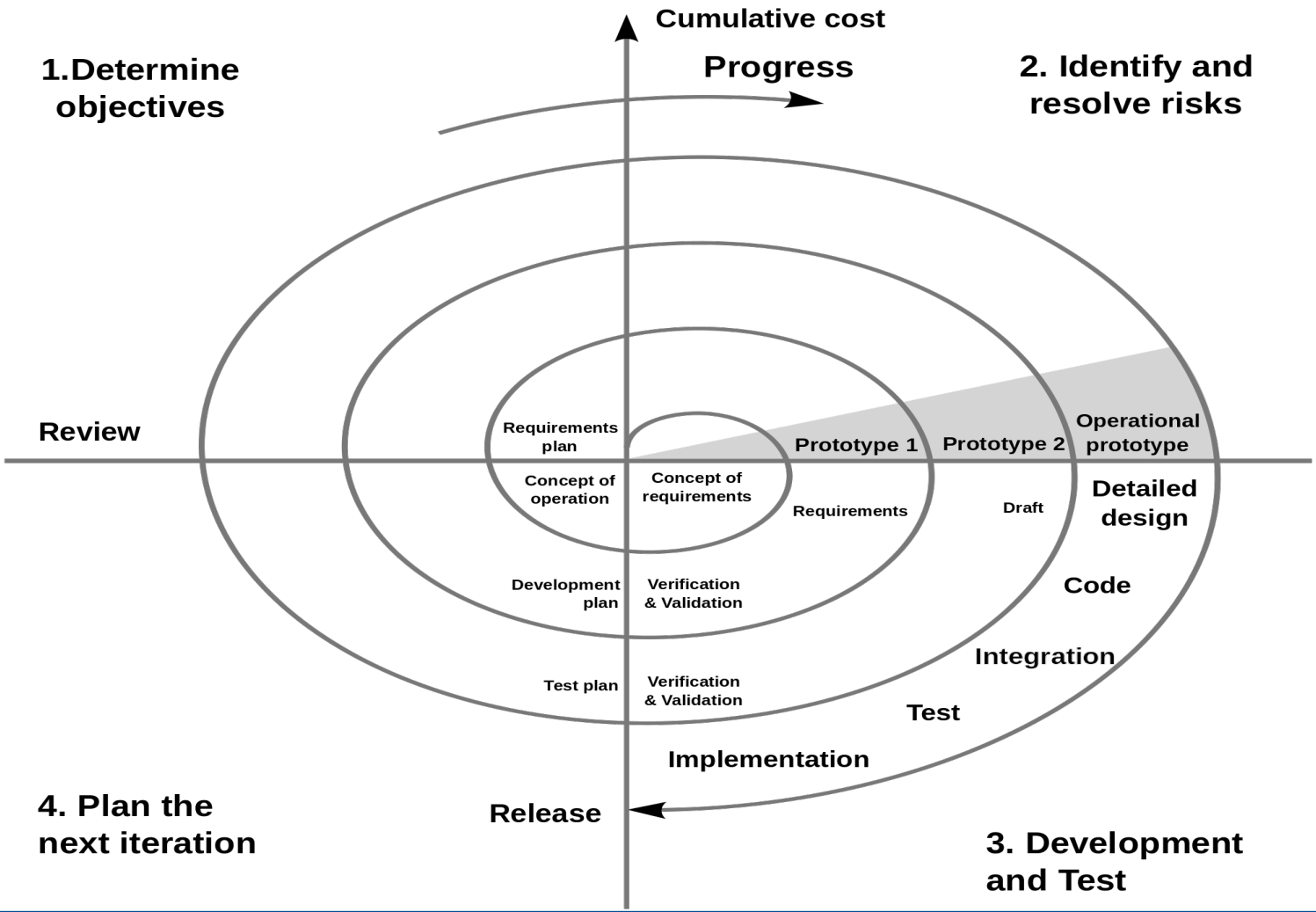
Ein klassischer Softwareentwicklungs-Prozess, „Wasserfall“:



Wer modelliert, und wann?

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Alternativer iterativer Ansatz, „Spiral“:



Wer modelliert, und wann?

„Sonderfälle“:

- Agile Software Development
- Open Source Development
- Model Driven Engineering
- . . .

Weitere Aspekte

Weitere wichtige Gesichtspunkte sind:

- Analyse:
  - Ist ein Modell korrekt? Ist es in sich konsistent?
  - Stimmt das Modell mit der späteren Implementierung überein? (Hier werden Verfahren zum Testen und zur Verifikation benötigt.)
- Werkzeuge, Software-Tools:
  ... werden benötigt zum Zeichnen, zum Darstellen (und Wechseln zwischen verschiedenen Darstellungen), zum Archivieren, zur Code-Generierung, zur Analyse, ...

Inhalt der Lehrveranstaltung

UNIVERSITÄT DUISBURG ESSEN
Offen im Denken

Inhalt (nicht exakt in dieser Reihenfolge)

- Mathematische Grundlagen
- Graphen für statische und dynamische Systembeschreibungen
- Petrinetze
- UML (Unified Modeling Language)

Aus Gründen der Übersichtlichkeit und Didaktik werden wir uns hauptsächlich mit kleinen Modellen befassen.

Die Verwendung von Modellen zur Verifikation von Eigenschaften wird nicht zentral sein, aber immer wieder mal thematisiert.

Inhalt der Lehrveranstaltung

Statische Modellierung von Operationen (teils als Vorbereitung für UML)

Statische Modellierung von Operationen

- Beim Entwurf eines Systems in der Informatik, eines Programms, oder vielleicht auch einer Datenbank, stellt sich oft zunächst die Frage, welche Operationen angeboten und umgesetzt werden sollen, und auf was für Daten diese arbeiten müssen.
- Dabei geht es noch nicht darum, was und wie die Operationen etwas genau „tun“ werden, sondern welche Aufrufe/Verwendungen syntaktisch erlaubt sind und wie Operationen kombiniert werden können.
- Mindestens kann das später der Dokumentation dienen; im Idealfall hilft es bereits bei der Software-Implementierung, etwa durch präzises Erfassen, welche Fälle von Eingaben überhaupt behandelt werden müssen, oder durch die Möglichkeit der Konsistenzprüfung und damit Vermeidung von Fehlern (etwa bei versuchter Verwendung von Operationen in nicht sinnvoller Kombination).

Beispiel: Arithmetische Operationen

Zum Beispiel könnten (etwa beim Entwurf einer Taschenrechner-App) übliche arithmetische Operationen auf der Menge $\mathbb{N}$ der natürlichen Zahlen, inklusive Null, wie folgt statisch zu modellieren begonnen werden:

$$+ \ : \ \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$* \ : \ \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$/ \ : \ \mathbb{N} \times \mathbb{N} \to ?$$

Beispiel: Arithmetische Operationen

Um verbotene Division durch Null auszuschließen, Einschränkung des zweiten Arguments auf die Menge $\mathbb{N}_+ \subset \mathbb{N}$ der positiven natürlichen Zahlen:

$$/ \ : \ \mathbb{N} \times \mathbb{N}_+ \to ?$$

Entscheidung darüber, ob nur ganzzahlige Division umgesetzt, oder auch rationale Zahlen unterstützt werden sollen:

$$/ \ : \ \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N} \quad \text{vs.} \quad / \ : \ \mathbb{N} \times \mathbb{N}_+ \to \mathbb{Q}$$

Dabei gibt es keine „richtige“ oder „falsche“ Entscheidung, sondern es kommt auf den Anwendungszweck und -kontext an. Dies sinnvoll herauszuarbeiten, wäre hier gerade Teil des Modellierens.

Beispiel: Arithmetische Operationen

Mit mehreren vorliegenden „Datentypen“ ($\mathbb{N}$, $\mathbb{N}_+$, eventuell $\mathbb{Q}$) können einige statische Informationen zu Operationen noch präzisiert werden, zum Beispiel durch Festhalten von:

$$+ : \mathbb{N}_+ \times \mathbb{N} \to \mathbb{N}_+$$

$$+ : \mathbb{N} \times \mathbb{N}_+ \to \mathbb{N}_+$$

$$* : \mathbb{N}_+ \times \mathbb{N}_+ \to \mathbb{N}_+$$

aber nicht etwa:

$$* : \mathbb{N}_+ \times \mathbb{N} \to \mathbb{N}_+$$

Beispiel: Arithmetische Operationen

Dann können wir rein anhand der statischen Informationen, also ohne wirklich etwas auszurechnen, feststellen, dass etwa $3/((5+0)*4)$ okay ist, $3/((5*0)*4)$ aber nicht sicher.

Umgang mit solchen arithmetischen Ausdrücken oder „Termen“ kennen Sie natürlich aus der Schule, insbesondere neben dem Aufstellen auch das Umformen von Termen.

Interessant aus Modellierungssicht ist nun, dass so ein algebraischer Zugang aber nicht nur für Zahlen und Operationen auf diesen möglich ist, sondern auch allgemein für Operationen in praktisch beliebigen anderen Anwendungsdomänen.

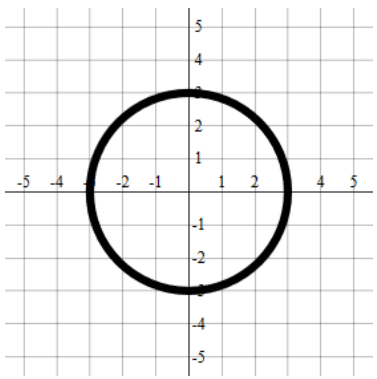
Beispieldomäne: Vektorgrafik

Angenommen, wir möchten für ein Grafikprogramm diverse mögliche Operationen zum Zeichnen und Manipulieren von Bildern modellieren.

Statt mathematischen Zahlenbereichen wie eben, sind hier neben grundlegenden Datentypen, die Sie etwa aus Python oder Java kennen oder in GPT bald kennenlernen werden (Int, Float, String), domänenspezifische Typen wie Color, Points, Picture relevant.

Eine Operation könnte dann wie folgt „getypt“ sein:

$$\text{circle} \;:\; \text{Float} \rightarrow \text{Picture}$$

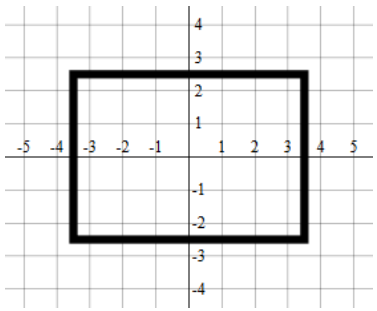


circle(3)

Beispieldomäne: Vektorgrafik

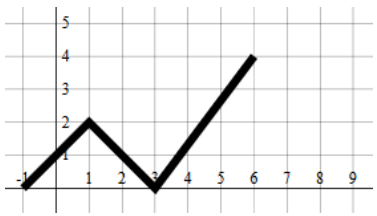
Weitere Operationen für Grundfiguren:

$$\text{rectangle} \;:\; \text{Float} \times \text{Float} \to \text{Picture}$$



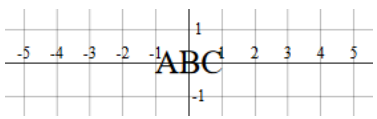
rectangle$(7,5)$

$$\text{path} \;:\; \text{Points} \to \text{Picture}$$



path$([(-1,0),\ldots])$

$$\text{print} \;:\; \text{String} \to \text{Picture}$$

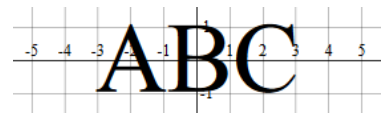


print("ABC")

Beispieldomäne: Vektorgrafik

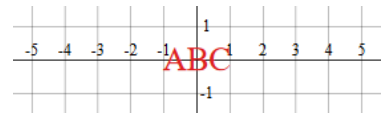
Operationen zur Manipulation bestehender Figuren:

$$\text{scale} : \text{Picture} \times \text{Float} \times \text{Float} \rightarrow \text{Picture}$$



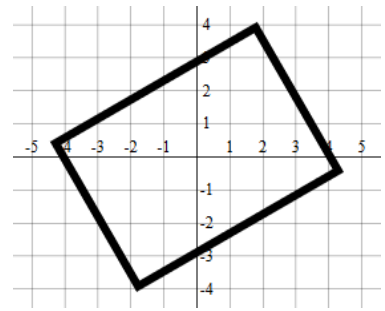
scale(print("ABC"), 3, 3)

$$\text{color} : \text{Picture} \times \text{Color} \rightarrow \text{Picture}$$



color(print("ABC"), red)

$$\text{rotate} : \text{Picture} \times \text{Float} \rightarrow \text{Picture}$$

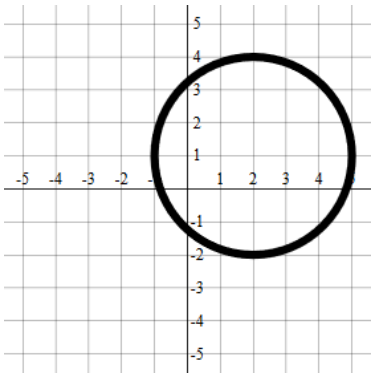


rotate(rectangle(7, 5), 30)

Beispieldomäne: Vektorgrafik

Operationen zur Positionierung:

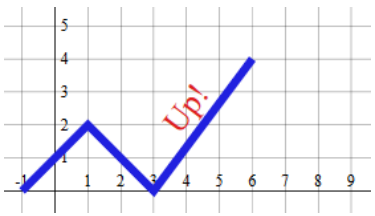
$$\text{move} \;:\; \text{Picture} \times \text{Float} \times \text{Float} \to \text{Picture}$$



move(circle(3), 2, 1)

. . . und zur Komposition aus Teilbildern:

$$\& \;:\; \text{Picture} \times \text{Picture} \to \text{Picture}$$



color(path([. . .]), blue) & move(rotate(color(print("Up!"), red), 53), 4, 2.5)

Beispieldomäne: Vektorgrafik

Anhand der statischen Informationen können wir nun wieder bestimmte Aufrufe/Kombinationen von Operationen als nicht sinnvoll erkennen.

Zum Beispiel:

- circle(circle(3)) – circle(3) ist Picture, nicht Float
- print(circle(3)) – ein Picture ist kein druckbarer Text
- print(3) – korrekt wäre: print("3")
- scale(print("ABC"),3) – falsche Anzahl der Argumente
- color(red,print("ABC")) – falsche Reihenfolge der Argumente
- rotate(30,30) – Zahl ist kein Bild
- "ABC" & "DEF" – Komposition nur für Bilder vorgesehen

Andererseits sind natürlich nicht alle Programmierfehler automatisch so erkennbar (z.B. rectangle(7,5) vs. rectangle(5,7)).