Funktionale Programmierung

Thanh Viet Nguyen

9. Dezember 2024

HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

- Was ist ein Funktionale Programmierung?
- Ursprung
- Unterschiede zur imperativen Programierung
- Historische Entwicklung
- Wichtige Konzepte: Funktionen als erste Klasse, Unveränderlichkeit und Rekursion
- Praktische Beispiele
- Ziel der Präsentation: grundlägige Verständnis zur funktionalen Programmierung

Weise Worte:

- The proper use of comments is to compensate for our failure to express ourselves in code."(Robert C. Martin, Functional Programming in Java, Vorwort)
- In programming the hard part isn't solving problems, but deciding what problems to solve."(Paul Graham, Functional Programming in Java, Vorwort)
- Testing by itself does not improve software quality. Test results are an indicator of quality, but in and of themselves, they don't improve it. Trying to improve software quality by increasing the amount of testing is like trying to lose weight by weighing yourself more often."(Steve McConnell, Functional Programming in Java, Vorwort)

- Lambda Kalkül (1930): Alonzo Church (14.06.1903 - 11.08.1995)
- Vor Maschinen, die solchen Code ausführen konnten, gab es bereits Ansätze dazu.
- Mathematische Themen gebiete:
  - Typen Theorie
  - Kategorie Theorie

Lambda Kalkül

- kleinste Universelle Programmiersprache, die keine Machine benötigt

$$\langle \text{expression} \rangle := \langle \text{name} \rangle \mid \langle \text{function} \rangle \mid \langle \text{application} \rangle$$
$$\langle \text{function} \rangle := \lambda \langle \text{name} \rangle . \langle \text{expression} \rangle$$
$$\langle \text{application} \rangle := \langle \text{expression} \rangle \langle \text{expression} \rangle$$

Typen Theorie

Formales System zur Klassifizierung von Objekten in verschiedene Typen, um Paradoxien zu vermeiden, mit einer hierarchischen Struktur, die Typen als Objekte betrachtet, und Anwendungen in der formalen Verifikation, Logik und Grundlagenforschung; oft konstruktivistisch und eng mit intuitionistischer Logik verbunden.

- Werte durch Typen klassifiziert, die definieren, welche Art von Daten ein Wert repräsentieren kann.
- Statische Typisierung: Typen werden zur Compile-Zeit überprüft, was bedeutet, dass viele Fehler frühzeitig erkannt werden können.
- : Dynamische Typisierung: Typen werden zur Laufzeit überprüft, was mehr Flexibilität bietet, aber auch zu Laufzeitfehlern führen kann.

Kategorie Theorie

Konzept, das Strukturen und Beziehungen zwischen ihnen durch Objekte und Morphismen beschreibt, um verschiedene mathematische Disziplinen zu vereinheitlichen und zu analysieren, wobei zentrale Begriffe wie Funktoren, natürliche Transformationen und Kategorien verwendet werden.

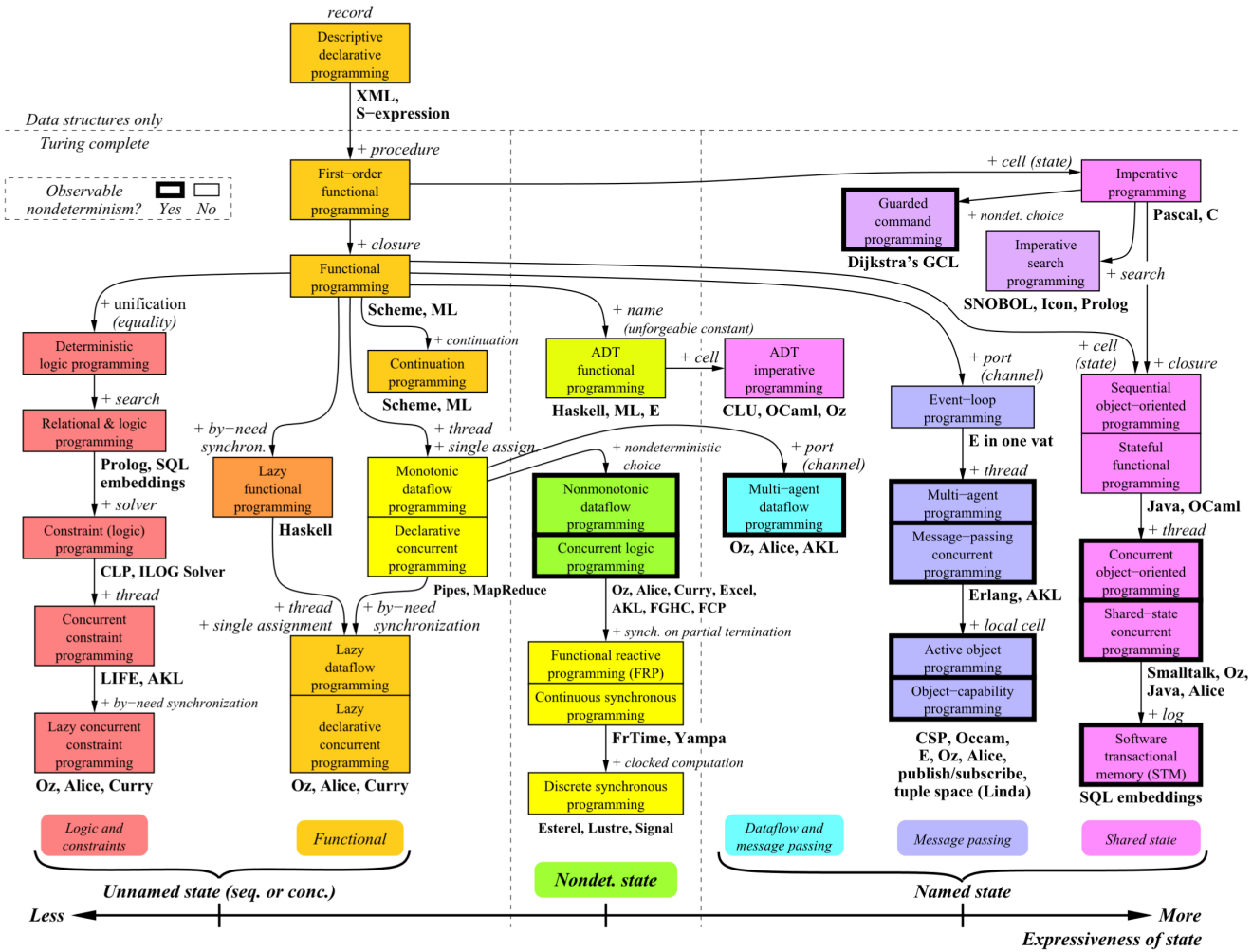
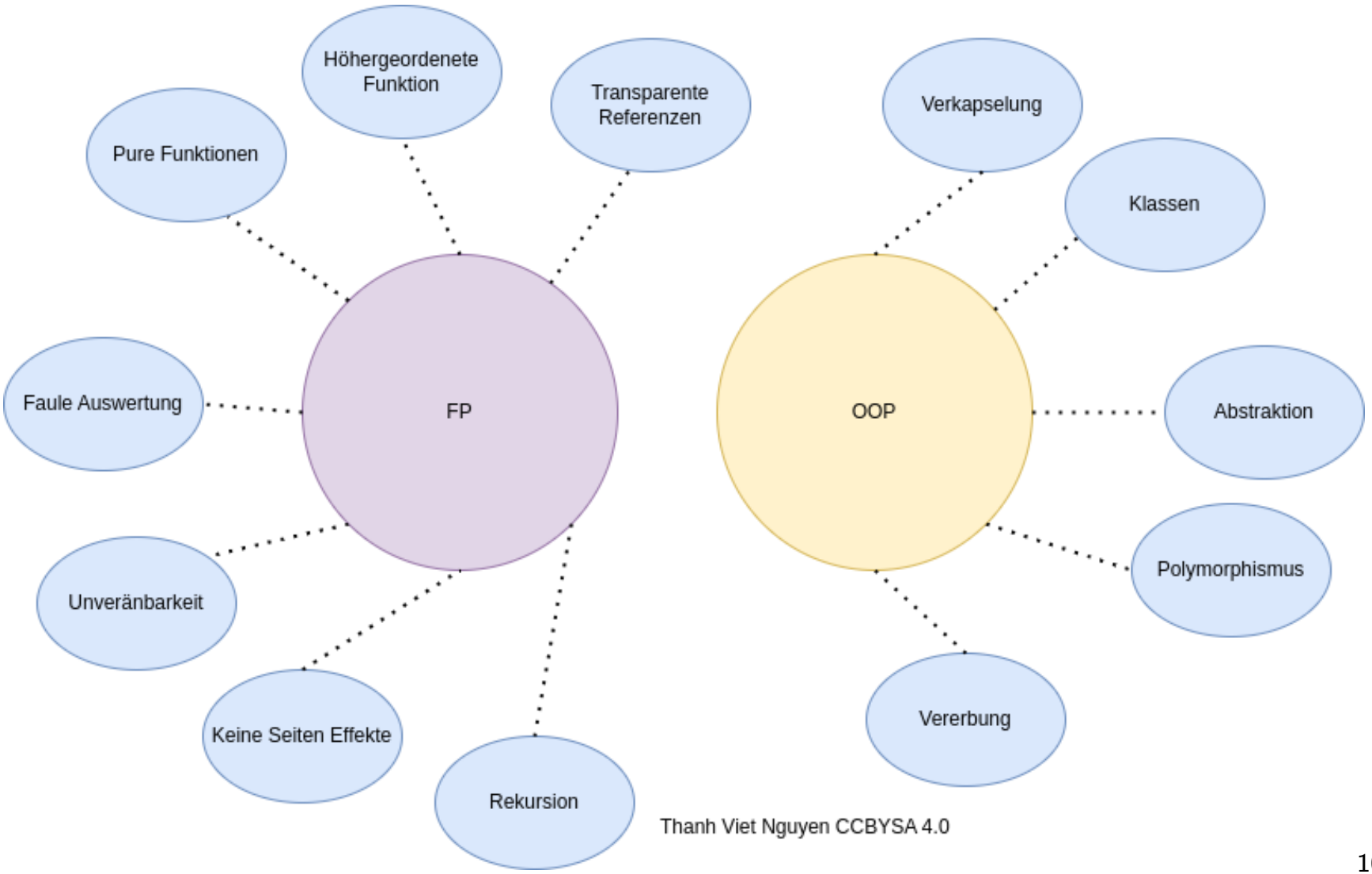


Figure 2. Taxonomy of programming paradigms

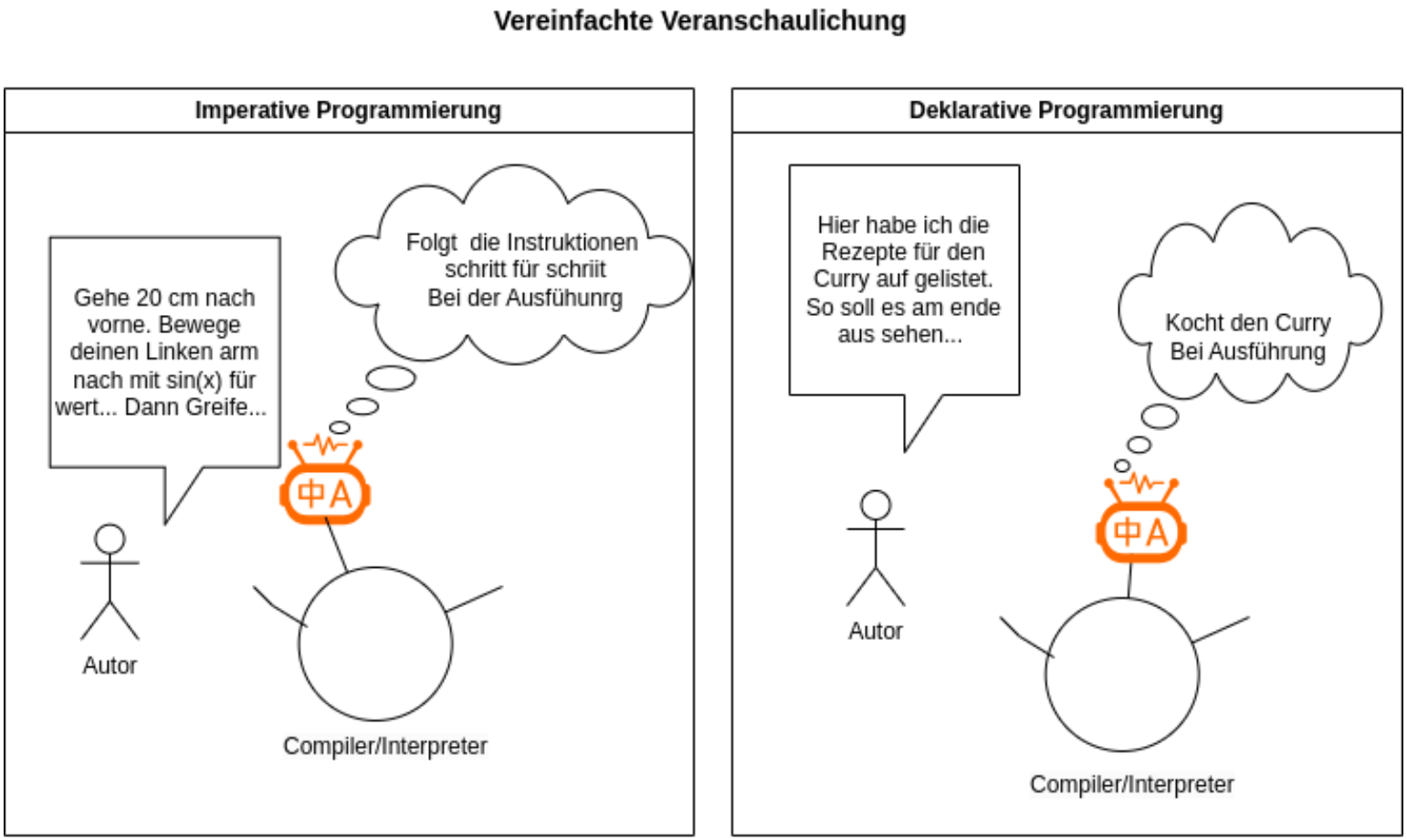
https://blog.acolyer.org/2019/01/25/programming-paradigms-for-dummies-what-every-programmer-should-know/

Objekt Orientierung und Funktionale Programmierung

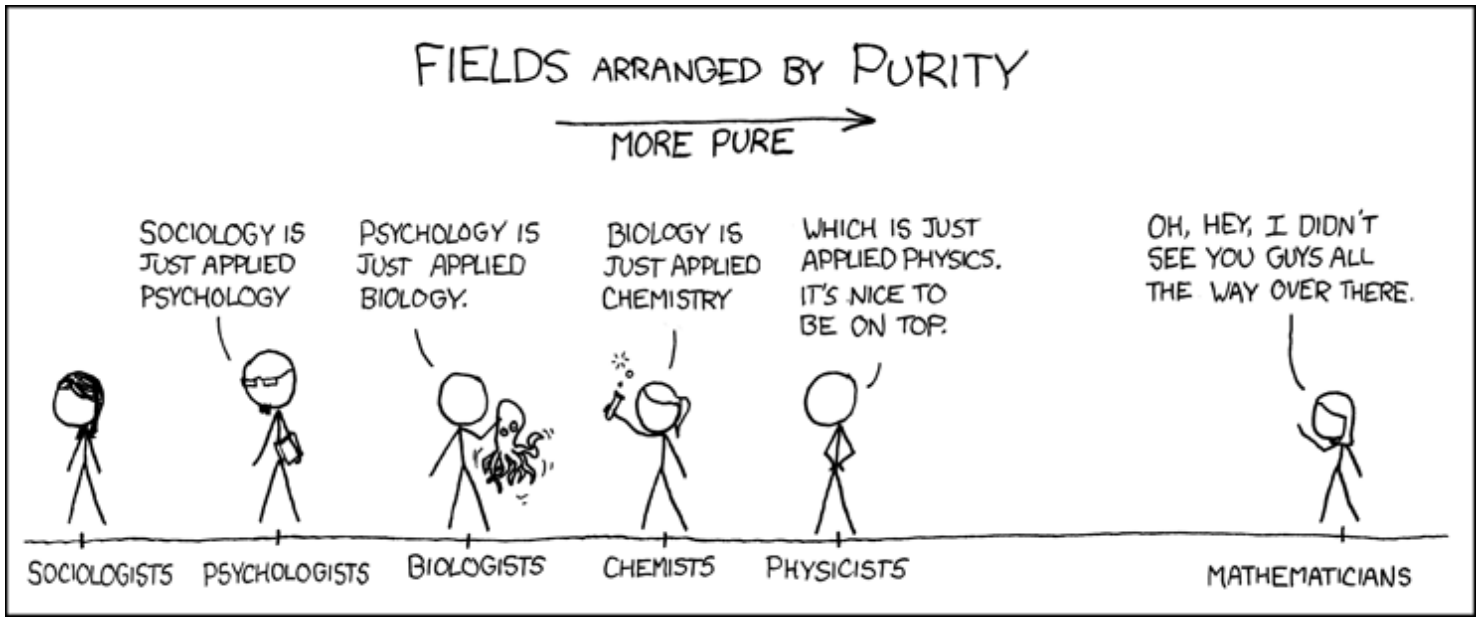


- ” Object oriented programming makes code understandable by encapsulating moving parts. Functional programming makes code understandable by minimizing moving parts.”
(Michael Feathers, Functional Programming in Java, Vorwort)

Deklaritive und Imperativen Programmiereung



Thanh Viet Nguyen CCBYSA 4.0



https://xkcd.com/435/

Lisp Maschinen

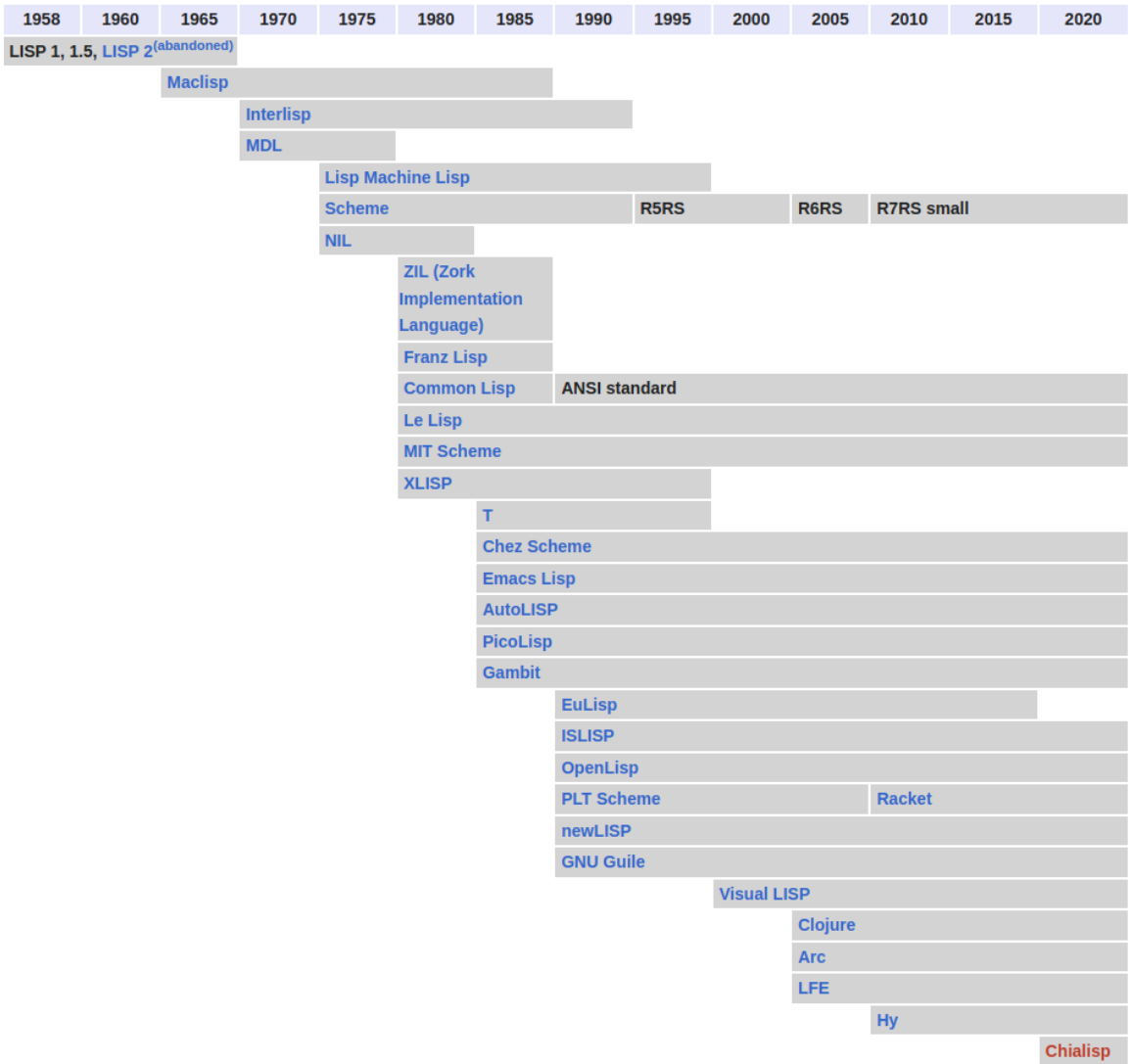
- 1960er
- Hoffnung an Digitales Computing für Mathematischeproblemlösungen
- Erforschung Künstlicher Intelligenz


https://en.wikipedia.org/wiki/Space-cadet_keyboard
https://en.wikipedia.org/wiki/Lisp_machine

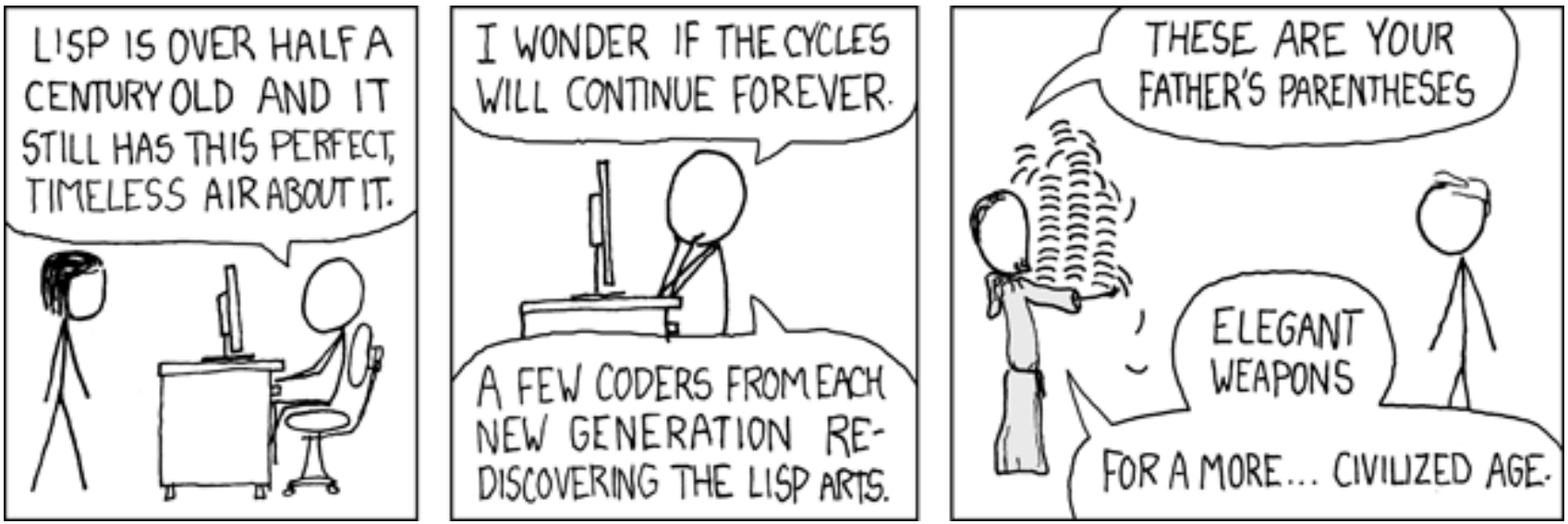
```
(print "Ich bin nicht alt, sondern nachhaltig!")
(define fact
    (lambda (n)
    (if (= n 0) ; Basisfall
        #;(= n 1)
        1 ; Identitaet von *
        (* n (fact (- n 1))))))
```

https://en.wikipedia.org/wiki/Lisp_(programming_language)

Lisp Dialekte



https://en.wikipedia.org/wiki/Lisp_(programming_language)



https://www.explainxkcd.com/wiki/index.php/297:_Lisp_Cycles

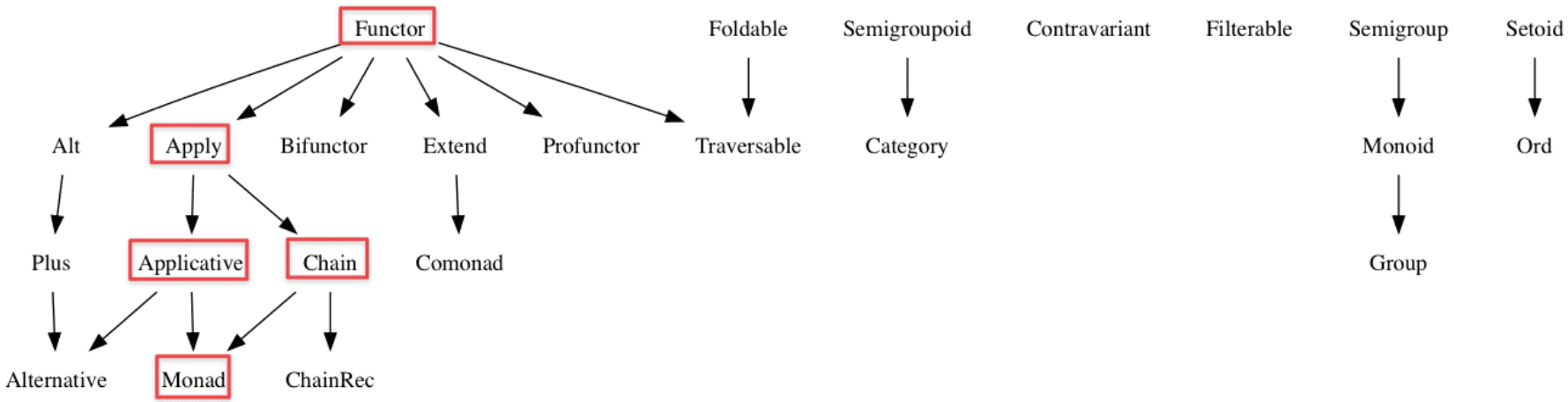
- Fokus auf unveränbare Daten
- Wenn eine Funktion pur ist, folgt: `Kein State -> keine Seiteneffekte`
- Basiert sich auf Aussagen
- Rerefrenzen sind Transparent: `input -> output`

Konzepte von FP part 1/2

- Rekursion: Funktion, die sich selbst aufruft.
- Höhere Geordnete Funktionen: Rufen andere Funktionen als Argumente zu übergeben oder als Ergebnisse zurückzugeben.
- Currying: Umwandlung von Funktionen, die mehrere Argumente annehmen, in eine Kette von Funktionen, die jeweils ein Argument akzeptieren.
- Functor:
  - Functor: Funktion mit mehreren Argumenten in eine Kette von Funktionen umgewandelt wird, die jeweils ein Argument akzeptieren, was die teilweise Anwendung von Argumenten ermöglicht.
  - wird über die Funktion "fmap" Bereitsgestellt.
  - Befolgt Identitätsregel und Kompositionsregel.

Konzepte von FP part 2/2

- Applificat:
  - Erweiterung des Funktors: Funktionen, die in einem Kontext sind, auf Werte in einem anderen Kontext anzuwenden.
  - Verwendet Funktionen " pure" oder "return", um Werte in den Kontext zu bringen
  - Befolgt Identitäts- und Assoziativitätsregeln.
  - Heißt Applikation in der Mathematik: Anwendung einer Funktion auf ihre Argumente, wobei das Ergebnis der Funktion zurückgegeben wird.
- Monad:
  - Spezielle art des Funktors: Bietet zusätzlich Möglichkeit, Werte in einem Kontext zu verketten.
  - Abstraktion um mit Effekten wie Zustandsänderungen, Ein- und Ausgabe oder Fehlern umzugehen
  - Befolgt Identitätsregel und Assoziativitätsregel.



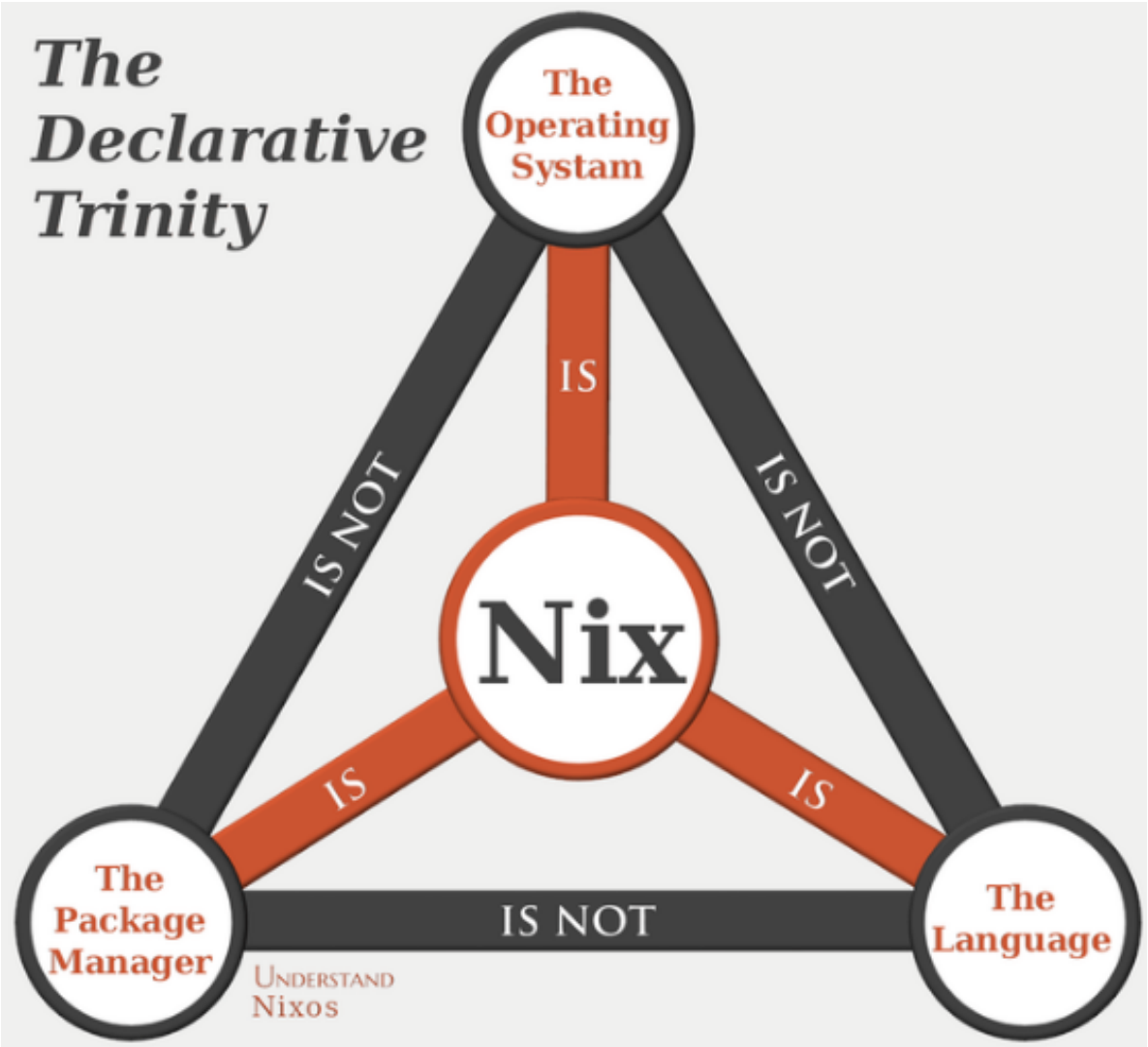
https://yabai.tw/category/monad/overview/

Vorteile	Nachteile
Mehr Sicherheit beim Entwickeln: Nebenläufigkeit, Wartbarkeit und Testbarkeit	keine Zustandskontrolle
Parallele Verarbeitung	meist speicherintensiver bei der Laufzeit
Code ist in der Regel lesbarer und ordentlicher	Lesbarkeit $\neq$ Verständlichkeit
Modularität	
Höhere Abstraktion	

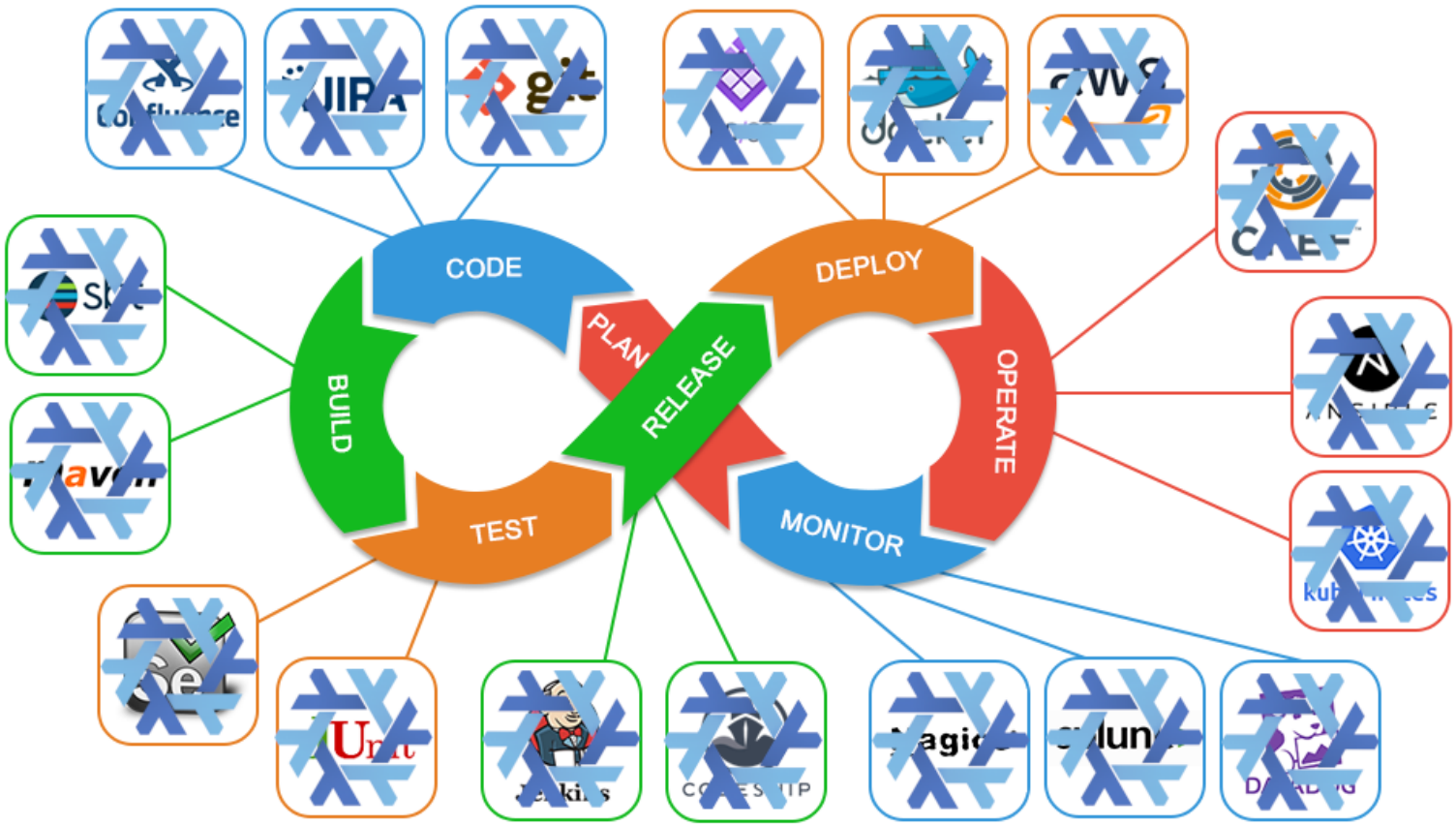
- Software Anwendungen
- Systemkonfigurationen
- Beweisasistent für Mathematische Theoreme sowie Automatische Beweisführung
  - Roqc für Lisp Dialekt Liebhaber
  - Lean für nicht Lisp Dialekt Liebhaber
- Verarbeitung von Daten

Deklarative Systemkonfigurationen und Reproduzierbare Bauten

- NixOS: GNU/Linux Distro
- Guix: GNU/Linux Distro, welches mit Guile Scheme Dialekt Konfiguriert wird
- Code als Basis für Infrastrukturen
- Emacs: Ein Texteditor mit vielen erweiterungen, welches mit Lisp Dialekt (Elisp)
- Debian GNU/Linux verwendung von OCAML für Reproduzierbare Checks



https://github.com/gytis-ivaskevicius/high-quality-nix-content/tree/master/memes



https://github.com/gytis-ivaskevicius/high-quality-nix-content/tree/master/memes

- Grundverständnis zur Funktionalen Programmiereung
- Lambda Kalkulus etwas kennen gelernt
- Einige frühere pur funktionale Programmiersprachen beinflussten moderne nicht pur funktionale Programiersprachen.
- Schwächen und Stärken der Funktionalen Programmierung
- Anwendungsgebiete die von FP geführt werden
- Wird in der Zukunft häufiger verwendet durch Entwickeln mit KI und Quantencomputer
- FP ist nicht die Lösung für alle Probleme in der Softwareentwicklung.

Mehr erfahren?
Hier sind einige Einsteiger freundliche Projekte

- Hier zu der Rekursionspräsentation
- Mehr über Lisp Dialekt: `https://docs.racket-lang.org/`
- Mehr über Haskell:`https://learnyouahaskell.github.io/`
- FP für weitere Programmier Sprachen: `https://learnfp.org/`
- Falls jemand nicht genug hat: `https://nixos.org/learn/`
-
```
(defun display-message ()
(let ((message Falls jemand immer noch nicht
genug hat: https://guix.gnu.org/cookbook/"))
(display-message)))
```

Literatur

- Mathematics in Programming (Xinyu Liu)
- Functional and Logic Programming (Gerhard Goos, Juris Hartmanis)
- Revised^6 Report on the Algorithmic Language Scheme (Robert Bruce Findler, Jacob Matthews)
- Functional Programming in Java (Pierre-Yves Saumont)
- The Design and Implemention of Programming Languages (John Hughes)
- A short introduction to the Lambda Calculus (Achim Jung)
- Lambda Calculus (Tobias Nipkow)
- Functional Programming with Overloading and Higher-Order Polymorphism (Mark P. Jones)
- A Tutorial Introduction to the Lambda Calculus (Raúl Rojas)
- Mathematical Components (Assia Mahboubi and Enrico Tassi)

Weblinks

- `https://yabai.tw/category/monad/overview/`
- `https://bitsavers.org/pdf/mit/cadr/chinual_6thEd_Jan84/`
- `https://www.mdc-berlin.de/research/publications/pigx-reproducible-genomics-analysis-pipelines-gnu-guix`
- `https://docs.racket-lang.org/`
- `https://learnyouahaskell.github.io/`
- `https://rand.cs.uchicago.edu/cufp_2015/`
- `https://www.adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html`

https://github.com/SpiXFamily/FP-Beginners-Presi