FAIBLE.nrw

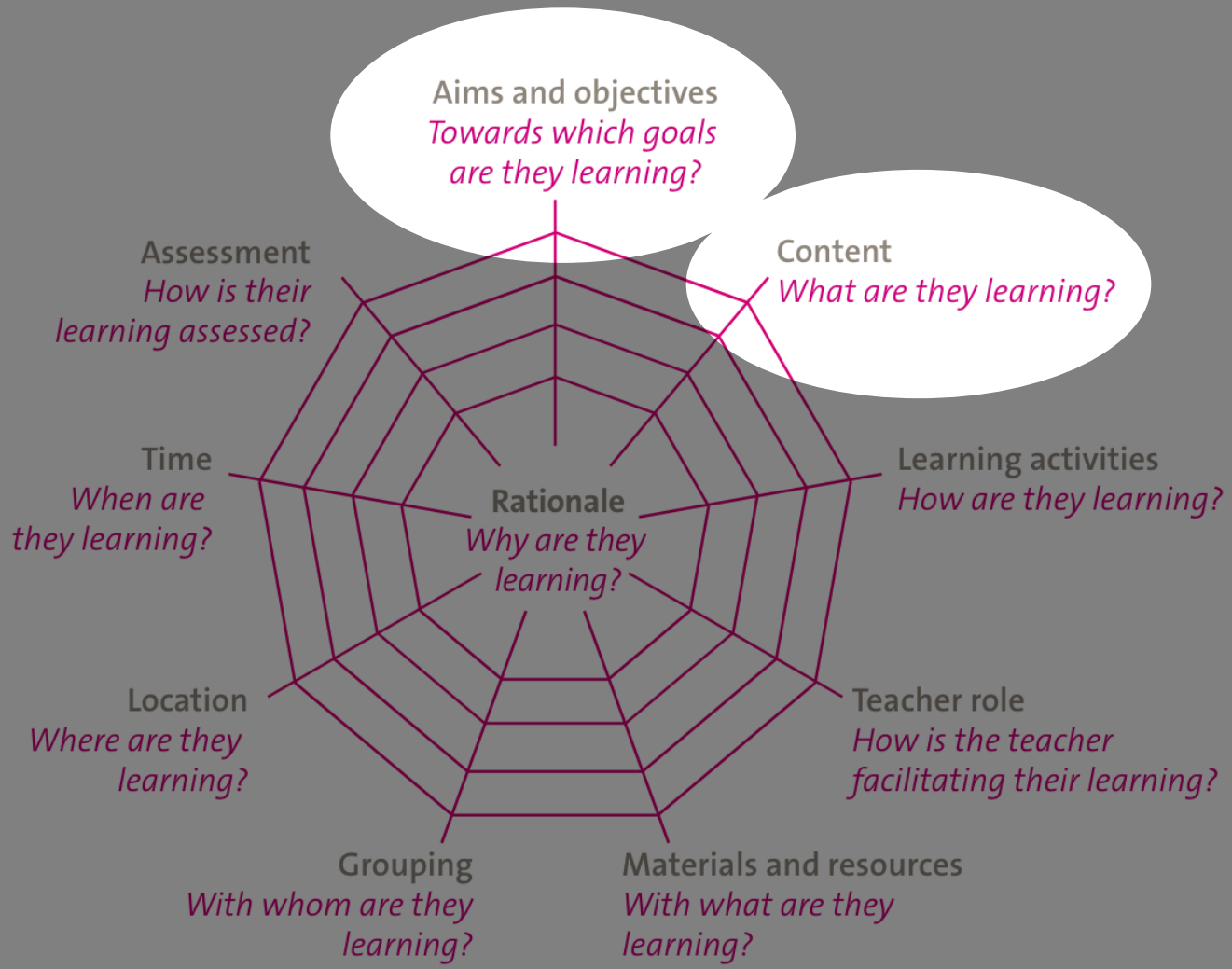
Didaktik der Programmierung

UNIVERSITÄT PADERBORN

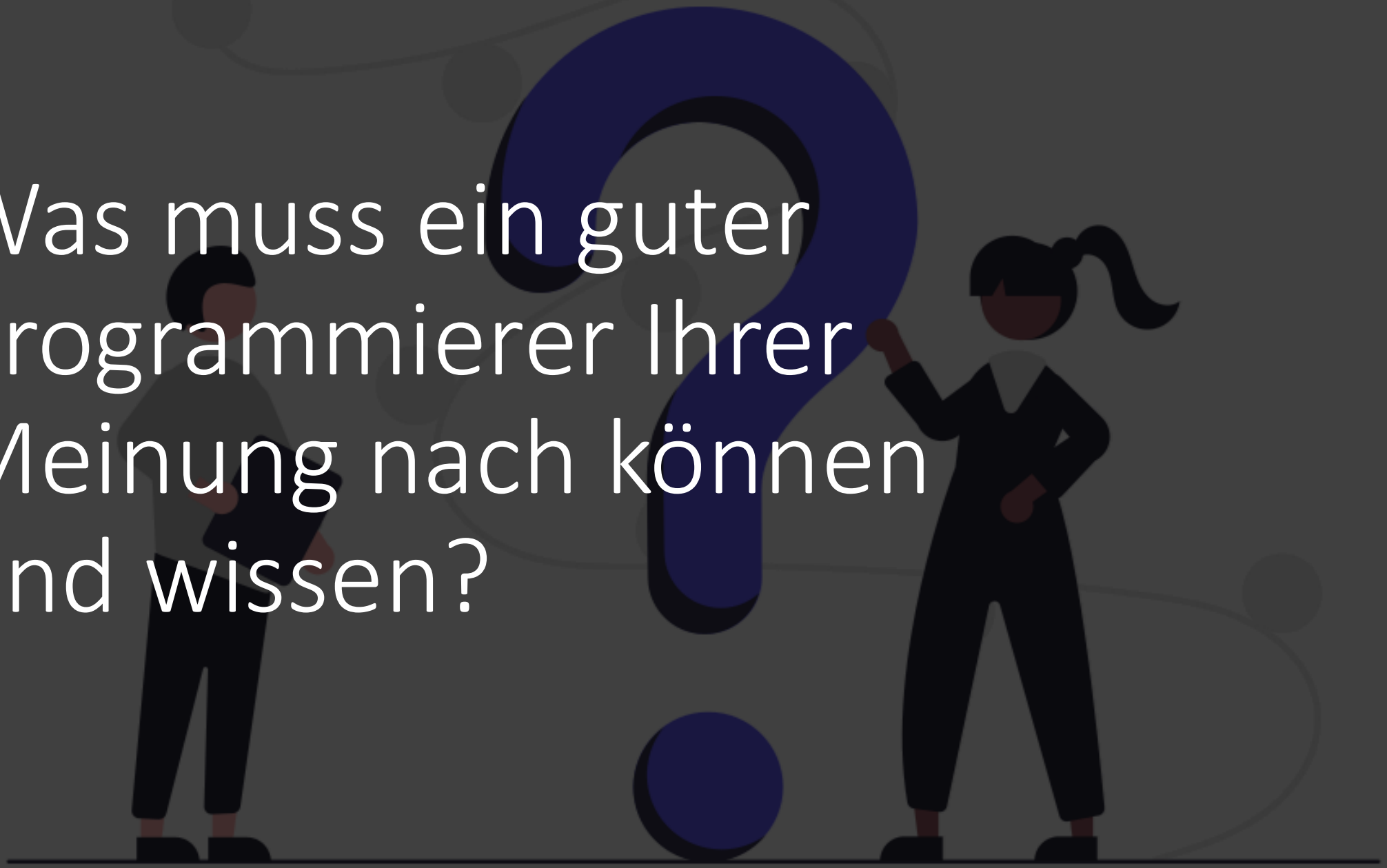

Notional Machine
Modellieren
Projektunterricht
Bildung
Pair Programming
PRIMM
Programmiersprachen
Software Life Cycle
4C/ID
Kompetenzen
Tools
Softwareentwicklung
Problemfelder
Lernroboter
Implementieren
Unterrichtsmethoden
Lernerfolg- und Leistungsbewertung
Worked Example
Inhalte
Lernziele
Programmverstehen
Digitale Welt
Debugging
Cognitive Load
STREAM
Block-basierte Sprachen
Computational Thinking
Programmieren
Use-Modify-Create
Programmierkonzepte
Computational Notebooks
Softwareprojekte
Lehrpläne

Kapitelübersicht

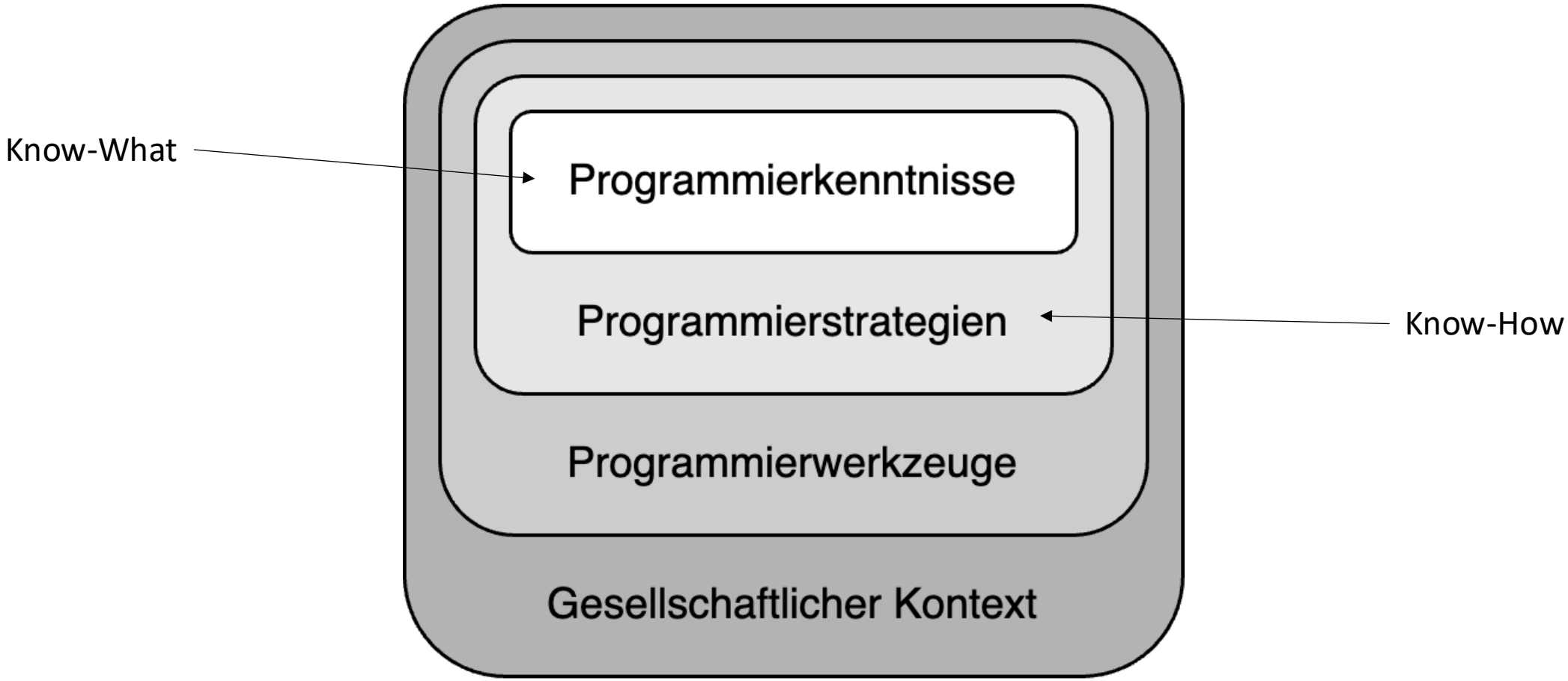
1. Einführung
2. **Ziele und Inhalte**
3. Lerntheorien
4. Unterrichtsplanung
5. Programmiersprachen und Umgebungen
6. Programmieren im Team
7. Lernerfolgskontrolle und Leistungsbewertung


Aims and objectives
Towards which goals are they learning?
Content
What are they learning?
Assessment
How is their learning assessed?
Time
When are they learning?
Rationale
Why are they learning?
Learning activities
How are they learning?
Location
Where are they learning?
Teacher role
How is the teacher facilitating their learning?
Grouping
With whom are they learning?
Materials and resources
With what are they learning?

Was muss ein guter Programmierer Ihrer Meinung nach können und wissen?



Programmierkompetenz

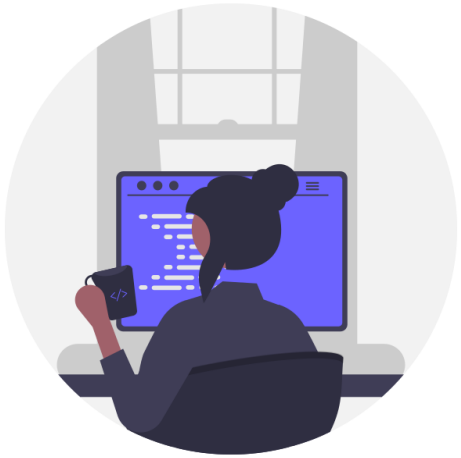


Programmierkenntnisse und Strategien

(Davies, 1993)

- Programmierkenntnisse *(Knowledge)*
  - Deklaratives Wissen
  - „e.g. being able to state how a ‚for' loop works"
- Programmierstrategien *(Strategies)*
  - Angewandtes Wissen (Prozedurales Wissen)
  - „e.g. using a ‚for' loop appropriately in a program"

Davies, S. P. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39(2), 237–267. [DOI]



Programmierkenntnisse

Know-What (Faktenwissen)

Beispiele für Programmierkonzepte

(vgl. Schulte & Bennedsen, 2006)

- Anweisungen
- Operatoren
- Kontrollstrukturen
  - Bedinge Anweisungen (if, else)
  - Schleifen (for, while)
- Variablen & Datentypen
  - Einfache Datentypen
  - Komplexe Datentypen
- Datenstrukturen
  - (Verkette) Listen / Arrays
  - Dictionaries
- Algorithmen
- Funktionen
  - Rekursion
- Objekte und Klassen
  - Attribute
  - Methoden
  - Vererbung
- Namensräume
- Entwurfsmuster
  - Observer
  - Model-View-Controller
- Kommentare
- Debugging
- …

Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? [DOI]

Was unterrichten Lehrekräfte in der Einführung in die Programmierung? *(Schulte & Bennedsen, 2006)*

- In einer 2006 durchgeführten Online-Umfrage wurden 457 Informatiklehrkräfte an Hochschulen und Schulen befragt (242 vollständige Antworten)
- Dänemark, Deutschland, USA und weitere Länder
- Wie **relevant** und **schwierig** sind die folgenden Themen in Einführungskursen zur Programmierung?

Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? [DOI]

Übung

- Ordnen Sie die folgenden Themen hinsichtlich **Relevanz** und **Schwierigkeitsgrad**

Rekursion

Laufzeitanalyse

Polymorphismus und Vererbung

Syntax

Datentypen

Problemlösestrategien

Parameter

Ethische Aspekte

Programmierbibliotheken

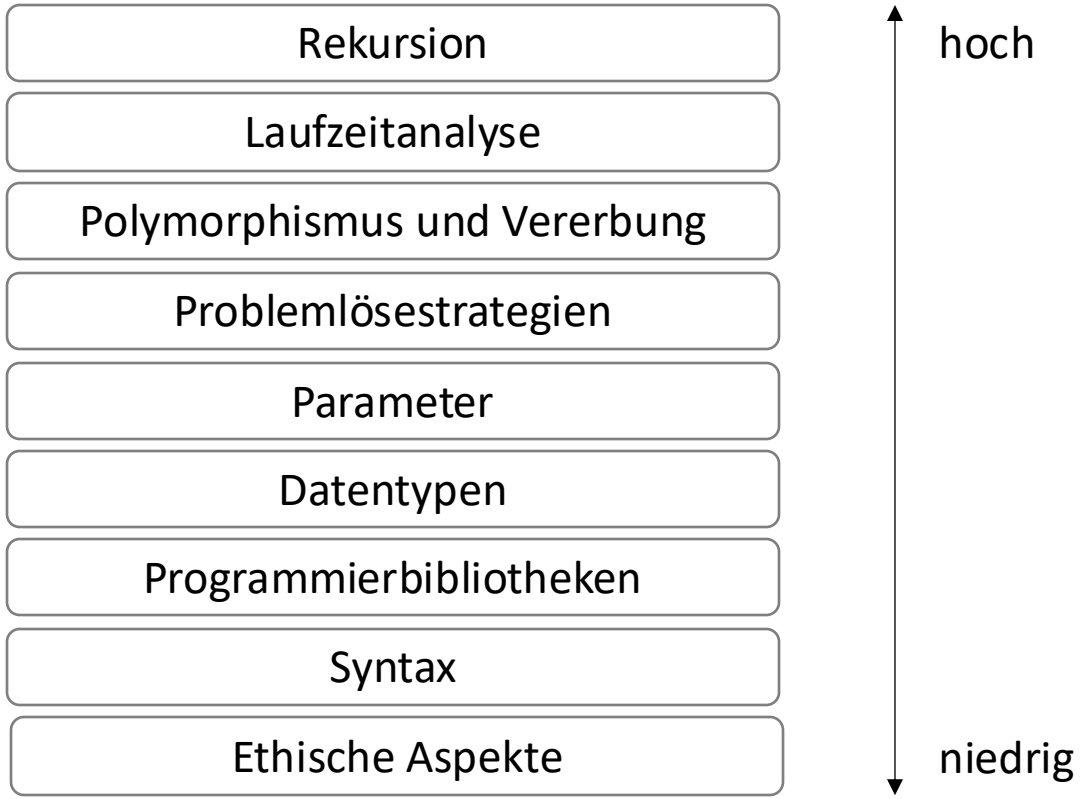
Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? [DOI]

Antworten der Lehrkräfte (Ausschnitt)

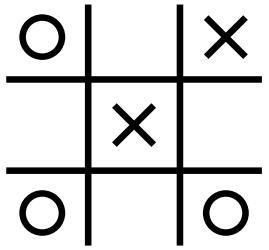
Relevanz

- Parameter
- Syntax
- Datentypen
- Problemlösestrategien
- Programmierbibliotheken
- Polymorphismus und Vererbung
- Rekursion
- Ethische Aspekte
- Laufzeitanalyse

Schwierigkeitsgrad



Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? [DOI]



Programmierstrategien

Am Beispiel: Rollen von Variablen

Rollen von Variablen I

In Programmen auf Anfängerniveau *(Sajaniemi, 2002)*

- In einer Studie von Sajaniemi (2002) wurden 109 von Experten geschriebene Programme auf Anfängerniveau untersucht
- Dabei wurden 9 Variablenrollen identifiziert, die 98,9 % aller verwendeten Variablen abdecken
- Die Rollen sind Beispiele für die Anwendung von Programmierstrategien (Know-How)

Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. [DOI]

Rollen von Variablen II

In Programmen auf Anfängerniveau *(Sajaniemi, 2002)*

- Fixed Values (Konstante)
  - Variable, deren Wert sich nach der Initialisierung nicht ändert
- Stepper (Zähler)
  - Variable, die eine Reihe von Werten durchläuft, die **nicht** von den Werten anderer nicht konstanter Variablen abhängen
- Follower
  - Variable, die eine Reihe von Werten durchläuft, die von den Werten eines Steppers oder eines anderen Followers abhängen, aber nicht von anderen nicht konstanten Variablen (z.B. Attribut eines Steppers)
- Most-recent Holder
  - Variable, die den letzten Wert enthält, der beim Durchlaufen einer Reihe von Werten angetroffen wird
- Most-wanted Holder
  - Variable, die den „besten“ Wert enthält, der beim Durchlaufen einer Reihe von Werten gefunden wurde (z.B. Maximum)

Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. [DOI]

Rollen von Variablen III

In Programmen auf Anfängerniveau *(Sajaniemi, 2002)*

- Gatherer (Sammler)
  - Variable, die eine Reihe von Werten kumuliert (z.B. Laufsumme)
- One-way Flag
  - Boolean, welcher effektiv nur ein einziges Mal geändert werden kann (z.B. eine Variable, die angibt, ob das Ende der Eingabe erreicht wurde)
- Temporary
  - Variable, die den Wert einer anderen Variablen nur für eine sehr kurze Zeit speichert
- Organizer
  - Array, das nur dazu verwendet wird, seine Elemente nach der Initialisierung neu anzuordnen (z.B. ein Array, das zum Sortieren von Eingabewerten verwendet wird)

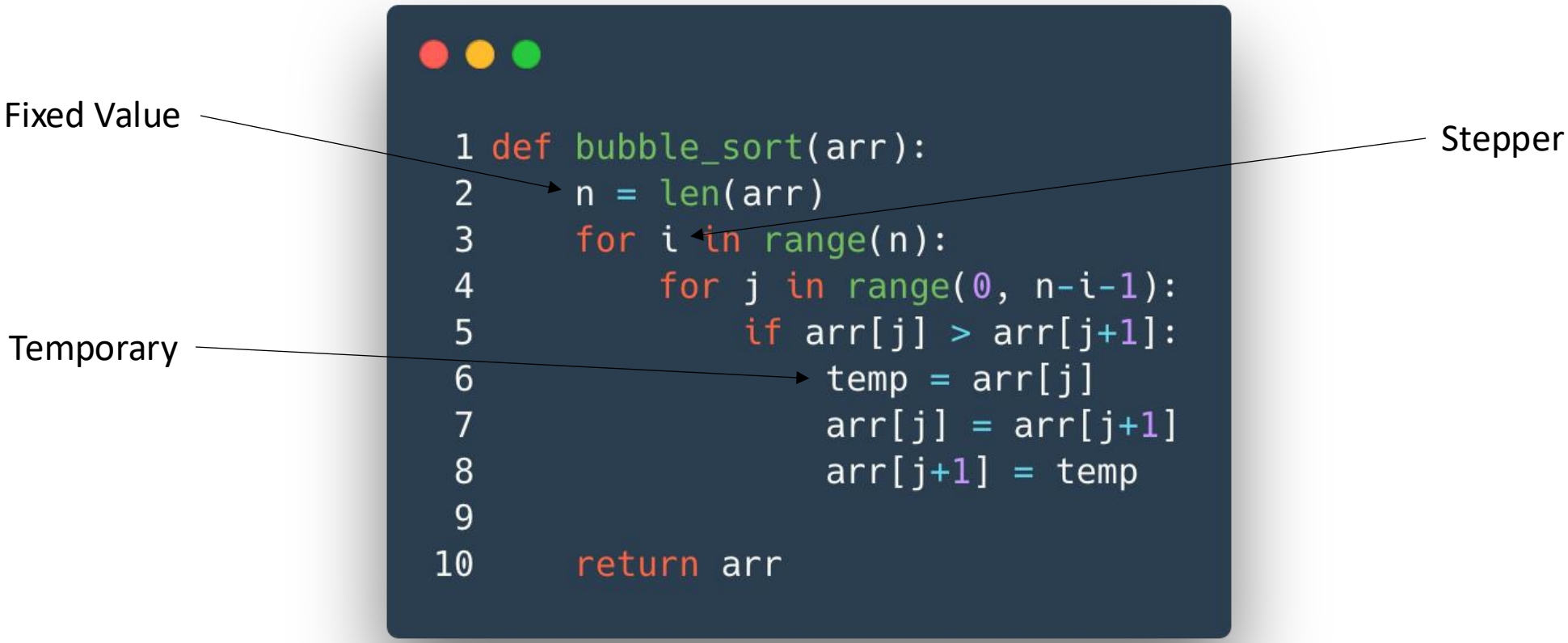
Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. [DOI]

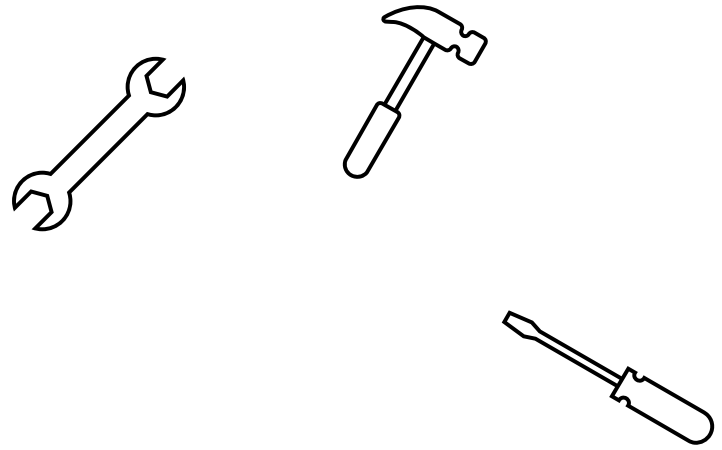
Beispiel: Bubble Sort

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                temp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = temp

    return arr
```

Beispiel: Bubble Sort



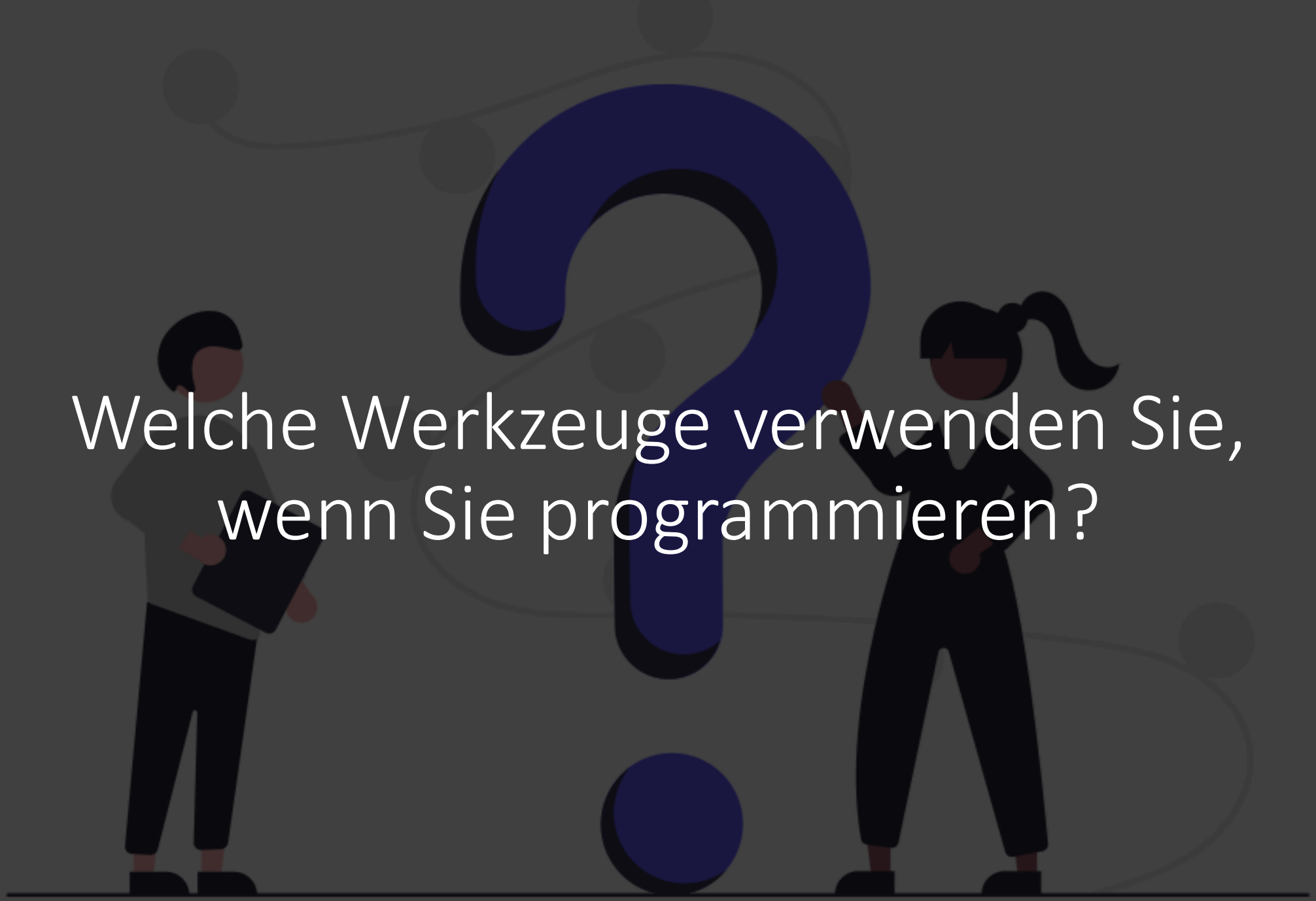


Programmierwerkzeuge

Entwicklungsumgebungen

KI-Assistenten

Debugger



Welche Werkzeuge verwenden Sie, wenn Sie programmieren?

Programmierwerkzeuge

Beispiele

- Programmiersprachen (z.B. Python, Java)
- Programmierumgebungen (z.B. Eclipse, PyCharm, Scratch)
- Bibliotheken, Frameworks, APIs
- Informationsquellen (Dokumentation, StackOverflow)
- Versionsverwaltung (Git)
- Künstliche Intelligenz (z.B. Microsoft Copilot, ChatGPT)
- Projektmanagement (z.B. SCRUM, Kanban-Board)
- Peripherie (Maus, Tastatur)

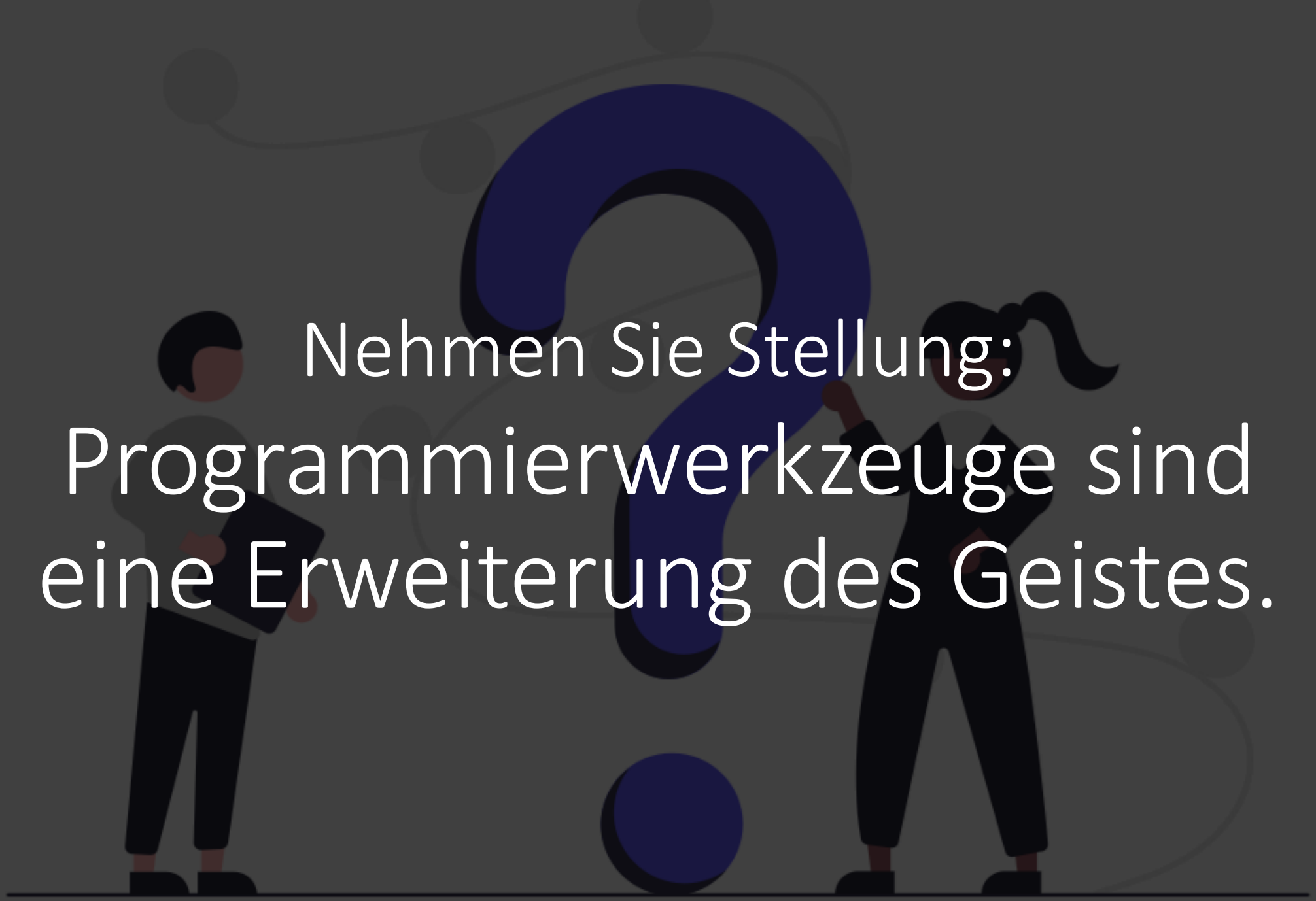
→ Ein Programmierer ist in hohem Maße auf solche Werkzeuge angewiesen!

„Standing on the shoulders of giants“

Extended Mind Thesis

- Embodiment Thesis *(Shapiro & Spaulding, 2024)*
  - **“**Many features of cognition are embodied in that they are deeply dependent upon characteristics of the physical body of an agent, such that the agent's beyond-the-brain body plays a significant causal role, or a physically constitutive role, in that agent's cognitive processing.”
- Die These besagt, dass einige Objekte in der äußeren Umgebung Teil eines kognitiven Prozesses sein können und auf diese Weise als Erweiterungen des Geistes selbst fungieren.

Shapiro, L., & Spaulding, S. (2024). Embodied cognition. In E. N. Zalta & U. Nodelman (Hrsg.), *The Stanford encyclopedia of philosophy* (Summer 2024)



Nehmen Sie Stellung:

Programmierwerkzeuge sind eine Erweiterung des Geistes.

Gesellschaftlicher Kontext

Programmierung als Kulturtechnik & Ingenieurwissenschaft

Ethische Aspekte der Programmierung

Die drei Traditionen der Informatik

(Tedre und Apiola, 2013)

- Die Informatik als Disziplin hat sich aus drei unterschiedlichen, miteinander verflochtenen Traditionen entwickelt
- Jeder dieser drei Traditionen liegen unterschiedliche Konzepte und Prinzipien zugrunde

Mathematische Tradition	**Ingenieurwissenschaftliche Tradition**	**Wissenschaftliche Tradition**
Annahmen	Aktionen	Beobachtungen
Axiome und Theoreme	Prozesse, Regeln und Heuristiken	Modelle, Theorien und Gesetze
Analytisch; deduktiv	Empirisch; konstruktiv	Empirisch; deduktiv und induktiv
Befasst sich mit Strukturen	Befasst sich mit Prozessen	Befasst sich mit Ursachen
Transformationen zwischen abstrakten Ideen	Konkretisierungen von abstrakten Ideen	Verallgemeinerungen aus bestimmten Erkenntnissen
"Publish or perish"	"Demo or die"	"Publish or perish"
Macht selten Behauptungen, die nicht bewiesen werden können	Muss in der Lage sein, mit sehr wenig Informationen zu agieren	Zögert, Behauptungen aufzustellen, wenn nicht genügend Informationen vorhanden sind

Tedre, M., & Apiola, M. (2013). Three computing traditions in school computing education.

Programmieren als Ingenieurstätigkeit

- Programmieren ist eine Ingenieurstätigkeit
  - Programmierer entwickeln und konstruieren technische, digitale Artefakte (Quelltext und Programm)
  - Diese werden von anderen Menschen genutzt
  - Der Programmierer trifft Designentscheidungen, die Auswirkungen auf den Benutzer und das System haben
  - Die Vorstellungen und Werte des Programmierers fließen in das Artefakt
- Der Programmierer muss dazu sowohl über die technische Umsetzung (Architektur) als auch über die gesellschaftlichen Implikationen (Relevanz) nachdenken

O'neil, C. (2017). *Weapons of math destruction: How big data increases inequality and threatens democracy*.

Duale Natur digitaler Artefakte

Architektur

- Analytische Perspektive
- Empirisch beobachtbare und objektiv-messbaren Eigenschaften des digitalen Artefakts
- Struktur, Mechanismen
- “How does it work?”

Relevanz

- Interpretative Perspektive
- Beabsichtigten Ziele, Zwecke und gewünschten Wirkungen des digitalen Artefakts
- Intention, Bedeutung, Funktion, Interpretation
- “What is it for?”

Beide Perspektiven sind untrennbar miteinander verbunden.

- Architektur ohne Relevanz ist buchstäblich bedeutungslos.
- Ohne Architektur ist keine Interpretation möglich

Schulte, C., & Budde, L. (2018). A Framework for Computing Education: Hybrid Interaction System: The Need for a Bigger Picture in Computing Education.

Ethische Aspekte der Programmierung im IU

(Fiesler et al., 2021)

- Beispiel (Personalisierte Werbung):
  - Aufgabenstellung
    - Ihre Aufgabe ist es, ein Programm zu schreiben, das entscheidet, welche Werbung einer Person auf einer Social-Media-Plattform angezeigt werden soll. Das Programm für personalisierte Werbung fragt den Benutzer nach Informationen und gibt dann einen Text zurück, der die Werbung auf der Grundlage der Eingaben des Benutzers beschreibt.
  - Informatische Konzepte
    - Bedingte Anweisungen und Verzweigung

Fiesler, C.; Friske, M.; Garrett, N.; Muzny, F.; Smith, J. J. & Zietz, J. (2021). Integrating Ethics into Introductory Programming Classes. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 1027–1033. [DOI]

Ethische Aspekte der Programmierung im IU

(Fiesler et al., 2021)

- Beispiel (Hochschulzulassungen):
  - Aufgabenstellung
    - Wie Sie wissen, werden im Rahmen des Zulassungsverfahrens für Hochschulen viele verschiedene Daten von Studieninteressierten zur Entscheidungsfindung herangezogen. Da die Zahl der Bewerber zunimmt, könnten sich die Hochschulen auf Algorithmen stützen, um auszuwählen, welche Bewerbungen intensiver von Menschen geprüft werden sollen. Ein Algorithmus könnte auf der Grundlage quantitativer Daten - wie GPA und SAT-Score - erste Empfehlungen aussprechen. […]. Ihre Aufgabe besteht darin, ein Programm zu erstellen, das eine Liste von Datenpunkten durchläuft und eine Empfehlung abgibt, welche Studieninteressenten wahrscheinlich die besten Kandidaten für eine Zulassung sind.
  - Informatische Konzepte
    - Listen
    - Einlesen und Speichern von Dateien

Fiesler, C.; Friske, M.; Garrett, N.; Muzny, F.; Smith, J. J. & Zietz, J. (2021). Integrating Ethics into Introductory Programming Classes. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 1027–1033. [DOI]

Exkurs
Lernziele im Kernlehrplan Informatik NRW (2014)

Inhaltsfelder

Auszug aus dem Kernlehrplan NRW, 2014
(Leistungskurs)

1. Daten und ihre Strukturierung
   - Objekte und Klassen
     - ermitteln bei der Analyse von Problemstellungen Objekte, ihre Eigenschaften, ihre Operationen und ihre Beziehungen (M)
     - dokumentieren Klassen (D)
     - implementieren Klassen in einer Programmiersprache auch unter Nutzung **dokumentierter Klassenbibliotheken** (I)
   - Datenbanken
     - implementieren ein relationales Datenbankschema als Datenbank (I)
2. Algorithmen
   - Analyse, Entwurf und Implementierung von Algorithmen
     - **analysieren und erläutern** Algorithmen und Programme (I)
     - modifizieren Algorithmen und Programme (I)
     - implementieren **iterative und rekursive Algorithmen** auch unter Verwendung von dynamischen Datenstrukturen (I),
     - **testen Programme systematisch** anhand von Beispielen und mithilfe von Testanwendungen (I).
   - Algorithmen in ausgewählten informatischen Kontexten
     - implementieren Operationen dynamischer (linearer oder nichtlinearer) Datenstrukturen (I)
     - implementieren und erläutern iterative und rekursive **Such- und Sortierverfahren** unterschiedlicher Komplexitätsklassen (Speicherbedarf und Laufzeitverhalten) (I)
     - entwickeln und implementieren Algorithmen und Methoden zur **Client-Server-Kommunikation** (I)

Auszug aus dem Kernlehrplan NRW, 2014
(Leistungskurs)

3. Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - nutzen die **Syntax und Semantik** einer Programmiersprache bei der Implementierung und zur Analyse von Programmen (I)
     - beurteilen die syntaktische Korrektheit und die Funktionalität von Programmen (A)
     - **interpretieren Fehlermeldungen** und **korrigieren den Quellcode** (I)
     - verwenden die Syntax und Semantik einer Datenbankabfragesprache, um Informationen aus einem Datenbanksystem zu extrahieren (I)
   - Scanner, Parser und Interpreter für eine reguläre Sprache
     - modellieren und implementieren Scanner, Parser und Interpreter zu einer gegebenen regulären Sprache (I)
4. Informatiksysteme
   - Nutzung von Informatiksystemen
     - wenden didaktisch orientierte **Entwicklungsumgebungen** zur Demonstration, zum Entwurf, zur Implementierung und zum Test von Informatiksystemen an (I)

Auszug aus dem Kernlehrplan NRW, 2014
(Leistungskurs)

5. Informatik, Mensch und Gesellschaft

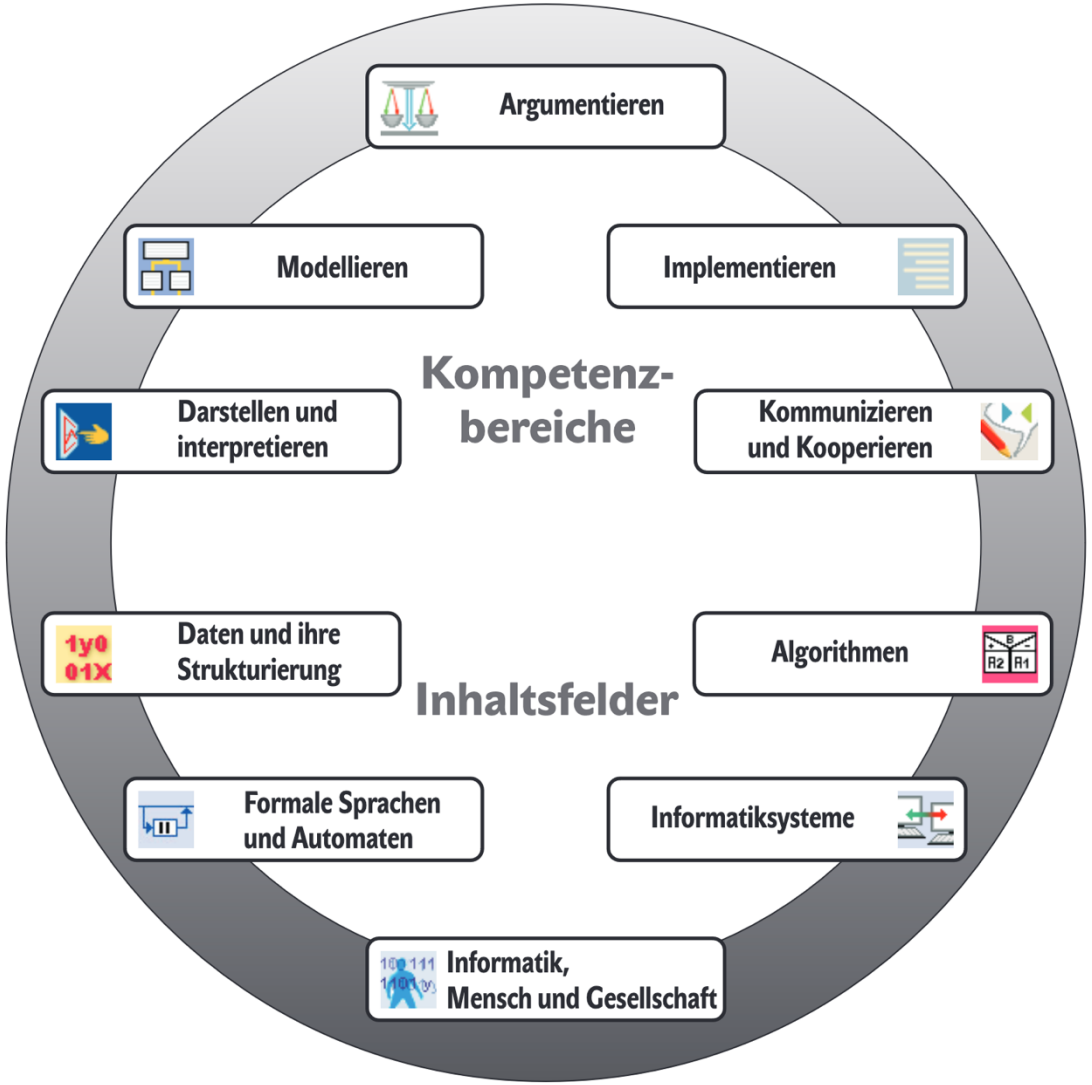
- Wirkungen der Automatisierung
  - Untersuchen und bewerten anhand von Fallbeispielen Auswirkungen des Einsatzes von Informatiksystemen sowie Aspekte der Sicherheit von Informatiksystemen, des Datenschutzes und des Urheberrechts (A)
  - untersuchen und bewerten Problemlagen, die sich aus dem Einsatz von Informatiksystemen ergeben, hinsichtlich rechtlicher Vorgaben, ethischer Aspekte und gesellschaftlicher Werte unter Berücksichtigung unterschiedlicher Interessenlagen (A)
- Grenzen der Automatisierung
  - untersuchen und beurteilen **Grenzen des Problemlösens** mit Informatiksystemen (A)

Zusammenfassung & Reflexion

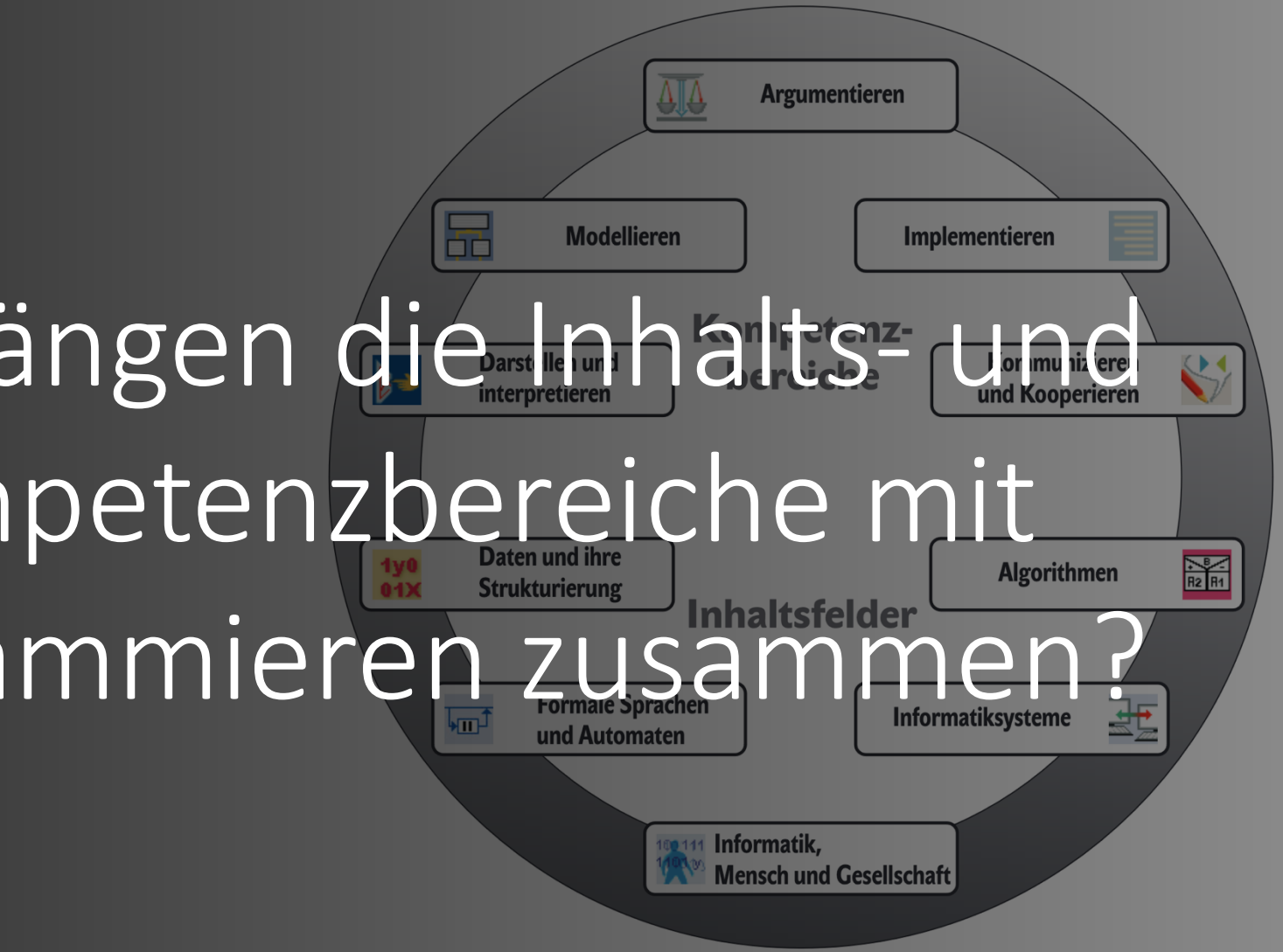
- Programmieren und Modellieren untrennbar miteinander verknüpft
  - Programmieren wird als Umsetzung des Modells aufgefasst
- Programmieren nimmt einen hohen Stellenwert ein (auch wenn der Begriff vermieden wird)
- Programmieren umfasst nicht nur Code schreiben (implementieren), sondern auch
  - Umgang mit Entwicklungsumgebungen
  - Testen, Debuggen und Analysieren von Code / Software
  - Verwendung von Bibliotheken / Lesen von Dokumentationen
  - Programmiersprachen
  - (Softwareprojekte)

Kompetenzerwartungen und inhaltliche Schwerpunkte

(Kernlehrplan Informatik, GOSt, NRW, 2014)



Wie hängen die Inhalts- und Kompetenzbereiche mit Programmieren zusammen?



Reflexion

- Die **digitale Welt** und die Informatik als Fachwissenschaft unterliegen einem **ständigen Wandel**
  - Neue Programmiersprachen, Technologien und Tools entstehen, während andere an Bedeutung verlieren
  - Lehrpläne haben Schwierigkeiten, diese dynamischen Trends zeitnah aufzugreifen und hinken hinterher (siehe ML, NLP oder LLM)
  - Schüler:innen werden im Laufe ihres Lebens mit Technologien konfrontiert, von denen wir heute noch nichts wissen oder die gerade erst entwickelt werden
- Wie können wir die Schüler:innen auf eine ungewisse Zukunft vorbereiten?
- Schüler:innen müssen die Fähigkeit entwickeln, sich **eigenständig in neue Technologien einzuarbeiten (Lebenslanges Lernen)**. Das beinhaltet unter anderem …
  - Die Fähigkeit, sich eigenständig in Programmiersprachen und die damit verbundenen Konzepte einzuarbeiten
  - Die Fähigkeit, sich mit neuen Technologien wie APIs, Bibliotheken, Frameworks und (Open Source) Projekten vertraut zu machen
  - Die Fähigkeit Beispielen und Tutorials für die Umsetzung eigener Projekte zu verwenden

Referenzen

- Crompton & Burke (2024). The Educational Affordances and Challenges of ChatGPT. In, TechTrend 68, 380 – 392. https://doi.org/10.1007/s11528-024-00939-0.
- Davies, S. P. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39(2), 237–267. https://doi.org/10.1006/imms.1993.1061.
- Du Boulay, B. (1986). Some difficulties of learning to program. In, Journal of Educational Computing Resarch 2 (1) (S. 57 – 73). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9.
- Fiesler, C.; Friske, M.; Garrett, N.; Muzny, F.; Smith, J. J. & Zietz, J. (2021). Integrating Ethics into Introductory Programming Classes. In, Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 1027 – 1033. https://doi.org/10.1145/3408877.3432510.
- Repenning, A.; Basawapatna, A. & Escherle, N. (2016). Computinal thinking tools. In, 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 218 – 222. https://doi.org/10.1109/VLHCC.2016.7739688.
- Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. In: Proceedings IEEE 2002 Symposium on Human Centric Computing Languages and Environments (S. 37 – 39). https://doi.org/10.1109/HCC.2002.1046340.

Referenzen

- Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? Proceedings of the second international workshop on Comupting education resarch, 17 – 28. https://doi.org/10.1145/1151588.1151593.
- Schulte, C., & Budde, L. (2018). A Framework for Computing Education: Hybrid Interaction System: The Need for a Bigger Picture in Computing Education. *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. https://doi.org/10.1145/3279720.3279733
- Turner, R., Falcone, M., Sharif, B., & Lazar, A. (2014). An Eye-Tracking Study Assessing the Comprehension of C++ and Python Source Code. Proceedings of the Symposium on Eye Tracking Research and Applications, 231–234. https://doi.org/10.1145/2578153.2578218.
- Wing, J. M. (2008). Computational thinking and thinking about computing. In Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences 366 (1881) (S. 3717 – 3725). https://doi.org/10.1098/rsta.2008.0118.

Illustrationen

- Illustrationen von Limpitsouni, K. unter freier Lizenz via https://undraw.co

FAIBLE.nrw

Beteiligte Hochschulen:

- RWTH-Aachen
- Westfälische Wilhelms-Universität Münster
- Universität Duisburg-Essen
- Universität Bonn
- Universität Paderborn
- Technische Universität Dresden
- Carl von Ossietzky Universität Oldenburg

ORCA.nrw
Das Landesportal für Studium und Lehre.

Ein Kooperationsvorhaben empfohlen durch die:
DIGITALE HOCHSCHULE NRW
INNOVATION DURCH KOOPERATION

gefördert durch:
Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen