

FAIBLE.nrw

Didaktik der Programmierung



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

Bildung

Kapitelübersicht

1. Einführung

2. Ziele und Inhalte

3. Lerntheorien

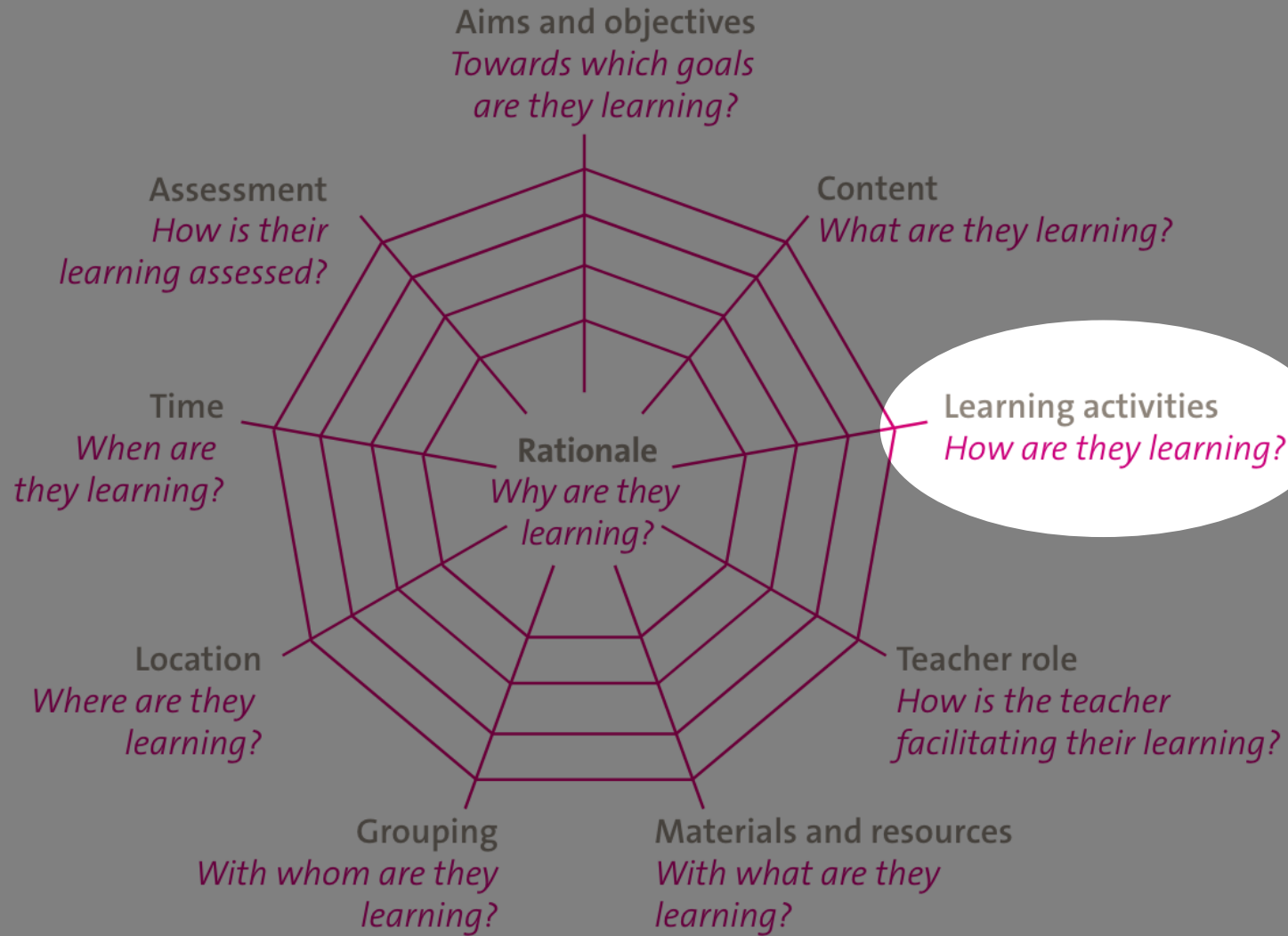
4. Unterrichtsplanung

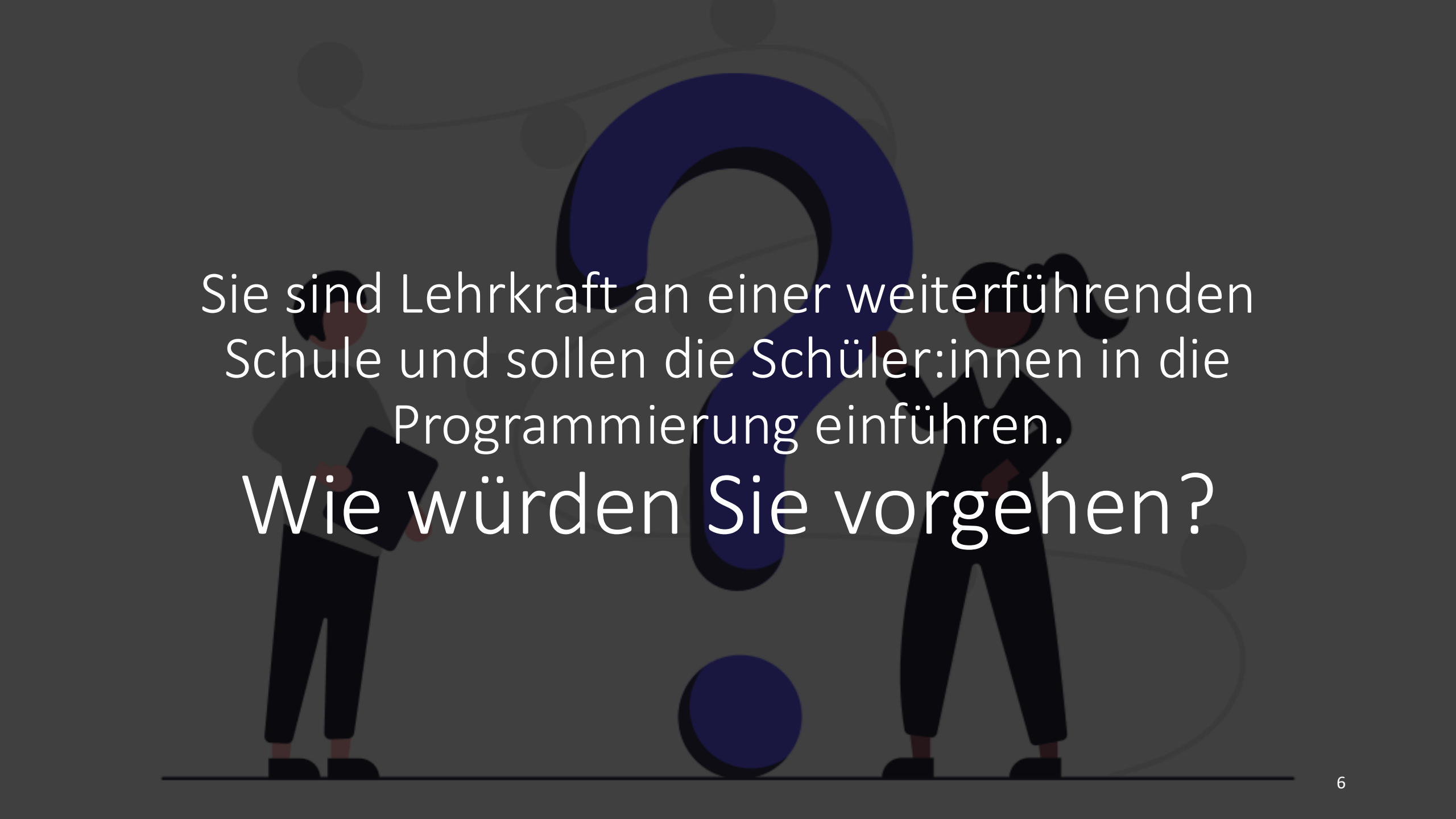
5. Programmiersprachen und Umgebungen

6. Programmieren im Team

7. Lernerfolgskontrolle und Leistungsbewertung





The background features a large, dark blue question mark. To the left of the question mark is a silhouette of a person holding a laptop. To the right is a silhouette of a person with their hands on their hips. The overall scene is set against a dark grey background with faint, light-colored circular patterns.

Sie sind Lehrkraft an einer weiterführenden
Schule und sollen die Schüler:innen in die
Programmierung einführen.

Wie würden Sie vorgehen?

Zehn Merkmale guten Unterrichts

(Meyer, 2003)

1. Klare Strukturierung des Lehr-Lernprozesses
2. Intensive Nutzung der Lernzeit
3. Stimmigkeit der Ziel-, Inhalts- und Methodenentscheidungen
4. Methodenvielfalt
5. Intelligentes Üben
6. Individuelles Fördern
7. Lernförderliches Unterrichtsklima
8. Sinnstiftende Unterrichtsgespräche
9. Regelmäßige Nutzung von Schüler-Feedback
10. Klare Leistungserwartungen und -kontrollen

Kapitelübersicht

- Ansätze zur Unterrichtsplanung

- Use-Modify-Create ← Computational Thinking (*siehe Kapitel 2*)
- PRIMM ← Umsetzung des Leseansatzes (*siehe Kapitel 3*)
- STREAM ← Objektorientierung, agile Softwareentwicklung

- Ansatz zur Reihenplanung

- 4C/ID Modell ← Cognitive Load Theory (*siehe Kapitel 3*)

„Not mine“ → „Mine“

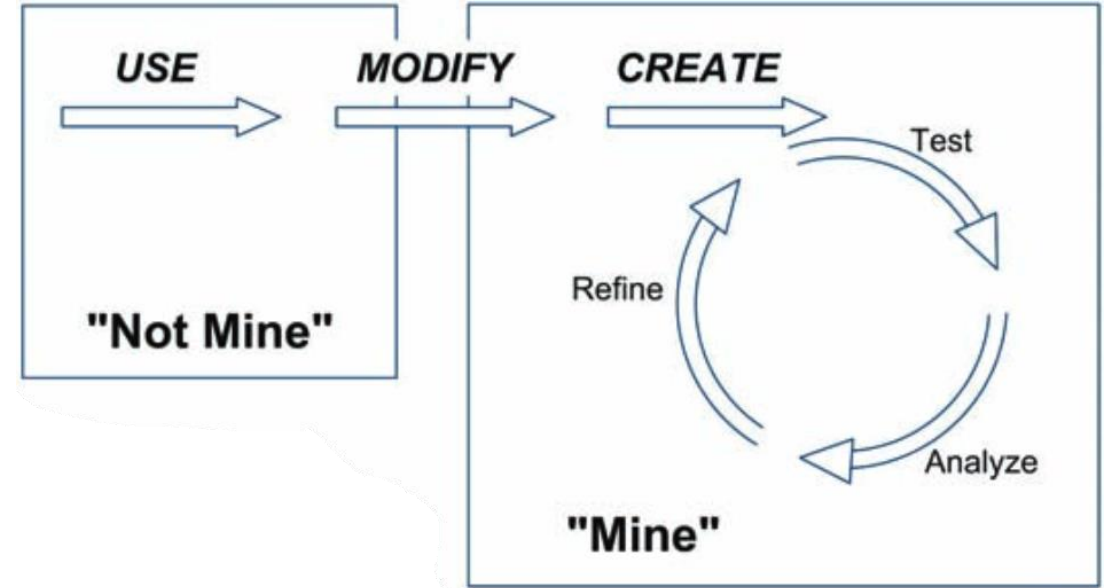
Use-Modify-Create

(Lee et al., 2011)

Use-Modify-Create

(Lee et al., 2011)

- Basiert auf dem Computational Thinking Ansatz (siehe Kapitel 2)
 - Abstraktion, Automatisierung, Analyse
- 3 Phasen
 - Use, Modify, Create
- Schwierigkeitsgrad steigt schrittweise an
- Mehrere Iterationen
- Ziel
 - Ein Maß an Herausforderung aufrechterhalten und gleichzeitig Ängste nehmen



Use-Modify-Create Learning Progression (Lee, et al., 2011)

Use-Modify-Create

(Lee et al., 2011)

1. **Benutzung** eines gegebenen Artefakts (Use)

- Schüler:innen sind Konsumenten
- Sie führen eine Simulation oder Programm aus, spielen ein Computerspiel, interagieren mit einem Roboter

2. **Modifikation** eines existierenden Artefakts (Modify)

- Veränderung der Parameter
 - Hinzufügen einer neuen Funktionalität
 - “what was once someone else’s becomes one’s own”
- } erfordert ein Verständnis der in dem Artefakt enthaltenen Abstraktion und Automatisierung

3. **Kreation** eines neuen Artefakts (Create)

- Basierend auf den Ideen und Erfahrungen aus den vorangegangenen Phasen wird ein eigenes, neues Artefakt entwickelt

Computational Thinking in der Praxis

(Lee et al., 2011)

- Beispiele für CT in drei Domänen (*entnommen aus Lee et al., 2011*)

	Abstraktion	Automatisierung	Analyse
Modellierung & Simulation	Auswahl von Merkmalen der realen Welt, die in ein Modell aufgenommen werden sollen	Die Simulation in diskreten Zeitschritten durchlaufen	Wurden die richtigen Abstraktionen vorgenommen? Spiegelt das Modell die Realität wider?
Robotik	Konzeption eines Roboters, der auf eine Reihe von Bedingungen reagiert	Programm prüft Sensoren zur Überwachung der Bedingungen	Gibt es Fälle, die nicht berücksichtigt wurden?
Spieleentwicklung	Spiele werden auf eine Reihe von Szenen mit Akteuren abstrahiert	Das Spiel reagiert auf Benutzeraktionen	Erhöhen die eingebauten Elemente den Spielspaß?

PREDICT > RUN > INVESTIGATE > MODIFY > MAKE

PRIMM

(Sentance et al., 2019)

PRIMM

(Sentance et al., 2019)

- **PRIMM** ist eine Unterrichtsmethode für den Einstieg in die Programmierung, die aus dem Leseansatz dem Use-Modify-Create Ansatz hervorgeht
- **Problem:** Programmieranfänger sollen im Informatikunterricht Programmcode schreiben bevor sie Programmcode lesen können
(Lister et al., 2004)
- **Schwerpunkte:**
 - Fokus auf dem **Lesen von Programmcode**
 - **Kollaboratives Arbeiten:** Gemeinsam über Code sprechen und diskutieren, Verwendung geeigneter Fachsprache *(Walqui, 2006)*
 - **Verringerung der kognitiven Belastung** (Cognitive Load) durch das gestützte Arbeiten mit Beispielen
 - **Scaffolding:** Zone der proximalen Entwicklung *(Vygotsky, 1978)*
 - Beispielcode wird schrittweise zum „**eigenen Code**“

Sentance, S.; Waite, J., & Kallia, M. (2019). Teaching Computer Programming with PRIMM: A Sociocultural Perspective.

Lister, R. et. al. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers.

Vygotsky, L. S., & Cole, M. (1978). Mind in society: Development of higher psychological processes.

Walqui, A. (2006). Scaffolding Instruction for English Language Learners: A Conceptual Framework.

Die fünf Phasen

- Eine Unterrichtseinheit wird dabei in **fünf Phasen** unterteilt:
 1. **Predict:** Die Schüler:innen diskutieren über ein Programm und stellen eine Vorhersage über dessen Ausgabe auf
 2. **Run:** Die Schüler:innen führen das Programm aus, um ihre Vorhersage zu überprüfen
 3. **Investigate:** Die Lehrkraft stellt diverse Aufgabe, um die Struktur des Codes zu erforschen; dazu gehören Aktivitäten wie Nachverfolgen des Programmablaufs (Tracing), Erklären, Kommentieren und Debuggen
 4. **Modify:** Die Schüler:innen modifizieren das Programm, um dessen Funktionalität zu ändern oder zu erweitern. Die Schwierigkeit sollte dabei zunehmen. Der Code wird mit jeder Änderung ein Stück mehr zum „eigenen Code“
 5. **Make:** Die Schüler:innen erstellen ein neues Programm und nutzen dabei die Struktur des Beispiels, um ein anderes Problem zu lösen

Evaluation PRIMM

(Sentance et al., 2019)

- Auswirkungen auf Programmiererfolg
 - Kontroll- und Versuchsgruppe (Traditionell versus PRIMM)
 - Versuchsgruppe schnitt signifikant besser ab in Testfragen (n=493, $r = .13$)
- Rückmeldung der Lehrkräfte
 - Routine (Code lesen, diskutierten, schreiben) führt zu einem besseren Verständnis und gesteigertem Selbstvertrauen der Schüler:innen
 - Leistungsschwächere und -stärkere Schüler:innen profitieren gleichermaßen
 - Phase *Investigate* zu schwierig für Schüler:innen (Anpassung an ZpE)
 - Betonung der Fachsprache fördert präzise Kommunikation
 - Mehr Planungssicherheit (Leitfaden)

Stubs > Tests > Representation > Evaluation > Attributes > Methods

STREAM

(Caspersen und Kolling, 2009)

STREAM

(Caspersen und Kolling, 2009)

- **STREAM**

- Vereinfachte Variante eines vollständigen agilen Softwareentwicklungsprozesses
- Richtet sich an Programmieranfänger:innen
- Speziell für die objektorientierte Programmierung
- Schrittweise Verbesserung der Software

- **Ziel**

- Schnelleres und effektiveres Lernen
- Grundlage für die Vertiefung fortgeschrittener Aspekte der Softwaretechnik

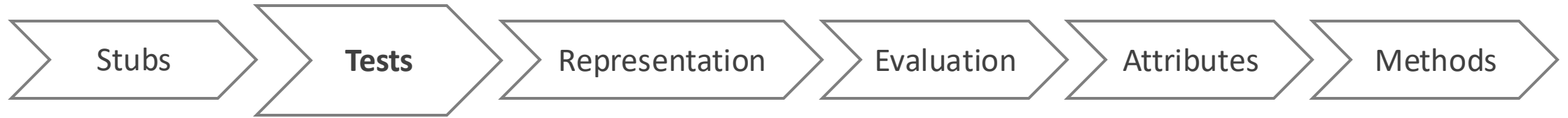


Caspersen, M. E. & Kolling, M. (2009). STREAM: A First Programming Process. [\[DOI\]](#)



- Klassengerüst erstellen

```
1 class Customer:
2     def __init__(self, name, wallet):
3         pass
4
5     def order(self, item, price):
6         pass
7
8     def pay(self):
9         pass
10
```



- Testfälle definieren

```
1 customer = Customer("Alice", 100)
2 customer.order("Shirt", 20)
3 customer.order("Jeans", 50)
4 customer.order("Shoes", 80)
5 assert(len(customer.cart), 3)
```



- Alternative Repräsentationen auflisten und evaluieren
 - R_1 : Einkaufswagen als Liste
 - R_2 : Einkaufswage als Dictionary

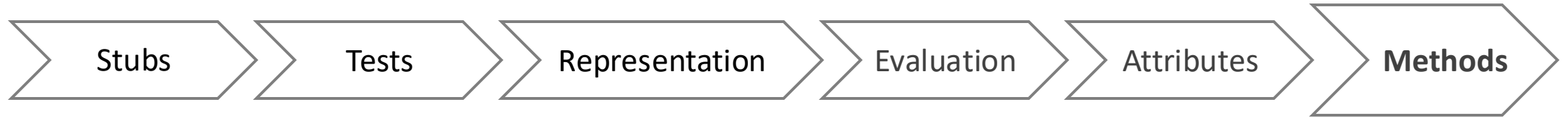
	R_1	R_2
order	niedrig	mittel
pay	niedrig	hoch

Programmieraufwand



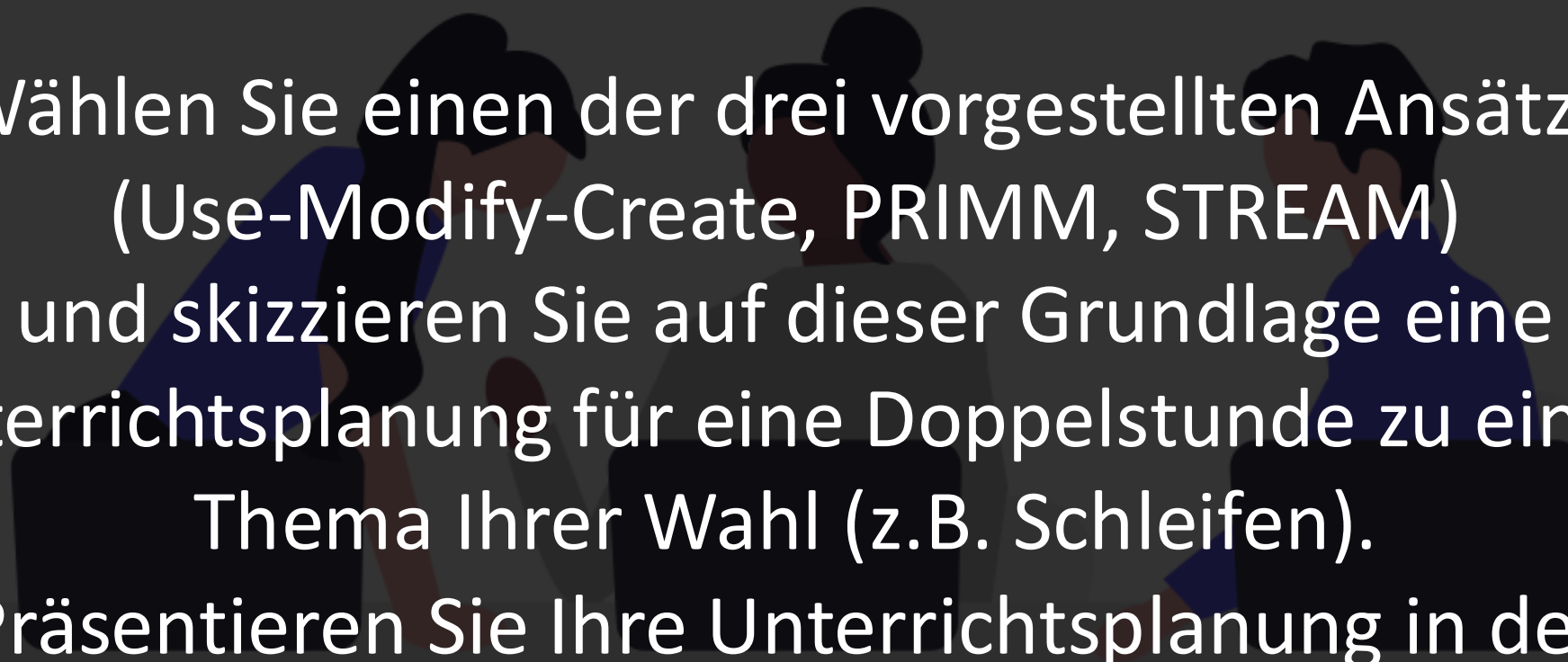
- Klassenattribute festlegen

```
1 class Customer:
2     def __init__(self, name, wallet):
3         self.name = name
4         self.wallet = wallet
5         self.cart = []
```



- Methoden füllen

```
1 class Customer:
2     def __init__(self, name, wallet):
3         self.name = name
4         self.wallet = wallet
5         self.cart = []
6
7     def order(self, item, price):
8         if self.wallet >= price:
9             self.cart.append((item, price))
10            print(f"{self.name} added {item} to cart.")
11        else:
12            print("Insufficient funds to add item to cart.")
13
14    def pay(self):
15        total_price = sum(item[1] for item in self.cart)
16        if self.wallet >= total_price:
17            self.wallet -= total_price
18            print(f"{self.name} paid ${total_price} for the items.")
19            self.cart = []
20        else:
21            print("Insufficient funds to complete the purchase.")
```



Wählen Sie einen der drei vorgestellten Ansätze
(Use-Modify-Create, PRIMM, STREAM)
und skizzieren Sie auf dieser Grundlage eine
Unterrichtsplanung für eine Doppelstunde zu einem
Thema Ihrer Wahl (z.B. Schleifen).
Präsentieren Sie Ihre Unterrichtsplanung in der
nächsten Sitzung im Plenum.

4C/ID Modell

Unterrichtsreihenplanung

Cognitive Load & Instructional Design

4C/ID Modell

(Van Merriënboer, et al., 2002)

- Four-Component (4C) Instructional Design (ID)
- Modell zur systematischen und effizienten Vermittlung komplexer Kompetenzen und Wissensgebiete (z.B. Programmierung)
- Charakteristiken
 - Realitätsnahe Aufgabe
 - Klare Lernziele
 - Gezielte Unterstützung der Lernenden
 - Schrittweiser Aufbau von Fachwissen
- Vier Komponenten

Die vier Komponenten (4C)

(Van Merriënboer, et al., 2002)

1. Lernaufgaben (Learning Tasks)

- Konkrete, authentische Aufgaben, die dem Lernenden zur Verfügung gestellt werden, um den Aufbau von Schemata zu fördern (*siehe Cognitive Architecture Kapitel 3*)
- Abstraktion vom Konkreten (Induktion)

2. Lernhilfen (Supportive Information)

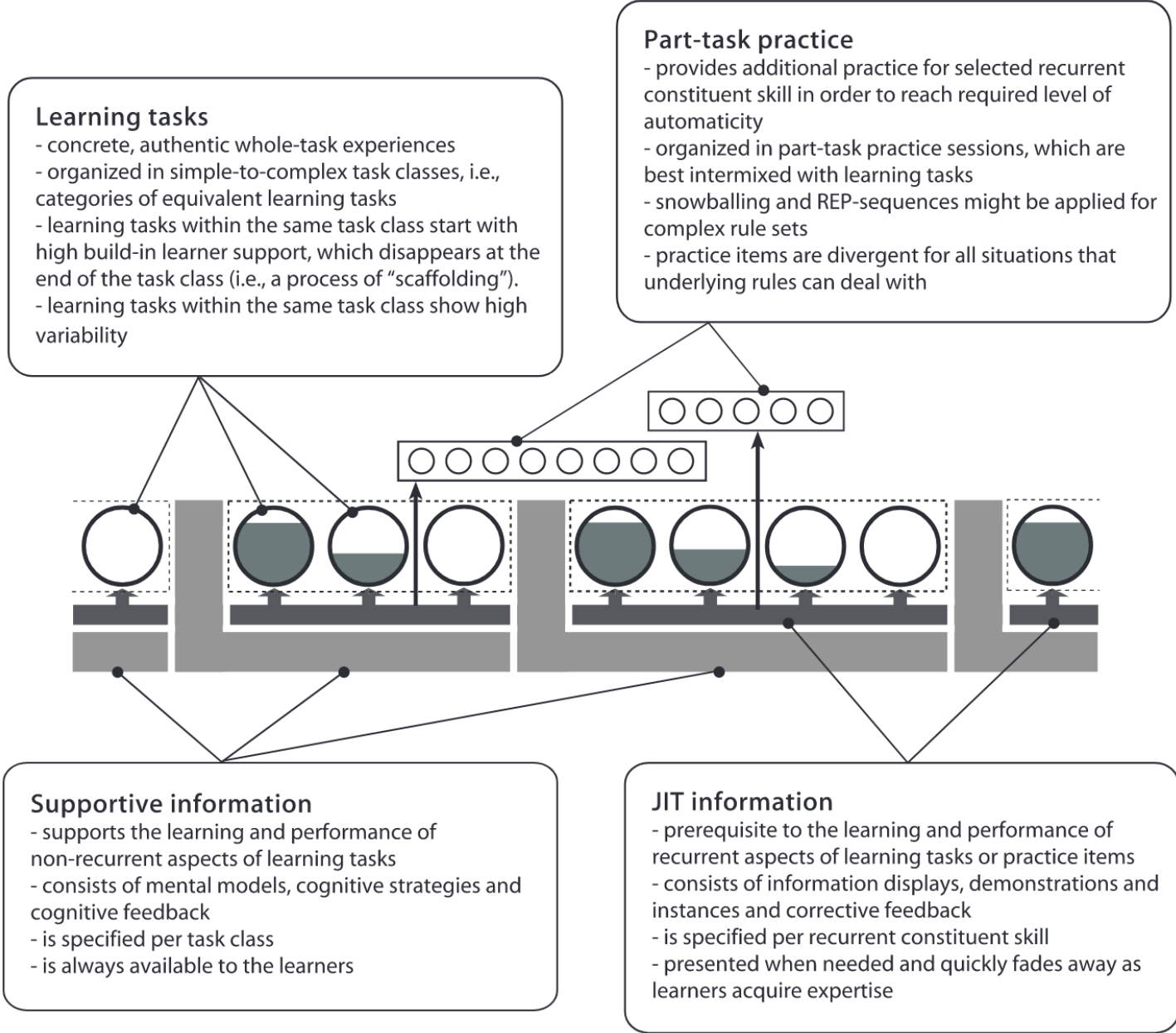
- Informationen, die das Lernen und die Ausführung von nicht wiederkehrenden Aspekten der Lernaufgaben unterstützen
- Brücke zwischen dem Vorwissen und den Lernaufgaben
- Elaboration: Verbesserung von Schemata durch die Herstellung von Beziehungen zwischen neuen Elementen und dem, was die Lernenden bereits wissen (vgl. Konstruktivismus)

3. Just-in-Time Informationen (Procedural Information)

- Informationen, die Voraussetzung für das Erlernen und Ausführen wiederkehrender Aspekte von Lernaufgaben sind
- Einbettung von prozeduralen Informationen in Regeln (Herausbildung von Regeln)

4. Übungsaufgaben (Part-task Practice)

- Übungsaufgaben, die den automatisierten Einsatz von Regeln für wiederkehrende Aspekte der Fähigkeit fördern.
- Synthese und Festigung

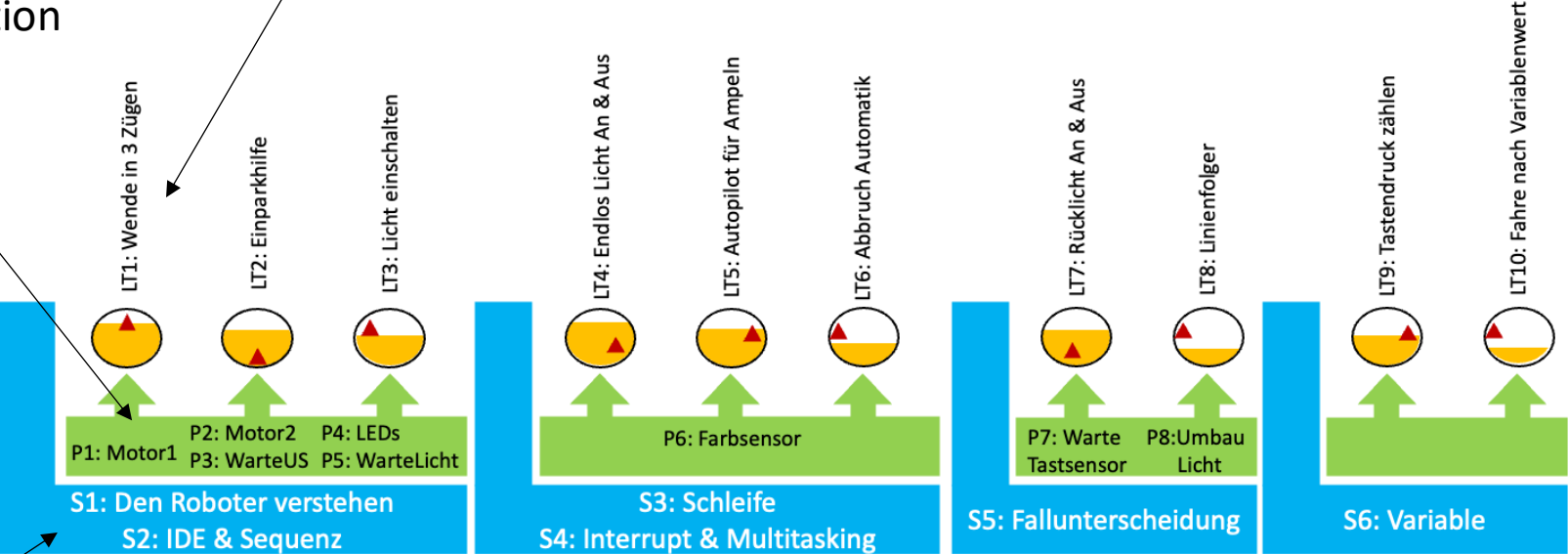


Beispiel

Learning Task

Lernpfad für die Grundlagen der Programmierung mit EV3-Robotern

Procedural Information



Eigene Abbildung

Supportive Information

Referenzen

- Caspersen, M. E. & Kolling, M. (2009): STREAM: A First Programming Process. In, ACM Transactions on Computing Education 9 (1), S. 1 – 29. <https://doi.org/10.1145/1513593.1513597>.
- Lee, I.; Martin, F.; Denner, J.; Coulter, B.; Allan, W.; Erickson, J.; Malyn-Smith, J. & Werner, L. (2011). Computational thinking for youth in practice. In ACM Inroads 2 (1), S. 32 – 37. New York: ACM. <https://doi.org/10.1145/1929887.1929902>.
- Lister, R.; Adams, E. S.; Fitzgerald, S.; Fone, W.; Hamer, J.; Lindholm, M.; McCartney, R.; Moström, J. E.; Sanders, K.; Seppälä, O.; Simon, B. & Thomas, L. (2004). A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In ACM SIGCSE Bulletin 36 (4), S. 119 - 150. <https://doi.org/10.1145/1041624.1041673>.
- Sentance, S.; Waite, J., & Kallia, M. (2019). Teaching Computer Programming with PRIMM: A Sociocultural Perspective. In Computer Science Education 29 (2 – 3), S. 136 – 176. <https://doi.org/10.1080/08993408.2019.1608781>.
- Van Merriënboer, J. J. G.; Clark, R. E. & De Croock, M. B. M. (2002). Blueprints for complex learning: The 4C/ID-model. In Educational Technology Research and Development 50 (S. 39 – 61. <https://doi.org/10.1007/BF02504993>.
- Vygotsky, L. S., & Cole, M. (1978). Mind in society: Development of higher psychological processes. Harvard university press. <https://doi.org/10.2307/j.ctvjf9vz4>.

Referenzen

- Walqui, A. (2006). Scaffolding Instruction for English Language Learners: A Conceptual Framework. In International Journal of Bilingual Education and Bilingualism 9 (2), S. 159 – 180. <https://doi.org/10.1080/13670050608668639>.
- Meyer, H. (2003). Zehn merkmale guten unterrichts. Empirische Befunde und didaktische Ratschläge. Pädagogik, 10, 36–43.

Illustrationen

- Illustrationen von [Limpitsouni, K.](#) unter freier [Lizenz](#) via <https://undraw.co>

Das vorliegende Gesamtwerk wurde im Rahmen des Projektes FAIBLE.nrw von der Universität Paderborn und der Universität Bonn erstellt und ist unter der (CC BY 4.0) - Lizenz veröffentlicht. Ausdrücklich ausgenommen von dieser Lizenz sind alle Logos! Weiterhin kann die Lizenz einzelner verwendeter Materialien, wie gekennzeichnet, abweichen. Nicht gekennzeichnete Bilder sind entweder gemeinfrei oder selbst erstellt und stehen unter der Lizenz des Gesamtwerkes (CC BY 4.0).

Sonderregelung für die Verwendung im Bildungskontext:

Die CC BY 4.0-Lizenz verlangt die Namensnennung bei der Übernahme von Materialien. Da dies den gewünschten Anwendungsfall erschweren kann, genügt dem Projekt FAIBLE.nrw bei der Verwendung in informatikdidaktischen Kontexten (Hochschule, Weiterbildung etc.) ein Verweis auf das Gesamtwerk anstelle der aufwändigeren Einzelangaben nach der TULLU-Regel. In allen anderen Kontexten gilt diese Sonderregel nicht!

Das Werk ist Online unter <https://www.orca.nrw/> verfügbar.



<https://creativecommons.org/licenses/by/4.0/deed.de>

