PRIMM

Predict	Run	Investigate	Modify	Make

Jonathan Pfeiffer
Wilhelm-Röpke-Schule Ettlingen

Gliederung

- Problemstellung
- PRIMM
  - Use-Modify-Create-Ansatz
  - Dialog & Sprache beim Lernen
  - Tracing + Explaining = Writing
  - Block Modell
  - Phasen [Predict, Run, Investigate, Modify, Make]

Problemstellung

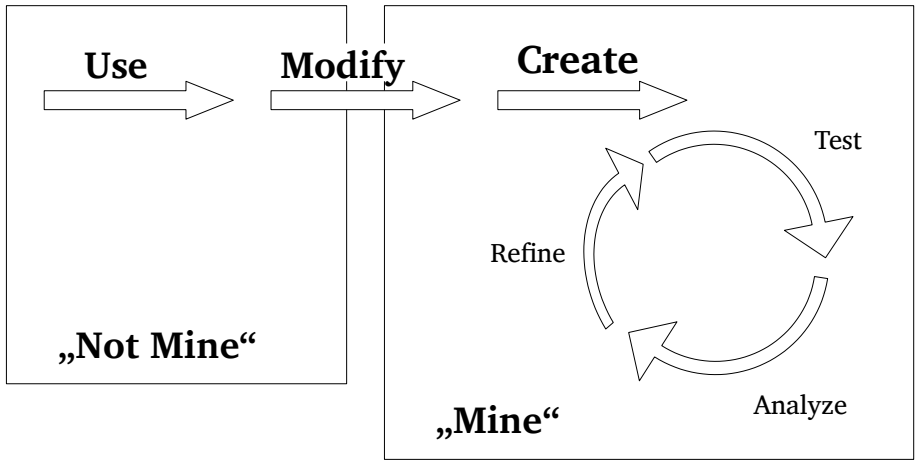
Programmieren lernen fällt vielen Anfängern schwer
Neben Syntax & Semantik haben Programmieranfänger Schwierigkeiten die einzelnen Teile zusammenzufügen und ein Programm zu schreiben (Robinson et al. 2003)
Code schreiben ist besonders schwierig für Programmieranfänger (Qian & Lehman 2017)
Auch Code Tracing ist für viele Programmieranfänger herausfordernd (Vainio & Sajaniemi 2007)
Kognitive Belastung: • Manche Instruktionsdesigns setzen ein Arbeitsgedächtnis voraus, das die Grenzen von Lernenden überschreitet (Sweller 1994) • Eigenständiges Arbeiten beim Programmieren verursacht eine höhere kognitive Belastung als in Partnerarbeit (Tsai et al. 2015)
Einsatz nicht adäquater Unterrichtsmethoden tragen ihren Teil dazu bei (Sentance & White 2023)

Relevanz

"Programming is a key part of computer science and computing; it is a skill that cannot sit separately from the theoretical components of computing. Rather, programming is the application of concepts that are often hard to understand until they are put into practice. Programming practice is not simply skill reinforcement; it is the route to understanding."

(Sentance & White 2023, 275)

Use-Modify-Create-Lehrkonzept



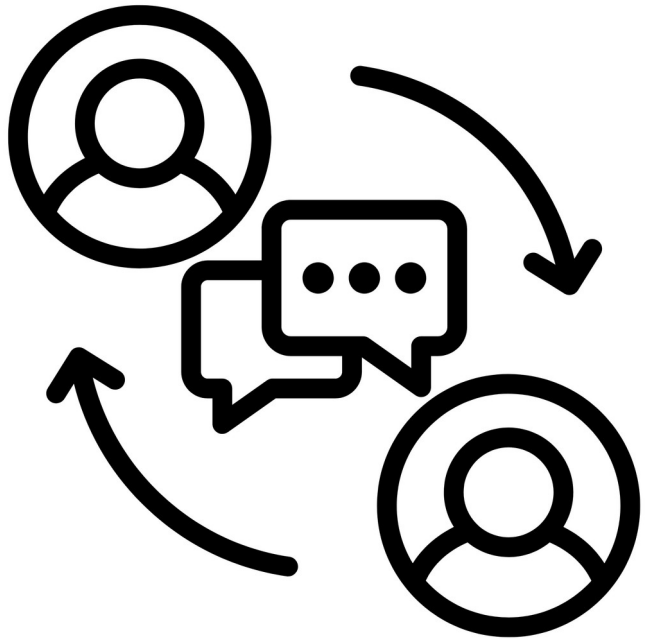
Lee et al. 2011, 35

- Programmieranfänger nutzen fertige Programme von Dritten („nicht meins“)
  - haben somit keine **emotionale Beziehung** zu diesen Programmen
  - Reduktion der **kognitiven Belastung**, da es zunächst um das Nutzen, Ausprobieren, Analysieren und Verstehen geht
  - Fokus kann z.B. auf **Fehlkonzepte** gelegt werden
  - In der Modify-Phase verändern die Lernenden Codezeilen oder ergänze diese
  - Mit erworbenen Fähigkeiten und Erfolgserlebnissen geht es dann in die abschließende Create-Phase und das Schreiben eigener Programme („meins“)

Lee et al. 2011

Dialog & Sprache beim Lernen

- Lernaufgaben in **Partnerarbeit**
- Ziel:
  - **Dialoge & Diskussionen** anregen
  - Lernende **artikulieren** und **versprachlichen** ihre Vorhersagen, Hypothesen, Vorstellungen, Analysen und Vorschläge
- Pair programming (driver & navigator) & Peer instruction (vote & re-vote, talk through, troubleshooting)
- Live Coding (thinking aloud, asking questions)



Sentance & Waite 2023

Welche programmierbezogenen Fähigkeiten gehen dem Schreiben von Programmcode voraus?

- *Code Tracing*
  - Ausführung eines Programms manuell und systematisch Schritt für Schritt zu verfolgen
- *Code erklären*
  - Programm(code) in eigenen Worten erklären
- Die Kombination aus *Code Tracing* und *Code erklären* korreliert stark mit *Code Writing*. 66% der Varianz in der Leistung beim Code schreiben kann durch die Leistung beim Code Tracing und Erklären von Code erklärt werden. (Lister 2022, 16)
- Programmieranfänger brauchen mindestens eine Genauigkeit von 50% beim Code Tracing, bevor sie eigenständig und selbstbewusst Code schreiben können (Venables et al. 2009)

PRIMM

Predict	Zusammenfassen was ein gegebener Code bei seiner Ausführung macht
Run	Code ausführen, um die Vorhersage zu testen
Investigate	Code Tracing, Kommentieren, Debuggen, Erklären
Modify	Code bearbeiten, um seine Funktionalität zu verändern
Make	Neues Programm schreiben

- Phasen können sich über mehrere Unterrichtsstunden erstrecken
- Predict, Run und Investigate können mehrmals wiederholt werden, bis Lernende für Modify und Make bereit sind

Sentance & Waite 2023

Predict

- Lernende bekommen einfachen, motivierenden Programmcode auf Papier bereitgestellt (und müssen ihn nicht kopieren)
- Lernende auffordern darüber zu sprechen und Vorhersagen zu machen
- Lernende sollten ihre Vorhersagen festhalten (Zeichnung, Ausgabe notieren)

Predict

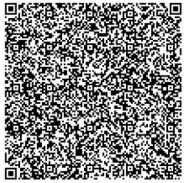
```python
def konversation():
    print("+++ Willkommen zu meinem Konversationsprogramm! +++ ")
    print()
    print("Magst du Radfahren? Antworte mit ja oder nein ")
    antwort = input()
    if antwort == "ja":
        print("Das ist gut - du wirst fit werden! ")
    else:
        print("Bestimmt magst du einen anderen Sport! ")
    print("Ciao.")

konversation()
```

Partnerarbeit
Lest den Programmcode und schreibt exakt auf was passiert, wenn der Programmcode ausgeführt wird.

Predict	**Run**	**Investigate**	**Modify**	**Make**

Run

- Lernende bekommen nun den ausführbaren Programmcode bereitgestellt, als Datei oder Link (pythontutor.com oder UUhistle)
- Lernende führen den Programmcode aus und testen ihre Hypothesen
- Besprechung der Ergebnisse im Plenum



Predict	**Run**	Investigate	Modify	Make

Investigate

- Didaktische Analyse, insb. zu Fehlkonzepten (siehe Übersicht bei Sorva (2012))
- Block Model (Schulte 2008) zur Orientierung bei der Planung nutzen
- Die Lernenden beantworten Fragen und absolvieren Aktivitäten zum Code, um das Verständnis zu vertiefen

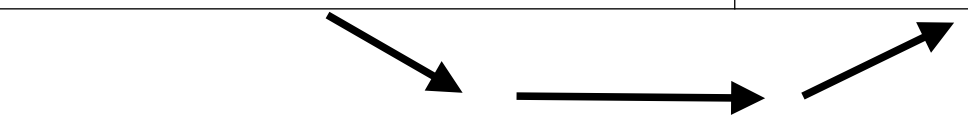
Predict	Run	**Investigate**	Modify	Make

Investigate – The Block Model

- Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching
- Fokus auf Lesen [Verstehen] (anstatt Schreiben) und Programmverstehen
- Zur Planung und Analyse von Programmier-Anfangsunterricht
- Nicht alle Zellen/Blöcke müssen Berücksichtigung finden und können in unterschiedliche Reihenfolge angeordnet werden
- Lernenden sollten Kompetenzen in jeder Zelle erwerben und zwischen den Zellen springen können
- Verstehensprozess kann durch die Lernumgebung beeinflusst werden (Brücken bauen)

Investigate – The Block Model

The Block Model (Schulte 2008)			
Macro structure	(Understanding) overall structure of the program text	Understanding the „algorithm“ of the program	Understanding the goal/the purpose of the program (in its contxt)
Relationships	References between blocks, e.g.: method calls, object creation, accessing data,...	sequence of method calls „object sequence diagrams“	Understanding how subgoals are related to goals how function is achieved by subfunctions
Blocks	„Regions of Interest“ (ROI) that syntactical or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen a sub-goal
Atoms	Language elements	Operation of a statement	Function of a statement. Goals onlye understandable context
	Text Surface	**Program execution (data flow and control flow)**	**Functions (as means or as purpose), goals of the program**
Duality	**„Structure“**		**Function**



understanding as bridging structure and function

Investigate - The Block Model

Makrostruktur	• Code kommentieren • Diagramm zeichnen, um die Gesamtstruktur/Funktions-abhängigkeiten darzustellen • „unsauberen“ Code überarbeiten	• Eingabewerte identifizieren, um alle Programmteile zu testen • Wird Zeile X je ausgeführt?	• Passenden Namen für ein Programm wählen • Satz verfassen, der den Zweck eines Programms beschreibt
Beziehungen	• Geltungsbereich von Variablen identifizieren • (Bestimmte) Funktionsaufrufe markieren • Variablenaufruf mit Variablendeklaration verbinden	• Kontrollfluss zeichnen • Redundante Bedingungszweige ermitteln • Programmausführung für Eingabewerte mit Funktionsaufrufen	• Passenden Namen für eine Variable/Funktion wählen • Sind zwei Code-Abschnitte funktional äquivalent?
Blöcke	• Codeblöcke identifizieren (Schleifen, Bedingungen, Funktionsdefinitionen, etc.)	• Reihenfolge der Codezeilen berichtigen • Parson’s Problems (Distraktoren, Verschachtelung) • Standard-Codeblöcke erkenne (z.B. Werte-Tausch zweier Variablen)	• Zweck eines Codeblocks erklären • Blöcke finden, die ein Teilproblem lösen
Atome	• Anweisungen und Ausdrücke markieren (Integer-Variablen auflisten, Funktionsköpfe markieren, etc.) • Syntax-Checkliste abarbeiten	• Variablenwerte verfolgen (ohne Funktionsaufrufe) • Verlaufstabelle zu Variablenwerten erstellen	• Zweck einer einzelnen Codezeile erklären
	Textoberfläche	**Programmausführung**	**Zweck**

Raspberry Pi & National Centre of Computing Education: Pedagogy Quick Reads - Understanding program comprehension using the Block Model
Sentance & Waite 2023

Investigate – weitere Aufgabenideen

Fachbegriffe Definitionen zuordnen	**MC-Fragen zu Codebeispiel**	**Programm in eigenen Worten erklären (schriftlich, mündlich, Audio, Screencast)**
Syntax-Checkliste zum Abarbeiten	**Debuggen (Syntaxfehler) – fehlerhafter Programmcode**	**Warum-Fragen**

Lopez et al. 2008, 103 ff.

Investigate – Fehlkonzepte berücksichtigen

No. [VarAssign]	Description
9	A variable can hold multiple values at a time / ‘remembers’ old values.
10	Variables always receive a particular default value upon creation.
11	Primitive assignment works in opposite direction.
12	Primitive assignment works both directions (swaps)
13	Limited understanding of expressions which lacks the concept of evaluation.
14	A variable is (merely) a pairing of a name to a changeable value (with a type). It is not stored inside the computer.
15	Primitive assignment stores equations or unresolved expressions.
16	Assignment moves a value from a variable to another.
17	The natural-language semantics of variable names affects which value gets assigned to which variable.
18	The order of declaration of variable names affects which value gets assigned to which variable.
19	There is no size or precision limit to the values we can store in a programming variable.
20	Incrementing a counter variable is an indivisible operation (no separate evaluation of right-hand side).
21	Primitive types (in Java) have no default value.
22	Unassigned variables of primitive type (in Java) have no memory allocated.

Sorva 2012. 367f.

Block Analysis

The Program text	Name	Goal
1 public static void sort (int[] array) { 2 int tmp=0;		-
3 for (int i=0; i<array.length-1; i++) {	Outer loop	Repeat stepwise moving for each bubble /variable
4 for (int j=0; j<array.length-1; j++){	Inner loop	Move currently smallest Bubble to the end
5 if (array[j] < array[j+1]) {	Com-pari-son	Check whether a move is needed
6 tmp = array[j]; 7 array[j] = array[j+1]; 8 array[j+1] = tmp;	Swap	Move Bubble one step (posi-tion in array)
9 } 10 } 11 } 12}		-

Block analysis of Bubblesort (Schulte 2008, 157)

- Relevante Blöcke und Aspekte des Programmcodes identifizieren, die wichtig für das Lernen sind und Lernhindernisse darstellen könnten
- Neben Lernzielen, Vorwissen, etc.
- Fehlkonzepte siehe Sorva (2012)
- Mehrere Erläuterungen zu einem Block geben (siehe Block Model; Struktur vs. Funktion)

Modify

- Lernende **verändern** den Programmcode (und damit seine Funktionaliät) oder **ergänzen** Code-Bestandteile in neuen Code-Beispielen)
- Besitzerschaft verändert sich: „nicht meins“ zu "teilweise meins“

Predict	Run	Investigate	**Modify**	Make

Make

- Lernenden lösen neue Programmieraufgaben mit Hilfe der gewonnen Programmierkenntnise aus den vorherigen Schritten
- Besitzerschaft nun „meins“
- Lehrkräfte unterstützen bei der Planung und dem Lösungsentwurf

Weitere Tools & Konzepte

- Notional Machine (Ude & Pankratz 2025)
- Abstraction Transition Taxonomy [Englisch – CS Speak Code] & Levels of Abstraction [Problem – Design – Code – Running the code] (Waite et al. 2018)
- teachcomputing.org → Quick Reads
- primmdebug.web.app
- …

Literaturverzeichnis

- Lee, I., Martin, F., Denner, J. Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011): Computational thinking for youth in practice. ACM Inroads 2011 (2/1) S. 32-37
- Lister, R. (2022). Some thoughts on designing eye movement studies for novice programmers. In 2022 IEEE/ACM 10th International Workshop on Eye Movements in Programming (EMIP) (S. 15 –22). Pittsburgh: ACM.
- Lopez, M., Whalley, J., Robbins, P. & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. ICER ’08: International Computing Edcuation Research Workshop, 101–112.
- Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. Computer Science Education, 13(2), 137–172.
- Qian, Y., and Lehman, J. (2017), Students Misconceptions and Other Difficulties in Introductory Programming: A Literature Review, ACM Transactions on Computing Education, 18: 1.
- Schulte, C. (2008). Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. ICER ’08: International Computing Education Research Workshop.
- Sentance, S. , & Waite, J. (2023). Programming in the Classroom. In S. Sentance , E. Barendsen , N.R. Howard & C. Schulte (Ed.). Computer Science Education: Perspectives on Teaching and Learning in School (S. 275–290). London: Bloomsbury Academic.
- Sentance, S., Waite, J., & Kallia, M. (2019), Teaching Computer Programming with PRIMM: A Sociocultural Perspective. Computer Science Education, 29 (2–3): 136–76.
- Sorva, J. (2012). Visual program simulation in Introductory Programming education.
- Sweller, J. (1994), Cognitive Load Theory, Learning Difficulty, and Instructional Design, Learning and Instruction, 4 (4): 295–312.
- Tsai, C. Y., Yang, Y. F., and Chang, C. K. (2015), Cognitive Load Comparison of Traditional and Distributed Pair Programming on Visual Programming Language. 2015 International Conference of Educational Innovation through Technology (EITT), 143–6.
- Ude, N. & Pancratz, N. (2025). Notional Machines als Mittel zur didaktischen Rekonstruktion von Vorstellungen zum Variablenkonzept im Programmieranfangsunterricht.
- Vainio, V., & Sajaniemi, J. (2007), Factors in Novice Programmers - Poor Tracing Skills. In Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 236–40.
- Venables, A., Tan, G., and Lister, R. (2009), A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. In Proceedings of the Fifth International Workshop on Computing Education Research, 117–28.
- Waite, J., Curzon, P., Marsh, D., Sentance, S., & Hawden-Bennett, A. (2018), ‘Abstraction in Action: K-5 Teachers’ Uses of Levels of Abstraction, Particularly the Design Level, in Teaching Programming’, International Journal of Computer Science Education in Schools, 2 (1): 14–40.