Coding Culture Oberberg

Anja Bertels

2026
Gefördert durch die Hans Hermann Voss Stiftung

Erzähle mir und ich vergesse. Zeige mir und ich erinnere. Lass es mich tun und ich verstehe.

—Konfuzius

1 Einleitung

1.1 Motivation

Wir leben in einer Welt, die fundamental von Software und Algorithmen geprägt ist. Vom Smartphone in unserer Tasche über die Logistik, die unsere Supermärkte füllt, bis hin zu den Werkzeugen der modernen Wissenschaft und Industrie – digitale Prozesse sind allgegenwärtig. Doch während wir als Gesellschaft Milliarden in digitale Infrastruktur und Endgeräte investieren, wie es der Digitalpakt in Deutschland vorsieht, vernachlässigen wir das Wesentliche: die Kompetenz, diese Technologien nicht nur zu *nutzen*, sondern sie zu *verstehen*, zu *gestalten* und *souverän* mit ihnen umzugehen.

Es entsteht eine gefährliche Lücke zwischen den Anwendern und den Gestaltern. Genau hier setzt die Vision einer *Coding Culture* an: die Etablierung von Programmierung und algorithmischem Denken als eine selbstverständliche, grundlegende Kulturtechnik, gleichberechtigt neben Lesen, Schreiben und Rechnen.

Die doppelte Notwendigkeit: Gesellschaft und Wirtschaft

Die Notwendigkeit für dieses Umdenken ist zweifach. Zum einen ist das Verstehen von Algorithmen und Datenstrukturen keine Nischenkompetenz mehr. Es ist eine Voraussetzung für

die digitale Mündigkeit in allen gesellschaftlichen Bereichen. Wer nicht zumindest grundlegend versteht, wie Daten verarbeitet werden und nach welchen Logiken Entscheidungen automatisiert werden, kann sich in der digitalisierten Welt nicht mehr souverän bewegen.

Zum anderen klafft eine dramatische Lücke auf dem Fachkräftemarkt. Das Interesse an MINT-Fächern, insbesondere an der Informatik, ist nach wie vor zu gering, um den zukünftigen Bedarf zu decken. Dies gilt insbesondere für Schülerinnen. Gleichzeitig dringt die Softwareentwicklung tief in traditionelle Disziplinen wie die Ingenieurswissenschaften ein, sei es im Rahmen von Industrie 4.0 oder bei der Entwicklung von Remote-Collaboration-Tools. Wir brauchen zukünftig eine breite Basis an Studien- und Berufsanfängern, die Programmierkenntnisse so selbstverständlich wie Mathematik oder Fremdsprachen einsetzen können.

Eine *Coding Culture* zielt darauf ab, diese Kompetenzen in der Breite zu verankern, die Diversität in den technischen Berufsfeldern zu erhöhen und insbesondere den Frauenanteil signifikant zu steigern.

Die drei Hürden auf dem Weg zur "Coding Culture"

Warum ist Programmieren noch keine Selbstverständlichkeit im deutschen Schulsystem? Die Analyse der aktuellen Situation offenbart drei zentrale Hindernisse:

Veraltete Stereotypen und kulturelle Vorurteile

Das Bild der Informatik ist oft noch geprägt von Klischees aus den 1980er Jahren, die in der Populärkultur hartnäckig weiter-

getragen werden: der männliche "Nerd", der allein im Keller sitzt. Aussagen wie "Das ist nur was für Jungs" oder "Das ist was für Zocker" sind Symptome einer systematischen Benachteiligung von Frauen in diesem Feld, die im westlichen Kulturkreis gut belegt ist. Initiativen wie "Coding for Girls" oder "Girls Days" sind wichtig, doch sie bekämpfen die Symptome. Eine echte *Coding Culture* muss die Wurzel des Problems angehen und Informatik als kreatives, kommunikatives und gesellschaftsgestaltendes Feld neu definieren.

Fehlende Lehrkräfte und vage Lehrpläne

Guter Informatikunterricht ist oft das Ergebnis des besonderen Engagements einzelner Lehrkräfte, nicht eines stabilen Systems. Die Lehrpläne sind häufig sehr allgemein und vage gehalten, wodurch die tatsächliche Entwicklung von Software und das algorithmische Denken oft zu kurz kommen. Es fehlt vielerorts an qualifiziertem Personal, da die Informatik in der Lehramtsausbildung lange eine untergeordnete Rolle spielte.

Mangel an zeitgemäßen didaktischen Materialien

Die vielleicht größte Hürde ist die Abstraktheit der Informatik. Konzepte wie Datenstrukturen, Algorithmen oder Rekursionen haben oft kein direktes Äquivalent in der Alltagserfahrung. Es fehlt an didaktischen Materialien, die diese Brücke schlagen. Schlimmer noch: Einige existierende Materialien bauen sogar Fehlkonzepte auf, die später mühsam korrigiert werden müssen. Wir können nicht erwarten, dass Schülerinnen und Schüler abstrakte Konzepte lieben lernen, wenn wir ihnen keine greifbaren Zugänge bieten.

Der Weg nach vorn: Algorithmen erlebbar machen

Eine *Coding Culture* kann nicht verordnet, sie muss erlebt werden. Um die genannten Hürden zu überwinden, müssen wir radikal umdenken, wie wir Programmieren vermitteln – insbesondere in den frühen Klassenstufen.

Das Ziel muss sein, das Interesse am *Computational Thinking* – den Denkmethoden der Informatik – frühzeitig und spielerisch zu wecken.

Hierzu brauchen wir didaktische Ansätze, die das Abstrakte greifbar machen. Im angelsächsischen Raum gibt es hierfür bereits zahlreiche Ansätze:

- **Haptische Materialien und Metaphern:** Algorithmen und Datenstrukturen müssen durch anschauliche Metaphern und physische Materialien buchstäblich "begreifbar" gemacht werden.
- **Spielerische Elemente:** Programmierspiele (auch als Brettspiele) können komplexe Logik vermitteln, ohne dass eine Zeile Code geschrieben wird.
- **Handlungsorientierte Angebote:** Lernroboter und leicht zu bedienende Technikbausteine (Maker-Kits) verknüpfen den digitalen Code direkt mit einer sicht- und erlebbaren Aktion in der physischen Welt.

Indem wir solche Materialien, Workshops und didaktische Anleitungen entwickeln, die sich am deutschen Schulsystem orientieren, können wir die grundlegenden Strukturen für Schülerinnen und Schüler erlebbar machen. Es geht darum, eine Kultur zu schaffen, in der das Entwickeln eigener kleiner Anwendungen als normaler, kreativer und lohnender Prozess wahrgenommen wird – eine Kultur, die neugierig macht, statt abzuschrecken.

1.2 Projektvorstellung

Viele zentrale Konzepte der Informatik – Algorithmen, Datenstrukturen, Speichermodelle – sind durch einen hohen Abstraktionsgrad gekennzeichnet. Für Lernende gibt es oft kein Äquivalent in der Alltagserfahrung, was den Einstieg in die Programmierung erschwert.

Genau hier setzt das Projekt **Coding Culture Oberberg** an, gefördert durch die Hans Hermann Voss Stiftung. Das Kernziel ist es, die abstrakten Konzepte der Informatik im wörtlichen Sinne "begreifbar" zu machen und so eine neue, intuitive Lernkultur zu etablieren.

Von der Abstraktion zum Begreifen: Die Theorie des Embodiments

Der didaktische Ansatz des Projekts basiert auf der Theorie des *Embodiment of Learning*. Diese geht davon aus, dass Wissensproduktion nicht von unserem Körper, unserer Sprache und unserer sozialen Entwicklung getrennt werden kann. Geist und Körper sollten nicht als getrennt, sondern als eine interagierende Einheit gesehen werden.

Für das Lernen bedeutet dies: Wenn wir Lerninhalte körperlich erfahren, verankern sie sich tiefer. Mentale Prozesse werden durch körperliche Systeme wie Bewegungen, das motorische System und den Wahrnehmungsapparat unterstützt. Das Wissen selbst wird verkörpert.

Für die Informatikdidaktik ist dieser Ansatz revolutionär:

- **Sinnliche Erfahrung:** Statt Code nur auf einem Bildschirm zu sehen, werden Konzepte durch haptische (fühlbare) und spielerische Materialien sinnlich erfahrbar.

- **Aktive Exploration:** Lernende können mit physischen Artefakten "herumspielen", Anordnungen ausprobieren und dynamische Veränderungen von Zuständen direkt wahrnehmen.
- **Strukturelle Ähnlichkeit:** Der Lerneffekt ist dann am größten, wenn die körperlichen Bewegungsabläufe eine strukturelle Ähnlichkeit zu den abstrakten Konzepten aufweisen.

Ein einfaches Beispiel ist der Ablauf einer `for`-Schleife (einer zentralen Kontrollstruktur), der durch wiederholte Operationen auf physische Gegenstände, etwa das Sortieren von Bausteinen, direkt erfahren werden kann.

Mehr als nur Grundstrukturen: Komplexe Konzepte und Metaphern

Während es für einfache Kontrollstrukturen bereits Materialien für das Vorschulalter gibt, geht "Coding Culture Oberberg" einen entscheidenden Schritt weiter. Das Projekt zielt darauf ab, *komplexere Strukturen* zugänglich zu machen, wie zum Beispiel:

- Die Modellierung von Datenstrukturen
- Die Verarbeitung von Listen (z. B. Verkettung)
- Komplexe Speichermodelle (z. B. Stapelspeicher/Stack)

Um diese Abstraktionen zu vermitteln, spielen *Metaphern* eine zentrale Rolle im Erkenntnisprozess. Das Projekt greift etablierte Metaphern aus der Didaktik der Informatik auf und entwickelt für sie anschauliche, physische und "begreifbare" Materialien. Ein Stapelspeicher wird so zu einem echten, physischen

Stapel, dessen Operationen (`push`/`pop`) man manuell durchführen muss.

Die drei Säulen des Projekts

Um diese Ziele zu erreichen und eine nachhaltige *Coding Culture* in der Region zu verankern, werden im Rahmen des Projekts drei zentrale Maßnahmen umgesetzt:

1. **Entwicklung haptischer Materialien:** Es werden spielerische und physische Materialien zum aktiven Explorieren von Informatik-Konzepten, Algorithmen und Datenstrukturen entwickelt und erprobt. (Dies umfasst die Evaluierung bestehender Produkte sowie die Neuentwicklung eigener Materialien).
2. **Durchführung von Workshops:** Die neuen Materialien und Konzepte werden in Workshops für Schülerinnen und Schüler direkt erprobt, sowohl in den Schulen vor Ort als auch auf dem Campus Gummersbach der TH Köln.
3. **Erstellung von OER-Materialien:** Die didaktischen Anleitungen, Workshop-Konzepte und Begleitmaterialien werden evaluiert und aufbereitet.

Das Ziel: Frei zugängliche Bildung für alle

Der nachhaltige Kern des Projekts liegt im Open-Source-Gedanken. Alle erfolgreichen Materialien, Aufgabenstellungen und Workshopkonzepte werden umfassend dokumentiert und als *frei zugängliche Bildungsressourcen* (Open Educational Ressource, OER) veröffentlicht.

Dadurch wird sichergestellt, dass Lehrkräfte und Schulen über das Projektende hinaus eigenständig und langfristig mit den im Projekt entwickelten Angeboten arbeiten können.

1.3 Zielgruppe

Zielgruppe und didaktischer Fokus des Projekts

Um eine *Coding Culture* nachhaltig zu etablieren, muss die Begeisterung für das Gestalten mit Code frühzeitig geweckt werden. Das Projekt "Coding Culture Oberberg" legt seinen Fokus daher gezielt auf den **Programmiereinstieg** und richtet sich primär an Schülerinnen und Schüler der **Klassen 5 bis 7**.

Diese Altersgruppe befindet sich in einer idealen Phase: Sie ist neugierig, offen für spielerische Ansätze und legt den Grundstein für das formale, abstrakte Denken, das die Informatik erfordert.

Didaktischer Scope: Worauf es in Klasse 5-7 ankommt

Eine Analyse der Kernlehrpläne für die Sekundarstufe I in NRW (Gymnasium, Realschule, Gesamtschule) zeigt, welche fundamentalen Kompetenzen in dieser frühen Phase gefordert sind. Der didaktische Scope des Projekts leitet sich direkt aus diesen Anforderungen ab und konzentriert sich auf drei Kernbereiche:

1. Algorithmen und Kontrollstrukturen (Das *Wie?*)

Dies ist der absolute Kern des Programmierens. In dieser Stufe lernen die Schülerinnen und Schüler, Probleme in logische Handlungsschritte zu zerlegen.

- **Fundament:** Alle Lehrpläne für diese Stufe fordern das Verständnis und die Anwendung der *algorithmischen Grundkonzepte*.
- **Projektfokus:** Die haptischen Materialien und Workshops zielen darauf ab, die drei zentralen *Kontrollstrukturen* erlebbar zu machen: **Sequenz** (Die richtige Reihenfolge), **Verzweigung / Bedingung** (Entscheidungen treffen), **Iteration / Schleife** (Wiederholungen)

2. Information und Daten (Das *Was?*)

Programmieren bedeutet, Daten zu verarbeiten. Ein Grundverständnis dafür, was ein Computer verarbeitet, ist unerlässlich.

- **Fundament:** Die Lehrpläne betonen die Unterscheidung zwischen Information und Daten, die Notwendigkeit der *Codierung* (z. B. Binärcode) und die Identifikation von *Datentypen* oder *Attributen* von Objekten.
- **Projektfokus:** Die Materialien helfen, die abstrakte Idee von *Daten* greifbar zu machen. Wie wird aus einem Buchstaben eine Zahl? Was ist ein *Datentyp* (z. B. eine Zahl vs. ein Wort)? Wie werden gleichartige Daten erfasst und strukturiert?

3. Formale Sprachen und Paradigmen (Das *Regelwerk*)

Die Schülerinnen und Schüler müssen intuitiv verstehen, dass ein Computer eine präzise, formale Sprache benötigt.

- **Fundament:** Die Lehrpläne führen *einfache Automaten* und die Notwendigkeit *formaler Sprachen* ein. Es wird die Basis für das Verständnis von *Syntax* (Wie muss ich es schreiben?) und *Semantik* (Was bedeutet es?) gelegt.

- **Projektfokus:** Während die Vermittlung der Kontrollstrukturen stark auf das *imperative Paradigma* (Schritt-für-Schritt-Befehle) einzahlt, schränkt sich das Projekt bewusst *nicht* darauf ein. Die Lehrpläne selbst deuten durch die frühe Einführung von *Objekten* und *Attributen* (Basis der Objektorientierung) oder *Entscheidungsbäumen* bereits andere Denkweisen an. Die Materialien des Projekts sollen daher flexibel sein, um auch diese unterschiedlichen Lösungsansätze und Denkmodelle (Paradigmen) spielerisch anzubahnen.

1.4 Vorgehen

Im Rahmen des Projekts wurden verschiedene Artefakte iterativ generiert und validiert. Eine Reihe von Spielen für das spielerische Lernen von Programmieren wurde getestet und evaluiert, sowohl im Forschungsteam als auch mit Schüler:innen aus dem Oberbergischen Kreis. Verschiedene Workshop Konzepte wurden entworfen und getestet. Alle Workshops und Veranstaltungen wurden evaluiert. Entwickelte Materialien wurden erprobt und evaluiert. Veröffentlichte Pattern und Ressourcen haben einen Peer-Review-Prozess durchlaufen. In mehreren Interviews mich Lehrkräften wurden verschiedene Meinungen und Perspektiven mit einbezogen und über den gesamten Projektverlauf hinweg eingeholt und berücksichtigt.

Das Buch soll auf verschiedene Arten genutzt werden können. Als Anleitung zur Vorbereitung von Workshops und Unterrichtseinheiten, als Nachschlagewerk für einzelne Ansätze und Projekte, als Orientierung und Inspiration und für die Differenzierung zwischen verschiedenen Abstraktionsebenen der Lehre von Programmieren an Schulen.

2 Analysen

2.1 Lehrplan

Eine detaillierte Analyse der aktuellen Lehrpläne für das Fach Informatik in Nordrhein-Westfalen (für Gymnasium, Realschule, Gesamtschule und Sekundarschule) zeigt deutlich, welche Kernkompetenzen im Bereich der Programmierung in der frühen Sekundarstufe I (Klassen 5-7) gelegt werden sollen.

Die Untersuchung bezog sich primär auf die Stufen, die diese Altersgruppe abdecken:

- **Gymnasium:** Klassen 5/6
- **Gesamtschule / Sekundarschule:** Erste Progressionsstufe
- **Realschule:** Klassen 7/8 (als relevanter Anknüpfungspunkt für die 7. Klasse)

Bei der Auswertung der Inhalte sticht ein Bereich als zentrales Fundament für die Programmierung heraus: *Algorithmen*.

Das Kernkonzept: Algorithmisches Denken

Über alle Schulformen hinweg ist der Kompetenzbereich *Algorithmen und algorithmische Grundkonzepte* die wichtigste Säule des frühen Programmierunterrichts. Das Ziel ist es, Schülerinnen und Schüler an das algorithmische Denken heranzuführen

– die Fähigkeit, ein Problem in klare, eindeutige und endliche Handlungsschritte zu zerlegen.

Für die Klassen 5 bis 7 bedeutet dies konkret:

1. **Algorithmen verstehen und anwenden:** Die Schülerinnen und Schüler lernen, was ein Algorithmus ist und wie er im Alltag (z. B. ein Kochrezept oder eine Bauanleitung) und in der Informatik verwendet wird.

2. **Algorithmische Grundkonzepte:** Im Mittelpunkt stehen die fundamentalen Bausteine, aus denen jede Programmierung besteht: *Sequenz* (Die korrekte Reihenfolge von Anweisungen), *Verzweigung/Bedingung* (Das Treffen von Entscheidungen WENN... DANN... SONST...), *Schleife/Wiederholung* (Das mehrmalige Ausführen von Anweisungen).

3. **Implementation:** Der Lehrplan für das Gymnasium (Klasse 5/6) nennt explizit die *Implementation von Algorithmen*. Dies unterstreicht die Notwendigkeit, diese logischen Konzepte auch praktisch umzusetzen, sei es durch visuelle Programmiersprachen (wie Scratch oder Blöcke) oder einfache textbasierte Umgebungen.

Wichtige Grundlagen für die Programmierung

Um Algorithmen implementieren zu können, definieren die Lehrpläne zwei weitere wichtige Stützpfeiler:

- **Information und Daten:** Programmierung ist die Verarbeitung von Daten. Daher ist der Bereich "Daten und ihre Codierungëssenziell. Die Schüler müssen ein grundlegendes Verständnis dafür entwickeln, wie ein Computer Informationen (z. B. Zahlen, Buchstaben, Bilder) darstellt, um sie sinnvoll verarbeiten zu können. - **Sprachen und Automaten:** In den Lehr-

plänen der Gesamtschule und Realschule wird der Punkt "Formale Sprachen und einfache Automaten"genannt. Dies unterstützt das Programmieren, indem es die Notwendigkeit einer exakten, unmissverständlichen Sprache (Syntax) vermittelt, die ein Computer (Automat) verstehen kann.

Was ist für Klasse 5-7 wichtig?

Zusammenfassend lässt sich sagen, dass der Fokus für die Klassen 5 bis 7 nicht auf dem Erlernen einer komplexen, spezifischen Programmiersprache liegt. Vielmehr geht es um die Vermittlung der *Grundlagen des Computational Thinking*.

Die zentralen Programmierinhalte sind:

1. Das Entwickeln von *schrittweisen Problemlösungen* (Algorithmen).
2. Das Verständnis und die praktische Anwendung der *Grundkontrollstrukturen* (Sequenz, Bedingung, Schleife).
3. Ein Grundverständnis für die *digitale Repräsentation von Daten* (Codierung).
4. Das Erlernen von *Präzision und Logik* im Umgang mit formalen Anweisungen.

2.2 Interviews

Um sicherzustellen, dass nicht an der Schulrealität vorbeigeplant wird, wurden im Vorfeld qualitative Interviews mit Informatik-Lehrkräften verschiedener Schulformen geführt. Ziel war es, die konkreten Hürden im Alltag, die didaktischen Bedürfnisse und die infrastrukturellen Rahmenbedingungen zu ermitteln.

Die Ergebnisse zeigen ein differenziertes Bild der aktuellen Herausforderungen und definieren klare Anforderungen an die zu entwickelnden Materialien.

Realitätscheck: Heterogenität und elementare Hürden

Ein zentrales Ergebnis aller Gespräche ist die extreme Leistungsheterogenität in den Klassen 5 bis 7.

- **Elementare Einstiegshürden:** Entgegen der Annahme, dass "Digital Natives" intuitive Computerkenntnisse besitzen, berichteten Lehrkräfte von grundlegenden motorischen und technischen Defiziten. Die Materialien müssen daher bereits *vor* dem eigentlichen Programmieren ansetzen: Probleme wie die Bedienung der Maus, das Zehn-Finger-System, die Nutzung der Shift-Taste (Großbuchstaben) oder der Login-Prozess müssen berücksichtigt werden.
- **Extreme Spannbreite:** Während einige Schülerinnen und Schüler bereits Vorkenntnisse aus der Grundschule (z. B. Scratch) oder dem privaten Bereich mitbringen, fangen andere bei Null an.
- **Lösung durch Niveaustufen:** Das Material muss zwingend binnendifferenziert sein. Gefordert wird ein *Level-System* (ähnlich einem Videospiel), bei dem alle Stufe 1 absolvieren müssen, leistungsstarke Lernende aber bis zu zwei Stufen überspringen können.

Didaktik: Vom Haptischen zum Digitalen

Die Interviews bestätigten den pädagogischen Kernansatz des Projekts: Reine Bildschirmarbeit überfordert zu Beginn oft das Abstraktionsvermögen.

- **Haptik und "Unplugged":** Es besteht der explizite Wunsch nach analogen Materialien (Spielfelder, physische Karten, greifbare Objekte), um Konzepte wie Schleifen oder Bedingungen erst körperlich und logisch nachzuvollziehen, bevor der Computer eingeschaltet wird.
- **Kleinschrittigkeit und Feedback:** Die Aufgabenstellungen müssen in kleine, überschaubare Einheiten ("Chunks") zerlegt werden. Da die Frustrationstoleranz vieler Lernender als gering eingeschätzt wird, sind schnelle Ergebnisse und unmittelbares Feedback essenziell.
- **Methodenvielfalt:** Um unterschiedliche Lerntypen anzusprechen, soll ein Mix aus Methoden angeboten werden: Stationenlernen, Expertenpuzzle und offene Projektphasen.

Motivation durch Gamification und Storytelling

Um das Interesse langfristig hochzuhalten, empfehlen die Lehrkräfte, Elemente aus der Lebenswelt der Schülerinnen und Schüler zu adaptieren.

- **Game Maps und Fortschritt:** Der Lernfortschritt soll visualisiert werden, beispielsweise über "Game Maps" (Lernpfade), auf denen Schülerinnen und Schüler ihren Weg durch leichte, mittlere und schwere Aufgaben nachverfolgen können.

- **Belohnungssysteme:** Konzepte wie Sammelkarten, Charakterbögen mit "XP-Boosts" oder das Freischalten von Levels sollen den Ehrgeiz wecken.
- **Storytelling:** Einbettung der Aufgaben in eine Rahmenerzählung (z. B. ein Spion-Camp oder ein Exit-Game-Szenario), um kognitive Konflikte spielerisch aufzulösen.

Organisatorische Rahmenbedingungen (Der "Koffer")

Damit die Materialien im stressigen Schulalltag tatsächlich genutzt werden, müssen sie "schlüsselfertig" sein.

- **Plug & Play:** Das Konzept "Koffer aufmachen und loslegen" ist zentral. Lehrkräfte benötigen klare Übersichten über benötigte Zeiträume, Voraussetzungen und Materialien.
- **Classroom Management:** Es bedarf Handreichungen, die nicht nur den Inhalt, sondern auch die Organisation klären: Wie werden Gruppen eingeteilt? Was tun, wenn Technik streikt? Wie wird der Koffer wieder ordentlich übergeben?
- **Infrastruktur-Unabhängigkeit:** Da WLAN an Schulen oft instabil ist, sollten digitale Inhalte auch offline verfügbar sein. Zudem ist eine Kompatibilität mit verschiedenen Endgeräten (Tablets, PCs) notwendig.
- **Sprachbarrieren:** Aufgrund der aktuellen Situation in vielen Klassen wurde der Wunsch geäußert, Materialien sprachlich flexibel zu gestalten (Umschaltmöglichkeiten, z. B. Deutsch/Englisch/Ukrainisch), um Integration zu fördern.

Nachhaltigkeit: Open Source und Erweiterbarkeit

Die Lehrkräfte betonten, dass starre Lehrwerke oft schnell veralten.

- **Open Educational Resources (OER):** Die Materialien sollen als offene Bildungsressourcen zur Verfügung stehen. Dies ermöglicht es engagierten Lehrkräften, eigene Anpassungen vorzunehmen, Fehler zu korrigieren oder neue "Level" hinzuzufügen.
- **Lösungen und Selbst-Evaluation:** Zu allen Aufgaben müssen Musterlösungen (Quellcode) bereitliegen. Zudem sollen Instrumente zur Diagnose (durch die Lehrkraft) oder zur Selbst-Evaluation der Schülerinnen und Schüler integriert werden.

Zusammenfassend zeigt die Analyse, dass der Erfolg des Projekts nicht allein von der fachlichen Tiefe abhängt, sondern maßgeblich von der **Niederschwelligkeit**, der **motivierenden Aufbereitung** und der **organisatorischen Durchdachtheit** der Materialien.

2.3 Anforderungen

Basierend auf einer umfassenden Triangulation von Datenquellen – bestehend aus der Evaluation durchgeführter Workshops, der Analyse der NRW-Lehrpläne, Experteninterviews sowie didaktischer Fachliteratur (insbesondere zu *Patterns* in der Informatikdidaktik) – wurde ein detaillierter Anforderungskatalog.

Die Anforderungen wurden priorisiert und in drei Kategorien unterteilt: Essenzielle Basisanforderungen (*Must Have*), didaktische Qualitätsmerkmale (*Should Have*) und unterstützende Rahmenbedingungen (*Nice To Have*).

Essenzielle Basisanforderungen (Must Have)

Diese Kriterien sind unverzichtbar, um die Projektziele in der Zielgruppe der Klassen 5 bis 7 zu erreichen und die identifizierten Hürden (Heterogenität, Motivation) zu überwinden.

Motivation durch Sinnhaftigkeit (Purpose):

Die Schülerinnen und Schüler müssen erkennen, wozu sie programmieren. Das Endprodukt darf keine abstrakte Berechnung sein, sondern muss Spaß machen und benutzbar sein.

- **Playability:** Mit dem entwickelten Produkt kann im Anschluss tatsächlich gespielt werden.
- **Joy of Use:** Das Ergebnis erzeugt positive Emotionen.

Frustrationsprävention und Selbstwirksamkeit:

Angesichts der oft geringen Frustrationstoleranz ist das Management von Fehlern zentral.

- **Frühe Erfolgserlebnisse:** Schnelle, sichtbare Ergebnisse motivieren zum Weitermachen.
- **Debugging-Kultur:** Fehler werden als Teil des Prozesses normalisiert.
- **Gezielte Hilfestellungen:** Unterstützung erfolgt *on demand*, beispielsweise durch KI-gestützte Tools oder klare Hilfekarten, um Sackgassen zu vermeiden.

Individualisierbarkeit (Binnendifferenzierung):

Das Material muss der extremen Leistungsheterogenität begegnen.

- **Levels:** Verschiedene Schwierigkeitsstufen, angepasst an Vorwissen oder Alter.
- **Sozialformen:** Flexibilität für Einzel-, Partner- oder Gruppenarbeit.
- **Sprache:** Eine variable Ansprache, die die Spanne zwischen einfacher Sprache (für Lernende mit Sprachbarrieren) und Fachsprache abdeckt.

Abstraktionsmanagement:

Der Übergang vom konkreten Handeln zum abstrakten Code muss begleitet werden.

- **Syntax-Spanne:** Das System sollte den Übergang von blockbasierter zu textueller Syntax ermöglichen oder vorbereiten.
- **Storytelling & Metaphern:** Abstrakte Konzepte (wie Variablen oder Schleifen) werden durch Geschichten und bildliche Vergleiche erklärt.

“Self-contained Kit”:

Um technische Hürden in Schulen zu minimieren, muss das Kit in sich geschlossen funktionieren (“alles Notwendige ist enthalten”) und plattformunabhängig (Multiplatform) einsetzbar sein.

Didaktische und methodische Ausgestaltung (Should Have)

Diese Anforderungen definieren die didaktische Qualität und die Art der Vermittlung.

Instruktionsdesign:

Die Anleitungen müssen so gestaltet sein, dass sie kognitive Überlastung vermeiden.

- **Zieltransparenz:** Das Ziel ist von Anfang an sichtbar ("Das bauen wir heute").
- **Kleinschrittigkeit:** Informationen werden in kleinen, klaren Schritten (Chunks) präsentiert, um Textwüsten zu vermeiden.
- **Multimedialität:** Einsatz von Videos und Bildern statt reiner Textanleitungen.

Modularität:

Die Inhalte sollten nicht monolithisch, sondern modular aufgebaut sein. Dies ermöglicht das Verknüpfen verschiedener Bausteine ("Connect the Pieces") und erlaubt flexible Unterrichtsplanung.

Exploratives Vorgehen:

Das Material soll zum Experimentieren einladen. Es muss möglich sein, Dinge ohne strikte Anleitung auszuprobieren und eigene Erweiterungen zu entwickeln, um das reine "Abarbeiten" von Aufgaben zu verhindern.

Physische Verankerung:

Entsprechend des "Embodiment"-Ansatzes ist die Verknüpfung von Code und physischen Objekten zentral. Projekte sollten stets einen konstruktiven "Bau-Teil" enthalten, um Informatik greifbar zu machen.

Erweiterungen und Nachhaltigkeit (Nice To Have)

Diese Aspekte erhöhen den Komfort für Lehrkräfte und die Nachhaltigkeit des Lernerfolgs.

Support-Strukturen für Lehrkräfte:

Um die Hürde für den Einsatz im Unterricht zu senken, sollten umfassende Materialien bereitgestellt werden:

- Online-Support / FAQ-Handbuch.
- Anleitungen zur Reparatur (z. B. Nachdrucken von 3D-Teilen) oder zum Ersatz von Materialien.
- Didaktische Integrationsguides für den regulären Unterricht.

"Take it Home" Faktor:

Lernen endet nicht an der Schultür. Es sollte Möglichkeiten geben, Ergebnisse zu persistieren (Abspeichern des Codes) oder physische Modelle (oder Teile davon) mit nach Hause zu nehmen, um die Identifikation mit dem Projekt zu stärken.

3 Didaktik

3.1 Pädagogische Grundprinzipien

Einordnung im Didaktischen Dreieck: Abgrenzung von Prinzipien, Methoden und Sozialformen

Die pädagogische Fachsprache ist oft von begrifflicher Unschärfe geprägt. Insbesondere die Begriffe Prinzip, Methode und Sozialform werden im Praxisalltag häufig synonym verwendet. Für eine professionelle, planvolle Gestaltung von Informatikunterricht ist eine klare taxonomische Trennung jedoch unerlässlich. Die Tatsache, dass gezielte Suchen nach klaren Abgrenzungen dieser Begriffe oft zu unspezifischen Ergebnissen führen 3, ist selbst ein Indikator für diese weit verbreitete begriffliche Unschärfe. Dieses Kapitel schafft daher zunächst die notwendige terminologische Klarheit.

In der klassischen Didaktik (z.B. nach Klafki) geht es um die Beantwortung der Fragen *Was?* (Inhalte), *Warum?* (Ziele) und *Wie?* (Methoden). Pädagogische Grundprinzipien sind auf der *Warum*-Ebene angesiedelt. Sie sind die normativen, philosophischen Leitlinien, die das pädagogische Handeln begründen und ihm eine Richtung geben. Sie sind allgemeiner als konkrete Methoden.

- **Pädagogische Grundprinzipien** sind die übergeordneten, normativen Leitsätze. Sie beantworten die Frage: *Warum*

und *wozu* gestalten wir Unterricht auf eine bestimmte Weise? (z.B. Schülerorientierung, Handlungsorientierung).

- **Didaktische Methoden** sind die konkreten, planbaren Verfahrensweisen und inszenierten Lehr-Lern-Arrangements zur Umsetzung dieser Prinzipien. Sie beantworten die Frage: *Wie* setzen wir das Prinzip um? (z.B. Projektarbeit, Stationenlernen).
- **Sozialformen** sind die organisatorischen Interaktionsstrukturen innerhalb einer Methode. Sie beantworten die Frage: *Wer* lernt mit *wem*? (z.B. Einzelarbeit, Partnerarbeit, Gruppenarbeit, Plenum).

Kernprinzipien für einen zeitgemäßen Informatikunterricht

Aus dem breiten Kanon pädagogischer Prinzipien haben sich für die Informatikdidaktik vier als besonders zentral herauskristallisiert.

Das Prinzip der Handlungsorientierung und der Konstruktionismus

Handlungsorientierung postuliert, dass Lernen am effektivsten durch aktives, selbstgesteuertes Handeln und die Erstellung eines greifbaren, sinnhaften Produkts stattfindet. Dieses Prinzip wurzelt in John Deweys "Learning by Doing" und findet seine prägnanteste Ausformung im "Konstruktionismus" von Seymour Papert. Paperts Vision war, dass Lernende Wissen am besten konstruieren, indem sie *öffentliche Artefakte* (public entities) – wie z.B. ein Programm – erschaffen.

Für den Informatikunterricht ist die Abgrenzung zur reinen *Anwendungsorientierung* fundamental. Handlungsorientierung bedeutet, dass Schülerinnen und Schüler zu *Produzenten* digitaler Artefakte werden (z.B. eine App programmieren, eine Datenbank modellieren, einen Roboter steuern). Anwendungsorientierung hingegen beschränkt sie auf die Rolle des *Bedieners* (z.B. eine Textverarbeitung nutzen).

Ein exzellentes Beispiel für gelebte Handlungsorientierung ist der in der Forschung diskutierte Einsatz des *App Inventors*. Hier erstellen Schülerinnen und Schüler eigene, funktionale Apps für Smartphones. Sie durchlaufen einen vollständigen kreativen und technischen Prozess, von der Idee bis zum Produkt.

Im Fach Informatik besitzt die Handlungsorientierung eine besondere, fast rekursive Qualität. Während in anderen Fächern eine Handlung (z.B. das Bauen eines Modells im Geschichtsunterricht) oft ein *Medium* zur Darstellung des Inhalts ist, ist im Informatikunterricht die *Handlung selbst* (das Programmieren, das Modellieren, das Strukturieren von Daten) häufig *identisch mit dem Inhaltsgegenstand* (Algorithmen, Datenstrukturen, systemisches Denken). Lernende, die eine App entwickeln, lernen nicht nur *über* Informatik – sie *betreiben* Informatik. Das Prinzip ist dem Fach somit immanent.

Das Prinzip der Problemorientierung und des Forschenden Lernens

Das Prinzip der Problemorientierung stellt den Lernprozess vom Kopf auf die Füße: Der Unterricht beginnt nicht mit der Theorie oder der Vorstellung eines Werkzeugs, sondern mit einem *authentischen, relevanten und komplexen Problem*. Das Problem ist der Motor des Lernprozesses; die zu erlernende Theorie oder

das Werkzeug wird als notwendiges Mittel zur *Lösung* dieses Problems eingeführt.

Dieses Vorgehen ist eng verwandt mit dem "Inquiry-Based Learning" oder dem "forschenden Lernen". Der Unterricht wird als *forschendes Lernen in domänspezifischen Situationen* inszeniert. Qualitativ hochwertiger Informatikunterricht ähnelt hierbei einem gut designten *digitalen Spiel*: Er stellt ein klares Problem, gibt dem Lerner Autonomie (agency) und unterstützt den Problemlöseprozess "unter Ausnutzung sozialer Interaktionen".

Im Kontext der Informatik ist die Problemorientierung das didaktische Vehikel für die Vermittlung von *Computational Thinking* (CT). CT, als die domänspezifische Problemlösekompetenz der Informatik, wird nicht als isolierte Technik (z.B. "Heute lernen wir Schleifen"), sondern im Kontext der Lösung eines realen Problems (z.B. "Wie bringen wir einen Roboter dazu, einer schwarzen Linie zu folgen?") erlernt.

Dieses Prinzip dient Lehrkräften zudem als wichtiger *Filter für Inhalte*. Das Fach Informatik ist geprägt von einer Flut an neuen Werkzeugen, Programmiersprachen und Hypes. Eine Lehrkraft, die dem Prinzip der Problemorientierung folgt, entkommt dieser "Content-Falle". Die Leitfrage verschiebt sich von "Muss ich Werkzeug X unterrichten?" zu "Welches relevante Problem können meine Schülerinnen und Schüler mit Werkzeug X lösen?". Der Fokus liegt somit auf der übertragbaren Problemlösekompetenz, nicht auf der kurzlebigen Werkzeugbeherrschung.

Das Prinzip der Schülerorientierung und Differenzierung

Das Prinzip der Schülerorientierung (oder Schüler*innenorientierung) rückt die Lernenden ins Zentrum der Unterrichtsplanung. Es fordert, den Unterricht an den Erfahrungen und Vorkenntnis-

sen der Schülerschaft anzuknüpfen und auf deren differenzierten Fähigkeiten und Entwicklungsstände einzugehen. Das Ziel ist es, jede Schülerin und jeden Schüler zu einem bestmöglichen Leistungspotenzial zu führen.

Im Informatikunterricht ist die Beachtung dieses Prinzips von herausragender Bedeutung. In kaum einem anderen Schulfach ist die Heterogenität der Vorkenntnisse so extrem. Die Spreizung reicht von Lernenden, die als *digital natives* zwar intensiv Medien konsumieren, aber keine Vorstellung von einer Dateistruktur haben, bis hin zu solchen, die in ihrer Freizeit bereits eigene Server administrieren oder Spiele modifizieren.

An Vorkenntnissen anknüpfen ist in der Informatik daher besonders anspruchsvoll. Es bedeutet nicht nur, die *Interessen* der Schüler (z.B. Spiele, Social Media) aufzugreifen, sondern vor allem, ihre oft *impliziten oder falschen mentalen Modelle* (Präkonzepte) über die Funktionsweise digitaler Systeme (z.B. "Die Cloud ist ein magischer Ort") aktiv aufzudecken, zu thematisieren und zu korrigieren.

Dieses Prinzip kann jedoch in einem *Spannungsfeld* zu anderen Prinzipien stehen. Ein komplexes, *handlungsorientiertes* Projekt kann Schülerinnen und Schüler mit geringen Vorkenntnissen überfordern und somit das Prinzip der *Schülerorientierung* verletzen. Umgekehrt kann eine übertriebene Schülerorientierung, die sich nur an oberflächlichen Interessen orientiert, die fachliche Tiefe und das Prinzip der *Ganzheitlichkeit* verfehlen. Die Kunst besteht darin, diese Prinzipien kontextabhängig auszubalancieren.

Das Prinzip der Ganzheitlichkeit und der gesellschaftlichen Einbettung

Das Prinzip der Ganzheitlichkeit fordert, Informatiksysteme niemals isoliert als technischen Code oder neutrale Hardware zu betrachten. Stattdessen müssen sie als *soziotechnische Systeme* verstanden werden, die untrennbar mit dem Individuum, Organisationen und der Gesellschaft verwoben sind.

Dieses Prinzip schlägt die entscheidende Brücke zu den übergeordneten Zielen des Faches. Die "Bildungs- und Lehraufgabe" des Informatikunterrichts umfasst explizit nicht nur "digitale Kompetenz", sondern auch "Medienkompetenz und politische Kompetenz". Das Prinzip der Ganzheitlichkeit ist das didaktische Mittel, um diese "politische Kompetenz" zu schulen.

Ein *ganzheitlicher* Blick auf einen Algorithmus (z.B. für Marktforschung oder in sozialen Medien) *muss* dessen gesellschaftliche Implikationen (z.B. Bias, Diskriminierung, Datenschutz, Machtkonzentration) einschließen. Dieses Prinzip verbietet einen *reinen* Programmierunterricht und fordert die ständige, reflexive und kritische Auseinandersetzung mit den Auswirkungen der Technologie.

Damit ist das Prinzip der Ganzheitlichkeit das vielleicht wichtigste Identitätsmerkmal des allgemeinbildenden Informatikunterrichts 2 und grenzt ihn fundamental von anderen Ausbildungen ab. Ein Universitätskurs der Informatik mag sich primär auf die Effizienz eines Algorithmus konzentrieren. Eine berufliche Ausbildung (z.B. zum Fachinformatiker) mag die Implementierung in ein Firmensystem fokussieren. Nur der allgemeinbildende Schulunterricht, geleitet vom Prinzip der Ganzheitlichkeit, muss die *politische und gesellschaftliche Dimension* in den Mittelpunkt stellen und definiert so seine einzigartige Bil-

dungsaufgabe.

Die vier vorgestellten Grundprinzipien – Handlungsorientierung, Problemorientierung, Schülerorientierung und Ganzheitlichkeit – bilden den **normativen Kompass** (das *Warum*) für die Gestaltung von Informatikunterricht. Sie geben die Richtung vor. Die **didaktischen Methoden** sind die *Fahrzeuge* (das *Wie*), die Lehrende wählen, um sich in die vorgegebene Richtung zu bewegen. Die Prinzipien *bedingen* die Methodenauswahl. Die Entscheidung für ein Prinzip schließt bestimmte Methoden aus und legt andere nahe.

3.2 Didaktische Methoden

In Interviews mit erfahrenen Informatiklehrkräften wurden bewährte Methoden identifiziert, die sich im Unterricht der Sekundarstufe I als besonders effektiv erwiesen haben. Diese Erkenntnisse fließen direkt in die Konzeption der Projektmaterialien ein. Dabei zeigt sich ein klarer Konsens: Erfolgreicher Informatikunterricht findet nicht ausschließlich am Bildschirm statt, sondern benötigt vielfältige Zugänge, die Planung, soziale Interaktion und haptisches Begreifen integrieren.

"Unplugged" & Veranschaulichung: Informatik ohne Computer

Ein zentrales Ergebnis der Befragung ist die hohe Bedeutung der Phase *vor* der eigentlichen Programmierung. Lehrkräfte betonen, dass algorithmisches Denken oft besser "begriffen" wird, wenn es zunächst losgelöst von der Technik vermittelt wird.

- **Planung auf Papier:** Bevor die erste Zeile Code geschrieben wird, entwickeln Schülerinnen und Schüler ihre Ide-

en analog. Bewährte Mittel sind hier das Skizzieren von Pseudocode, das Zeichnen von Struktogrammen oder UML-Diagrammen. Der Planungsanteil nimmt dabei oft einen größeren Raum ein als die spätere Umsetzung am PC.

- **Haptische Methoden:** Um abstrakte Konzepte greifbar zu machen, setzen Lehrkräfte auf physische Hilfsmittel. Algorithmen werden durch das Legen von Plättchen visualisiert, Konzepte der Objektorientierung (OOP) mithilfe von Karten eingeführt oder Rekursion am Beispiel der "Türme von Hanoi" erlebbar gemacht.
- **Rollenspiele & Metaphern:** Komplexe Logik wird durch körperlichen Einsatz simuliert. So spielen Schülerinnen und Schüler Sensoren nach (z. B. "Hände und Augen als Linienfolger"), stellen Referenzen dar, indem sie aufeinander zeigen, oder nutzen Alltagsmetaphern (wie "Bademeister und Rutsche" für Schleifen), um technische Abläufe zu verstehen.

Kooperative Lernformen & Soziale Interaktion

Der Einzelkämpfer am PC ist ein Auslaufmodell. Die Interviews zeigen einen starken Trend hin zu kooperativen Lernformen, die den Austausch über Lösungswege fördern.

- **Think-Pair-Share:** Diese Methode wird häufig genutzt, um individuelle Denkphasen mit kooperativem Austausch zu verbinden: Erst überlegt jeder allein, dann wird die Idee in Partnerarbeit besprochen, bevor sie im Plenum präsentiert wird.
- **Gruppen- & Expertenpuzzle:** Besonders bei komplexeren Themen (z. B. Sortieralgorithmen) eignen sich arbeits-

teilige Gruppenverfahren. Dies ermöglicht auch eine Binnendifferenzierung, indem stärkere Schülergruppen anspruchsvollere Teilaufgaben übernehmen.

- **Peer-Teaching:** Helfersysteme spielen eine wichtige Rolle. Erfahrene Schülerinnen und Schüler fungieren als Experten, die herumgehen und unterstützen, oder es werden bewusst Tandems aus leistungsstärkeren und schwächeren Lernenden gebildet.
- **Gemeinsamer Diskurs:** Das "Sprechen über Code" im Plenum, das Vergleichen verschiedener Lösungswege oder das gemeinsame Tracing (Nachvollziehen) von Codezeilen am Beamer sind essenziell für das Verständnis.

Projektbasiertes & Exploratives Lernen

Um die Motivation hochzuhalten und Selbstständigkeit zu fördern, setzen viele Lehrkräfte auf offenere Formate.

- **Projektarbeit:** Das Ziel ist oft ein konkretes, sinnhaftes Produkt, etwa ein eigenes Minispiel oder eine Datenbankanwendung mit Alltagsbezug.
- **Exploration:** Lernende erhalten offene Aufgabenstellungen, bei denen sie selbstständig herausfinden (explorieren) müssen, was mit den gegebenen Tools möglich ist. Die freie Projektwahl stärkt dabei die Eigenverantwortung.
- **Gamification:** Spielerische Ansätze dominieren den Einstieg. Allerdings weisen einige Lehrkräfte darauf hin, dass ein Gleichgewicht nötig ist: Zu viel Spiel kann vom ernsthaften Verständnis der Grundlagen ablenken.

Strukturierung & Differenzierung

Angesichts der enormen Leistungsheterogenität in den Klassen sind klare Strukturen und Differenzierungsmaßnahmen unverzichtbar.

- **Kleinschrittigkeit:** Komplexität wird reduziert, indem Inhalte in kleine Häppchen zerlegt werden. Strenge, kleinschrittige Anleitungen geben gerade schwächeren Lernenden Sicherheit.
- **Differenzierung:** Um dem unterschiedlichen Lerntempo gerecht zu werden, kommen abgestufte Hilfen, Lernvideos oder Learning Management Systeme (LMS) zum Einsatz, die ein individuelles Fortschreiten ermöglichen.
- **Stationenlernen:** Methoden wie Stationenlernen (ggf. mit Laufzetteln) erlauben es, analoge und digitale Phasen sowie Einzel- und Partnerarbeit flexibel zu kombinieren.

3.3 Workshop Konzepte

Um die unterschiedlichen Altersgruppen und Vorerfahrungen der Schülerinnen und Schüler adäquat abzuholen, wurden im Projekt spezifische Workshop-Konzepte entwickelt. Diese folgen einer gemeinsamen didaktischen Grundstruktur, variieren jedoch stark in Inhalt, Komplexität und methodischem Ansatz.

Allgemeine Agenda und Struktur

Unabhängig von der Zielgruppe folgt jeder Workshop einem bewährten strukturellen Rahmen, der eine Balance aus Input, Aktivität und Reflexion gewährleistet:

- **Einführung und Zielsetzung:** Klare Transparenz schaffen – Was machen wir heute? Wozu dient das Endprodukt?
- **Theoretische Grundlagen:** Kurze, prägnante Vermittlung der notwendigen Konzepte (z. B. "Was ist eine Schleife?"), oft unterstützt durch "Unplugged"-Methoden.
- **Praktische Übungen:** Der Kern des Workshops. Hier wird entwickelt, gebaut und programmiert.
- **Feedback und Diskussion:** Vorstellung der Ergebnisse (Show & Tell) und Reflexion über aufgetretene Probleme (Debugging-Kultur).
- **Abschluss und Zusammenfassung:** Sicherung der Lernergebnisse.
- **Gimmick:** Jeder Workshop endet mit einem besonderen Highlight oder einem "Take-Home"-Element (z. B. das Spiel auf dem eigenen Handy, ein physisches Artefakt), um eine positive emotionale Verankerung zu sichern.

Zielgruppenspezifische Ausrichtung

Damit die Workshops erfolgreich sind, basieren sie auf drei zentralen Säulen, die für alle Altersstufen gelten:

1. **Interaktion:** Frontalunterricht wird auf ein Minimum reduziert. Der Fokus liegt auf dem "Machen".
2. **Relevanz:** Die Inhalte müssen an die Lebenswelt der Teilnehmenden anknüpfen (z. B. Smartphone-Apps, Spiele).
3. **Gruppenarbeit:** Die Förderung von Teamkompetenzen durch kooperative Arbeitsformen (Pair Programming, Projektteams).

Altersdifferenzierte Konzepte

Die inhaltliche Ausgestaltung orientiert sich an der kognitiven Entwicklung und den typischen Interessen der jeweiligen Altersstufen:

Altersgruppe 13–15 Jahre (Klasse 7–9): Der spielerische Einstieg

In dieser Phase steht der niedrigschwellige Zugang im Vordergrund. Vorerfahrungen werden nicht vorausgesetzt, aber aktiv abgefragt, um Gruppen homogen zu bilden.

- **Inhalte:** Fokus auf visuellen, blockbasierten Sprachen wie *Scratch* oder dem *MIT App Inventor*.
- **Ziel:** Schnelle Sichtbarkeit von Ergebnissen. Das Verständnis von Algorithmen wird spielerisch erlernt, ohne durch Syntax-Fehler frustriert zu werden. Die Erstellung eigener kleiner Apps oder Spiele motiviert durch direkten Alltagsbezug.

Altersgruppe 15–17 Jahre (Klasse 9–11): Der Übergang zur Textprogrammierung

Hier erfolgt oft der Schritt von der visuellen zur textuellen Programmierung. Die Projekte werden komplexer und technischer.

- **Inhalte:** Einführung in textbasierte Sprachen (z. B. `C`) oder fortgeschrittenes Game Development.
- **Projektbeispiel:** Die Programmierung eines funktionstüchtigen Taschenrechners oder eines komplexeren Spiels.

- **Fokus:** Vertiefung des algorithmischen Verständnisses, Einführung in Datentypen und logische Operatoren. Die Schülerinnen und Schüler lernen, abstraktere Probleme in Code zu übersetzen.

Altersgruppe 18–19 Jahre (Oberstufe / Berufskolleg): Simulation der Berufswirklichkeit

Bei den ältesten Teilnehmenden verschiebt sich der Fokus vom reinen *Coden* hin zum professionellen Prozess der Softwareentwicklung.

- **Inhalte:** Durchlaufen eines kompletten Software-Entwicklungs-Zyklus.
- **Methodik:** Simulation eines realen Entwicklungsteams. Die Teilnehmenden übernehmen verschiedene Rollen (z. B. Projektmanagement, Frontend-Entwicklung, Backend-Entwicklung, Testing).
- **Ziel:** Verständnis für die Arbeitsteilung und die Prozesse in der IT-Industrie (Requirements Engineering, agile Methoden) entwickeln, als Vorbereitung auf Studium oder Ausbildung.

4 Projekte

Die im Rahmen von „Coding Culture Oberberg“ entwickelte Web-App fungiert als zentraler Hub für alle Beteiligten: Sie ist Wissensspeicher, Materialdatenbank und Community-Plattform in einem. Ziel der Anwendung ist es, die Hürden für den Einstieg in den Informatikunterricht so gering wie möglich zu halten, indem alle Ressourcen an einem Ort gebündelt und visuell ansprechend aufbereitet werden.

Dieses Kapitel gibt einen Überblick über den Aufbau der Plattform und stellt beispielhaft drei Projekte vor, die im Rahmen der Initiative entstanden sind und evaluiert wurden.

4.1 Aufbau der Web-App

Die Struktur der App wurde bewusst übersichtlich gehalten, um eine intuitive Navigation zu ermöglichen. Die Hauptbereiche gliedern sich wie folgt:

Landing Page Der Einstiegspunkt bietet prägnante Informationen über das Gesamtprojekt, die beteiligten Partner und die übergeordneten Ziele der Initiative. Sie dient als Visitenkarte und als Orientierungshilfe.

Projekte Das Herzstück der Anwendung. Hier finden Lehrkräfte eine gefilterte Übersicht aller evaluierten Coding-Projekte. Jedes Projekt wird übersichtlich mit Schwierig-

keitsgrad, benötigter Zeit und den adressierten Lernzielen dargestellt.

Didaktik Dieser Bereich fasst die pädagogischen Erkenntnisse zusammen, die aus den Interviews mit Lehrkräften gewonnen wurden. Er dient als theoretischer Unterbau für die praktische Arbeit im Klassenzimmer.

Muster (Patterns) Hier werden didaktische Entwurfsmuster veröffentlicht, die sich im Projektverlauf als „Best Practices" herauskristallisiert haben. Sie bieten Lösungen für wiederkehrende pädagogische Herausforderungen im Informatikunterricht.

Vorlagen Ein Download-Bereich für standardisierte Templates. Von Projektkarten über Präsentationsvorlagen bis hin zu Feedbackbögen können Lehrkräfte hier Materialien herunterladen, um den Unterricht effizient vorzubereiten.

4.2 Highlights

Um die theoretischen Konzepte mit Leben zu füllen, werden im Folgenden drei beispielhafte Projekte vorgestellt, die unterschiedliche didaktische Schwerpunkte setzen.

4.2.1 Mission Weltraum

Dieses Projekt entstand aus der konkreten Aufgabenstellung, ein Konzept für einen „Coding-Koffer" für die Klassen 5 bis 7 zu entwickeln. Die Herausforderung bestand darin, abstrakte Programmierkonzepte lehrkraftfreundlich, motivierend und binnendifferenziert zu vermitteln, ohne die Schülerinnen und Schüler durch trockene Theorie zu frustrieren.

Das Konzept: Storytelling trifft Gamification

Um die Motivation der Lernenden dauerhaft zu sichern, bettet das Projekt alle Aufgaben in eine durchgehende **Raumschiff-Geschichte** ein. Die Storyline gibt Struktur und einen Sinnzusammenhang für die fünf Lerneinheiten (Kapitel) à 45 Minuten:

1. **Sequenzen:** Der Start der Mission.
2. **Schleifen & Bedingungen:** Navigation durch Asteroidenfelder.
3. **Variablen:** Ressourcenmanagement an Bord.
4. **Algorithmen & Methoden:** Komplexe Flugmanöver.
5. **Finale:** Wiederholungs-Quiz und ein Scratch-Abschlussprojekt.

Didaktischer Ansatz: Rollenspiel und Skeuomorphismus

Die Klasse wird in 4er-Gruppen aufgeteilt. Jedes Gruppenmitglied übernimmt pro Lerneinheit die Rolle eines Charakters aus der Geschichte (z. B. Navigator, Ingenieur). Diese Rollen rotieren, sodass jedes Kind unterschiedliche Verantwortungen trägt und die Teamdynamik gestärkt wird.

Ein Kernmerkmal ist die Nutzung von **Abstraktionen und Skeuomorphismus**: Technische Konzepte werden durch physische Gegenstände erklärt. So werden Variablen beispielsweise nicht abstrakt als Speicheradressen definiert, sondern durch echte Becher repräsentiert, in denen Werte „gelagert“ werden.

Der physische Koffer

Um die Hürde für Lehrkräfte zu minimieren, passt das gesamte Basismaterial in einen einfachen Pappkarton:

- Ein USB-Stick mit allen PDFs, digitalen Inhalten und Druckvorlagen.
- 8 Becher zur Veranschaulichung des Variablen-Konzepts.

Zusätzlich werden lediglich Scheren, Spielfiguren und ein internetfähiges Gerät pro Gruppe benötigt.

4.2.2 Das Labyrinth

Dieses Projekt verbindet handwerkliches Geschick mit algorithmischem Denken und Robotik. Es eignet sich besonders, um den Transfer von Code in die physische Welt zu demonstrieren.

Ablauf:

1. **Bauphase:** Die Schülerinnen und Schüler konstruieren zunächst physisch ein Labyrinth aus Pappe.
2. **Programmierung:** Ein Micro:bit-Mikrocontroller, montiert auf einem Maqueen-Roboterchassis, wird programmiert. Ziel ist es, dass der Roboter mithilfe seiner Sensoren (Ultraschall oder Linienfolger) selbstständig Hindernisse erkennt.
3. **Algorithmus:** Der Wegfindungs-Algorithmus wird gemeinsam in der Gruppe entwickelt (z. B. die „Wandfolger"-Strategie).
4. **Wettbewerb:** Den Abschluss bildet ein Rennen: Welcher Roboter bewältigt das Labyrinth am schnellsten?

4.2.3 Tug of War

Basierend auf den *littleBits*-Modulen bietet dieses Projekt einen spielerischen Zugang zu Elektronik und Logik ohne Schreibarbeit am Computer.

Konzept:
Schülerinnen und Schüler bauen eine digitale Version des klassischen Tauziehens. Hierbei kommen Eingabemodule (Buttons/-Sensoren) und Ausgabemodule (LED-Leisten oder Motoren) zum Einsatz. Durch schnelles Drücken oder Interagieren muss ein Signal auf die eigene Seite „gezogen“ werden. Das Projekt fördert das Verständnis für digitale Signale, Zustandswechsel und den Aufbau einfacher Schaltkreise.

5 Beitrag

Das Projekt „Coding Culture Oberberg“ ist kein statisches Werk, sondern eine lebendige Plattform, die von den Erfahrungen und Ideen der Community lebt. Die Web-App und die Materialsammlung sind als Open-Source-Projekt konzipiert. Das bedeutet: Jede Lehrkraft, jede Schülerin und jeder Schüler sowie Entwicklerinnen und Entwickler sind eingeladen, Fehler zu korrigieren, neue Features zu entwickeln oder eigene Projektideen beizusteuern.

Dieses Kapitel beschreibt, wie Sie technisch zur Web-App beitragen, wie Sie neue Projekte in das System einspeisen und wie Sie die bereitgestellten Vorlagen nutzen und erweitern können.

5.1 Anleitung zur Weiterentwicklung der Web App

Die Web App ist das Herzstück des Projekts. Sie dient als Katalog für Projekte, als Wissensspeicher und als Werkzeugkasten für den Unterricht. Gehostet wird die Anwendung über **GitHub Pages**, der Quellcode liegt offen auf dem dazugehörigen GitHub-Repository.

Der Technologie-Stack

Für technisch interessierte Mitstreiter: Die App basiert auf einem modernen, performanten Stack:

- **Svelte:** Als Frontend-Framework für eine reaktive und schnelle Benutzeroberfläche.
- **shadcn/ui:** Für ein zugängliches, konsistentes und ansprechendes Design der Komponenten.

Zusammenarbeit via GitHub

Die Kollaboration findet zentral auf der Plattform GitHub statt. Hier gibt es verschiedene Wege, sich einzubringen:

1. **Issues (Fehler melden & Ideen einbringen):** Wenn Sie einen Fehler (Bug) in der App finden oder eine Idee für eine neue Funktion haben, können Sie im Reiter *Issues* ein neues Ticket erstellen. Beschreiben Sie das Problem so genau wie möglich.
2. **Discussions (Austausch):** Über den Tab *Discussions* können Sie in Kontakt mit anderen Lehrkräften und den Entwicklern treten. Hier ist der Platz für pädagogische Fragen, Erfahrungsaustausch oder Diskussionen über Best Practices im Informatikunterricht.
3. **Pull Requests (Code beisteuern):** Wer selbst programmieren kann, darf den Code direkt erweitern.
    - **Forken** Sie das Repository (erstellen Sie Ihre eigene Kopie).
    - Erstellen Sie einen neuen **Branch** für Ihre Änderung.

- Reichen Sie Ihre Änderungen als **Pull Request (PR)** ein. Das Kern-Team prüft den Code und übernimmt ihn in die Hauptversion.

Git & GitHub in aller Kürze

Für Einsteiger in die Versionierung mit Git sind dies die wichtigsten Schritte im Terminal:

```
git clone [URL]
# Laedt das Repository auf Ihren Rechner.

git checkout -b feature/mein-feature
# Erstellt einen neuen Arbeitszweig.

git add .
git commit -m "Beschreibung"
# Speichert Ihre Aenderungen.

git push origin feature/mein-feature
# Laedt die Aenderungen hoch, um einen Pull
    Request zu stellen.
```

5.2 Entwicklung und Integration neuer Projekte

Damit die Plattform wächst, brauchen wir stetig neue Projektideen. Das Hinzufügen eines neuen Projekts in die Web-App erfordert **keine Programmierkenntnisse** in Svelte. Die App generiert die Projektübersicht dynamisch aus einer zentralen Datendatei.

Der Prozess über das GitHub-Interface

Sie können neue Projekte direkt im Browser über die GitHub-Oberfläche hinzufügen:

1. Navigieren Sie im Repository zur Datei `app/static/data/projects/temp` (oder dem entsprechenden Daten-Ordner).
2. Klicken Sie auf das Stift-Symbol (*Edit this file*).
3. Fügen Sie einen neuen Eintrag nach dem untenstehenden JSON-Schema hinzu.
4. Klicken Sie unten auf *Commit changes* und wählen Sie *Create a new branch for this commit and start a pull request*. Damit schlagen Sie Ihr neues Projekt zur Aufnahme vor.

Die JSON-Struktur

Ein Projekt wird durch ein JSON-Objekt definiert. Hier ist die Vorlage mit Erklärungen zu den einzelnen Feldern:

```
{
        "id": "UUID",
        "name": "Name",
        "description": "Beschreibung",
        "product": "Produkt",
        "content": [
                "Keyword 1",
                "Keyword 2",
                "Keword 3"
        ],
        "complexity": 1,
        "language": ["Python"],
        "minDuration": 30,
        "maxDuration": 60,
```

```
        "minGroup": 1,
        "maxGroup": 3,
        "type": "Konsolenspiel",
        "materials": [
                "Computer",
                "Schere",
                "Papier"
        ]
}
```

Erklärung der Felder

id Eine einzigartige Identifikationsnummer. Nutzen Sie einen Online-UUID-Generator, um eine zufällige ID zu erzeugen. Dies verhindert Datenbankkonflikte.

name & description Titel und Teaser-Text, der auf der Übersichtskarte in der App erscheint.

product Das zentrale Produkt, um das es sich dreht.

content Eine Liste von Lerninhalten (Tags), nach denen Nutzer filtern können.

complexity Eine Skala von 1 (Einfach) bis 3 (Schwer).

language Die verwendete(n) Programmiersprache(n).

duration Zeitrahmen in Minuten (von/bis).

group Gruppengröße (von/bis).

type Die Art von Projekt, z.B. Einplatinencomputer, Brettspiel, etc.

materials Physische oder digitale Materialien, die vorbereitet werden müssen.

Sobald Ihr Pull Request akzeptiert ist, erscheint das Projekt automatisch in der Web-App und kann von Lehrkräften überall genutzt werden.

5.3 Nutzung und Anpassung von Vorlagen

Um den Vorbereitungsaufwand für Lehrkräfte zu minimieren, bietet die Web-App einen eigenen Reiter **„Vorlagen“**. Hier finden sich standardisierte Dokumente, die für einen reibungslosen Ablauf der Projekte sorgen. Diese Vorlagen sind ebenfalls Open Source und können bei Bedarf angepasst werden.

Folgende Kategorien stehen zum Download bereit:

5.3.1 Projektkarten

Dies sind visuell aufbereitete Kurzübersichten (*One-Pagers*) für jedes Projekt. Sie eignen sich hervorragend für offene Lernphasen oder Stationenlernen. Schülerinnen und Schüler können anhand der Karten selbstständig entscheiden, welches Projekt sie bearbeiten möchten. Sie enthalten den Titel, das Zielprodukt, den Schwierigkeitsgrad und die benötigte Zeit auf einen Blick.

5.3.2 Präsentationsvorlagen

Die bereitgestellten Folien-Templates geben eine Struktur vor, um den Unterricht, orientiert an den Projekten und im Workshop-Charakter, zu präsentieren.

5.3.3 Feedbackbögen zur Evaluation

Um die Qualität des Unterrichts und der Materialien zu sichern, stehen Feedbackbögen bereit. Diese können von den Lernenden ausgefüllt werden, um Rückmeldung zu den Projekten zu geben.

5.3.4 Anleitungstemplates (Handreichungen)

Für jedes neue Projekt können zusätzlich Schülerhandreichungen erstellt werden. Das Anleitungstemplate bietet eine didaktisch sinnvolle Struktur und ein einheitliches Design.

Wir ermutigen alle Nutzenden, verbesserte Versionen dieser Vorlagen oder gänzlich neue Hilfsmittel über den oben beschriebenen GitHub-Workflow wieder in das Projekt zurückfließen zu lassen.