Prozessorientierte Software-Entwicklung im Informatikunterricht

Hanno Schauer [1]

Abstract: Das Thema Softwareentwicklung ist geeignet, wertvolle Schlüsselkompetenzen zu vermitteln. Es ist aber gleichzeitig ein besonders herausfordernder Unterrichtsgegenstand. Denn es gilt, die Komplexität realer Softwareprojekte im Unterricht erlebbar zu machen, ohne diesen zu überfrachten. Der Beitrag präsentiert eine speziell auf die Bedürfnisse der Schule abgestimmte Software-Entwicklungsmethode, die durch die Anwendung von Prozessmodellierungstechniken dazu beiträgt, viele der Herausforderungen deutlich abzumildern.

Keywords: Software-Entwicklung, Schulinformatik, Prozessmodellierung, Prozessorientierung.

1 Motivation

Die Softwareentwicklung ist ein ebenso reizvolles wie herausforderndes Unterrichtsthema. Sie ist reizvoll, da sich hier in besonderem Maße das Wesen der Informatik zeigt, und weil Schlüsselqualifikationen gefördert werden, die nicht nur für die Informatik von besonderem Wert sind: Das Gestalten von Technologien, das methodische Vorgehen, die prozessorientierte Sichtweise sowie das Arbeiten in Projekten.

Gleichzeitig ist die Softwareentwicklung ein herausforderndes Thema. Denn es gilt, sowohl die Komplexität realer Softwareprojekte als auch die Anwendung informatischer Planungs- und Modellierungsmethoden im Unterricht erlebbar zu machen, ohne diesen zu überfrachten. Dabei sollten charakteristische Aufgaben, Phasen und (typische) Meilensteine für den Lerner klar erkennbar und (jede Phase für sich) sinnhaft sein. Um diesen Spagat zu meistern, bedarf es geeigneter Unterrichtskonzepte.

Eine in Schulbüchern verbreitete Möglichkeit, den dargestellten Herausforderungen didaktisch zu begegnen, ist es, die Softwareentwicklung anhand innerinformatischer Problemstellungen zu behandeln: anhand eines Taschenrechners oder Fahrkartenautomaten [Hu09] oder eines Simulationsprogramms für eine Populationsentwicklung [Du09]. Ein anderer Ansatz ist der Verweis auf ein Pflichtenheft, welches bereits alle Anforderungen an eine Software beschreibt [Br09].[2]

In beiden Fällen sind die Anforderungen an eine Software weitestgehend vorgegeben. Die praktischen Übungen reduzieren sich auf die Phasen des Software-Entwurfs (mit UML-Klassendiagrammen; zuweilen auch mit Zustandsdiagrammen [Hu09]) sowie die Implementierung der Software. Insbesondere die frühen Phasen der Softwareentwick-

[1] Mons-Tabor-Gymnasium Montabaur, Von-Bodelschwingh-Straße 35, 56410 Montabaur, hanno.schauer@uni-due.de

[2] Andere Schulbücher klammern das Thema Softwareentwicklung gänzlich aus (z. B. [En06], [Ha10]).

lung, die Analyse- und frühe Entwurfsphasen, bei denen in realen Software-Projekten mit Anwendern bzw. Auftraggebern zusammengearbeitet wird, werden hierbei ausgeklammert. Dabei werden die frühen Phasen in den Einheitlichen Prüfungsanforderungen der KMK explizit für das Abitur gefordert ([KM04], S. 5): „Grundprinzip des Modellierens als zielgerichtetes Vereinfachen und strukturiertes Darstellen von Ausschnitten der Wirklichkeit, Erstellen eines Modells auf der Grundlage der Problemanalyse."

Da in vielen allgemeinbildenden Schulen Informatik das einzige Technik- bzw. Ingenieursfach ist, ist die Softwareentwicklung das Thema, bei welchem das Wesen von Ingenieursberufen i. Allg. exemplarisch vermittelt werden kann. Bei den genannten bisherigen Ansätzen für Softwareentwicklung(sprojekte) im Unterricht spielt nicht zuletzt das für Ingenieurswissenschaften typische gestaltungsorientierte Moment nur eine eingeschränkte Rolle. Insbesondere fehlt den Schülerinnen und Schülern die Erfahrung, dass die Einführung von Software die jeweilige Domäne – z. B. ein Unternehmen bzw. dessen Organisationsstruktur oder Abläufe – in deutlicher Weise verändert.

In diesem Beitrag wird eine speziell für den Einsatz im Schulunterricht entworfene Softwareentwicklungsmethode präsentiert. Gemäß der eingesetzten Modellierungssprachen und Implementierung heißt die Methode *ProKlaMation – Prozesse, Klassen, Automation*. Die Methode unterstützt speziell dabei, (auch) die frühen Phasen der Softwareentwicklung im Unterricht angemessen zu adressieren. Die Methode orientiert sich an gängigen Vorgehensweisen der Informatik-Praxis. Sie nutzt Ansätze der (Geschäfts-) Prozessmodellierung.

Der Beitrag ist wie folgt aufgebaut: Zunächst werden Prozessmodellierungstechniken allgemein und die hier genutzte Business Process Model Notation (BPMN) grundlegend vorgestellt (Kap. 2). Danach folgen ein Überblick über die Methode (Kap. 3) sowie die Anwendung der Methode an einem Beispiel (Kap. 4). Didaktische Anmerkungen (Kap. 5) sowie ein Ausblick (Kap. 6) runden den Beitrag ab.

2 Konzeptuelle Prozessmodellierung

In der Wirtschaftsinformatik werden seit geraumer Zeit konzeptuelle Prozessmodellierungssprachen entwickelt. Auch viele Vertreter der Informatik beschäftigen sich mit Fragen der Prozessmodellierung (vgl. [SSF15], S. 15). Es handelt sich hierbei um grafische Modellierungssprachen, die darauf gerichtet sind, realweltliche Abläufe – z. B. Geschäftsprozesse eines Unternehmens – semi-formal abzubilden und zu visualisieren. Die Sprachen sind so konzipiert, dass sie (1) Ablaufstrukturen formalisieren und als Planungsinstrument der Software-Entwicklung genutzt werden können. Gleichzeitig (2) sind sie so einfach gehalten, dass sie für Nicht-Informatiker gut verständlich sind. Sprachen der konzeptuellen Modellierung können somit insbesondere als Mittel der Kommunikation zwischen Domänenexperten und Software-Entwicklern genutzt werden.

Die Sprachen dienen hierbei sowohl als Instrument, um existierende Abläufe zu modellieren und zu analysieren (Ist-Prozess), als auch, um Prozesse zu reorganisieren (Soll-Prozess). Besonderes Interesse gilt hierbei auch den Veränderungen in Prozessen, die

durch eine avisierte Computerunterstützung hervorgerufen werden. Diesbezüglich sind Prozessmodelle auch ein konkretes Mittel der Technologiefolgeneinschätzung.

Typische Notationselemente einer konzeptuellen Prozessmodellierungssprache sind Aktivitäten, ein Kontrollfluss samt Verzweigungen sowie Ereignisse, die den Prozess auslösen, beeinflussen, oder (Zwischen-) Ergebnisse symbolisieren (siehe Abb. 1).

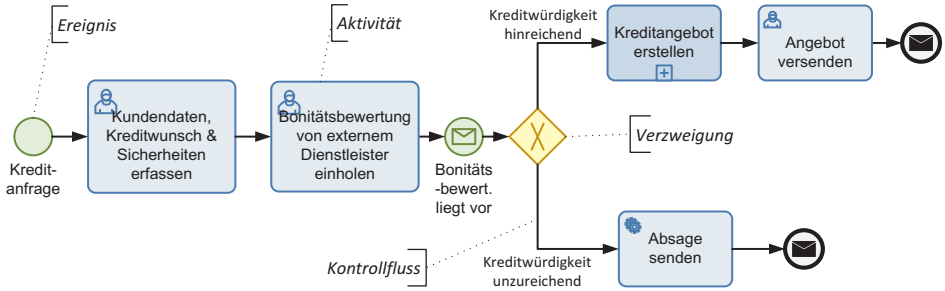

Abb. 1: Vereinfachtes Modell des Prozesses einer Kreditwürdigkeitsprüfung (in BPMN)

Die wesentlichen Strukturelemente ähneln den aus der Kerninformatik bekannten Programm-Ablauf-Plänen, weshalb sich Prozessmodellierungssprachen gut in den Unterricht integrieren lassen. Allerdings gibt es auch zentrale Unterschiede: Prozessmodellierungssprachen sind, wie bereits erwähnt, dediziert semi-formal und als Kommunikationsinstrument zwischen Entwicklern und Domänenexperten konzipiert. Daher beinhalten sie zusätzliche Elemente, die zur Abbildung realweltlicher Prozesse besonders hilfreich sind. Besonders augenfällig ist hierbei bspw. die Kennzeichnung einer Aktivität dahingehend, ob sie manuell ( ), semi-automatisch (Mensch interagiert mit Maschinen:) oder voll-automatisch abläuft ( ).

Ein deutlicher Unterschied zur softwarenahen Modellierung besteht zudem in der Art, wie die Sprachelemente typischerweise verwendet werden: Wenn z. B. ein Ablauf nicht eindeutig zu beschreiben ist, weil die handelnden Personen Freiheitsgrade haben, ist es üblich, unvollständig zu modellieren – z. B. über vage Aufgabenbeschreibungen. Auch bieten die Sprachen Notationselemente, die eine nur teilweise spezifizierte Modellierung unterstützen (z. B. unspezifische Verzweigungen). Sehr gängig in der Praxis ist es zudem, unklare oder nur aufwendig formalisierbare Tatbestände durch zusätzliche *textuelle Annotationen* natürlichsprachlich zu beschreiben.

Im Unterschied zu Klassendiagrammen (i. W. Spezifikation von Softwareschnittstellen) oder Zustandsdiagrammen (i W. Spezifikation gültiger Zustände eines Systems) sind konzeptuelle Prozessmodelle keine reduzierte, aber formale Sicht auf eine Software, sondern eine nur teilweise formalisierte Darstellung realweltlicher Prozesse. In der Software-Entwicklung sind Prozessmodelle daher Instrumente in frühen Phasen, nämlich der Analysephase und frühen Entwurfstätigkeiten (Prozess-Design).

Es gibt eine Reihe von konzeptuellen Prozessmodellierungssprachen, die alle für die hier vorgeschlagene Software-Entwicklungsmethode geeignet wären. Zu denken ist hier insbesondere an die *MEMO Process Modelling Language* (MEMO-PML) [Fr11] oder

die *Ereignisgesteuerten Prozessketten* [Sc00]. Aufgrund der Verfügbarkeit verschiedener, auch freier Modellierungseditoren wird in diesem Beitrag die *Business Process Modell Notation* (BPMN) [OM15] genutzt.

3 Die Methode ProKlaMation im Überblick

Die zentrale Herausforderung der Softwaretechnik ist – generalisierend gesprochen – die Komplexitätsreduktion. Das methodische Mittel zur Komplexitätsreduktion sind Modellierungssprachen der Software-, Projekt- oder Finanzplanung. Diesen ist gemein, dass sie die jeweilige Planung gezielt auf bestimmte Aspekte fokussieren. Die hier vorgestellte Methode ProKlaMation ist darauf gerichtet, Planungssprachen der Informatik so zu verknüpfen, dass die Phasen der Software-Entwicklung authentisch im Unterricht durchlaufen werden können, ohne diesen zu überfrachten.

Das Vorgehensmodell der Methode ProKlaMation orientiert sich grundsätzlich am Wasserfallmodell. Die Tätigkeiten und Ergebnisse der einzelnen Phasen sind hierbei klar voneinander unterscheidbar, bauen aber gleichzeitig aufeinander auf. ProKlaMation ist eine prozessorientierte Methode. Die Prozessmodellierung kommt insbesondere in den frühen Phasen der Softwareentwicklung zum Einsatz und wird (zusätzlich) mit den Modellen späterer Phasen verknüpft. Die Methode unterscheidet sich diesbezüglich in den Phasen Analyse, Prozess-Design und Software-Entwurf von den Vorgehensweisen in gängigen Schulbüchern (siehe oben), weswegen im Folgenden insbesondere auf diese Phasen eingegangen werden soll[3]:

- Gegenstand der Analyse-Phase ist es, einen oder mehrere Abläufe eines Anwendungsbereiches (einer Domäne) mithilfe einer Prozessmodellierungssprache zu modellieren (Ist-Prozessmodell) und dieses insbesondere im Hinblick auf eine Softwareunterstützung zu analysieren.
- In der *Prozess-Design-Phase* wird das Ist-Prozessmodell unter Berücksichtigung der Anforderungen aus der Domäne zu einem Soll-Prozessmodell „reorganisiert“. Dieses berücksichtigt organisationale Verbesserungen und die durch eine Softwareunterstützung induzierten Veränderungen im Prozessablauf.
- In der *Software-Entwurfsphase* werden aus dem Soll-Prozessmodell diejenigen Informationen abgeleitet, die innerhalb des Prozesses benötigt werden und damit von der zu erstellenden Software zu verwalten sind. Das Soll-Prozessmodell wird dazu durch ein UML-Klassendiagramm (Fachmodell) ergänzt. Dieses wird im nächsten Schritt angereichert mit Konzepten, welche für das Funktionieren der Software zuständig sind (z. B. Controller-Klassen oder GUI-Elemente), was zu einem Implementierungsmodell führt.

Abb. 2 zeigt das Phasenmodell und die jeweils genutzten Modelltypen.

[3] Für ein Unterrichtskonzept für die Test-Phase siehe [Sc13].

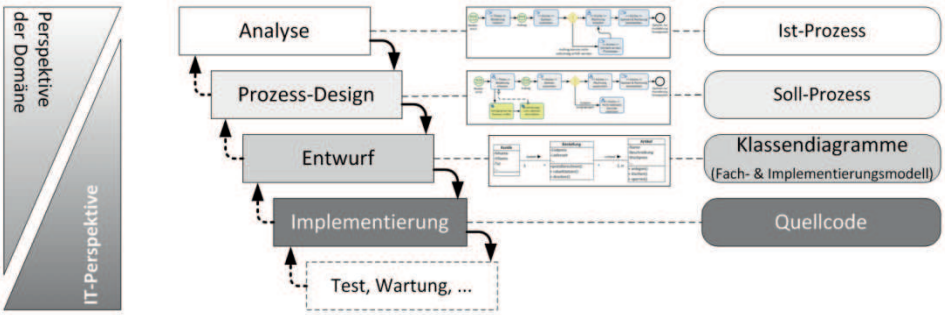

Abb. 2: Phasenmodell und jeweils genutzte Modelltypen

Das Zusammenspiel der Modelle erlaubt eine durchgängige Entwicklung und ein durchgängiges anschauliches Erleben des Softwareentwicklungsprozesses. Die weitestgehend intuitiv verständlichen Sprachelemente der Prozessmodellierung erlauben Schülern ohne besondere Voraussetzung einen schnellen Einstieg in die Softwareentwicklung.

4 Methode an einem Beispiel (Pizzadienst)

Die Anwendung und – damit einhergehend – mögliche schulische Aufgabenstellungen werden im Folgenden anhand eines Demonstrationsbeispiels gezeigt. Die Darstellung fokussiert hierbei auf die Phasen Analyse, Prozess-Design und Implementierung. Ausgangspunkt des Demonstrationsbeispiels sei das Fallbeispiel einer Pizzeria:

Die Pizzeria „Luigi" hat ihr Geschäftsfeld um einen Lieferservice erweitert. Luigi, der Inhaber, ist ein begnadeter Koch aber ein weniger guter Manager. Er organisiert den Lieferservice wie ein stationäres Restaurant: Die Speisen werden per Telefon am Tresen bestellt. Die Bestellannahme und die Lieferung werden von unterschiedlichen Personen übernommen. Rückfragen beim Gast gestalten sich daher schwierig und Liefertouren lassen sich schlecht im Vorfeld planen. Das Geschäftsergebnis erfüllt nicht die Erwartungen. Zudem häufen sich Beschwerden, weil Lieferzusagen nicht eingehalten werden.

4.1 Analyse (Ist -Prozess)

Zweck der Analysephase ist es, die Optionen einer Software-Unterstützung einer Domäne zu ergründen und Anforderungen abzuleiten. In der Analysephase würden Schüler ein Prozessmodell wie das in Abb. 3 zur derzeitigen „Bestellannahme und Speisenzubereitung" entweder selbst erstellen oder in einer Aufgabe als Ist-Prozess erhalten.

Aus dem Ist-Prozessmodell (mit der Fallbeschreibung) lassen sich Vorschläge und Ideen zur Reorganisation des Prozesses und Optionen einer Unterstützung durch Software ableiten. Die Schülerinnen und Schüler übernehmen dabei die Rolle eines externen und somit kritischen IT-Beraters oder Softwareentwicklers. Für dieses Fallbeispiel wäre bspw. festzustellen, dass (1) die Koordination zwischen Theke und Küche verbesse-

rungswürdig ist, insbesondere daraufhin, dass die Auslastung der Küche und ggf. nicht mehr vorhandenen Zutaten schon bei der Bestellung bekannt sein sollten, sowie (2) dass die vielen manuellen Tätigkeiten zu Fehlern und Medienbrüchen bei der Weitergabe der Bestellung und der Rechnungserstellung führen.

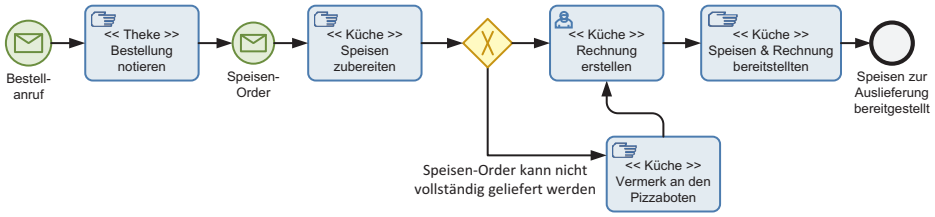

Abb. 3: Modell des Teilprozesses „Bestellannahme und Speisenzubereitung“ (BPMN)

4.2 Prozess-Design (Soll-Prozess)

Auf Basis der Problembeschreibung und Empfehlungen der Analyse-Phase ist ein reorganisierter Prozess zu modellieren (Soll-Prozess), der neben organisatorischen Verbesserungen insbesondere auch durch Software unterstützt werden kann. Abb. 4 zeigt einen möglichen Soll-Prozess unseres Pizzeria-Beispiels.

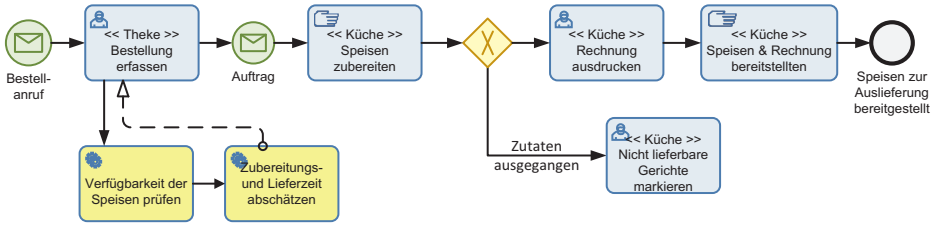

Abb. 4: „Reorganisierter“ Soll-Prozess (BPMN)

Im Unterricht ist es empfehlenswert, zu berücksichtigen, dass konzeptuelle Prozessmodellierungssprachen große Freiheit in der Modellierung erlauben (siehe oben). Es ist daher sinnvoll, alternative Entwürfe eines Soll-Prozesses zu vergleichen und sich im Diskurs auf eine Alternative zu einigen.

4.3 Software-Entwurf (Fachmodell, Implementierungsmodell)

Aus den Einzelschritten eines Prozessmodells lässt sich gut der jeweilige Informationsbedarf ableiten. Aus dem Soll-Prozessmodell sind die im Prozess durch ein Informationssystem zu verwaltenden Daten abzuleiten. Diese sind in einem UML-Klassendiagramm zu spezifizieren (*Fachmodell*). Im genannten Fallbeispiel ist ein Klassendiagramm für ein Informationssystem zu entwerfen, welches das Angebot und die Bestellungen des Pizzadienstes abbildet. Abb. 5 zeigt ein mögliches Fachmodell für den Pizzaservice. Die gestrichelten Kanten deuten hierbei an, in welcher Aktivität welche Daten

erhoben bzw. besonders benötigt werden.

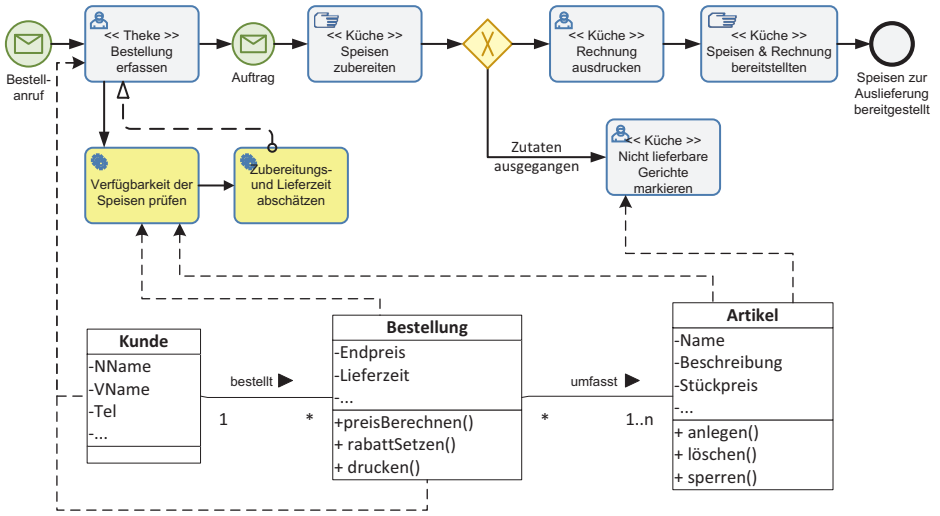

Abb. 5: Um ein Fachmodell (Klassendiagramm) erweitertes Prozessmodell

Fachmodelle sind nicht hinreichend für die Implementierung einer Software. Nun gilt es, technische Details der Implementierung zu bestimmen. Da im schulischen Kontext für gewöhnlich weder über die genutzten Technologien oder Programmiersprachen noch über die Software-Architektur zu bestimmen ist, gilt es, das Fachmodell mit zusätzlichen Software-Artefakten anzureichern, die für den Ablauf der Software von Nöten sind. Hier ist bspw. zu denken an Controller-Klassen, die den Prozessablauf steuern, oder die GUI-Komponenten einer grafischen Oberfläche (*Implementierungsmodell*).

5 Didaktische Einordnung

Die verschiedenen Aufgaben, Tätigkeiten und Ergebnisse der Methode ProKlaMation im Unterricht sind in Tab. 1 aufgelistet. Die präsentierte Methode muss dabei nicht ausschließlich an durchgängigen Beispielen behandelt werden, sondern es lassen sich in jeder Phase einzelne Aufgaben stellen – bspw. zur Modellierung von Fallbeispielen, zur Analyse gegebener Prozesse, zur Ableitung eines Fachmodells aus einer Kombination aus Prozessmodell und Fallbeispiel.

Dadurch, dass die Methode kein durchgängiges Beispiel erfordert, lassen sich Unterrichtsreihen gut phasieren. Ein zeitliches Auseinanderdriften von Schüler-Arbeitsgruppen ist nach Erfahrung des Autors diesbezüglich stets unproblematisch.

Im Folgenden werden Vor- und Nachteile bzw. Herausforderungen der Methode noch einmal eingeordnet. Diese beruhen auf Unterrichtserfahrungen des Autors einerseits sowie auf Erfahrungen und Feedback aus Lehrerfortbildungen, in denen Teile der Methode vorgestellt worden sind.

Phase	Typische Aufgaben / Tätigkeiten im Unterricht	Meilenstein / Ergebnis
Analyse	• Prozessmodelle zu Fallbeispielen erstellen • Prozessmodelle vergleichen	Ist-Prozessmodell
Prozess-design	• Analyse des Ist-Prozessmodells (und ggf. der Fallbeschreibung) hinsichtlich Reorganisation und Softwareunterstützung. • Redesign des Ist-Prozessmodells mit Blick auf dessen Unterstützung durch Software • Prozessmodelle vergleichen • Technologiefolgen an konkreten Beispielen diskutieren.	Soll-Prozessmodell (ggf. mehrere Alternativen)
Software-Entwurf	• Ableitung der Daten, die im Soll-Prozess benötigt werden, und Entwurf eines UML-Klassendiagramms • Verfeinerung des Klassendiagramms um Konzepte, die für die Software von Bedeutung sind. • Modelle vergleichen	Fachmodell Implementierungsmodell

Tab. 1: Didaktische Aufgaben, Tätigkeiten und Meilensteine der Methode ProKlaMation.

5.1 Unterrichtspraktische Anmerkungen

Zur Auswahl von Fallbeispielen:

- Die Arbeit an realistischen Fallbeispielen ist geeignet, die Software-Entwicklungs-Reihe in einen für die Schüler ansprechenden Kontext zu setzen. Allerdings ist an die Auswahl der Fallbeispiele bzw. Kontexte immer die Voraussetzung zu knüpfen, dass sie erstens mit Prozessmodellierungssprachen angemessen zu beschreiben sind, und zweitens, dass es sich um realweltliche Prozesse handelt, die durch Software unterstützt werden können. Dies gilt i. d. R. für Geschäfts-, Produktions- und Verwaltungsprozesse sowie für die Beschreibung komplexerer Software-Anwendungsszenarien (Use Cases).
- Empfehlenswert ist es, bei einem durchgehenden Fallbeispiel Musterlösungen für Meilensteine vorzuhalten, auf welche Schüler aufbauen können, die selbst keine geeigneten Zwischenergebnisse erzielen konnten. Im Unterricht des Autors hat es sich bewährt und als abwechslungsreich herausgestellt, wenn spezifische Aufgaben zu jeder Phase mit ein oder zwei durchgängigen Fallbeispielen gemischt werden.

Zum Umgang mit Prozessmodellierungssprachen:

- Prozessmodellierungstechniken sind für Schüler leicht verständlich, so dass sie grundsätzlich auch erst während der Reihe Softwareentwicklung eingeführt werden können. Es empfiehlt sich aber, die Techniken schon vorher zu nutzen. Insbesondere kann die Sprache BPMN Programm-Ablauf-Pläne ersetzen und so schon während der Algorithmik/Programmierung im Unterricht eingesetzt werden.

- In der Unterrichtspraxis erstellen Schüler häufig unterschiedliche Modelle, die auch unterschiedliche Softwarelösungen nahelegen. Hier kommt der Aufgabe, alternative Schülerlösungen im Diskurs zu vergleichen, eine wichtige Rolle zu.
- Da bei der Nutzung semi-formaler Modellierungssprachen das Abstraktionsniveau nicht strikt vorgegeben ist, ist es hilfreich, Hinweise zu geben, wie detailliert ein Modell zu erstellen ist. Eine Faustregel lautet: So detailliert modellieren, wie dies ein Domänenexperte (hier: der Pizzabäcker) für angemessen halten würde.
- Zur Modellierung von Prozessen sind verschiedene Tools erhältlich – auch kostenlose Werkzeuge (z. B. yEd oder bpmn.io). Leider unterstützen die kostenlosen Werkzeuge aktuell keine Simulation.

5.2 Didaktische Bewertung

Lernsystematisch besehen ist die Software-Entwicklung eine Vertiefung der Algorithmik, der Programmierung und der Modellierung. Schüler mit Schwächen in der Programmierung sind bei programmierintensiven Unterrichtsansätzen der Softwaretechnik schnell überfordert. Prozessmodellierungssprachen allerdings sind für Schüler gut verständlich. Sie erlauben es auch den in der Programmierung schwachen Schülern, sich am Unterricht erfolgreich zu beteiligen.

Die vorgeschlagene Methode ist eine didaktische Reduktion von in der Informatik-Praxis gängigen Ansätzen. Die Methode steht dabei authentisch für die Informatik und ist gleichzeitig ein prototypisches Beispiel für die Software-Entwicklung als solches. Durch die Nutzung verschiedener Modellierungs- bzw. Programmiersprachen in den verschiedenen Phasen der Softwareentwicklung (Prozessmodelle in der Analyse- und Prozess-Design-Phase, Klassendiagramme im Entwurf, eine Programmiersprache in der Implementierung), bleiben die Phasen und deren Herausforderungen für die Schüler klar wahrnehmbar. Gleichzeitig knüpfen die Meilenstein-Ergebnisse der einzelnen Phasen aneinander an, so dass die Durchgängigkeit der Entwicklung erkennbar wird.

Die Einführung und Nutzung einer Software in einem Anwendungskontext verändert diesen vielfach in deutlicher Weise. Abläufe und Organisationsstrukturen sind hiervon betroffen. Die Auswirkungen eines Informationssystems auf dessen Kontext müssen bei dessen Planung berücksichtigt werden. Nach den Erfahrungen des Autors erhalten Schülerinnen und Schüler über die Anwendung der Methode ein gutes Gespür für die Herausforderungen der Analysephase und für die Auswirkungen von Software (allgemeiner Technologien) auf Anwendungsfälle.

6 Zusammenfassung und Ausblick

Die Praxis der Softwaretechnik stellt nicht die Programmierung in den Mittelpunkt. Die Kernherausforderungen sind fachlicher und organisationaler Natur. Fachlich gilt es, schrittweise einen Software-Entwurf zu erstellen (Datenmodell, Klassendiagramm, Spezifikation). Dies geschieht auf Grundlage einer methodischen Analyse des Gegenstands-

bereiches, der sog. „Domäne“ einer Software. Größere Softwareprojekte umfassen viele Beteiligte mit unterschiedlichen Interessen und fachlichen Hintergründen. Organisational besteht die Herausforderung im Projektmanagement, um Einigungsprozesse zwischen den Interessengruppen herzustellen und eine klare Aufgabenverteilung zu ermöglichen. Für beide Herausforderungen stellt die Softwaretechnik Modellierungssprachen bereit, die aus verschiedenen Perspektiven die Software oder deren Domäne beschreiben.

Die vorgestellte Methode der Softwareentwicklung im Unterricht greift die zentrale Rolle der Prozessmodellierung in der betrieblichen Praxis auf. Modellierungskonzepte werden phasenübergreifend und durchgängig angewendet. Die Herausforderungen der Phasen der Softwareentwicklung werden erlebbar und die Prozessmodelle werden im Unterricht selbst als Kommunikationsmedium zum Vergleich von geänderten und durch neue Software transformierte Abläufe („mögliche Welten“) genutzt.

Literaturverzeichnis

[Br09] Brichzin, P. et.al.: Informatik Oberstufe 1 – Datenstrukturen und Softwareentwicklung. Oldenbourg: München et al. 2009.

[Du09] Duden-Verlag (Hg.): Informatik – Objektorientierte Programmierung mit BlueJ. Duden: Berlin, Mannheim, 2009.

[En06] Engelmann, L. (Hg.): Informatik – Gymnasiale Oberstufe. Duden: Berlin 2006.

[Fi06] Fischer, H. et.al.: Grundlagen der Informatik II. Oldenbourg: München et al. 2006.

[Fr11] Frank, U.: MEMO Organisation Modelling Language (2): Focus on Business Processes. ICB-Research Report Nr. 49, Institut für Informatik und Wirtschaftsinformatik, Universität Duisburg-Essen, 2011. (www.icb.uni-due.de/ fileadmin/ICB/research/research_reports/ICB-Report-No49.pdf; 31.01.2015)

[Ha10] Hattenhauer, R.: Informatik für Schule und Ausbildung. Pearson: München et al. 2010.

[Hu09] Hubwieser, P. et.al.: Informatik 4 – Lehrwerk für Gymnasien – Rekursive Datenstrukturen, Softwaretechnik. Ernst Klett: Stuttgart 2009.

[KM04] Kultusministerkonferenz (Hg.): Einheitliche Prüfungsanforderungen Informatik. Beschluss vom 01.12.1989 i. d. F. von 05.02.2004. (www.kmk.org/fileadmin/ veroeffentlichungen_beschluesse/1989/1989_12_01-EPA-Informatik.pdf; 31.1.2015)

[OM15] Object Management Group (Hg.): Business Process Model and Notation. (www.bpmn.org; 31.01.2015)

[Sc00] Scheer, A.-W.: ARIS: Business Process Modeling. 3. Aufl., Springer: Berlin et al., 2000.

[Sc13] Schauer, H.: Debugging-Aufgaben – Programmfehler als Lernchance. In: Breier, N. et.al. (Hg.): INFOS 2013: Informatik erweitert Horizonte – 15. GI-Fachtagung Informatik und Schule. Lecture Notes in Informatics Nr. P-219, Gesellschaft für Informatik, Bonn, 2013, S. 97 – 106.