Herausgeber von *bwp@* : Karin Büchter, Martin Fischer, Franz Gramlinger, H.-Hugo Kremer und Tade Tramm

Berufs- und Wirtschaftspädagogik - *online*

Norbert SEULBERGER
(Ausbilder, Altenholz)

Fachmethoden als Ausbildungsmethoden – Erfahrungslernen am Arbeitsplatz im Ausbildungsberuf Fachinformatiker/-in Anwendungsentwicklung

Online unter:

http://www.bwpat.de/ausgabe28/seulberger_bwpat28.pdf

in

bwp@ Ausgabe Nr. **28** | Juni 2015

Berufliche Lehr-Lernforschung

Hrsg. v. **Tade Tramm, Martin Fischer & Carmela Aprea**

www.bwpat.de | ISSN 1618-8543 | *bwp@* 2001–2015

bwp@

www.bwpat.de

ABSTRACT (SEULBERGER 2015 in Ausgabe 28 von ***bwp@***)

Online: http://www.bwpat.de/ausgabe28/seulberger_bwpat28.pdf

Die Berufsausbildung hat nicht nur die Vermittlung der notwendigen beruflichen Fertigkeiten, Kenntnisse und Fähigkeiten zum Ziel, sondern muss darüber hinaus den Erwerb der erforderlichen Berufserfahrung ermöglichen (BBIG, §1 (3)). Erfahrungslernen am Arbeitsplatz eröffnet die Möglichkeit, die Berufspraxis zu erleben.

Das Berufsbild des Ausbildungsberufs Fachinformatiker/-in, Fachrichtung Anwendungsentwicklung, wird wesentlich von einem zentralen Produktionsprozess, dem Softwareentwicklungszyklus, bestimmt. Das neuere Vorgehen der agilen Softwareentwicklung gründet auf einem ganzheitlichen Ansatz, der alle Kompetenzfelder und Lernbereiche anspricht, sodass die Nutzung von Fachmethoden der Anwendungsentwicklung zur Gestaltung des Erfahrungslernens am Entwicklerarbeitsplatz nahe liegt.

Der vorliegende Artikel identifiziert und beschreibt verschiedene Fachmethoden und integriert diese in ein Lehr-Lern-Arrangement am Arbeitsplatz. Abweichend von häufig anzutreffenden Szenarien, die den beruflichen Alltag in Gruppenarbeit von Auszubildenden untereinander nachstellen, erlebt der Auszubildende die berufliche Realität in der Zusammenarbeit mit erfahrenen Kollegen.

Bei der Methodenwahl spielen neben der lernförderlichen Gestaltung des Arbeitsplatzes die Fachwissenschaft Softwareengineering und ihre praktischen Umsetzung in den Betrieben eine wesentliche Rolle. Einen wichtigen Zwischenschritt stellt die Klärung dar, welche Inhalte die Berufspraxis am Arbeitsplatz prägen.

Es handelt sich um einen Vorschlag, der dem betrieblichen Ausbilder ein pragmatisches, gleichwohl fachlich vollwertiges wie didaktisch ausgewogenes Vorgehen an die Hand geben soll.

Specialist techniques as training methods – experiential on-the-job learning in vocational training for IT / application development specialists

The aim of vocational training is not only to impart the necessary professional skills, knowledge and capabilities but, additionally, to facilitate acquisition of the requisite vocational experience (Germany's Vocational Training Act/BBIG, §1 (3)). Experiential on-the-job learning opens up the possibility to gain work experience.

The job profile of an IT / application development specialist is determined largely by a central production process, referred to as the software development cycle. The more recent approach of agile software development is holistic and addresses all fields of expertise and learning, such that it seems reasonable to use specialist application development techniques for the purpose of experiential learning at developers' workplaces.

This article identifies and describes different specialist methods and integrates them into an on-the-job teaching/learning arrangement. Departing from frequently encountered scenarios that simulate the occupational routine of trainee group work, the trainee undergoes vocational reality in working together with experienced colleagues.

The specialist science of software engineering and its practical implementation in enterprises, in addition to a workplace layout conducive to learning, play a key role where the choice of methods is concerned. Clarifying what content matter has a formative part in on-the-job work experience represents an important intermediate step.

This is a proposal designed to offer in-company instructors a pragmatic and, at the same time, technically complete as well as didactically balanced approach

NORBERT SEULBERGER (Ausbilder, Altenholz)

Fachmethoden als Ausbildungsmethoden – Erfahrungslernen am Arbeitsplatz im Ausbildungsberuf Fachinformatiker/-in Anwendungsentwicklung

1 Einleitung

Erfahrung bezeichnet erworbene Fähigkeiten, Fertigkeiten oder Erkenntnisse, die eine Bewältigung von Lebenssituationen ohne expliziten Rückgriff auf theoretisches Wissen erlauben. Naturgemäß wird Erfahrung nicht durch intentionales Lehren, sondern durch informelles Lernen erworben (vgl. Schelten 2005, 173ff.).

In der Berufsausbildung erlebt der Auszubildende die erforderliche Berufserfahrung in der tagtäglichen Beschäftigung mit Aufgaben und Kollegen seines Berufs. Dabei muss sich Erfahrungslernen am Arbeitsplatz „durch einen besonders intentional angelegten pädagogischen Prozess auszeichnen“ (Schelten 2005, 183). Andernfalls droht eine reine „Anpassungsqualifizierung“ (Dehnbostel 2008, 7), die „das Lernen in der Arbeit auf den unmittelbaren betrieblichen Nutzen verengt und von rein zweckgerichtetem Handeln bestimmt wird“ (Dehnbostel 2008, 8).

Der vorliegende Artikel behandelt die Fragestellung, wie Erfahrungslernen am Arbeitsplatz im Ausbildungsgang Fachinformatik Anwendungsentwicklung gestaltet werden kann. Da der zentrale Leistungsprozess, der sogenannte Softwareentwicklungszyklus, und seine Teilprozesse von in der Theorie des Softwareengineerings umfassend beschriebenen Fachmethoden unterstützt werden, liegt der Ansatz nahe, Fachmethoden als Ausbildungsmethoden zu nutzen. Die Fokussierung auf diesen einzelnen IT-Beruf begründet sich aus der Prozess- und Methodenspezialisierung des Fachgebiets.

Das Vorgehen orientiert sich an den z. B. bei Bonz beschriebenen Grundsätzen der Methodenwahl in der Berufsbildung, insbesondere in der betrieblichen Berufsausbildung. Die Auswahl der Methoden setzt die Analyse des didaktischen Feldes voraus (vgl. Bonz 2009, 9ff. sowie 247ff.). Neben Ziel und Inhalt sind typische Rahmenbedingungen wie z. B. curriculare Vorgaben, das Berufsfeld oder betriebliche Rahmenbedingungen zu beachten.

Das Ziel, die Berufspraxis am realen Entwicklerarbeitsplatz erfahrbar zu machen, bestimmt die fachlichen Inhalte: Der Auszubildende erlebt den zentralen Produktionsprozess der Anwendungsentwicklung in allen Teilprozessen und typischen Tätigkeiten.

Der Artikel beleuchtet daher folgende Aspekte:

1. Wie lässt sich der Entwicklungszyklus sowohl allgemeingültig als auch im konkreten betrieblichen Umfeld interpretierbar beschreiben? Welche Teilprozesse erfordern eigene Fachmethoden?

2. Wie können Fachmethoden mit berufs- und arbeitspädagogischem Potential identifiziert werden?
3. Wie ist deren berufs- und arbeitspädagogische Eignung zu bewerten?

Die folgenden Rahmenbedingungen bilden dabei nicht nur notwendige Voraussetzungen, sondern dienen zusätzlich der Identifikation und Bewertung der Fachmethoden. Dabei werden berücksichtigt:

1. die lern- und kompetenzförderliche Gestaltung des Arbeitsplatzes,
2. Kompetenzfelder und Lernbereiche,
3. curriculare Rahmenbedingungen,
4. individuelle und betriebliche Voraussetzungen,
5. der Ausbildungsberuf und das Berufsfeld.

Weil gezielt Fachmethoden gesucht werden, nimmt der letztgenannte Punkt, für den sowohl empirische Ergebnisse als auch die Theorie des Fachgebiets Anwendungsentwicklung betrachtet werden, besonderen Raum ein.

Der Artikel unterbreitet einen Vorschlag, wie der betriebliche Praktiker, der überwiegend selbst als Anwendungsentwickler arbeitet, nur anteilig ausbildet und lediglich über die berufs- und arbeitspädagogische Qualifikation gemäß AEVO verfügt (zur Ausbildung der Ausbilder vgl. BIBB 2009 und Jacobs/Preuße 2013), ein Lern-Arrangement für Erfahrungslernen am Entwicklerarbeitsplatz, das sich wesentlich auf Fachmethoden stützt, entwickeln, anpassen und anwenden kann.

2 Rahmenbedingungen

2.1 Lern- und kompetenzförderliche Gestaltung des Arbeitsplatzes

Notwendige Voraussetzung für ein Erfahrungslernen am Arbeitsplatz ist eine lern- und kompetenzförderliche Gestaltung der Tätigkeiten, die sich durch folgende Merkmale auszeichnet (vgl. Dehnbostel 2008, 6).

Tabelle 1: **Kriterien lern- und kompetenzförderlicher Arbeit** (Dehnbostel 2008, 6)

Kriterien	**Kurzcharakteristik**
Vollständige Handlung/ Projektorientierung	Aufgaben mit möglichst vielen zusammenhängenden Einzelhandlungen im Sinne der vollständigen Handlung und der Projektmethode
Handlungsspielraum	Freiheits- und Entscheidungsgrade in der Arbeit, d. h. die unterschiedlichen Möglichkeiten, kompetent zu handeln (selbstgesteuertes Arbeiten)
Problem-, Komplexitätserfahrung	Ist abhängig vom Umfang und der Vielschichtigkeit der Arbeit, vom Grad der Unbestimmtheit und Vernetzung
Soziale Unterstützung/ Kollektivität	Kommunikation, Anregungen, Hilfestellungen mit und durch Kollegen und Vorgesetzte; Gemeinschaftlichkeit

Individuelle Entwicklung	Aufgaben sollen dem Entwicklungsstand des Einzelnen entsprechen, d. h., sie dürfen ihn nicht unter- oder überfordern
Entwicklung von Professionalität	Verbesserung der beruflichen Handlungsfähigkeit durch Erarbeitung erfolgreicher Handlungsstrategien im Verlauf der Expertiseentwicklung (Entwicklung vom Novizen bis zum Experten)
Reflexivität	Möglichkeiten der strukturellen und Selbstreflexivität

Die Erfüllung der Kriterien wird von folgenden Voraussetzungen an das Lernumfeld unterstützt:

1. Das Lehr-Lern-Arrangement ermöglicht die Ergänzung um formelle Lehr-Lern-Formen, ohne selbst formalisiert zu werden und seine charakteristischen Merkmale zu verlieren (vgl. Dehnbostel 2008, 7). „Eine intellektuelle und kognitive Durchdringung findet zusätzlich durch entsprechend angeleitete Reflexion und Verarbeitung der Arbeits- und Lernerfahrung statt sowie durch die Vermittlung von erforderlichem Hintergrund- und Vertiefungswissen“ (Schaper 2004, 5).
2. Der Lernende unterliegt keinen quantitativen Produktionsvorgaben („Soll-Mengen-Druck“, vgl. Schaper 2004, 5).

Während der Entwicklungs- und Erprobungsphase ist eine formative Evaluation des vorgeschlagenen Lehr-Lern-Arrangements mittels strukturierter Interviews oder Fragebögen, die sowohl die Ausbildungsbeauftragten als auch den Auszubildenden befragen, erforderlich (vgl. Schaper 2004, 10).

2.2 Kompetenz- und Lernbereiche

Eine wesentliche Anforderung für erfolgreiches Erfahrungslernen bildet die ganzheitliche Förderung aller Kompetenzfelder (vgl. Schaper 2004, 7). Während Fach- und Methodenkompetenz durch eine geschickte Wahl von Fachmethoden adressiert werden können, ist die Entwicklung von Eigen- und Sozialkompetenz in einem technischen Beruf nicht offensichtlich. „Die weitverbreitete Ansicht, Softwareentwicklung sei in erster Linie eine technische Aufgabe, entpuppt sich bei näherer Betrachtung des Entwicklungsprozesses als nur die halbe Wahrheit. Softwareentwicklung ist heutzutage auch ein komplexer sozialer Prozeß.“ (Oestereich 1998, 18). Inzwischen werden die sozialen Faktoren stark betont, z. B. im populären Agilen Manifest: „Individuals and interactions over processes and tools“ (Beck et al. 2001). Der in seiner Eigen- und Sozialkompetenz eingeschränkte Nerd ist ein Zerrbild, das dem tatsächlichen Berufsbild nicht entspricht. Paar- oder Gruppenarbeit als Sozialformen sind weit verbreitet (vgl. Brüggemann/Dehnbostel/Rohs 2010, 20ff.).

Anwendungsentwicklung ist unbestreitbar eine kognitive Tätigkeit. Agilität adressiert zusätzlich emotionale Aspekte, indem Wert propagiert und eine Haltung eingefordert werden. „[…] a style that celebrates a consistent set of values that serve both human and commercial needs […]” (Beck 1999, 29).

„Wahrscheinlich findet man in kaum einer anderen Ingenieurdisziplin so viel dilettantisches Vorgehen wie bei der Softwareentwicklung“ (Oestereich 1998, 22). Pragmatische Fertigkeiten, also das Vermögen, kognitives Wissen praktisch anzuwenden, bilden den Ausgangspunkt der Craftsmanship-Initiative (vgl. Martin et al. 2009). Die Ideen der Agilität werden um die Forderung nach handwerklicher Exzellenz und Methoden zu deren Erreichung ergänzt. Die „Clean Code“-Prinzipien (vgl. Martin 2009) bilden einen Katalog an Expertenerfahrungen, die in der täglichen Arbeit am Arbeitsplatz eingeübt werden müssen.

2.3 Curriculare Vorgaben: Ausbildungsordnung und Rahmenplan

Verbindliche Quelle für die Erstellung eines betrieblichen Ausbildungsplans bildet die Ausbildungsordnung, insbesondere der zugehörige Rahmenplan. Für Fachinformatik Anwendungsentwicklung werden die Lernziele, die sich auf Anwendungsentwicklung beziehen, in § 10 Ausbildungsberufsbild, dort schwerpunktmäßig in den laufenden Nummern

- 5. Herstellen und Betreuen von Systemlösungen,
- 6. Systementwicklung,
- 8. Informations- und telekommunikationstechnische Systeme,
- 9. Kundenspezifische Anwendungslösungen,

formuliert (vgl. BMWi 1997, 1745 sowie 1764ff.).

Bei der Umsetzung der Vorgaben in praktische Ausbildungsmaßnahmen gibt es große Schwierigkeiten. „Die […] Ausbildungsrahmenpläne sind aus didaktischer Sicht eher nach einem Mischkonzept mit nach wie vor teils ausgeprägter Fachsystematik konzipiert. Das Ergebnis ist, dass viele Betriebe […] die Vorgaben so gut wie gar nicht als unterstützendes Dokument für die Ausbildungsplanung und -umsetzung verstehen“ (Petersen/Wehmeyer 2003, 310). Sowohl die sachliche als auch die zeitliche Gliederung des Rahmenplans brechen die Themen so weit herunter, dass die Geschäfts- und Arbeitsprozesse kaum erkennbar sind (vgl. Petersen/Wehmeyer 2003, 307/310). Angesichts des konzeptionellen und technologischen Wandels seit 1997 dürfte der Rahmenplan eine inhaltliche Aktualisierung benötigen.

Auch ergänzende Materialien, die die Ausbildungsinhalte an Technologien oder Aufgabenstellungen exemplarisch illustrieren (z. B. Bleßmann/Büttner/Dax 2011; Kersken 2013), geben keine systematische Hilfestellung.

Im Weiteren entnehmen wir den curricularen Inhalten keine Anregung zur Gestaltung des Lern-Arrangements am Arbeitsplatz. In 4.3 wird die Erfüllung der curricularen Vorgaben belegt.

2.4 Betriebliche und individuelle Rahmenbedingungen

In der Fachdisziplin Anwendungsentwicklung sind einige Aspekte in der betrieblichen Praxis sehr verbreitet anzutreffen, so dass folgende Rahmenbedingungen sinnvolle Annahmen bilden.

1. Für jede Teildisziplin des Softwareengineerings gibt es jeweils eine begrenzte Anzahl an Industriestandards für Konzepte, Methoden und Technologien. Beispielsweise ist das derzeit und auf absehbare Zeit in der betrieblichen Praxis wichtigste Programmierparadigma die Objektorientierung. Bei den Projektmanagementmethoden dominieren inzwischen iterativ-inkrementelle Vorgehensmodelle, bei denen gleiche Teilprozessabfolgen mehrfach wiederholt werden, wobei das entstehende Produkt jeweils ein Stück wächst.

2. Anwendungsentwicklung findet in der Regel in einer Projektorganisation und -struktur statt, die nur teilweise den Charakter einer wirklichen Projektgruppe besitzt und viele Merkmale einer teilautonomen Arbeitsgruppe aufweist (zur Abgrenzung der Begriffe vgl. Antoni 1994, 33ff.), da das Team meist als funktionale Einheit der regulären Organisationsstruktur zusammenwirkt und die Rahmenbedingungen der einzelnen Entwicklungsvorhaben ähnlich sind.

3. Ein Softwareprojekt wird von einem Entwicklungsteam durchgeführt. Eine typische Teamgröße beträgt fünf bis zwölf Personen. Die Teammitglieder nehmen eine oder mehrere Rollen wahr, oft üben mehrere Personen eine funktionale Rolle gemeinsam aus. Ein Wechsel zwischen verschiedenen Rollen ist für einzelne Mitarbeiter eingeschränkt, aber nicht beliebig möglich, weil die Ausübung der einzelnen Rollen spezielle Kompetenzen erfordert.

4. Häufig gibt es in einem Entwicklungsteam nur einen oder zwei Auszubildende. Sobald der Auszubildende über solide Kenntnisse der im Projekt eingesetzten Programmiersprache und einen souveränen Umgang mit der Entwicklungsumgebung verfügt, kann er sinnvoll an einem Softwareentwicklungsprojekt teilnehmen. Das sollte in der Regel ein halbes Jahr nach Ausbildungsbeginn gegeben sein.

5. Der Ausbilder nimmt als Anwendungsentwickler an dem Projekt teil, in dem der Auszubildende eingesetzt wird, und erhält für die Ausbildung ein angemessenes, gegenüber konkurrierenden Bedarfen geschütztes Zeitbudget. Weil der Ausbilder die fachliche Breite der Anwendungsentwicklung nicht allein abdecken kann, wird er von Ausbildungsbeauftragten im Team unterstützt.

Im Gegensatz zu den vorgenannten Punkten 3-5 wird Erfahrungslernen am Arbeitsplatz in der Ausbildung häufig als Gruppenarbeit in folgendem Rahmen skizziert:

Mehrere Auszubildende aus verwandten Berufsgruppen organisieren selbstständig die Tätigkeiten in einem Qualifizierungsstützpunkt („Lerninsel“) von mehrwöchiger Dauer. Ein Aus-

bildungsbeauftragter steht als Lernberater unterstützend zur Verfügung. Die Auszubildenden entstammen überwiegend dem dritten Ausbildungsjahr und sind in der Lage, sich gegenseitig einzuarbeiten, nehmen also sowohl die Rolle von Lernenden als auch Lehrenden ein (vgl. Schaper 2004, 8). Für dieses Lern-Arrangement bieten sich die Projektmethode „als Idealform von handlungsorientierten Methoden“ (Bonz 2009, 121), alternativ die Leittextmethode an (vgl. Bonz, 213/216f.).

Ein Softwareprojekt als reines Ausbildungsprojekt in dem eben umrissenen Rahmen durchzuführen, birgt in der Praxis erhebliche Gefahren: Die berufliche Praxis kann falsch oder verzerrt erlebt werden, die erforderliche fachliche Tiefe wird nicht erreicht, Anwendungsentwicklung auf reines Codieren reduziert.

Daher liegt dem Weiteren folgendes Szenario zugrunde:

Der Auszubildende nimmt als Junior-Entwickler in einem erfahrenen Entwicklerteam an einem realen Entwicklungsprojekt teil, das den typischen Softwareentwicklungszyklus in einem iterativ-inkrementellen Vorgehensmodell durchläuft. Er übt dort alle Tätigkeiten gemäß dem jeweils aktuellen Ausbildungsstand und anhand geeigneter Betreuungs- und Sozialformen aus (zum Begriff Junior-Entwickler vgl. Tabelle 4: **Tätigkeiten eines Junior-Entwicklers**).

2.5 Das Berufsfeld Anwendungsentwicklung

2.5.1 Empirische Ergebnisse

Von 1999 bis 2004 wurde am Berufsbildungsinstitut Arbeit und Technik (biat) im Auftrag des Bundesinstituts für Berufsbildung (BIBB) eine Studie zur Evaluation der (damals neuen) IT-Berufe durchgeführt. Das Ziel der Studie bestand darin, die im Jahr 1997 neu gefassten IT-Ausbildungsberufe hinsichtlich der inhaltlichen Schneidung und Abgrenzung sowie der Akzeptanz durch die Ausbildungsbetriebe zu beleuchten. Den Untersuchungsergebnissen liegt eine bundesweite Befragung von zahlreichen Unternehmen und Auszubildenden zugrunde (vgl. Petersen/Wehmeyer 1999-2004).

Für das Teilprojekt 2 – Fallstudien wurden in ausgewählten Betrieben Expertenbefragungen durchgeführt, um konkrete IT-Geschäfts- und Arbeitsprozesse zu identifizieren, zu beschreiben und deren betriebliche Umsetzungen der IT-Ausbildung zu untersuchen (vgl. Petersen/Wehmeyer 2003, 3f.). Die fachliche Verallgemeinerung der in den Fallstudien ermittelten IT-Geschäfts- und Arbeitsprozesse erfolgte über eine Beschreibung in der GAHFA-Modellstruktur (IT-**G**eschäftsfeld, IT-**A**rbeitsfelder, IT-**H**andlungs**f**elder, IT-**A**rbeitsaufgaben (vgl. Petersen 2003)).

Im Berufs-Geschäftsfeld Fachinformatik wird das für das Berufsbild Fachinformatiker/in Anwendungsentwicklung zentrale Arbeitsfeld „IT-System- und Anwendungsentwicklung “ in sechs Handlungsfelder ähnlicher Komplexität unterteilt, die durch die in der rechten Spalte

der folgenden Tabelle beschriebenen Arbeitsaufgaben illustriert werden (vgl. Petersen/Wehmeyer 2003, 318):

Tabelle 2: **Anwendungsentwicklung – Handlungsfelder und Arbeitsaufgaben** (Petersen/Wehmeyer 2003, 318)

Handlungsfeld	**Arbeitsaufgaben**
Analyse und Beratung	IT-System- und Anwendungsanforderungen in Abstimmung mit Kollegen, dem Kunden oder Nutzern aus technischer Sicht analysieren, bewerten und beschreiben
	Konkrete Hard- und Software-Anforderungen bestimmen und spezifizieren
	Passend und angepasste IT-Lösungen empfehlen, beschreiben und präsentieren
System- und Arbeitsplanung	Eigene Arbeits- und Projektprioritäten sowie Arbeitsaufgaben auf der Basis des Arbeits- und Projektplans festlegen und beschreiben
	Realisierung von IT-Lösungen im Abgleich mit den Planungen bewerten und Verbesserungen vorschlagen
	IT-System- und Software-Entwicklungsumgebungen testen, auswählen und einrichten
Konzeption und Modellierung	System- und Softwarearchitekturen (Design-Modell, Entwurfsmuster, Framework) nachvollziehen sowie anwendungsbezogen mitgestalten
	SW-Entwicklungskonzepte, Verfahren, Algorithmen, Datenstrukturen und Daten(bank)-Modelle analysieren und mit Blick auf die Anwendung begründen
	Standards der objektorientierten Modellierung und des Entwurfs (i. d. R. UML) aufgabenbezogen einsetzen sowie bestehende Entwürfe analysieren und anpassen
Programmierung und Implementierung	Software-Bausteine bzw. Module, Schnittstellen, Attribute, E/A-Parameter etc. analysieren und festlegen
	SW- und Datenbank-Anwendungen implementieren, anpassen, kompilieren und dokumentieren
	Benutzer- und Bedienoberflächen sowie web-basierte Lösungen gestalten, programmieren und einbinden
Softwareanpassung und Test	Testabläufe und Testfälle (Test-Design) für den Komponententest festlegen und koordinieren
	Software- und Anwendungstest (Unit-, GUI-Tests etc.) durch Nutzung passender und verfügbarer Testmethoden, Tools und Daten durchführen und interpretieren
	Möglichkeiten der Fehlerbehebung und Softwareanpassung beschreiben und grundlegende Fehler durch Änderung, Erweiterung oder Anpassung beseitigen bzw. debuggen
Konfigurations-management und Dokumentation	Installations- und Nutzeranweisungen sowie Teile von System- und Applikationshandbüchern verfassen und verwalten
	System- und Software-Anwendungen dokumentieren, registrieren und versionieren (Konfigurations- und Reportmanagement)
	Arbeitsaufgaben und Ressourcen nach allgemeinen und den spezifisch betrieblichen Standards dokumentieren

Die Handlungsfelder bilden einen guten Ansatz, den Softwareentwicklungszyklus und seine Teilprozesse in sinnvoller Abstraktion zu erfassen, weichen aber von der Terminologie und dem Zuschnitt in der Fachdisziplin Softwareengineering ab. Beispielsweise bleibt der Begriff

„Beratung“ vage, „Dokumentation“ sollte in alle Teilgebieten und Projektphasen erfolgen. „Softwareanpassung und Test“ verbindet Tätigkeiten, die eher eigenständig betrachtet werden sollten.

Im Folgenden wird daher der grundsätzlich zielführende Ansatz, Anwendungsentwicklung in thematisch konsistente Teilgebiete zu unterteilen, innerhalb der Systematik des Softwareengineerings beschrieben.

2.5.2 Die Fachdisziplin Softwareengineering

Softwareengineering ist das Teilgebiet der praktischen Informatik, das sich mit der Entwicklung und Wartung von Software beschäftigt. Es handelt sich um eine Ingenieurwissenschaft bzw. deren praktische Anwendung (vgl. IEEE 2014, xxxi). Man könnte Softwareengineering als theoretische Grundlage der Anwendungsentwicklung oder als Synonym für Anwendungsentwicklung bezeichnen.

Eine gute Übersicht über die Gegenstände und Teilgebiete des Softwareengineerings bietet SWEBOK© V3.0, Guide to the Software Engineering Body of Knowledge (vgl. IEEE 2014). In SWEBOK wird das Softwareengineering in folgende Wissensgebiete („Knowledge Area“, KA) unterteilt:

- Software Requirements
- Software Design
- Software Construction
- Software Testing
- Software Maintenance
- Software Configuration Management
- Software Engineering Management
- Software Engineering Process
- Software Engineering Models and Methods
- Software Quality
- Software Engineering Professional Practice

Dazu kommen die Wissensbereiche Software Engineering Economics, Computing Foundations, Mathematical Foundations sowie Engineering Foundations. Eine ähnliche Systematik besitzen die Norm ISO 12207 zur Definition der Prozesse im Lebenszyklus einer Anwendung (vgl. ISO 2008, 14) oder Reifegradmodelle wie CMMI® for Development (vgl. Chrissis/Konrad/Shrum 2007, 33), wobei Anzahl und Zuschnitt der Teilgebiete jeweils etwas unterschiedlich ist.

Zusammenfassend lassen sich folgende Teilgebiete abstrahieren:

Tabelle 3: **Teilgebiete der Anwendungsentwicklung**

Teilgebiet der Anwendungsentwicklung	**SWEBOK**	**biat 2003**
Softwarenahes Projektmanagement, Vorgehensmodell	Software Engineering Management, Software Engineering Process	Arbeitsplanung
Anforderungsmanagement	Software Requirements	Analyse und Beratung (Dokumentation)
Softwaredesign	Software Design	Konzeption und Modellierung
Programmierung	Software Construction, Software Maintenance	Programmierung und Implementierung, Softwareanpassung (Dokumentation)
Softwaretest	Software Testing	Test
Entwicklungsinfrastruktur und Softwarekonfigurationsmanagement	Tools jeweils KA-bezogen, Software Configuration Management	Konfigurationsmanagement (Dokumentation)

Ein Junior-Entwickler (vgl. 2.4 Betriebliche und individuelle Rahmenbedingungen) übt in den Teilgebieten jeweils folgende Tätigkeiten aus:

Tabelle 4: **Tätigkeiten eines Junior-Entwicklers**

Teilgebiet	**Tätigkeiten eine Junior-Entwicklers**
Vorgehensmodell	Mitwirkung an Projektplanung, -durchführung und -controlling
Anforderungsmanagement	Mitwirkung an der Anforderungsermittlung und -dokumentation
Softwaredesign	Mitwirkung an Entwurf und Dokumentation der Softwarearchitektur
Programmierung	Erstellen, Prüfen, Ändern und Dokumentieren des Programmcodes
Softwaretest	Erstellung und Durchführung von Entwicklertests
Entwicklungsinfrastruktur	Verwaltung von Programmcode und Arbeitsaufgaben, Mitwirkung an der Versions- und Releaseverwaltung

2.5.3 Der Softwareentwicklungszyklus

Die Teilgebiete stehen bei einem iterativ-inkrementellen Vorgehen in komplexen Beziehungen, die stark vereinfacht wie folgt illustriert werden können:

Die Entwicklungsinfrastruktur und das Konfigurationsmanagement bilden die Basis, in der alle entstehenden Artefakte (Programmtexte, Anforderungsdokumente, Konfigurationsdateien, Architekturmodelle, Dokumentation etc.) verwaltet werden. Diese Artefakte entstehen im zyklischen Durchlaufen von Anforderungsmanagement, Softwaredesign, Programmierung und Softwaretest. Das Vorgehensmodell bildet das Regelwerk, wie die anderen Tätigkeiten miteinander verwoben sind und durchlaufen werden.

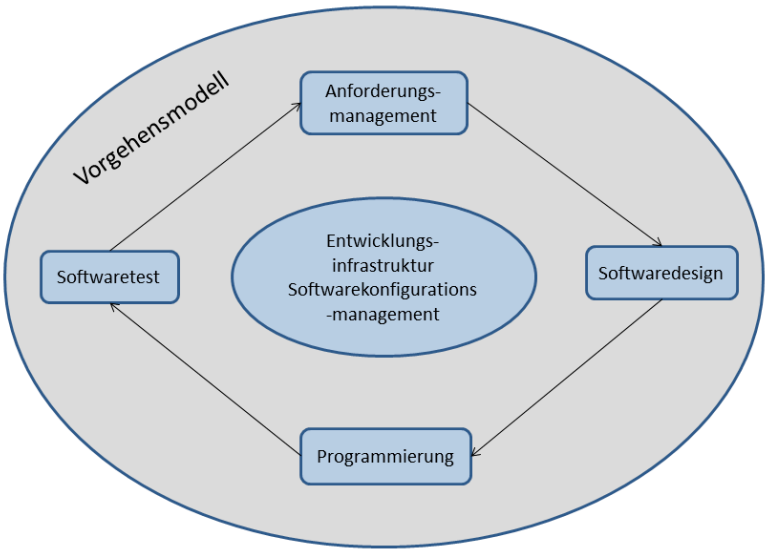

Abbildung 1: Der Softwareentwicklungszyklus (eigene Darstellung)

3 Fachmethoden

Für jedes Teilgebiet werden SWEBOK (vgl. IEEE 2014) ein oder mehrere Fachmethoden entnommen:

Tabelle 5: **Ausbildungsgeeignete Fachmethoden der Anwendungsentwicklung**

Fachmethode	**Teilgebiet**	**Fundstelle in SWEBOK**
Scrum	Vorgehensmodell	Software Engineering Models and Methods 9-9
Pair programming	Programmierung	Software Engineering Models and Methods 9-9 Software Engineering Professional Practice 11-10
Refactoring	Programmierung	Software Maintenance 5-10 Reengineering Software Engineering Models and Methods 9-9
Komponententest	Programmierung, Softwaretest	Software Construction 3-7 Software Testing 4-5 Software Quality 10-3
Objektorientierte Analyse (OOA)	Anforderungsmanagement	Software Engineering Models and Methods 9-7
Objektorientiertes Design (OOD)	Softwaredesign	Software Design 2-11 Software Engineering Models and Methods 9-7
Continuous integration	Entwicklungsinfrastruktur, Softwarekonfigurationsmanagement	Software Configuration Management 6-3

Das agile Vorgehensmodell **Scrum** besitzt folgende Merkmale:

- kurze Entwicklungszyklen (zwei- bis vierwöchige Sprints), deren Ablauf sich unter Fortschreiten des Produktumfangs mehrfach wiederholt (iterativ-inkrementell),
- ein einfaches Rollenmodell (Product owner als Vertreter des Kunden bzw. des Produktes, Scrum-Master als unterstützender Projektmoderator, das teilautonome Entwicklerteam),
- die gemeinsame Planung des Teams zum Sprintanfang einschließlich gemeinsamer Verpflichtung des Teams zum angestrebten Sprintergebnis,
- die Fertigstellung eines definierten Produktstandes zum Sprintende und Vorstellung des Sprintergebnisses in einem Review,
- tägliche Kurzmeetings zum Abgleich des Arbeitsstandes und zur Behebung von Problemen,
- maximale Transparenz des Arbeitsstandes durch eine allgemein zugängliche Aufgabentafel (Scrum board),
- Reflexion des Teams nach Sprintende über die Zusammenarbeit (Retrospektive).

(vgl. Schwaber/Sutherland 2013)

Scrum fördert einerseits die Selbstständigkeit der Teammitglieder, verlangt andererseits ausgeprägt kommunikative Fähigkeiten. Gegenüber anderen agilen Vorgehensmodellen (z. B. eXtreme Programming), die auf ähnlichen Prinzipien beruhen, setzt sich Scrum zunehmend als Industriestandard durch (vgl. BITKOM 2013, 26).

Objektorientierte Analyse (OOA) ist eine Methode zur Ermittlung und Dokumentation der Anforderungen, die die zu erstellende Software erfüllen soll. Es handelt sich um einen Verfeinerungsprozess, der von sehr allgemeinen Aussagen (Produktvision, Stakeholder, Systemkontext, Anwendungsfalldiagramme) über Zwischenstufen (essentielle Anwendungsfallbeschreibungen) zu detaillierten Anforderungsbeschreibungen (Aktivitätsdiagramme, detaillierte Anwendungsfallbeschreibungen, Fachklassenmodelle) gelangt. Die Ergebnisse werden teilweise natürlichsprachlich, teilweise in der Modellierungssprache UML notiert und bilden die Grundlage für die Programmierung (vgl. Oestereich 1998, 121ff.).

Eine modernes Softwaresystem besitzt eine Architektur, in der verschiedene Belange (fachliche Daten, deren Darstellung auf dem Bildschirm oder Speicherung in einer Datenbank) voneinander getrennt sind und über festgelegte Mechanismen miteinander kommunizieren. Die in der objektorientierten Analyse identifizierten Fachklassen, die Objekte der Fachdomäne beschreiben, bilden eine Teilmenge des Gesamtsystems, die im Rahmen des **objektorientierten Designs** (OOD) zu einer Gesamtarchitektur ergänzt wird (vgl. Oestereich 1998, 161ff.).

Pair programming bezeichnet die gemeinsame Arbeit zweier Entwickler an einem Computer, wobei sich die beiden auf unterschiedlichen Abstraktionsniveaus bewegen. Während der Bediener der Tastatur („Driver“) codiert, behält sein Partner („Observer“) die strategische Linie im Blick. Lautes Denken ist wichtig. Die Rollen wechseln regelmäßig oder nach Bedarf

zwischen den Partnern, innerhalb des Projektes werden häufig neue Paare gebildet (vgl. Williams 2000, 3; Brüggemann/Dehnbostel/Rohs 2010, 37).

Refactoring bezeichnet die Überarbeitung von bereits erstelltem Programmcode ohne Veränderung der fachlichen Funktionalität, um den Programmtext zu verbessern. Ziele sind bessere Verständlichkeit, bessere Wartbarkeit, Verallgemeinerung der Funktionalität zur Wiederverwendung, die Eliminierung von schlecht programmierten Stellen (vgl. Fowler 1999, xvi). Refactoring wird insbesondere bei der Softwarepflege eingesetzt und kann im Pair programming erfolgen.

Komponententests (oder Unit-Tests) sind Testprogramme, die die eigentlichen Programme prüfen (vgl. Spillner/Linz 2004, 40ff.). Entwickler testen damit die eigenen Arbeitsergebnisse systematisch und stellen dadurch eine grundsätzliche Qualität sicher. Komponententests unterstützen ein iterativ-inkrementelles Vorgehen, indem verhindert wird, dass neuer Code die bestehenden Programmteile mit Fehlern infiziert. Komponententests bewähren sich besonders beim Refactoring und können automatisiert ausgeführt werden.

Die Entwickler synchronisieren regelmäßig die lokal auf dem eigenen Arbeitsplatzrechner bearbeiteten Programmtexte mit denen der anderen Projektmitglieder, indem der Code auf einem zentralen Versionsverwaltungssystem gespeichert bzw. von dort bezogen wird. Regelmäßig, meist nachts, wird der gemeinsame Stand automatisiert von einem Erstellungssystem zusammengeführt und geprüft. Dabei werden u. a. die Komponententests ausgeführt. Zum Dienstbeginn am nächsten Morgen finden die Entwickler ein Protokoll vor, das die dabei aufgetretenen Fehler ausweist. Dadurch wird eine permanente Qualitätssicherung erreicht. Den Gesamtprozess nennt man **Continuous integration** (vgl. Fowler 2006).

4 Berufs- und arbeitspädagogische Bewertung

Das berufs- und arbeitspädagogische Potential der vorgeschlagenen Fachmethoden zeigt sich besonders deutlich bei Scrum und Pair programming.

Pair programming unterstützt und initiiert Lernprozesse bei beiden Partnern („Pair-Learning“, vgl. Williams, 45ff.) und stößt bereits in der Berufs- und Arbeitspädagogik auf Interesse (vgl. Brüggemann/Dehnbostel/Rohs 2010, 27/44). Während es anfangs als Vormachen und Beobachten eher instruktiv angelegt ist, ermöglicht Pair programming mit zunehmendem Leistungsstand des Auszubildenden selbstgesteuertes, konstruktives Lernen.

Scrum adressiert besonders die Teilautonomie des Teams, unterstützt damit ein job enrichment und job enlargement und eröffnet dem Auszubildenden ein reichhaltiges, realistisches Bild des Ausbildungsberufs. Die unterstützenden Elemente (Scrum-Master als Team supporter, Daily Scrum als Frühwarnindikator, Ausbilder im Entwicklerteam) verhindern, dass der Auszubildende unbemerkt an unbewältigten Lernproblemen festhängt.

4.1 Kompetenzfelder und Lernbereiche

Die gute Überdeckung der Kompetenzfelder und Lernbereiche ergibt sich aus dem Zusammenwirken sehr unterschiedlicher Methoden.

Tabelle 6: **Abdeckung der Kompetenzfelder und Lernbereiche**

Methode	Kompetenzen				Lernbereiche		
	F	M	E	S	k	a	p
Scrum	(x)	x	x	x		x	x
Pair programming	x	x	x	x	x	x	x
Refactoring	x	x	x		x	(x)	x
Komponententest	x	x	x		x	(x)	x
OOA	x	x		x	x		x
OOD	x	x		(x)	x		x
Continuous Integration	x	x	x	x			x

(F = Fachkompetenz, M = Methodenkompetenz, E = Eigenkompetenz, S = Sozialkompetenz, k = kognitiv, a = affektiv, p = pragmatisch)

Der Ansporn, den eigenen Programmcode zu testen (Komponententest), laufend zu verbessern (Refactoring) und systematisch zu verwalten (Continuous Integration), fördert das implizite Einüben personaler Eigenschaften wie Zuverlässigkeit, Verantwortungs- und Pflichtbewusstsein.

Der pragmatische Lernbereich, also das Erlernen des Handwerks Anwendungsentwicklung (vgl. 2.2 Kompetenz- und Lernbereiche) wird von allen Methoden, ganz intensiv von Pair programming, Refactoring, Komponententest und Continuous Integration angesprochen.

4.2 Lern- und Kompetenzförderung

Auch die Lern- und Kompetenzförderung lässt sich auf die Methodenvielfalt zurückführen.

Tabelle 7: **Erfüllung der Lern- und Kompetenzförderung**

Kriterium	**Umsetzung**
Vollständige Handlung	Der komplette Softwareentwicklungszyklus wird einschließlich aller wichtigen Teilprozesse durchlaufen.
Handlungsspielraum	Scrum beruht auf der Teilautonomie des Teams. Jedes Teammitglied kann die Art und Weise der Aufgabenbearbeitung selbst gestalten.
Problemerfahrung	Der Auszubildenden wirkt in einem realen Softwareprojekt mit.
Soziale Unterstützung / Kollektivität	Scrum zeichnet sich durch starke soziale Interaktion aus. Pair programming ermöglicht die Unterstützung in einer geschützten Paarumgebung.
Individuelle Entwicklung	Im Rahmen von Pair programming kann das Aufgabenniveau wachsen. Der Verfeinerungsprozess von OOA führt schrittweise in die Komplexität der Problemstellung ein.
Professionalität	Es handelt sich um Fachmethoden, die aktuelle Industriestandards sind.
Reflexivität	Pair programming, Komponententest und Refactoring trainieren die individuelle Selbstreflexion. Die Retrospektive von Scrum dient der Reflexion auf Teamebene.

Sofern neben der Tätigkeit als Fachkraft die Ausbildertätigkeit mit geschützten Freiräumen und Zeitkontingenten abgesichert ist und der Auszubildende keinem Soll-Mengen-Druck unterliegt, sind Einschübe von instruktiven Lehr-Lern-Einheiten möglich.

4.3 Die Ausbildungsordnung

Die curricularen Vorgaben der Ausbildungsordnung werden umfassend abgedeckt, allerdings müssen begleitende Lehrveranstaltungen (Kurse, Lehrgespräche) den systematischen Erwerb kognitiven Wissens und die Erfolgssicherung unterstützen.

Tabelle 8: **Abdeckung des Rahmenplans**

Lfd. Nr.	**Teil des Ausbildungsberufsbildes**	**Fachmethoden**
2.1(a)	Leistungserstellung und -verwertung	Scrum, OOA, OOD, Pair programming, Komponententest, Continuous integration
3.1	Informieren und Kommunizieren	Scrum (Daily scrum, Review, Retrospektive), OOA, Pair programming
3.2	Planen und Organisieren	Scrum (Sprintplanung, Scrumboard)
3.3	Teamarbeit	Scrum
5.1(d)	Ist-Analyse und Konzeption: Datenmodelle entwerfen	OOA, OOD

5.2	Programmiertechniken	OOD, Pair programming, Refactoring
6.1(a)	Analyse und Design: Vorgehensmodelle und -methoden	Scrum
6.1(a)	Analyse und Design: Entwicklungsumgebungen	Continuous integration
6.1(b-e)	Analyse und Design	OOA, OOD
6.2(a,c-e)	Programmerstellung und -dokumentation: Programmierung	OOD, Pair programming, Refactoring, Komponententest
6.2(b,f,g)	Programmerstellung und -dokumentation: Softwarekonfigurationsmanagement	Continuous integration
6.3	Schnittstellenkonzepte	OOA, OOD, Pair programming, Refactoring, Komponententest
6.4	Testverfahren (Teilaspekt Entwicklertest)	Komponententest
8.1(b)	Architekturen	OOA, OOD
8.2	Datenbanken und Schnittstellen	OOA, OOD, Pair programming, Refactoring
9.1	Kundenspezifische Anpassung und Softwarepflege	OOA, OOD, Pair programming, Refactoring, Komponententest, Continuous integration
9.2	Bedienoberflächen	Pair programming, OOA (GUI-Prototypen)
9.3	Softwarebasierte Präsentation	Scrum (Review)
9.4(a,c,d)	Technisches Marketing	Scrum (Review)

4.4 Evaluation und Lernkontrolle

Zusätzlich zu den in 2.1 erwähnten Evaluationsmöglichkeiten kann der Ausbilder aus den Ergebnissen der Lernkontrolle weitere Hinweise über die Wirksamkeit des Lern-Arrangements gewinnen. Der Auszubildende erstellt allein oder in Paararbeit laufend typische Artefakte der Anwendungsentwicklung (Programmcode, Anforderungsbeschreibungen, Modelle etc.), anhand derer der Ausbilder im Lehrgespräch eine Erfolgskontrolle durchführen kann. Die zyklische Wiederholung der wesentlichen Verfahrungsschritte unterstützt die Erfolgssicherung.

Dagegen dürfte die Durchführung formaler Leistungsbeurteilungen dem offenen Charakter des Lehr-Lern-Arrangements entgegenstehen, so dass solche Beurteilungen außerhalb der Methode erfolgen müssen.

5 Fazit

Die vorgeschlagenen Fachmethoden der Anwendungsentwicklung besitzen ein großes berufs- und arbeitspädagogisches Potential und können in der vorgestellten Kombination das Erfahrungslernen am Arbeitsplatz fördern. Als Besonderheit ist hervorzuheben, dass Fachmethoden, die curriculare Lerninhalte bilden, im Lernprozess nachhaltig eingeübt werden.

In der jüngeren Vergangenheit fanden verschiedene Arbeitsmethoden der Anwendungsentwicklung über die offensichtliche Wirksamkeit in der Praxis Anerkennung und wurden erst später wissenschaftlich untersucht (zu eXtreme Programming vgl. Williams 2000, 10). Daher wäre eine weitere empirische Untersuchung von Lehr-Lern-Arrangements am Entwicklerarbeitsplatz, die wesentlich auf Fachmethoden zurückgreifen, unter frühzeitiger Zusammenarbeit von Berufspädagogen und Anwendungsentwicklern wünschenswert.

5.1 Andere IT-Ausbildungsberufe

Der Vorschlag beschränkt sich auf den Ausbildungsberuf Fachinformatiker/in Anwendungsentwicklung, weil die vorgeschlagenen Methoden der Fachdisziplin eigen sind, und lässt sich auf den Ausbildungsberuf Mathematisch-technische/r Softwareentwickler/in übertragen.

Ob andere IT-Berufe einen ähnlichen Zugang erlauben, hängt davon ab, ob dort ebenfalls Fachmethoden und Prozesse mit didaktischem Potential vorhanden sind. Dies kann nur unter Beteiligung von Spezialisten dieser Berufe geklärt werden und muss an dieser Stelle offen bleiben.

5.2 Agilität versus klassische Vorgehensmodelle

Das didaktische Potential der vorgeschlagenen Fachmethoden ergibt sich aus der anthropozentrischen Grundkonzeption der Agilität, die sich sehr deutlich bei Scrum zeigt. Alternative agile Vorgehensmodelle dürften ähnlich große berufs- und arbeitspädagogische Möglichkeiten bieten. Beispielsweise nutzt eXtreme Programming ebenfalls Pair programming, Refactoring, Komponententest und Continuous Integration (vgl. Beck 1999, 54ff.).

Agilität gilt mittlerweile als wichtiger Erfolgsfaktor in der Softwareindustrie (vgl. BITKOM 2013). Gleichwohl gibt es viele Skeptiker der agilen Softwareentwicklung, gerade in Deutschland. Es gibt zahlreiche Firmen, die mit klassischen, formalen Vorgehensmodellen (z. B. V-Modell XT) erfolgreich arbeiten. Auch in solch einem Projektumfeld lassen sich die propagierten Fachmethoden gewinnbringend für das Erfahrungslernen einsetzen (natürlich mit Ausnahme von Scrum).

Für ein klassisches Vorgehensmodell der Anwendungsentwicklung ist ein strukturierterer Ansatz zu erwägen, beispielsweise die Modellierung der lernrelevanten Arbeits- und Geschäftsprozesse als ereignisgesteuerte Prozessketten (vgl. Rebmann/Schlömer 2009). Ob dabei Fachmethoden sinnvoll integriert werden können, bedarf der weiteren Untersuchung.

Literatur

Antoni, C. (Hrsg.) (1994): Gruppenarbeit in Unternehmen. Weinheim.

Beck, K. (1999): Extreme Programming Explained. Reading.

Beck, K. et al. (2001): Manifesto for Agile Software Development. Online: http://agilemanifesto.org/ (09.03.2015)

BIBB Bundesinstitut für Berufsbildung (2009): Empfehlung des Hauptausschusses des Bundesinstituts für Berufsbildung zum Rahmenplan für die Ausbildung der Ausbilder und Ausbilderinnen. In: Bundesanzeiger Nr. 111/2009 vom 30.07.2009. Online: http://www.bibb.de/dokumente/pdf/empfehlung_135_rahmenplan_aevo.pdf (07.03.2015)

BITKOM Bundesverband Informationswirtschaft Telekommunikation und neue Medien e. V. (2013): Agiles Softwareengineering Made in Germany. Online: http://www.bitkom.org-/de/publikationen/38337_76498.aspx (07.03.2015)

Bleßmann, J./Büttner. A./Dax, E. (2011): Basiswissen IT-Berufe: Anwendungsentwicklung. 5. Auflage. Köln.

Bonz, B. (2009): Methoden der Berufsbildung. 2. Auflage. Stuttgart.

Brüggemann, A./Dehnbostel, P./Rohs, M. (2010): eXtreme working – eXtreme learning? Münster.

BMWi Bundesministerium für Wirtschaft (1997): Verordnung über die Berufsausbildung im Bereich der Informations- und Telekommunikationstechnik. In: Bundesgesetzblatt 1 vom 15. Juli 1997, 1741-1799. Bonn.

Chrissis, M./Konrad, M./Shrum, S. (Hrsg.) (2007): CMMI® Guidelines for Process Integration and Product Improvement. Upper Saddle River.

Dehnbostel, P. (2008): Lern- und kompetenzförderliche Arbeitsgestaltung. In: Berufsbildung in Wissenschaft und Praxis (BWP) 2/2008, Bonn, 5-8.

Fowler, M. (1999): Refactoring: Improving the Design of Existing Code. Reading.

Fowler, M. (2006): Continuous Integration. Online: http://www.martinfowler.com/articles/continuousIntegration.html (07.03.2015)

IEEE Computer Society (2014): SWEBOK© V3.0, Guide to the Software Engineering Body of Knowledge. Online: http://www.computer.org/portal/web/swebok (07.03.2015)

ISO International Organization for Standardization (2008): ISO/IEC 12207 Systems and software engineering – Software life cycle processes. Genf.

Jacobs, P./Preuße, M. (2013): Kompaktwissen AEVO in vier Handlungsfeldern. 2. Auflage. Köln.

Kersken, S. (2013): IT-Handbuch für Fachinformatiker. 6. Auflage. Bonn

Martin, R. C. (2009): Clean Code - Refactoring, Patterns, Testen und Techniken für sauberen Code. Heidelberg.

Martin, R. C. et al. (2009): Manifesto for Software Craftsmanship. Online: http://manifesto-.softwarecraftsmanship.org (07.03.2015)

Oestereich, B. (1998): Objektorientierte Softwareentwicklung. Analyse und Design mit der Unified modeling language. 4. Auflage, München.

Petersen, W./Wehmeyer, C. (1999-2004): Die neuen IT-Berufe auf dem Prüfstand – Eine bundesweite Studie im Auftrag des Bundesinstituts für Berufsbildung BiBB. Flensburg. Online: http://www.biat.uni-flensburg.de/BIBB-IT (07.03.2015)

Petersen, W. (2003): Betriebliche Geschäfts- und Arbeitsprozesse als Grundlage und neue didaktische Orientierung der Berufsbildung. Flensburg. Online: http://www.biat.uni-flensburg.de/elektroberufe-online/Uebersicht/GAHFA/GAHPA-GAHFA-Modellstruktur.htm (07.03.2015)

Petersen, W./Wehmeyer, C. (2003): Aufgedeckt: IT-Arbeitsprozesse und Ausbildung in der Betriebspraxis. Teilprojekt 2 Betriebliche Fallstudien. Flensburg. Online: http://www.biat.uni-flensburg.de/datenbank/Materialien/biat-BiBB-IT-T2-Fallstudien-01-2004.pdf (07.03.2015)

Rebmann, K./Schlömer, T. (2009): Lernen im Prozess der Arbeit. Berufs- und Wirtschaftspädagogik online PROFIL 2. Online: http://www.bwpat.de/profil2/rebmann-schloemer-profil2.pdf (07.03.2015)

Schaper, N. (2004): Fach-, Methoden- und Sozialkompetenz durch arbeitsbezogenes Lernen in der betrieblichen Ausbildung. Online: http://groups.uni-paderborn.de/psychologie/scha-Arbeitsbezogenes-Lernen.pdf (07.03.2015)

Schelten, A. (2005): Grundlagen der Arbeitspädagogik. 4. Auflage. Stuttgart.

Schwaber, K./Sutherland, J. (2013): Der Scrum Guide. Online: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf (07.03.2015)

Spillner, A./Linz, T. (2004): Basiswissen Softwaretest. 2. Auflage. Heidelberg.

Williams, L. A. (2000): The collaborative software process. Salt Lake City. Online: http://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf (07.03.2015)

Dieser Beitrag wurde dem ***bwp@*-Format:** ☑ **BERICHTE & REFLEXIONEN** zugeordnet

Zitieren dieses Beitrages

Seulberger, N. (2015): Fachmethoden als Ausbildungsmethoden – Erfahrungslernen am Arbeitsplatz im Ausbildungsberuf Fachinformatiker/-in Anwendungsentwicklung. In: *bwp@* Berufs- und Wirtschaftspädagogik – online, Ausgabe 28, 1-18. Online: http://www.bwpat.de/ausgabe28/seulberger_bwpat28.pdf (22-06-2015).

Der Autor

Dipl.-Math. NORBERT SEULBERGER

Wiesengrund 14, 24161 Altenholz

norbert.seulberger@gmx.de

https://www.xing.com/profile/Norbert_Seulberger