Dezentrale E-Learning-Plattformen – einige Überlegungen

Hans-Gert Gräbe[1]
Universität Leipzig
Institut für Informatik
Johannisgasse 26
04103 Leipzig

Zusammenfassung

In der E-Learning-Debatte wird viel von „elektronischem Lernen“ gesprochen und wenig über die Anforderungen an die zum Einsatz kommende Software. Noch weniger gesprochen wird über die Anforderungen, die sich aus Einsatzdynamik, Betrieb, Installation und Weiterentwicklung solcher Softwarestrukturen ergeben, obwohl entsprechendes Know-How über Entwicklung und Einsatz komplexer IT-Strukturen im akademischen Bereich reichlich vorhanden ist.

In diesem Aufsatz werden die Anforderungen und Rahmenbedingungen in Sachsen unter gerade einem solchen Blickwinkel genauer analysiert und mit einschlägigen Erfahrungen verglichen. Das Ergebnis ist ein klares Plädoyer für eine dezentrale E-Learning-Software-Landschaft für die operative Dimension der zu begleitenden Prozesse.

In der E-Learning-Debatte wird viel von Lernen, Lerninhalten, Medienkompetenz, „Life Long Learning“ usw. gesprochen, kaum jedoch über Vor- und Nachteile der eingesetzten Software oder gar IT-Strukturen. Man kann sich des Eindrucks nicht erwehren, dass ein großer Teil der Akteure der Meinung ist, dass die genaue Ausprägung dieser technischen Basis vollkommen zweitrangig ist – Hauptsache „E-“. Entsprechende Entscheidungen werden weitgehend hemdsärmlig getroffen wie in den Anfangszeiten des elektronischen Zeitalters, obwohl sich inzwischen herumgesprochen haben sollte, dass sich die wirklichen Kosten anfänglicher Erfolge oft erst im Nachhinein ergeben. In den 40 Jahren seines Bestehens als Disziplin sammelt und systematisiert das Software-Engineering diese auf anderen Feldern teuer erkauften Erfahrungen, so dass es mir angezeigt erscheint, deren Systematik einmal an die aktuellen Debatten anzulegen.

Ich möchte deshalb hier – in aller gebotenen Kürze – einige Gedanken zum Thema „Softwaretechnische Basis der E-Learning-Aktivitäten in Sachsen“ skizzieren und dabei eine Reihe impliziter Vorannahmen auf den Prüfstand stellen, die das Herangehen bisher – nach meinem Verständnis funktional vollkommen unbegründet – einschränken.

1 Software als techno-soziales Konstrukt

Ein erster solcher Kritikpunkt bezieht sich auf die starke Technikzentriertheit der Debatte, die wenig von einer souveränen Beherrschung der technischen Möglichkeiten zeugt. Eine solche souveräne Beherrschung technischer Möglichkeiten kommt vor allem in der Fähigkeit zum Ausdruck, diese dann *nicht* anzuwenden, wenn sie dem praktischen Kontext nicht angemessen sind. Ohne eine solche – nicht nur theoretische, sondern auch entscheidungsmächtige – Fähigkeit, technische Lösungen auf ihre Adäquatheit für die zu begleitenden Prozesse hin

[1] graebe@informatik.uni-leipzig.de

zu bewerten, bewegt sich die Diskussion faktisch auf der Ebene der Debatte unter Zauberlehrlingen über ihre neuen Spielzeuge. Um die Metapher weiterzuspinnen: Es bleibt dabei nur zu hoffen, dass ein Meister mit dem rechten Spruch zur rechten Zeit den Besen aufzuhalten vermag, wenn er unserer Gewalt entglitten ist. Oder die Natur selbst die aus dem Ruder gelaufenen Entwicklungen geraderückt wie im phantastischen Finale von Bulgakows ausgezeichneter Novelle „Die verhängnisvollen Eier“[2].

Die eigentlich triviale Forderung, die Fähigkeit zu bewahren, auf die Anwendung einer technischen Lösung verzichten zu können, enthält ein doppeltes Spannungsverhältnis. Dies ist zum einen die Frage, *wer* diesen Verzicht entscheidet, und zum anderen die Frage der Reproduktion der dazu erforderlichen technologischen *Kompetenz* dieser Entscheidungsträger. Wobei ich das im gegenwärtigen politischen Kontext immer wohlfeilere Wort „Entscheidungsträger“ an dieser Stelle gern durch das altmodische Wort „Entscheidungsprozesse“ ersetzt sehen möchte, denn ich halte ein solches Zurück zum Prinzip des „primus inter pares“ einem akademischen Kontext noch immer angemessener als das „Betriebsleiterprinzip“. Letzeres ist ja nicht ernsthaft zu denken, ohne zugleich darüber nachzudenken, was denn eine „insolvente Universität“ ist, wenn man nicht letztlich bei einem „insolventen Staat“ landen möchte, wie eine nicht allzu weit zurückliegende historische Erfahrung lehrt.

Ich möchte meinen kleinen hochschulpolitischen Ausflug damit beenden, denn das ist ja hier nicht mein Thema. Gleichwohl kommt man ohne einen Blick auf diese Hintergründe nicht aus, denn die Entscheidungen über Software-Architekturen sind vielfältig verzahnt mit dem allgemeinen Ringen um Entscheidungen und Entscheidungsspielräume, was Lawrence Lessig in der prägnanten Kurzformel „code is law“[3] auf den Punkt gebracht hat.

Kehren wir also zurück zum Thema „Software als techno-soziales Konstrukt“. Die Softwaretechnik unterscheidet ganz grundlegend zwischen *technischer Software*, mit der der Gestaltungswille einzelner Akteure praktisch unterstützt wird und die in technischen Artefakten ihre lebensweltliche Realisierung erfahren, und *kaufmännisch-administrativer Software*, mit der die Kommunikation und Interaktion gestaltungswilliger (und -mächtiger) Akteure unterstützt wird. Diese Unterscheidung ist substanziell und beginnt bereits bei der Entwicklungsmethodik: technische Software wird in der Regel nach der Inside-Out-Methode entwickelt, kaufmännisch-administrative dagegen nach dem Outside-In-Verfahren. Lehrbücher der Softwaretechnik wie das einschlägige deutschsprachige Standardwerk von Helmut Balzert beschränken sich deshalb explizit oder implizit auf Ausführungen zur Entwicklung von Software der zweiten Sorte, die betrieblich einer stärkeren Standardisierung unterliegt als z. B. Prozessleitsysteme, die im betrieblichen Umfeld ebenfalls eine wichtige Rolle spielen.

Der Unterschied ergibt sich, grob gesprochen, daraus, dass für technische Software der „Gestaltungswille“ zunächst einmal genügend genau expliziert und detailliert werden muss, das Modell der Software also dem Kopf entspringt. Kaufmännisch-administrative Software dagegen orientiert sich primär an den realen Interaktions- und Kommunikationsbedürfnissen der beteiligten Akteure, die es zu unterstützen gilt. Kurz, solche Software wird in der Regel „nach der realen Welt“ modelliert. Ich bezeichne diese Art von Software im Weiteren als *Interaktionssoftware*.

Bereits diese grundlegende Unterscheidung wird beim Design von E-Learning-Umgebungen, die heute im Gespräch sind, kaum berücksichtigt, denn die relevanten Anwendungsfälle lassen sich grob unterteilen in

[2] Michail Bulgakow (1983): Kleine Prosa I. Verlag Volk und Welt, Berlin. S. 203 ff.

[3] Siehe `http://code-is-law.org`

(A) Unterstützung der Planungsebene von Lernprozessen: Autorenwerkzeuge für Konzeption, Aufbau und Organisationsplanung von Kursen in der Designphase;

und

(B) Unterstützung der Durchführung von Lernprozessen: Unterstützung von administrativ-organisatorischen und kommunikativen Aspekten.

(A) ist zweifelsohne der Domäne technischer Software zuzuordnen, (B) dagegen der Domäne der Interaktionssoftware. Heute existierende E-Learning-Software kann die Herkunft aus einer der beiden Quellen meist nur schwer leugnen: OLAT[4] etwa ist ganz klar aus (A) entstanden und (B) „draufgesetzt", unser Elate-Portal[5] dagegen kann die Herkunft aus dem Kontext (B) nicht verleugnen und wird nun mit einiger Mühe um Autorenwerkzeuge für bestimmte Zwecke erweitert. Dass im Übrigen hier der Ansatz „ein System kann alles" nicht zielführend ist, kann als Binsenweisheit der heutigen Softwaretechnik betrachtet werden.

2 Techno-soziale Aspekte von Interaktionssoftware im Allgemeinen

Im weiteren Text dieses Aufsatzes werde ich mich auf Interaktionssoftware, also den Einsatzbereich (B) beschränken, für den nun zunächst die enge Verbindung zwischen Software und realen Interaktions- und Kommunikationsprozessen analysiert werden soll.

Die Begriffe „Interaktion" und „Kommunikation" verwende ich hier in einem – aus philosophischer Perspektive allein sinnvollen – sehr weiten Verständnis, dem Begriffe wie „Wissen", „Information", „Lernen" etc. nachgelagert sind, um die es im engeren E-Learning-Kontext geht.

Klar ist[6], dass Interaktionssoftware nur als *Werkzeug* zum Einsatz kommen kann und die realen Akteure bei der Umsetzung ihrer realen Bedürfnisse unterstützen muss, wenn sie überhaupt nennenswerte Akzeptanz finden soll. Um die Bedeutung einer solchen Aussage auszuleuchten, wollen wir zunächst dem traditionellen Ansatz der Softwareentwicklung folgen und von einem Entwicklungsszenario im Rahmen eines Produzenten-Kunden-Verhältnisses ausgehen. Ein solches Szenario erfordert, dass die Software-Entwickler zunächst einmal ein detailliertes Bild der Kundenbedürfnisse erstellen und dabei diese oft nur verschwommen und informell artikulierten Bedürfnisse so weit formalisieren, wie dies für die Erstellung der Software erforderlich ist. Diese als *Anforderungsanalyse* bezeichnete Phase im Softwarelebenszyklus ist für klassische Anwendungen im kaufmännischen Bereich, die bereits auf der lebensweltlichen Seite durch ein entsprechendes Vertragsrahmenwerk vorstrukturiert sind,

[4] Siehe `http://www.olat.org`.

[5] Siehe `http://www.elateportal.de`.

[6] So klar nun auch wieder nicht. Klaus Fuchs-Kittowski betont seit über 30 Jahren, dass der Gedanke der Prozessautomatisierung in diesem Bereich nicht nur inhumane Kosequenzen hat, sondern schlicht disfunktional ist. Unter der Überschrift *Wider die Doktrin der Identifizierung von Automat und Mensch* schreibt er retrospektiv: „Es war damals wie heute die Frage: Welche Stellung hat der Mensch im hochkomplexen Informationstechnologischen System? Unsere Antwort auf die Frage war immer: Der Mensch ist die einzig kreative Produktivkraft, er muß Subjekt der Entwicklung sein und bleiben. Daher ist das Konzept der Vollautomatisierung, nach dem der Mensch schrittweise aus dem Prozeß eliminiert werden soll, verfehlt!". Klaus Fuchs-Kittowski: Wissens-Ko-Produktion. Verarbeitung, Verteilung und Entstehung von Informationen in kreativ-lernenden Organisationen. In Christiane Floyd, Christian Fuchs, Wolfgang Hofkirchner (Hrsg.): Stufen zur Informationsgesellschaft. Peter Lang-Verlag, Frankfurt 2002.

gut studiert und wird im B2B-Bereich vielfach erfolgreich umgesetzt. Auf die selbst in einem solchen Kontext häufigen Missverständnisse zwischen Produzenten und Kunden, deren Ursachen etwa auf der Basis des GAP-Modells genauer untersucht werden, gehe ich an dieser Stelle nicht ein.

Eine zweite Voraussetzung für die Anwendung klassischer softwaretechnischer Entwicklungsansätze ist die relative Stabilität der Bedürfnisse der Anwender. Schließlich erfasst die Anforderungsanalyse nicht die Bedürfnisse der Anwender schlechthin, sondern nur einen Schnappschuss dieser Bedürfnisse zu einem gewissen Zeitpunkt. Mit klassischen Methoden lässt sich also keine Software für agile Einsatzszenarien mit rasch und grundlegend wechselnden Nutzerbedürfnissen entwickeln. Genau solche Rahmenbedingungen findet man aber im E-Learning-Bereich vor, wenn man etwa die vielfältigen Umbrüche im Studienbetrieb im Rahmen des Bologna-Prozesses betrachtet. Ich komme darauf weiter unten genauer zu sprechen.

Zunächst möchte ich aber *klassische* softwaretechnische Entwicklungsansätze für Interaktionssoftware weiter verfolgen, denn auch diese stehen vor der Frage, wie sie mit der Variabilität von Anforderungen des zu modellierenden Zielbereichs umgehen. Diese Variabilität hat zwei Dimensionen. Neben der zeitlichen Dimension, die ich weiter oben bereits erwähnt hatte, ist dies die personelle Dimension, da im selben Interaktionskontext natürlich Personen mit unterschiedlichen Zielen und Bedürfnissen agieren, die punktuell sogar widersprüchlich bis konfliktär sind. Lessigs Formel „code is law“ thematisiert den Umstand, dass Software die Art und Herangehensweise an die Lösung solcher Konflikte vorstrukturiert und damit Verhalten und Entscheidungsspielräume der Akteure ähnlich stark beeinflusst wie gesetzliche Vorgaben, ohne einen vergleichbar guten gesellschaftlichen Legitimationsprozess durchlaufen zu haben. Der in Amtsstuben häufige Satz „Der Computer lässt mich das nicht anders machen“ legt von dieser demokratietheoretisch gefährlichen Tendenz beredtes Zeugnis ab.

Widersprüchlich heißt in diesem Zusammenhang, dass sich die Interessensphären der Nutzer abgrenzen lassen, in denen die Software verschieden funktional operieren muss, ohne dass es zum lebensweltlichen Konflikt der Nutzer kommt. Dies ist aus betriebswirtschaftlicher Sicht doppelt interessant, da die hier zum Einsatz kommenden Lösungen meist auch geeignet sind, dieselbe Software für mehrere Anwender mit je entsprechenden Modifikationen einzusetzen.

Konfliktär heißt, dass der Konflikt lebensweltlich nicht zu vermeiden ist, die Software also die Konfliktlösung selbst unterstützen muss – eine auch aus softwaretechnischer Sicht um Größenordnungen kompliziertere Angelegenheit.

An dieser Stelle ist es Zeit, den Begriff „Software“ genauer zu fassen. Die Softwaretechnik unterscheidet zwischen dem Quellcode der Software (genauer: dem Quellcode in einer bestimmten Version; ich bezeichne dies im Weiteren als *die Software*), verschiedenen *Installationen* dieser Software und dem *Betrieb* einer solchen Installation. Mit einer derartigen Unterscheidung in verschiedene *Sichten*, deren Aufzählung hier bei weitem nicht vollständig ist, wird den sehr verschiedenen Rollen und Interessenbündeln Rechnung getragen, die durch den Einsatz einer Software tangiert sind.

Beim Entwickeln einer Software sind diese Sichten zu berücksichtigen und insbesondere die nachgelagerten Phasen von Installation und Betrieb bereits mit zu planen. Installation einer Software bedeutet in der Regel, diese in eine bereits bestehende Hard- und Softwareumgebung zu integrieren und die neue Software in diesem Prozess auch noch zu *konfigurieren*, also den speziellen Einsatzwünschen im Rahmen der in der Software vorgesehenen Möglichkeiten anzupassen. Dies geschieht in der Regel vor Ort durch Systemadministratoren, die

denselben Installations- und Konfigurationsprozess für eine Vielzahl verschiedener Softwarepakete koordinieren und auf das Problemfeld „Software am Laufen halten“ spezialisiert sind. Ihr Berufsprofil zeichnet sich durch eine gute Kenntnis der verschiedenen informatischen Kopplungs- und Kompatibilitätsaspekte aus, die beim Zusammenspiel von Software unterschiedlicher Entwickler zu berücksichtigen oder herzustellen sind. Ich bezeichne diese Rolle im Weiteren als *Administrator* der Software.

Davon zu unterscheiden sind die Anforderungen, die sich aus dem laufenden Betrieb der Software unmittelbar ergeben. Der erforderliche Betreuungsaufwand richtet sich dabei vor allem auf die Konsistenz der von der Software verwalteten Daten und wird in der Regel durch eine oder mehrere Personen mit den erforderlichen Kenntnissen aus dem jeweiligen Fachgebiet organisiert, die zu allem Überfluss meist auch „Administratoren“ genannt werden, obwohl sie nichts mit der Rolle der Systemadministratoren zu tun haben. Ich bezeichne diese Rolle im Weiteren als *Betreiber* der Software. Solche Betreiber sind im Übrigen auch erforderlich, wenn die Softwareinstallation selbst an zentraler Stelle erfolgt und allein der Betrieb dezentral organisiert ist.

Ich breite diese Selbstverständlichkeiten hier so detailliert aus, weil mir scheint, dass gerade im Zusammenhang mit E-Learning-Software diese Unterscheidungen auf der konzeptionellen Ebene in keiner Weise selbstverständlich sind und insbesondere die personellen Erfordernisse in den verschiedenen Rollen sowohl hinsichtlich des Arbeitsaufwands als auch der Qualifikation sträflich unterschätzt werden.

Kommen wir auf den Ausgangspunkt dieses Abschnitts zurück – die wechselnden realweltlichen Bedürfnisse der Anwender. Es gibt also drei Ebenen, auf denen Software auf solche veränderten Bedürfnisse reagieren kann:

(1) Die Bedürfnisse sind beim Design der Software auf der Betriebsebene vorgesehen. Dem Nutzer stehen entsprechende Gestaltungselemente zur Verfügung, um die neuen Bedürfnisse auszudrücken. Dies erfordert eine direkte und/oder indirekte Kommunikation zwischen Entwicklern und Nutzern, in deren Rahmen letztere mit den verfügbaren Gestaltungsmöglichkeiten vertraut gemacht werden (Schulungen, Schulungsmaterialien, Hilfesysteme, Handbücher etc.)

(2) Variierende Bedürfnisse unterschiedlicher Kunden werden auf verschieden konfigurierte Installationen derselben Software abgebildet. Dies erfordert ein entsprechendes Konfigurationsmanagement auf Seiten des Anbieters der Software und zusätzlich zu (1) Kommunikation zwischen Entwicklern und Administratoren bzw. Betreibern über die Konfigurationsmöglichkeiten.

    Der letztere Aufwand kann natürlich durch eine zentrale Installation einer einzigen Software-Instanz vermieden werden und entsprechende Konzepte erleben derzeit im Zuge der Entwicklungen von Web 2.0 eine verstärkte Renaissance. Aber aus softwaretechnischer Sicht haben sie einen entscheidenden Pferdefuß – sie skalieren extrem schlecht.

(3) Neue Bedürfnisse werden im Weiterentwicklungszyklus der Software in neueren Versionen berücksichtigt. Dies ist bei rasch wechselnden oder sehr unklar spezifizierbaren Rahmenbedingungen der einzige Weg, auf Änderungen angemessen zu reagieren.

    In diesem Fall müssen im Rahmen der Qualitätssicherung eines solchen Projekts bereits im Entwurf Aspekte der Änderbarkeit und Wartbarkeit in gleichem Umfang berücksich-

tigt werden wie funktionale Aspekte der Software. Ersparnisse in frühen Entwicklungsphasen rächen sich durch übermäßigen Aufwand in späteren Projektphasen der Weiterentwicklung und Anpassung. Es gibt spezielle Entwicklungsmethodiken für derartige agile Szenarien.

3 Interaktionssoftware – die Datensicht

Im betrieblichen Kontext lässt sich die unterstützende Funktion von Interaktionssoftware in zwei Bereiche mit deutlich differierenden funktionalen Anforderungen unterteilen; in die operative Funktion und die dispositive Funktion. Dies kann grob mit dem Unterschied der Rolle von Sinnesorganen und Gehirn im Prozess der menschlichen Entscheidungsfindung verglichen werden, in denen die aktuale und historische Dimension von Entscheidungsprozessen abgebildet werden.

Entsprechend gliedert sich die Datenlandschaft eines Unternehmens in operative und dispositive Daten. Operative Daten haben einen hohen Detaillierungsgrad, aber geringe zeitliche Tiefe – es sind die Daten, die über die „Sinnesorgane“ des Unternehmens dauernd einlaufen und zusammen mit dem Erfahrungshintergrund der je konkret verdichteten dispositiven Daten die Basis für operative Entscheidungen bilden. Diese Daten sind zugleich der Input für die Fortschreibung des dispositiven Datenbestands. Zu diesem Zweck werden die operativen Daten gefiltert, konsolidiert, historisiert und schließlich transformiert – sie wandern vom Kurzzeit- ins Langzeitgedächtnis.

Die betriebliche Spezifik operativer Daten liegt auf deren Gewinnung, wofür spezielle softwareseitige Voraussetzungen geschaffen werden müssen. In der *Software* des entsprechenden Prozessleitsystems steckt in Größenordnungen firmeninternes Know-How. Ähnlich wie im Industrieanlagenbau werden deshalb entsprechende Großsysteme in enger Kooperation zwischen „Anlagenbauern“ und dem Auftraggeber entwickelt und betrieben. Es handelt sich in der Regel um maßgeschneiderte Einzelanfertigungen, die natürlich aus standardisierten Komponenten und nach bewährten Konstruktionsprinzipien aufgebaut sind, aber meist Unikate darstellen, für die eine Abtrennung einer speziellen Konfigurationsphase aus der allgemeinen Entwicklungsphase zumindest entwicklungslogisch keinen Sinn hat. Der Schwerpunkt liegt auf der funktionalen Seite der Software und weniger den Daten. Da diese Software hochgradig geschäftskritisch ist, wird diese in deutlich engerem Kontakt zwischen Entwicklern und Nutzern entwickelt und betrieben als sich dies in einem klassischen Produzenten-Kunden-Verhältnis widerspiegelt. In der Regel wird zwischen einer Entwicklungs- und einer Produktivversion der Software unterschieden, die durch spezielle Roll-out-Szenarien miteinander verbunden sind.

Eine deutlich andere Spezifikation gilt für dispositive Daten: hier liegt die Besonderheit nicht so sehr in der spezifischen Funktionalität der Software, sondern in einer für den jeweiligen betrieblichen Kontext angemessenen Verdichtung und Systematisierung der anfallenden Daten. Als Software wird deshalb meist ein gängiges ERP-System wie etwa das SAP Business Information Warehouse eingesetzt, denn der betriebliche Wert liegt hier mehr auf der speziellen Struktur des akkumulierten Datenbestands und weniger in der dafür verwendeten Software.

4 Die E-Learning-Software-Landschaft in Sachsen

Auf diesem Hintergrund stellen sich die Anforderungen an eine sächsische E-Learning-Software-Landschaft wie folgt dar:

Es ist deutlich zu unterscheiden zwischen (A) technischer Software zur Unterstützung von Autorenprozessen, die sich in der Verantwortung (und damit dem „Gestaltungswillen“) einzelner Hochschullehrer befinden und bis hin zur computergestützten Abwicklung von Tests und Prüfungen reichen, und Interaktionssoftware zur administrativ-organisatorischen Begleitung akademischer Prozesse. Im Bereich der Interaktionssoftware ist zwischen (B) operativen und (C) dispositiven Szenarien zu unterscheiden. Es ist nicht sinnvoll (und in einem betrieblichen Kontext nicht üblich), die Aspekte (B) und (C) in *einem* gemeinsamen Softwaresystem zu integrieren, da vollkommen verschiedene Anforderungen an die entsprechende Software bestehen.

Ich möchte mich nun auf E-Learning-Software im Sinne von (B) konzentrieren, mit der operative Aspekte der administrativ-organisatorischen Seite akademischer Prozesse unterstützt werden. Es geht also darum, ein entsprechendes „Prozessleitsystem“ aufzubauen und die dafür erforderliche Software nicht nur zu entwickeln, sondern daraus auch Installationen abzuleiten und diese zu betreiben.

Dem Bereich (B) kommt eine gewisse Scharnierfunktion zwischen den Bereichen (A) und (C) zu, so dass es sinnvoll ist, die Kräfte auf den Ausbau gerade dieses Bereichs zu konzentrieren. Auf Grund der hohen Autonomie der Akteure in den einzelnen Fachbereichen bzw. Fakultäten (nicht nur) sächsischer Hochschulen – bei ihnen liegt ja sowohl die fachliche Kompetenz als auch die administrative Verantwortung für die Gestaltung der operativen Prozesse – ist es sinnvoll, dabei auf dezentrale kooperationsfähige Installationen zu setzen, die in Verantwortung dieser administrativen Einheiten betrieben werden.

Die in diesen Systemen gehaltenen operativen Daten lassen sich über einen Konsolidierungsprozess leicht zu dispositiven Daten verdichten, was bereits heute etwa über die Prüfungsämter auf der realweltlichen Seite geschieht; oftmals mit nur unzureichender elektronischer Unterstützung. Eine bessere Übermittlung dieser Daten an zentrale dispositive Systeme lässt sich mit überschaubarem Aufwand im Zuge der Gestaltung entsprechender Schnittstellen realisieren, ohne dass hierfür monolithische hochschulweite Softwarelösungen auf operativer Ebene eingeführt werden müssen.

Es bleibt die Frage nach einer technisch und betriebswirtschaftlich sinnvollen Entwicklungsstrategie einer solchen dezentralen E-Learning-Software-Landschaft. Folgende Aspekte bilden die Ecksteine, welche dafür den strategischen Rahmen aufspannen:

- Wegen der dezentralen Zuständigkeit kann der *Betrieb* der einzelnen Installationen nur in Verantwortung der jeweiligen Struktureinheiten erfolgen. Mit Blick auf die vorhandene rechentechnische Ausstattung sind dafür kaum Sachinvestitionen erforderlich, so dass der Fokus auf die *Personalressourcen* gerichtet werden muss, deren Stärke, Aufgabenspektrum und vor allem Qualifikation genauer bestimmt werden müssen, da sie nutzerseitig in die Weiterentwicklung der Softwarebasis zumindest als technisch versierte Konsultanten einzubeziehen sind.

  Unter den gegebenen finanziellen Rahmenbedingungen lässt sich diese Aufgabe nur bewältigen, wenn dabei auch studentische Hilfskräfte zum Einsatz kommen, die durch Personal mit entsprechender fachlicher und didaktischer Erfahrung angeleitet werden. Sowohl aus übergreifenden Ausbildungsgesichtspunkten als auch demokratietheore-

tischen Aspekten (angemessene studentische Beteiligung an akademischen Entscheidungsprozessen; Sensibilisierung von Studenten für solche Fragen) ist ein derartiger Einsatz auch wünschenswert.

- Diesen autonomen dezentralen Akteuren kommt auch eine entscheidende Rolle zu bei der Fortschreibung der strategischen Entwicklungen hin zu größerer Kohärenz und einer gemeinsamen Softwarebasis, mit der Synergien genutzt und die gerade in diesem Bereich bisher äußerst knappen Entwicklungsressourcen effizient eingesetzt werden können. Diese Akteure kennen die wechselnden örtlichen Bedingungen und Anforderungen und haben mit einer Reihe von Eigenentwicklungen auch bereits Erfahrungen gesammelt, wo und auf welche Weise das operative administrativ-organisatorische Geschäft durch elektronische Medien sinnvoll unterstützt werden kann.

- Eine konsistente Strategie der Auswahl, Konsolidierung und Weiterentwicklung der Softwarebasis muss die Schaffung und Fortschreibung entsprechender organisatorischer Rahmenbedingungen einschließen. Mit Blick auf die generell angespannte finanzielle Situation an den Hochschulen werden Teile der Software an den Einrichtungen selbst weiterzuentwickeln sein.

  Dabei ist eine Kopplung mit dem Betrieb der Installationen sinnvoll, was zusätzliche Qualifikationsanforderungen an das eingesetzte Personal mit sich bringt sowie entsprechende Freiräume für diese Entwicklungsarbeiten erfordert. Eine Verbindung mit der Ausbildung in Informatik-Studiengängen, wie bereits von unserer Abteilung praktiziert, bietet sich auch an dieser Stelle an, bringt aber mit der erforderlichen Kompetenzreproduktion zugleich einen weiteren agilen Faktor ein, der Berücksichtigung erfordert.

Damit ergibt sich wie von selbst ein Entwicklungsmodell, das gegenüber anderen operativen Systemen, in deren Entwicklung im Vergleich zur Entwicklung klassischer „Anwendungen von der Stange“ die Nutzer sowieso schon stärker in die Entwicklung einbezogen sind, eine nochmalige Aufwertung der Nutzerseite. Die klassische Produzenten-Nutzer-Dichotomie wird entwicklungsseitig aufgehoben durch eine Entwickler-Koentwickler-Perspektive und auf der betriebswirtschaftlichen Ebene durch die kooperative Bewirtschaftung einer gemeinsamen Codebasis.

Dies sind aber gerade die beiden Säulen, auf denen das Open Source (OS) Entwicklungsmodell ruht. Zentralen Einrichtungen kommt hier vor allem eine koordinierende Funktion zu, welche sich an den Bedürfnissen der Akteure orientiert und diese unterstützt, ihre Anforderungen zu artikulieren und diese in einem kooperativen Prozess zu verwirklichen. Kleinere bis mittlere OS-Projekte sind heute in der Lage, sich eine derartige koordinierende Infrastruktur, über die sich technisch-organisatorische Fragen behandeln lassen, ohne zusätzliche Mittel zu schaffen. Die entsprechenden Arbeitsanteile werden aus der „Grundlast“ der beteiligten Akteure finanziert nach dem Prinzip „Mache, was du sowieso tun musst, aber stelle deine Ergebnisse öffentlich zur Nachnutzung zur Verfügung“. Weitergehende koordinierende Leistungen (Projektmeetings, Schulungen, strategische Abstimmungen) müssen im Einzelfall explizit von den beteiligten Akteuren finanziert werden, wenn es keine Grundfinanzierung der Infrastruktur aus anderen Quellen, insbesondere Projektgeldern, gibt. In entwickelten Projekten wird diese Infrastrukturfinanzierung in vielen Fällen nicht auf der monetären, sondern der personellen Ebene realisiert, indem einzelne strategisch besonders fähige Personen ganz oder teilweise für die Organisation koordinierender Aktivitäten freigestellt und finanziell ausgestattet werden.

In Sachsen sind Bruchstücke einer solchen Infrastruktur bereits sichtbar, die es weiterzuentwickeln gilt. Die Entscheidung auf Landesebene, sich eng an das Open Source Projekt OLAT anzuschließen, bietet die Möglichkeit, auch über Sachsen hinausreichende Synergien und Projektinfrastrukturen zu nutzen, was im Lichte meiner Ausführungen als strategisch wichtige Weichenstellung zu betrachten ist, deren Potenzen sich aber nur in einem dezentralen Entwicklungs- und Betriebsansatz wirklich entfalten können. Ob es sich einzelne Einrichtungen dauerhaft leisten können, eine solche Infrastruktur für mehr als eine Plattform am Laufen zu halten, wird die Zukunft zeigen.