Lernobjekte im E-Learning - Eine kritische Beurteilung zugrunde liegender Konzepte anhand eines Vergleichs mit komponentenbasierter Software-Entwicklung

Victor Pankratius, Andreas Oberweis, Wolffried Stucky
Institut für Angewandte Informatik
und Formale Beschreibungsverfahren (AIFB)
Universität Karlsruhe (TH)
76128 Karlsruhe
{pankratius, oberweis, stucky}@aifb.uni-karlsruhe.de

Abstract. Im E-Learning-Umfeld haben sich Lernobjekte als eine spezielle Art von Komponenten herauskristallisiert, um Lerninhalte aller Art zu kapseln. Damit wird beabsichtigt, dass Lerninhalte strukturiert gespeichert, effizient wieder verwendet und flexibel zwischen verschiedenen E-Learning-Plattformen ausgetauscht werden können. Allerdings scheinen Lernobjekte im Hinblick auf technische, organisatorische und ökonomische Aspekte noch lange nicht so erfolgreich zu sein wie ursprünglich erhofft. Dieser Beitrag identifiziert einige der Gründe, indem Erkenntnisse aus dem Gebiet der komponentenbasierten Software-Entwicklung auf E-Learning und Lernobjekte übertragen werden.

Keywords: Lernobjekte, komponentenbasierte Software-Entwicklung, Software-Produktlinien, E-Learning-Produktlinien

1. Einführung

Die wachsende Komplexität von E-Learning-Kursen, die z.T. aus mehreren Dateien mit unterschiedlichen Formaten bestehen können, hat zur Entwicklung von Lernobjekten geführt. Lernobjekte kapseln Lerninhalte aller Art um sie effizient zu speichern, wieder zu verwenden und um sie zwischen verschiedenen Lernplattformen leichter austauschen zu können. Zahlreiche Standards, wie z.B. Dublin Core [6], IEEE LOM [14] oder ADL SCORM [25], um nur einige zu nennen, versuchen eine einheitliche Beschreibung von Metadaten zu definieren, um die Wiederverwendung und Komposition von Lernobjekten, beispielsweise zu größeren Kursen, zu vereinfachen. Obwohl heutige Learning-Management-Systeme (LMS) sich auf Lernobjekte stützen, ist das Konzept der Lernobjekte noch lange nicht so erfolgreich wie ursprünglich erhofft. Dies kann sowohl auf technische, als auch auf organisatorische und ökonomische Gründe zurückgeführt werden. Insbesondere sind auf der technischen Seite die gängigen Lösungen durch die Standards bezüglich Komposition, Wiederverwendung, Wartbarkeit und Interoperabilität von Lernobjekten für praktische Situationen noch nicht zufrieden stellend.

Dieser Beitrag benennt die wichtigsten Probleme der Lernobjekte und untersucht deren Ursprung, indem die Ansätze des verwandten Gebiets der komponentenbasierten Software-Entwicklung näher betrachtet werden. Dazu werden zunächst systematisch die Anforderungen an Komponentenmodelle, Kompositionstechniken und Kompositionssprachen aufgestellt, die für eine erfolgreiche Erstellung eines Softwaresystems aus standardisierten Komponenten unabdingbar sind. Benannt werden auch wichtige Kriterien für eine erfolgreiche Wiederverwendung von Komponenten. Anschließend werden die Erkenntnisse auf Lernobjekte im E-Learning übertragen. Durch einen Vergleich sollen bestehende Probleme identifiziert und Lösungsvorschläge entwickelt werden, die von Bedeutung für den erfolgreichen Einsatz von E-Learning sind.

2. Komponentenbasierte Software-Entwicklung

Dieses Kapitel fasst einige der wichtigsten Erkenntnisse aus der komponentenbasierten Software-Entwicklung zusammen. Die ersten Ideen in diesem Gebiet sind mittlerweile mehr als 35 Jahre alt [20]. Hauptsächlich geht es darum, dass zunächst einzelne Software-Komponenten entwickelt werden, bevor anschließend ein komplettes Software-System durch Komposition solcher Komponenten zusammengesetzt wird. Eine Komponente wird allgemein wie folgt charakterisiert [27]:

> *„A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties".*

Die obige Definition umfasst sowohl technische Aspekte von Komponenten, wie Unabhängigkeit im Sinne von Modularität und einen von außen nicht beobachtbaren Zustand, kontraktbasierte Schnittstellen und Komposition, als auch marktbasierte Aspekte wie dritte Parteien. Beispiele für Komponenten und Implementierungstechnologien sind in [27] zu finden. Obwohl ein komponentenbasiertes Vorgehen in manchen Fällen die Entwicklungszeit verkürzt und vergleichsweise kostengünstig ist, kann es in anderen Fällen problematisch sein, z.B. wenn vorhandene Komponenten nicht alle Anforderungen des Gesamtsystems erfüllen oder nicht zusammenpassen. Außerdem können durch die Verwendung von standardisierten Komponenten auch spezifische Konkurrenzvorteile aufgegeben werden. Viele Fragen sind immer noch unzureichend beantwortet [2]: Wie können Komponenten für Wiederverwendbarkeit vorbereitet werden? Wie können Komponenten während der Komposition flexibel angepasst werden? Wie soll ein komponentenbasierter Software-Entwicklungsprozess, bei dem möglicherweise mehrere Tausend Komponenten in verschiedenen Versionen beteiligt sind, idealerweise aussehen? Wir beginnen mit der Betrachtung von Komponentensystemen im Allgemeinen, unter Berücksichtigung der bislang gemachten Erfahrungen und betrachten anschließend die Anforderungen für eine erfolgreiche Wiederverwendung von Software-Komponenten.

2.1 Komponentensysteme

Eine allgemeine Taxonomie der Anforderungen an eine komponentenbasierte Software-Entwicklung wird in Abbildung 1 dargestellt [2]. Weiterhin zeigt sie, wie komponentenbasierte Systeme anhand von drei wesentlichen Kriterien verglichen werden können: Komponentenmodelle, Kompositionstechniken und Kompositionssprachen.

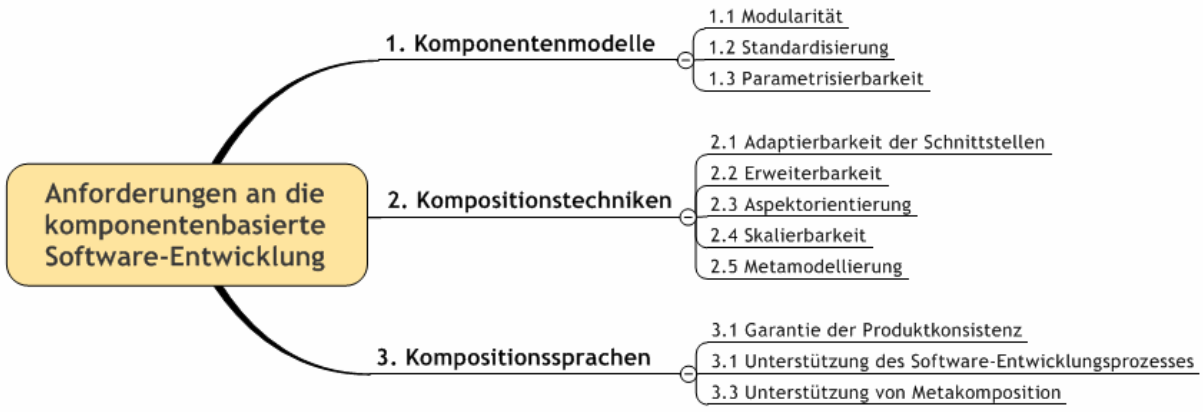

Abbildung 1: Anforderungen an Komponentensysteme nach [2]

Komponentenmodelle konzentrieren sich auf die Darstellung von Komponenten und legen präzise fest, wann zwei Komponenten austauschbar sind. *Kompositionstechniken* stellen Operationen für die Kom-

bination von Komponenten zur Verfügung. *Kompositionssprachen* beschreiben, wie aus einzelnen Komponenten größere Systeme gebaut werden können und liefern eine Beschreibung der Architektur.

Eine wichtige Anforderung im Rahmen der Komponentenmodelle ist *Modularität*, d.h. eine Komponente sollte nach dem Geheimnisprinzip gekapselt sein und Schnittstellen spezifizieren. Dabei reicht eine reine Schnittstellendefinition oft nicht aus, was zusätzlich den Einsatz von kontraktbasierten Spezifikationen für Schnittstellen erforderlich macht. Beispielsweise würde eine Komponente, die den Datentyp „Queue“ und die Operationen „Enqueue“ und „Dequeue“ in jeweils zwei getrennten Schnittstellen implementiert mit Hilfe eines Kontraktes ausdrücken können, dass alles was mit „Enqueue“ gespeichert wird mit „Dequeue“ wieder entfernt werden kann – eine Semantik, die aus den reinen Schnittstellenspezifikationen nicht ersichtlich ist. Solche Kontrakte können mit Hilfe von Vor- und Nachbedingungen als auch Invarianten spezifiziert werden (vgl. [27] für weitere Details). Die Verwendung verschiedener *Standards* für Komponenten vereinfacht den Software-Entwicklungsprozess, reduziert Fehlerquellen und erleichtert die Komposition von Komponenten. Zu beachten ist, dass ein erfolgreicher Standard dominierend sein muss, oder in einem Szenario mit mehreren konkurrierenden Standards jeder einzelne der Standards einen genügend großen Marktanteil hat. Es ist wichtig, bei zu vielen Standards die Anzahl zu reduzieren, damit ein einzelner Standard genügend Unterstützung findet. Standardisierung ist besonders schwierig in „horizontalen Märkten“, d.h., wenn ein Standard mehrere verschiedene Domänen schneidet und besonders viele verschiedene Interessen berücksichtigen muss [28]. Ein weiterer wichtiger Punkt im Komponentenmodell ist die *Parametrisierbarkeit* einer Komponente, um sie an den Kontext der Wiederverwendung anpassen zu können.

Im Hinblick auf Kompositionstechniken und einer flexiblen Wiederverwendung sollten die *Schnittstellen* einer Komponente für die Interaktion mit anderen Komponenten angepasst werden können, z.B. bezüglich ihrer verwendeten Typen von Parametern oder Protokollen. Sollte eine direkte Interaktion zwischen Komponenten aufgrund Ihrer Spezifikation nicht möglich sein, kann ein Komponentensystem sog. *„glue code“* als Mediator verwenden, der z.B. verschiedene Protokolle passend aufeinander abbildet. Weiterhin sollten vorhandene Komponentensysteme leicht um neue Teile und neue Funktionalität *erweiterbar* sein. Solche Updates sollten automatisiert durchgeführt werden können ohne die alten Teile des Systems gesondert editieren zu müssen. Eine *Aspektorientierung* sollte gewährleisten, dass Komponenten auch „Crosscutting Concerns“ [9] darstellen können, die sich über mehrere modulare Einheiten erstrecken können. Das bedeutet, dass die Implementierung einer Komponente, die zwar konzeptuell zusammengehört, physisch nicht an einer einzigen Stelle vorhanden sein muss, sondern dass Teile davon in verschiedenen anderen modularen Einheiten vorkommen können. Die *Skalierbarkeit* der Kompositionstechniken spielt auch eine entscheidende Rolle. Insbesondere sollten hierfür die Technik und der Zeitpunkt der Bindung von Komponenten verborgen bleiben. Eine *Metamodellierung* erleichtert die Beschreibung und Analyse von Komponenten und Kompositionstechniken.

Die Kompositionssprache eines Komponentensystems sollte die *Produktkonsistenz* des Endprodukts garantieren können, das mit Hilfe der Kompositionstechniken aus Komponenten erstellt wird. Unverzichtbar ist auch die Unterstützung des Software-Entwicklungsprozesses durch die Kompositionssprache, die vor allem mächtig genug sein sollte, um sowohl die benötigten Varianten, Versionen und Produktlinien als auch das Gesamtsystem beschreiben zu können. Gleichzeitig sollte eine Kompositionssprache nicht schwer zu verstehen sein. Letztendlich sollten Kompositionssprachen Metakomposition unterstützen, d.h., sie selbst sollten auch komponentenbasiert sein, um auf Architekturebene wieder verwendet werden zu können.

2.2 Wiederverwendung

Wiederverwendung wird als *Prozess* verstanden, bei dem existierende Artefakte benutzt werden um ein Software-System zu konstruieren. Alle vorhandenen Techniken der Wiederverwendung (vgl. [16,26]) haben Gemeinsamkeiten, die sich entlang der folgenden Dimensionen beschreiben lassen [16]:

- **Abstraktion:** Alle Ansätze verwenden irgendeine Form von Abstraktion, z.B. für die Beschreibung der Inhalte von Komponenten oder in der Anleitung zur Wiederverwendung.

- **Selektion:** Die meisten Ansätze erlauben es, wieder verwendbare Artefakte zu suchen, zu vergleichen und zu selektieren. Mit Hilfe von Klassifikation und Katalogisierung können Bibliotheken wieder verwendbarer Artefakte erstellt werden.

- **Spezialisierung:** Oft werden mehrere ähnliche Artefakte zu einem einzigen, allgemeineren Artefakt zusammengefasst. Dieses lässt sich dann später mit Hilfe von Parametern, Transformationen oder anderen Restriktionen an den Benutzungskontext anpassen.

- **Integration:** Typischerweise verwenden Wiederverwendungstechniken Frameworks, die genauer spezifizieren wie eine Menge von mehreren Artefakten wieder zu verwenden ist. Beispielsweise könnte ein Framework aus einer Menge von erweiterbaren konkreten und abstrakten Klassen bestehen, die bereits miteinander verzahnt sind. Während der Wiederverwendung werden individuelle Implementierungen der abstrakten Klassen erstellt [8].

Weiterhin wird in [16] bezüglich der Wiederverwendung gefordert:

1. *”For a software reuse technique to be effective, it must reduce the cogntive distance between the initial concept of a system and its final executable implementation.”*

2. *“For a software reuse technique to be effective, it must be easier to reuse the artifacts than it is to develop the software from scratch.”*

3. *“To select an artifact for reuse, you must know what it does.”*

4. *“To reuse a software artifact effectively, you must be able to “find it” faster than you could “build it.”*

In der ersten Forderung ist die kognitive Distanz kein genaues Maß, sondern eine informelle Bezeichnung dafür wie viel intellektueller Aufwand seitens eines Software-Entwicklers notwendig ist, um ein Software-System von einem Entwicklungszustand zum nächsten zu überführen. Beispielsweise kann die kognitive Distanz durch Abstraktion reduziert werden, wenn bestimmte Teile eines Systems hinter Schnittstellen „versteckt“ werden. Obwohl die Umsetzung dieser Forderung einfach zu sein scheint, ist sie in der Praxis schwierig durchzuführen (vgl. NASA Goddart Reuse Experiment [19]). Die zweite Forderung nimmt indirekt Bezug darauf, dass oft Quell-Code von Artefakten von anderen Entwicklern blockweise kopiert und manuell angepasst wird. Dieses Vorgehen erfordert gleichzeitig ein detailliertes Verständnis des benutzten Codes, so dass die Wiederverwendung unter Umständen schwieriger, langwieriger und teuerer sein kann als eine Neuprogrammierung. In der dritten Forderung wird deutlich gemacht, dass eine syntaktische Beschreibung von Schnittstellen eines Artefaktes nicht ausreicht, um das Verhalten, den Kontext und die geplante Art der Wiederverwendung zu beschreiben (vgl. Kontraktspezifikationen im vorangegangenen Abschnitt). Die letzte Forderung bezieht sich schließlich auf die Notwendigkeit einer adäquaten Organisation von Artefakten.

Der oft geäußerte Wunsch nach einer Maximierung der Wiederverwendung einer Komponente sollte wohl überlegt werden, da gleichzeitig die Anzahl der Kontextabhängigkeiten steigen kann. Die Umgebungen, in denen Komponenten verwendet und ausgeführt werden, als auch die Komponenten selbst werden oft in neuen Versionen weiterentwickelt (auch Schnittstellen o.ä). Aus Sicht einer Komponente müssen daher bei einer hohen angestrebten Wiederverwendung die unterschiedlichen Versionen der Zielumgebung berücksichtigt werden, was die Komponente vergrößert und deren Programmierung und Verständnis erschwert. Ebenso muss eine Zielumgebung mehrere Versionen verschiedener Kom-

ponenten unterstützen. Diese Effekte beeinflussen, anders als geplant, die Wiederverwendung in einer negativen Art und Weise [27].

Abschließend sei noch bemerkt, dass sich viele der derzeitigen Ansätze zur Wiederverwendung auf die Ebene von Quell-Code beziehen, die typischerweise etwa 20% der Entwicklungskosten eines Projekts ausmacht [11]. Wiederverwendung im Kontext der Software-Erstellung ist immer noch nicht zufrieden stellend gelöst. Die für Wiederverwendung gedachten Software-Bibliotheken enthalten aufgrund der wachsenden Komplexität eine immer größer werdende Zahl von Komponenten, die schwer auffindbar sind. Dementsprechend sind geringe Verwendungsgrade zu verzeichnen. Neuere Ansätze, wie z.B. Software-Produktlinien, versuchen diese Nachteile zu kompensieren, indem Wiederverwendung durch Analysieren von Gemeinsamkeiten mehrerer Programme auch auf Architektur- und Konstruktionsprozessebene betrieben wird und gleichzeitig die Zahl der Varianten eingeschränkt wird.

3. Lernobjekte im E-Learning

Die Erkenntnisse aus dem Gebiet der komponentenbasierten Software-Entwicklung werden in diesem Kapitel auf Lernobjekte im E-Learning-Umfeld übertragen, und es wird eine kritische Beurteilung zugrunde liegender Konzepte durchgeführt. Obwohl generell die Notwendigkeit von Lernobjekten anerkannt ist um Lernmaterial zu kapseln, leichter wieder zu verwenden und besser auszutauschen, gibt es bislang keine einheitliche Definition von Lernobjekten. Einige Beispiele, wie konkrete Lernobjekte aussehen können, sind in [24] zu finden. Anders als im Sinne der objektorientierten Programmierung kapseln Lernobjekte Daten und stellen überwiegend keine Methoden zur Verfügung [17]. In neueren Ansätzen, die auf Web Services basieren, wird die Aufhebung dieser Einschränkung bereits untersucht [23]. Im Folgenden werden wir die in Kapitel 2 erarbeiteten Forderungen an Komponentensysteme und die Kriterien für eine erfolgreiche Wiederverwendung im Kontext der Lernobjekte betrachten.

3.1 Lernobjekte aus Sicht von Komponentensystemen

Da Lernobjekte im Grunde genommen selbst Komponenten darstellen, die ein System bilden sollen (z.B. einen Kurs), müssen sie auch die Anforderungen erfüllen, die an Komponentensysteme gestellt werden:

1. **Komponentenmodelle:** Da momentan Uneinigkeit bei der Definition eines Lernobjektes herrscht, gibt es kein einzelnes, allgemeingültiges Komponentenmodell. Die Vergleichbarkeit von Lernobjekten ist kaum möglich, da unterschiedliche Annahmen bzgl. Aufbau, Inhalten, Granularitätsebenen, Dateiformaten oder Layout zugrunde liegen können.

    - **Modularität:** Diese Anforderung wird erfüllt, indem zusammengehöriges Lernmaterial (meist auf logischer Ebene) gekapselt wird.

    - **Standardisierung:** Es gibt momentan eine kaum überschaubare Vielzahl von Bestrebungen unterschiedlicher Gremien, Standards für Lernobjekte zu schaffen, z.B.: Dublin Core [6], Aviation Industry CBT Committee (AICC) [1], IMS Global Learning Consortium [15], IEEE Learning Object Metadata (LOM) [14], um nur einige zu nennen. Weiterhin gibt es Gremien, die von vorhandenen Standards neue Standards ableiten, wie z.B. EdNA [7] von Dublin Core oder andererseits CanCore [4], UK LOM Core [28] oder CELTS [30] von IEEE LOM. Der ADL Sharable Content Object Reference Model (SCORM) Standard [25] versucht schließlich, unterschiedliche Teile von mehreren anderen Standards zu einem eigenen Standard zu etablieren. Weitere Spezifikationen gibt es auch aus der Wirtschaft, wie das CISCO RLO/RIO Modell [3] oder das NETg Learning Object Model [18]. Die Ziele der verschiedenen Standards sind teilweise recht unterschiedlich. Während einige versuchen, Metadaten für Dateien zur Verfügung zu stellen um die Suche nach Inhalten zu erleichtern, sind andere eher bestrebt eine genauere Architekturbeschreibung von Lernobjekten zu liefern. Dabei wird auch übersehen, dass viele Dateiformate, die im E-Learning benutzt werden, wie z.B. PDF, Word, Powerpoint, bereits eigene Metadaten ent-

halten, die jedoch nicht in die Standards integriert werden. Es ist offensichtlich, dass das Verständnis, was ein Lernobjekt ist und wie es aufgebaut sein muss, weit auseinander driftet. Der folgende Vergleich soll das Spektrum beispielhaft verdeutlichen. Der IEEE-LOM-Standard definiert sehr allgemein: „a learning object is defined as any entity, digital or non-digital, that may be used for learning, education or training“ [14]. Der Dublin Core Standard definiert lediglich 15 Deskriptoren für ein Lernobjekt bzgl. Inhalt (“title, subject, description”), geistigem Eigentum (“creator, publisher, contributor, rights”) und Dokumentinstanzen (“date, type, format, identifier, source, language, relation, coverage”). Demgegenüber fordert das CISCO-Modell in einem “Reusable Learning Object” genau 7 ± 2 sog. “Reusable Information Objects”, die Informationen für ein bestimmtes Lernziel repräsentieren und die um weitere Teile für Übersicht, Zusammenfassung und Online-Tests ergänzt werden müssen.

- **Parametrisierbarkeit:** Wird momentan unzureichend von den Standards unterstützt, da Lernobjekte überwiegend eine Menge vordefinierter Daten kapseln. Der IMS-Standard erlaubt als Parametrisierung lediglich die Definition von mehreren, unterschiedlichen Organisationsstrukturen für einen Kurs, d.h., dass für dieselben Dateien in einem Lernobjekt mehr als eine Reihenfolge definiert werden kann. Auf Web Services basierende Lernobjekte könnten zukünftig eine flexiblere Parametrisierbarkeit erlauben, die auch dynamisch während der Präsentation von Inhalten eines Objektes durchgeführt werden kann.

2. **Kompositionstechniken:** Da Lernobjekte lediglich als statische Datenpakete angesehen werden, wird deren Komposition zu kompletten Kursen derzeit mit Hilfe Laufzeitumgebungen in E-Learning-Plattformen (sog. Learning Management Systeme, LMS) vorgenommen. Dies geschieht meist manuell durch einen Lehrenden. Obwohl eine Komposition auch automatisiert anhand der Metadaten durchgeführt werden könnte, führt dies momentan aufgrund der fehlenden Einheitlichkeit im Komponentenmodell oft nicht zum gewünschten Erfolg. Ein LMS besitzt typischerweise eine Präsentationsschicht, in der die Inhalte der Lernobjekte aufbereitet und den Lernenden angezeigt werden. Weiterhin kann ein LMS bei Bedarf Interaktionen mit den Lernenden (z.B. bei Online-Tests) koordinieren. Eine Interaktion zwischen Lernobjekten ist momentan in den heutigen Modellen noch nicht vorgesehen.

   - **Adaptierbarkeit der Schnittstellen:** Lernobjekte kapseln nur Daten und keine Funktionalität. Sie besitzen daher keine Schnittstellen im klassischen Sinne. Einige der Metadaten (z.B. IMS) könnten als eine Art von Schnittstellen verstanden werden, da sie zu einer logischen Kursorganisation den Ort der physischen Dateien angeben. Andere Standards, wie z.B. Dublin Core, machen keine Schnittstellen-ähnliche Angaben. Kontrakte für Schnittstellen spielen derzeit eine untergeordnete Rolle.

   - **Erweiterbarkeit:** Lernobjekte können häufig nur so wieder verwendet werden, wie sie vorhanden sind. Dies ist nicht nur auf die Standards, sondern auch auf die in Lernobjekten enthaltenen Dateiformaten zurückzuführen, die nicht änderbar sind. Beispielsweise werden die Dateiformate PDF oder Macromedia Flash häufig im E-Learning benutzt, ohne dass die zugehörigen Quelldateien weitergegeben werden. Ein weiterer Nachteil des Flash-Formates (auch der Quelldateien) ist die fehlende logische Trennung zwischen Darstellung und Inhalt.

   - **Aspektorientierung:** „Crosscutting Concerns“ als konzeptuell zusammengehörige Teile, die in mehreren getrennten Lernobjekten vorkommen (z.B. Online-Tests, Layout, Bepreisung) werden von den Standards nicht berücksichtigt, was zu einer redundanten Speicherung und Schwierigkeiten bei der Wartung führen kann [22].

   - **Skalierbarkeit:** Die Probleme hier sind ähnlich wie bei der Erweiterbarkeit.

   - **Metamodellierung:** Nur sehr rudimentär in einigen Standards berücksichtigt, z.B. als „Meta-Metadata“ Deskriptoren in IEEE LOM.

3. **Kompositionssprachen:** Obwohl einige Forschungsprototypen eine automatisierte Komposition von Lernobjekten durchzuführen versuchen, scheint das Interesse an einer allgemeinen, standardisierten Kompositionssprache für Lernobjekte gering zu sein.

- **Garantie der Produktkonsistenz:** Die Unterschiede und Ungenauigkeiten in den vorhandenen Komponentenmodellen erlauben keine garantierte Produktkonsistenz.
- **Unterstützung des Entwicklungsprozesses:** Manche Standards (z.B. Dublin Core) besitzen nicht die Ausdruckskraft, um den Entwicklungsprozess z.B. mit detaillierteren Konfigurations- und Schemainformationen zu unterstützen.
- **Unterstützung von Metakomposition:** Spielt momentan bei Lernobjekten eine untergeordnete Rolle.

3.2 Lernobjekte aus Sicht der Wiederverwendung

Die festgestellten Unzulänglichkeiten von Lernobjekten aus Sicht von Komponentensystemen wirken sich auch auf die Wiederverwendung aus. Dies wird auch durch eine empirische Studie zur Verwendung des IEEE-LOM-Standards bestätigt, die in den Jahren 2003-2004 von einem ISO/IEC-Komitee durchgeführt wurde [10]. In einem internationalen Rahmen wurden 5 größere Bibliotheken von Lernobjekten untersucht, die mehrere Sammlungen mit 75 bis über 3000 Lernobjekte zur Verfügung gestellt haben. Gemessen wurde die Häufigkeit der Verwendung von Elementen des LOM-Standards. Das Ergebnis, das die Metadaten-Attribute für Lernobjekte aus den LOM-Kategorien „General, Lifecycle, Meta-Metadata, Technical, Educational, Rights, Relation“ und „Annotation“ hervorhebt, wird in Abbildung 2 gezeigt.

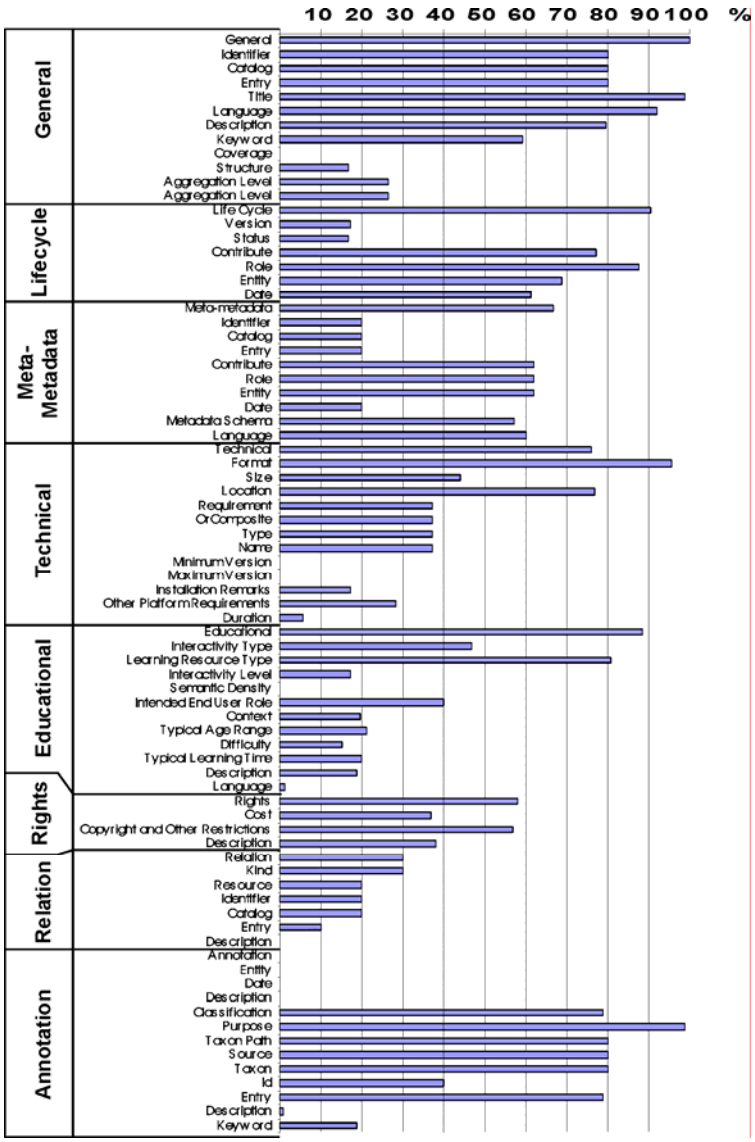

Abbildung 2: ISO-Studie: Häufigkeit der Verwendung von Elementen des LOM-Standards [10]

Es wurde festgestellt, dass nur wenige der potenziellen Attribute häufig benutzt werden und dass diese oft durch Dublin-Core-Attribute abgedeckt werden können. Das Vokabular für Schlüsselwörter u.ä. wird häufig nicht konsistent benutzt. Viele Attribute, die den Entwicklungsprozess und die Evolution eines Lernobjektes beschreiben, wie z.B. Versionierungsinformationen in der Kategorie „Technical", wurden gar nicht benutzt. Des Weiteren ist die Verwendung von Attributen in der Kategorie „Educational" sehr gering, obwohl dieser Standard gerade für die Beschreibung von Lernmaterial geschaffen wurde. Bei der Verwendung von IEEE LOM als Teil von SCORM oder UK LOM Core werden sogar alle „Educational"-Attribute als optional deklariert [10]. Die Attribute der Kategorie „Relation", die semantische Beziehungen zwischen Lernobjekten modellieren sollen, wurden in weniger als 30% der Fälle benutzt. Abschließend folgert die Studie, dass LOM nicht in der Lage ist eine einfache Interoperabilität und Portabilität der Daten zwischen verschiedenen Sammlungen von Lern-Ressourcen mit konventionellen Methoden der Datenverarbeitung zu gewährleisten.

Diese Ergebnisse beeinflussen wiederum die Dimensionen der Wiederverwendung für Lernobjekte:

- **Abstraktion:** Ein geeignetes Abstraktionsniveau scheint momentan bei der Beschreibung von Lernobjekten noch nicht erreicht worden zu sein. Dafür spricht auch der geringe Verwendungsgrad vieler Attribute in der besprochenen Studie (vgl. auch [12,29]). Weiterhin ist die Granularität der Inhalte in Lernobjekten oft nicht genau festgelegt (d.h. ein Lernobjekt kann z.B. einen Kurs, eine Lerneinheit oder nur einige Bildschirmseiten beinhalten).
- **Selektion:** Obwohl die vorhandenen Standards die Klassifikation und Katalogisierung von Lernobjekten erleichtern wollen, zeigt die empirische Studie, dass eine inkonsistente Verwendung eines Vokabulars sowie eine seltene Verwendung von Schlüsselwörtern hinderlich sein kann. Außerdem erschwert die Vielzahl von Standards sowohl die Annotation mit Metadaten, als auch die Suche nach Lernobjekten.
- **Spezialisierung:** Eine Anpassung von „fremden" Lernobjekten an eigene Bedürfnisse ist momentan praktisch nur mit großem Aufwand möglich. Typischerweise werden Lernobjekte genau so wieder verwendet, wie sie erstellt worden sind. Die Anpassbarkeit, Änderbarkeit und Wartbarkeit der Inhalte von Lernobjekten ist schwierig, da oft Dateiformate wie PDF oder Macromedia Flash benutzt werden, ohne dass die Quelldateien weitergegeben werden.
- **Integration:** Die Integration verschiedener Lernobjekte wird in E-Learning-Plattformen meist manuell durchgeführt. Problematisch ist, dass aufgrund der Vielzahl von Standards nicht alle Systeme alle existierenden Standards beherrschen.

Weiterhin wird die Forderung nach einer Reduktion der kognitiven Distanz (vgl. Abschnitt 2.2) zwischen dem Konzept eines Systems (hier beispielsweise eine Kurseinheit, ein Kurs oder ein Lehrplan) und seiner Implementierung, mit den vorhandenen Abstraktionsmechanismen schlecht erfüllt. Oft wird wegen fehlender Spezialisierungsmechanismen auch die Forderung verletzt, dass eine Wiederverwendung von Lernobjekten einfacher sein soll, als eine komplette Neuerstellung. Die besprochene Studie hat gezeigt, dass auf einer semantischen Ebene oft nicht ausreichend beschrieben wird, was ein Lernobjekt genau tut und wie es sich von anderen unterscheidet, was die dritte Forderung verletzt. Schließlich ist auch die Erfüllung der letzten Forderung nach einer schnellen Suche angesichts der vielen Standards problematisch.

3.3 Fazit

Die noch fehlende Konvergenz bei der Standardisierung von Lernobjekten hemmt momentan die Entstehung von präziseren Komponentenmodellen. Diese sind eine wichtige Voraussetzung für die Bereitstellung von Kompositionstechniken und Kompositionssprachen für Lernobjekte. Aus einer allgemeineren praktischen Sicht wird in [13] weiterhin bemängelt, dass eine Implementierung nicht als komplexer Prozess angegangen wird und dass beteiligte Akteure oft mangelnde Kompetenzen besitzen. Auch für Lernobjekte gilt im Sinne von [27]:

> "*It is far too simplistic to assume that components are simply selected from catalogs, thrown together, and magic happens. In reality, the disciplined interplay of components is one of the hardest problems of software engineering today*".

Dieses Zitat und die gemachten Ausführungen zu Komponentensystemen sollen klarmachen, dass die derzeitige Sicht der E-Learning-Community - alles sei überwiegend nur eine Frage der Standardisierung von Metadaten für die Suche von Lernobjekten in Katalogen - verworfen werden muss. Die Probleme im Bereich der komponentenbasierten Software-Entwicklung gehören zu den derzeit schwierigsten im Software-Engineering, an denen seit über 30 Jahren geforscht wird. Deswegen können für ähnliche Probleme bei Lernobjekten im E-Learning auch keine schnellen, allgemeingültigen Lösungen erwartet werden - insbesondere dann nicht, wenn die bereits gemachten Erfahrungen im Software-Engineering ignoriert werden.

Neben der Schaffung von genauer spezifizierten Komponentenmodellen, Kompositionstechniken und Kompositionssprachen wäre zusätzlich die Verwendung von Frameworks und Software-Produktlinien, angewendet auf Lernobjekte, von Vorteil. Auf diese Weise könnte man in einer Produktlinie für E-Learning-Material für eine *Menge* von Lernobjekten gemeinsame Anforderungen spezifizieren bzgl. des zu verwendenden Aufbaus, Inhalten, Granularitätsebenen, Dateiformaten, Organisation, Layout oder Metadatenstandards. Somit könnten auch gewisse Garantien für Lernobjekte, die Teil einer bestimmten E-Learning-Produktlinie sind, als auch für die durch deren Komposition entstandenen Ergebnisse, abgegeben werden. Weitere Details zu diesem Ansatz sowie zu Modellierungstechniken und Werkzeugen sind in [21] zu finden.

Abschließend soll noch diskutiert werden, ob grenzenlose Interoperabilität und unkomplizierte Migration von Lernobjekten aus der ökonomischen Sicht eines kommerziellen Anbieters von E-Learning-Material in allen Situationen wünschenswert sind. Aufgrund der zunehmenden Standardisierung der Funktionalität von Learning Management Systemen bleiben die E-Learning-Inhalte und damit Lernobjekte womöglich der letzte komparative Konkurrenzvorteil, mit dem sich ein Anbieter von anderen abheben kann. Konsequenterweise muss dieser einerseits Eintrittsbarrieren gegenüber den anderen Anbietern schaffen, was er durch qualitativ hochwertiges, aber proprietäres Lernmaterial erreichen kann, das beispielsweise nur in seiner speziellen Umgebung ausgeführt werden kann. Andererseits kann ein Anbieter die Kundenbindung vorhandener Kunden verstärken, indem er zusätzlich Austrittsbarrieren schafft. Dies geschieht, indem die Migration und Interoperabilität von Lernobjekten erschwert wird. Angesichts der hohen Kosten, die bei der Erstellung von Lernmaterial anfallen, dient dieses Vorgehen auch der Investitionssicherung. Andererseits zwingt solch ein Vorgehen zur Programmierung von proprietären Werkzeugen für die Erstellung von Inhalten, was kostenintensiver als eine Beschaffung bereits vorhandener Werkzeuge sein kann. Letztendlich kann jedoch nur eine genauere empirische Untersuchung klären, welche Vorteile überwiegen. Die Kenntnis dieser Kostenstrukturen ist für die Entwicklung von Anreizsystemen zur Benutzung von Standards wichtig.

4. Zusammenfassung und Ausblick

Dieser Beitrag identifiziert Unzulänglichkeiten im Konzept der Lernobjekte im E-Learning. Hierfür wurden die Entwicklungen und Erfahrungen aus dem Bereich der komponentenbasierten Software-Entwicklung dargestellt und im E-Learning-Kontext angewendet. Die verwendeten Kriterien können auch als Erfolgsfaktoren für den Einsatz von Lernobjekten angesehen werden. Weiterhin zeigt der Vergleich neue Perspektiven auf, wie z.B. die Verwendung von Frameworks und Software-Produktlinien im E-Learning, die konzeptionelle Lücken schließen können.

Literatur

1. *Aviation Industry CBT Committee (AICC),* http://www.aicc.org/, Juli 2005
2. Aßmann, U. *Invasive Software Composition*, Habilitationsschrift, Springer-Verlag, 2003
3. Barrit, C. *CISCO Systems Reusable Learning Object Strategy- Designing Information and Learning Objects Through Concept, Fact Procedure, Process, and Principle Templates*, Version 4.0. White Paper, CISCO Systems, Inc., November 2001.
4. *CanCore Metadata Standard*, http://www.cancore.ca/en, Juli 2005
5. Downes, S. *Design and Reusability of Learning Objects in an Academic Context: A New Economy of Education?* USDLA Journal Nr. 17, 2003
6. *Dublin Core Metadata Initiative*, http://dublincore.org/ , Juli 2005
7. *Edna (Education Network Astralia) Metadata Standard*, http://www.edna.edu.au/edna/go/pid/385 , Juli 2005
8. Fayad, M.E., Schmidt, D.C., *Object-oriented application frameworks*, Comm. ACM, 40(10), 32-38
9. Filman, R.E, et.al, *Aspect-Oriented Software Development*, Addison-Wesley, 2004
10. Friesen, N. *Final Report on the "International LOM Survey"*, ISO/IEC JTC1 SC36 N0871, 2004
11. Gomaa, H., *Designing Software Product Lines with UML : From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley, 2004
12. Heath, B.P. et.al., *Metadata lessons from the iLumina digital library*, Commun. ACM, 48(7), 2005, 68-74
13. Hohenstein, A., *Blended Learning erfolgreich gestalten? – eine Geschichte der Weiterbildungspraxis!*, in: Wilbers, K. *Stolpersteine beim Corporate E-Learning*, Oldenbourg, 2005
14. IEEE Computer Society, *IEEE Standard for Learning Object Metadata*, IEEE Std P1484.12.1TM-2002, September 6, 2002
15. *IMS Global Learning Consortium, Inc. – Specifications*, http://www.imsglobal.org/specifications.cfm, Juli 2005.
16. Krueger, C. W. *Software Reuse*, ACM Computing Surveys, Vol. 24, No. 2, Juni 1992
17. Knolmayer, G. F. *E-Learning Objects*, Wirtschaftsinformatik 46 (2004) 3, S. 222-224
18. L'Allier, J. J., *A Frame of Reference: NETg's Map to Its Products, Their Structures and Core Beliefs*, 1997 http://www.netg.com/research/whitepapers/index.asp
19. Leveson, N.G., Weiss, K.A., *Making embedded software reuse practical and safe*, Proc. 12th ACM SIGSOFT international symposium on Foundations of software engineering, 2004, 171-178
20. McIlroy, M.D., *Mass-produced software components*, Proc. NATO Conf. on Software Engineering, 1968
21. Oberweis, A., Pankratius, V., Stucky, W., *Product Lines in E-Learning*, Forschungsbericht des Instituts AIFB, Universität Karlsruhe, August 2005, ISBN 3-9810441-0-X
22. Pankratius, V. *Aspect-oriented Learning Objects*, 4th IASTED International Conference on Web-based Education, Grindelwald, Schweiz, 2005
23. Pankratius, V., Sandel, O., Stucky, W., *Retrieving Content with Agents in Web Service E-Learning Systems.* In IFIP Symposium on Professional Practice in AI, IFIP WG12.5, Toulouse, France, August 2004.
24. Pankratius, V., Vossen, G., Reengineering of Educational Material: A Systematic Approach, International Journal of Knowledge and Learning (IJKL), 1(3), 2005
25. *Sharable Content Object Reference Model (SCORM),* http://www.adlnet.org, Juli 2005.
26. Sommerville, I., *Software Engineering*, 7th edition, Pearson Addison Wesley, 2004
27. Szyperski, C., Gruntz, D., Murer, S., *Component Software*, Addison-Wesley, 2002
28. *UK Learning Object Metadata Core*, http://www.cetis.ac.uk/profiles/uklomcore/uklomcore_v0p1.doc, Draft, Mai 2004
29. Ward, J., *A quantitative analysis of unqualified Dublin Core metadata element set usage within data providers registered with the Open Archive Initiative*, Proc. Joint Conference on Digital Libraries, IEEE Computer Society, 2003, 315-317
30. Xian, X., et.al., *Introduction of the Core Elements Set in Localized LOM Model*, Chinese E-Learning Technology Standard, ISO/IEC JTC1 SC36 working document, Sept. 2003, http://mdlet.jtc1sc36.org/doc/SC36_WG4_N0059.pdf