ATHABASCA UNIVERSITY

AGILE AND OPEN E-LEARNING SYSTEMS ARCHITECTURE

BY

ALAN SWEENEY

An essay submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

November, 2008

© Alan Sweeney, 2008
This page is intentionally blank.
DEDICATION

I would like to dedicate this essay to my lovely wife Merrilyn who has encouraged and supported me, and without whom the completion of this essay would not have been possible.
ABSTRACT

E-learning systems are becoming increasingly important with the advances in both e-learning pedagogy and technology. These systems are susceptible to software architecture problems that are caused by changing technological, pedagogical and integration requirements that continuously challenge the dependency design of the software. The emergence of lightweight, agile software development methodologies and principles help address many of the challenges associated with changing requirements. Similarly, there are many advantages to using open source software and principles to address development and integration challenges. However, the principles and practices associated with the agile and open methodologies present interesting challenges for the software design and architecture. This essay reviews the characteristics of e-learning systems development and examines the advantages that agile and open development principles offer in addressing some of the bigger challenges in this field. These principles are examined from an architectural perspective to discover desirable characteristics of a lightweight, open architecture that can support open and agile methodologies. These characteristics are modeled as a template prototype and open source frameworks and tools are presented that could help implement an agile and open e-learning systems architecture.
ACKNOWLEDGEMENTS

I would like to acknowledge the support and encouragement from my supervisor, Dr. Dunwei Wen, whose respected expertise and encouraging words early on provided me with the confidence to finish this project. I would also like to acknowledge my mother, Dr. Lorine Sweeney, whose experience I relied on many times to guide me throughout this endeavour. Finally, I would like to thank my wife, to whom this paper is dedicated, as well as my daughter for their unending patience and support during my studies.
# TABLE OF CONTENTS

CHAPTER I - INTRODUCTION ................................................................................................. 1  
  Research Problem or Question ................................................................. 2
  Statement of Purpose .................................................................................. 3
  Significance ................................................................................................. 4
  Limitations .................................................................................................. 5
  Organization of the Essay .......................................................................... 6

CHAPTER II - REVIEW OF RELATED LITERATURE .............................................................. 8  
  E-learning Systems ..................................................................................... 8
    Overview ..................................................................................................... 9
    Technology .................................................................................................. 10
    Pedagogy ..................................................................................................... 12
    Integration ................................................................................................... 18
  E-learning Systems Development ................................................................. 21
  Software Development Methodologies and Principles ............................... 21
    E-learning Systems .................................................................................. 23
    Agile Software Development .................................................................... 25
    Free/Open Source Software ...................................................................... 29
  E-learning Systems Architecture .................................................................. 31
    Agile and Open Source Architecture ......................................................... 36
  Summary ...................................................................................................... 36

CHAPTER III - METHODOLOGY ......................................................................................... 38
  Introduction .................................................................................................. 38
  Agile Architecture ....................................................................................... 39
    Challenges ................................................................................................ 39
    Controversies ............................................................................................ 40
    Design Principles ....................................................................................... 41
    Design Patterns ........................................................................................ 44
    Lightweight Containers ............................................................................. 45
    Domain Model .......................................................................................... 46
    Data Mapper and Repository .................................................................... 46
  Agile Practices ............................................................................................... 48
    Test Driven Development .......................................................................... 48
    Continuous Integration ............................................................................ 49
    Refactoring ............................................................................................... 49
    Conclusion ............................................................................................... 51
  Open Source Architecture .......................................................................... 51
    Selecting FOSS Tools .............................................................................. 51
    FOSS Licenses ......................................................................................... 53
  Agile, FOSS and E-Learning ....................................................................... 55
LIST OF TABLES

Table 1. Sakai Abstract Architecture Layers.................................................................81
Table A1. E-learning Systems Architectural Technologies...........................................99
Table A2. E-learning Standards and Specifications.....................................................100
Table A3. Software Package Metrics...........................................................................103
LIST OF FIGURES

1. Logical Layered Architecture ................................................................. 62
2. Domain Model ........................................................................................ 64
3. Controller Layer ................................................................................... 66
4. Data Mapping Layer ............................................................................... 67
5. Inversion of Control Assembler .............................................................. 69
6. Sample Spring Framework Configuration ............................................. 70
7. Hibernate Data Mapping Layer .............................................................. 72
8. Sample Change Cases .......................................................................... 74
A1. Sample Spring Application Context ...................................................... 105
A2. Sample Hibernate Session Configuration ........................................... 106
A3. Sample Hibernate Mapping File .......................................................... 106
CHAPTER I

INTRODUCTION

E-learning systems have become a central part of the learning process and play an increasingly important role in personalized student-centred learning. The concept and application of e-learning has become progressively more prevalent in educational settings ranging from modern post-secondary institutions to the smallest and most remote rural schools; as well, e-learning systems now have an integral role in many educational organizations. The ability to manage and deliver online courses has become an important aspect of the learning models, and this importance has created a tremendous dependency upon e-learning systems as educators strive to deliver quality education to their students.

E-learning systems have become much more than course management and delivery systems as they increasingly facilitate the educational need to communicate, share data, problem solve and assist students as they cooperatively arrive at new understandings and solutions. Pedagogically, teachers are moving to a constructivist environment for learning that lends itself to the creation of new understandings and new ways of problem solving. This evolution of pedagogical methodology has in turn caused the required growth and flexibility of e-learning systems. New technologies are continuously being introduced to the e-learning systems landscape that allow students to have more control over their own learning, to think analytically and critically, and to work collaboratively in new environments.

As the reliance on these systems has increased at all levels of education,
so too has the need to provide solutions that can integrate the e-learning systems with other technical infrastructure within the organizations. It is a tremendous challenge to effectively integrate new e-learning applications and technology within an organization’s systems infrastructure in a manner that can ensure the efficiency of all processes. Software architects are challenged to provide a system design that possesses the flexibility to allow integration and the adaptability to support new technologies and emerging pedagogical practices.

**Research Problem or Question**

With e-learning systems becoming increasingly important, and e-learning pedagogy and technology continually evolving and advancing, systems are particularly susceptible to design problems that are caused by improper dependencies between modules of software. As the requirements change in ways that an initial design did not anticipate, the dependency architecture degrades and the design becomes rigid and immobile, causing a fragility in the software that stagnates development and hampers the maintainability of the system. In addition, e-learning systems that are becoming integrated in the learning institution’s overall infrastructure are introduced to dependencies on external systems that can magnify the problems associated with the dependency architecture.

These problems are certainly not unique in the design of software systems, and there are many well-documented approaches that provide guidelines for addressing them through development principles and practices. This essay proposes the use of lightweight, agile software development methodologies and
principles to help address many of the challenges associated with changing requirements. Similarly, there are many advantages to using open source software and open source development practices to address some of the issues associated with e-learning system development and integration. In addition, both of these philosophies (agile software development methodology and open source software) contain principles that are well suited for the fields of education and academia that are served by e-learning systems.

The challenge presented to an e-learning software architect is to design an architecture that is not hampered with dependency issues, uses open source frameworks and philosophies, and is designed in such a way that it is compatible with design principles that are encouraged as part of agile software development.

Statement of Purpose

The purpose of this research paper is to investigate lightweight agile software development and open source philosophies and their suitability for e-learning system development, and then to explore the challenges that these methodologies pose to the design and architecture of an e-learning system. These issues are examined with attention focused on the architectural challenges of implementing a model of an e-learning system that can provide functionality for new and existing e-learning techniques, technology, and standards, while following the principles of agile, lightweight and open development processes. The author proposes to provide guidelines for e-learning architectural decisions by:

● Investigating the fundamental requirements of e-learning architecture
by examining the characteristics of e-learning systems that makes their software development uniquely challenging.

- Reviewing lightweight development methodologies such as agile software development as well as open source philosophies to evaluate their effectiveness for addressing the challenges of e-learning systems development.

- Exploring architectural frameworks, standards and technologies that are used in e-learning systems development and discussing the future direction of these architectures.

- Proposing lightweight container architecture and associated frameworks, design patterns and technologies that address the requirements of e-learning system architecture while also following open source philosophy and can be implemented and maintained within an agile software development process.

E-learning software architecture needs to address complex issues, and these issues can be compounded by constraints put in place by software development methodologies. This paper addresses this by first examining the suitability of specific development methodologies for e-learning systems development, and then exploring architectural decisions that are supportive of the chosen development methodologies.

Significance

There has been a significant amount of research conducted in the area of open source software development for educational institutions, much of it
focused on how open source principles and practices can improve software development, and also benefit the educational community as a whole. Additionally, agile methodology has become a popular research topic in many different fields of software development where there is the need to develop adaptive software. The author hopes to build on this research in order to show how adopting agile methodologies and open source principles can be beneficial to the unique challenges in the development of e-learning systems. These methodologies can pose challenges for the system architecture, however, and most of the research done in these areas does not address the architectural decisions that need to be made to accommodate the open and agile approaches to development.

Research related to e-learning architecture addresses many different technical issues through software design, but sometimes architectural decisions result in designs and implementations that are incompatible with software development methodologies and philosophies employed by a project. By investigating architecture with regard to compatibility with appropriate development methodologies, this paper provides useful insight for developers and architects of e-learning systems.

Limitations

This research does not provide a design for the architecture of an e-learning system, but instead provides information that can be beneficial for making architectural decisions in an e-learning environment. A software developer can gain insight from the technologies, models, patterns and frameworks presented in
this research and use this information to make appropriate decisions that can overcome some of the architectural challenges faced when developing e-learning systems.

**Organization of the Essay**

The essay is organized into five chapters, which examine e-learning systems and software development methodologies and then investigate architectures to find and propose solutions for developing e-learning systems.

Chapter I – Introduction. This chapter is an introduction to the essay, an overview of the topics to be explored, and a general description of the purpose of the research and manner of conducting it.

Chapter II – Review of Related Literature. Introduces the reader to literature related to e-learning systems pedagogy and technology, presented as background for a discussion of the challenges of developing e-learning systems architecture that are to be addressed by the research. Lightweight software development methodologies as well as open source principles are presented and arguments are made to show their suitability for e-learning systems development. E-learning systems architecture is discussed and research in the area is reviewed that describes challenges that architectures will need to address now and in the future.

Chapter III – Methodology. This chapter examines the challenges that software architectures can present to open and agile development processes and examines desirable features that an open agile e-learning system architecture should possess in order to overcome these challenges. Design principles and
architectural frameworks are reviewed and their characteristics are evaluated for their effectiveness and compatibility within agile and open development processes. This chapter offers insight into how architectural decisions can be made for e-learning systems that can support these processes.

Chapter IV – Architecture Design and Strategies. This chapter proposes specific technologies, frameworks and patterns and presents architectural models that can be used to develop architecture with the features described in the previous chapters. The emergence of lightweight container architecture is investigated for its suitability in this regard, as well as associated design patterns and frameworks that can support an evolving, lightweight adaptable e-learning system.

Chapter V – Conclusions and Recommendations. The concluding chapter evaluates the models and comments on the feasibility of the results as implementable e-learning systems architecture. The chapter concludes with a discussion on the possibility of implementing the models with agile and open development principles, and the potential consequences this would have on e-learning systems development.
CHAPTER II

REVIEW OF RELATED LITERATURE

This chapter reviews the literature associated with (a) e-learning systems development, (b) software development methodology, and (c) e-learning systems architecture. The literature will demonstrate the importance that software development methodologies and the associated principles have on e-learning systems development and that software architecture can impact the effectiveness of these methodologies. The review begins by describing the evolution of e-learning systems technology and then explores the challenges that the advances in pedagogical approaches pose for systems architects, developers and integrators. Secondly, software development methodologies are explored; specifically agile and open source principles will be reviewed in an attempt to determine how they can be utilized to address some of the challenges in e-learning systems development. Thirdly, technical literature related to e-learning systems architecture will be reviewed and future challenges are discussed, which leads to an introduction to agile and open architectures. Lastly, the chapter summarizes the issues and introduces the characteristics that a potential architecture for agile, open e-learning systems development must possess.

E-learning Systems

In this section the reader is introduced to e-learning systems and the unique challenges associated with the development and architecture of these systems. This includes an examination of current and future e-learning technologies, a
discussion of education and the changing pedagogical practices within e-
learning, and a look at the emergence of the integration of e-learning in
enterprise systems. The section concludes by examining e-learning systems
development processes and discusses the impact that technology, pedagogy and
integration have on the e-learning systems development processes.

Overview. E-learning systems have grown out of the necessity to merge the
pedagogical world with the expanding world of technology. Computer technology
has been used for educational purposes for many years, starting with what was
known as computer assisted learning (CAL), which could describe any software
tool suitable for any number of learning activities, often by distributing learning
materials on discs and then later on CD-ROM (Attwell, 2004a). The arrival of the
web and widespread availability of home computers in the 1990s allowed e-
learning to become part of mainstream education. Almost ten years ago, Robson
defined web-based course support systems, and his definition is surprisingly
accurate for what is commonly termed e-learning today: “comprehensive software
packages that support WWW-based courses, meaning courses that depend on
the WWW for some combination of delivery, testing, simulation, discussion, or
other significant aspect” (1999, p. 271). In his overview, Robson asserted that
course-support systems got started shortly after the “birth of the WWW” in 1993.
E-learning systems have since evolved to include large integrated learning,
content management and delivery systems that can incorporate powerful
technologies. According to one article, “active e-learning employs a broad range
of Internet technologies such as personalization, simulation and mobility to
achieve pedagogic scenarios otherwise inaccessible to traditional forms of
learning” (Dagger, O'Connor, Lawless, Walsh & Wade, 2007, p. 28). Today’s e-learning platform is commonly called a learning management systems (LMS), which provides tools that support the creation, maintenance, delivery and management of learning materials.

**Technology.** This section first outlines current technologies employed by e-learning systems and provides an overview of the standards that have emerged for e-learning systems development. Some possible future technologies are briefly considered and then the challenges that are faced by e-learning systems developers to adapt to the changing technology and standards are reviewed.

The main technological emphasis has been on providing tools for educators to transfer content to the web, such as educational services that revolve around course management and scheduling, and finer grained activities such as quizzes and simulations (Dagger et al., 2007). There are also common services that are typical of any web server system, such as authentication, file sharing, logging and content management. Interactivity has been secondary, but typically there have also been features within these systems for chat rooms and forums for collaboration and communication (Attwell, 2004a).

There are three categories of these systems: (a) proprietary commercially licensed software such as WebCT/Blackboard, GradePoint, and Desire2Learn (b) open source projects such as Moodle, Sakai and ATutor; and c) in-house systems that are developed by an organization such as a university, college or school division for their own internal use, which may or may not be based on the open source software (Caplan, 2004; Dagger et al., 2007). There is a disparate range of technologies employed by these projects, including programming languages,
frameworks, libraries and databases. A brief listing of some of the architectural technology used by some of the e-learning systems is attached as Appendix A, and will be discussed in more detail in a later section.

The ad-hoc nature of e-learning systems development initially made it difficult to integrate systems and support portability and exchange of content, which prompted the development of technical standards for e-learning systems to help address these shortcomings (Friesen, 2004a). Friesen provides an overview of the three major organizations that contribute to the development of e-learning standards: The IMS Global Learning Consortium, the IEEE LTSC (Institute of Electrical and Electronics Engineers, Inc., Learning Technology Standards Committee), and the ISO/IEC (International Standards Organization/International Electrotechnical Commission). He stated that “in the context of e-learning technology, standards are generally developed for use in systems design and implementation for the purposes of ensuring interoperability, portability and reusability” (¶2). Specifications such as IMS Content Packaging and standards such as IEEE Learning Object Metadata (LOM) have been collected by Advanced Distributed Learning (ADL) into a Shareable Content Object Reference Model (SCORM), which supports the ability for systems to share course content objects (Dagger et al., 2007). These standards as well as others are discussed in subsequent sections, and a partial listing of important e-learning specifications and standards is attached as Appendix B.

E-learning systems development must strive to implement the standards as they are specified. Apostolakis & Varlamis (2006) suggested that “a major complaint about e-learning standards is that products claiming conformance do
not work together without further tweaking”, continuing on to argue: “it is necessary that e-learning standards must be adopted by all e-learning systems developers” (p. 73). This is very challenging for developers because the standards do not result in technical stability and are in fact another element of change that to which systems must adapt. Friesen (2004a) stated the following:

Perhaps especially in an emerging domain such as e-learning, standards development involves the difficult task of hitting a moving target from a position that is itself changing. The ‘target’ is represented by emerging and accepted practices, capabilities and requirements in the domain, and the ‘changing position,’ by the standards development dynamic itself. (¶ 25)

These standards will continue to evolve and change as e-learning continues into the next generation of interoperability, which includes not only content but needs to offer exchange of tools, functionalities, semantics to allow users to build custom e-learning platforms from a vast range of services (Dagger et al., 2007). In addition to supporting the changing standards, adapting e-learning systems to accommodate new technology such as mobile devices is another challenge for systems architects. Ally, Lin, McGreal & Woo noted that “the design for mobile devices has to be flexible to allow electronic learning [E-learning] materials to be delivered in heterogeneous computing platforms. Because of these heterogeneous platforms, some course materials may not be in a format that mobile devices would accept” (2005, p. 1).

Pedagogy. As e-learning technology and standards continue to advance, the pedagogical approaches for delivering education using the technology must advance as well. “We are now entering an era where streaming video, video and audio conferencing, and virtual worlds are readily available for educational use.
Thus, online learning theory needs to help educators to decide which of the many technological options is best suited for their application” (Anderson, 2004, p. 41). However, these approaches tend to be dominated by previous learning practices that simply take the new technology and use it as an extension of what was done previously. For example, distance learning institutions use the Internet and learning management systems as a new means of distributing materials which had previously been mailed in print form (Attwell, 2004a). Mainstream Learning Management Systems offer little in terms of interactivity, with the main emphasis on providing tools for lecturers to transfer their notes and assignments to the web. Managers or administrators make use of the technology monitor or control staff and to reduce costs, and even to try to replace the need for a teacher. While the technology gains wider acceptance, e-learning pedagogy isn't always being effectively implemented, explained by Stephen Downes (2003, as cited in Attwell, 2004b), who described the state of e-learning as the “educational equivalent of dictatorship. . . a manufactured environment where every movement, every idea, is carefully guided and nurtured” (p. 4).

E-learning software and standards seem to be consumed by solving the technical problems and pay little or no attention to the issues of pedagogy and how teaching methods and expectations impact the implementation of the e-learning systems. Attwell found that many technology developers “claim that their e-learning applications are pedagogy free or pedagogy neutral” (2004a, p. 1). An example of this is SCORM, a collection of standards and specifications mentioned previously, which calls itself “a pedagogically neutral means for designers and implementers of instruction to aggregate learning resources for the
purpose of delivering a desired learning experience” (Friesen, 2004b, Section Objection 2, ¶ 4). Dan Rehak, one of the chief architects of SCORM, found that the framework has a limited pedagogical model and is not well suited for some learning environments (Rehak, 2002, as cited in Friesen, 2004b).

While there may be nothing specific in the current standards and technology that limit the pedagogical approaches, the neutrality of the technology does inhibit the deployment of particular pedagogical approaches to learning. Friesen (2004b) suggested “specifications and applications that are truly pedagogically neutral cannot also be pedagogically relevant” (Section Objection 2, ¶ 7). Olivier and Liber have expressed the concern that: “eLearning standards will constrain Internet supported learning by freezing a sub-set of existing practices” (2003, as cited in Attwell, 2004a, p. 5). They also question “whether [standards] can be provided that can support the development of new, enhanced, but yet to be developed approaches to learning which the Internet makes possible” (p. 5). New pedagogical practices will evolve as the technology grows; developers need to pay more attention to the issue of pedagogies for e-learning and develop the e-learning technology in support of the pedagogical practices as they emerge to take advantage of new technology.

There is a growing need for software developers and educators to focus e-learning development to work more effectively with the pedagogical developments that can best utilize new technologies. The development of complex learning tools and modules that adhere to modern pedagogical standards needs learning systems architectures that embrace and promote these practices, rather than attempting to keep the technology free from particular
approaches to learning. The technology simply cannot be separated from the pedagogy, and it can be argued that attempts to do so are counter productive to the development of e-learning systems, and efforts should be made by software developers to embrace and enhance the continuous changing pedagogical approaches rather than inhibiting them by developing pedagogically agnostic e-learning systems.

As Internet technology grows, so too does the potential for e-learning systems to deliver unique and dynamic learning experiences to a diverse population of students; however, there has been a disconnect between this potential and the reality of the learning experience being delivered by these systems. Robson (1999) recognized this when he wrote “first generation course-support systems have dressed up old pedagogy in new technology” (p. 280). This concern prevails in current e-learning implementations that are seen by many as an extension to the distance learning paradigm and are used as an enhanced means of distributing materials and assignments to students, and fail to evolve the pedagogical practices to take more acute advantage of new technology (Attwell, 2004a). Robson argued that there needs to be awareness by software developers and it is necessary to focus on the needs of the learners and educators from the pedagogical perspective when designing e-learning systems, demanding that “they should be built so that they are flexible enough to allow teachers to experiment with new ideas” (Robson, 1999, p. 280). Attwell echoed his concern, writing: “Many educationalists have had great pedagogic ideas for the use of e-learning, only to be frustrated by the limitations of the technology applications” (2004a, p. 4). While LMSs have created an environment in which
teachers and learners can create and update their course content without the aid of programmers or designers, this leads to “potential for error and less-than-professional-standard output” (Anderson, 2004, p. 42).

The growing emphasis on lifelong learning is driving awareness of the importance of the different types of knowledge where e-learning systems need to become part of the action and it is necessary to refocus on the needs of the learner. Anderson (2004) wrote “the challenge for teachers and course developers working in an online learning context is to construct a learning environment that is simultaneously learning centered, content centered, community centered, and assessment centered” (p. 54). They went on to say that it is teachers that must learn develop their skills to produce the online activities that are most supportive of how people learn. However, there is also a growing need for e-learning software developers to be aware of and embrace the importance of different types of knowledge and to develop software that more easily supports these wider forms and contexts of learning. “Together creators, innovators and technical developers can begin to shape the applications we need for learning and knowledge development to occur” (Attwell, 2004b, p. 7).

There are many challenges for software developers to develop pedagogically supportive e-learning systems, as there is often a gap between the educators and their desires and the technologists’ domain knowledge that prevents the full pedagogical potential of new technology from being realized. Many educationalists have had great pedagogic ideas for the use of e-learning, only to be frustrated by the limitations of the e-learning systems because software developers are often content with simply making new technology
features available, but leaving it up to the educators to successfully implement their pedagogical practices using the new technology (Attwell, 2004a).

Pedagogies are evolving to use e-learning to explore, construct and develop knowledge and while these pedagogical ideas are not new, the e-learning technology that can support some of these ideas is just beginning to take form.

As an example, teachers are moving to a constructivist environment, which lends itself to the creation of new understandings and new ways of problem solving. A modern constructivist e-learning environment requires technology that can allow teachers to engage students in meaningful interactions. Felix (2005) wrote that: “emphasis is placed on students acquiring meta-skills and knowledge, shaping relevant, negotiated curricula, being involved in collaborative, lifelong, global learning with the help of real-life tutors and informants, a social constructivist approach presents itself as the natural pedagogical paradigm” (p. 96). The activities and lessons should promote exploration, experimentation, construction, collaboration and reflection, but these activities can be difficult within their current e-learning systems, which often use hypertext based environments. While hyperlinking allows students to create their own learning paths through content, which is congruent with constructivist learning theory (Anderson, 2004), oftentimes it is difficult for teachers to actively promote constructivist pedagogy because they lack the technical expertise to create the content effectively within the LMS.

E-learning systems developers need to actively engage the educators in order to allow the pedagogy to guide the requirements of the systems and to enable the educators to provide an e-learning environment for their students that
suits their pedagogical needs without the technical barriers. Attwell (2004b) suggested, “The technologist’s job in assisting ‘constructivist pedagogy’ is not to build ‘constructivist systems’ but to make it easier for innovators to make effective use of technologies” (p. 5). Ismail (2001) noted “The problem is, in many cases the development of e-learning projects devolved into a purely technical process, resulting in expensive software implementations, essentially unused by uninformed, fearful, or resentful employees” (p. 331). Attwell (2004a) went on to state “I do see the dawning of a new movement which focuses on the learner and on learning and in which e-learning can open opportunities for a new and rich pedagogy” (p. 5). This is the challenge faced by software developers; to provide e-learning systems that can adapt to the changing pedagogical environment and provide meaningful and useful tools to educators and users as the technology to implement new features is also evolving.

Integration. With the tremendous potential of e-learning systems, the trends in education are shifting towards a reliance and dependability upon central, integrated and robustly designed e-learning environments. Smith (2005) stated that the changes in e-learning “have led to the ascendance of distance education methods through the evolution of all traditional universities into dual mode institutions, offering e-learning courses, and supporting both on-campus and off-campus students through Internet-based delivery systems” (p. 3). There is a growing importance of the e-learning system as Davis (2004) reported, “online learning is now becoming ubiquitous at all levels of education, in all institutions of learning” (p. 97). The author also noted that “understanding how the entire system of course development and delivery occurs and how these systems link to
services and other components are vital aspects of ensuring effectiveness and quality” (pp. 97-98). Integration of the e-learning environment with the enterprise infrastructure of the organization is needed for efficient organizational information systems. Smith wrote that organizations are: “fundamentally rethinking and rewiring structure and infrastructure to become a more accessible university in a rapidly emerging e-world” (2005, p. 3).

Ideally, an e-learning system would be built from inception as an integrated component of the organization's infrastructure. However, many e-learning systems begin their existence as bottom-up uncoordinated experimentation of an individual educator or a small group of educators and technologists that result in systems that need to be integrated after they are in full productive use. As the e-learning system grows it needs to be integrated with other courseware development systems, library systems, course calendars, student information systems (SIS), and other student assessment or marks systems. Students will often access the LMS, as well as other related web-accessible components of an organization's system through a user friendly portal that has a single student login (Davis, 2004).

The e-learning systems are often separated functionally with little technical integration, requiring a great deal of administrative coordination and communication to integrate the systems. The online learning staff and systems need a lot of support and maintenance from the administrative computing unit, which requires clear statements of roles and responsibilities, processes, and policies to be established to manage the integration between the systems (Davis, 2004). To provide technical integration solutions requires the continued
communication amongst the departments. According to one e-learning integration project, “Almost all the different aspects of the e-University Project have involved key players from the Distance and e-Learning Centre, Information Technology Services, the Library, and Faculties, together with other stakeholders from areas such as Student Administration, the International Office, Student Support Services and Financial Services” (Smith, 2005, p. 11).

The amount of effort that it takes to integrate the e-learning and administrative systems is largely dependent upon the architecture of the e-learning system. For example, consider the task of ensuring that the LMS be linked to the SIS so that the right student information is easily available within the LMS. Davis (2004) wrote “this requires clever and robust programming in the LMS, a server to authenticate student log-ins and ensure a secure interface with the SIS, and some appropriate programming in the SIS” (p. 107). The e-learning systems architecture must provide flexibility and openness to achieve this type of integration.

E-learning systems with closed, proprietary architectures must inevitably be integrated using the vendor’s documented and supported methods, which are often inelegant solutions that place a heavy dependency on the vendor to provide a solution that will work in the organization's infrastructure. The system infrastructure becomes dependent on the vendor’s solution, which makes it fragile when the systems need to be changed or upgraded. Even proprietary systems that conform to interoperability standards can be difficult, as those standards exist to ensure interoperability between LMSs, but do little to promote interoperability between the LMS and other systems such as an SIS. Open
source e-learning systems and systems developed in-house that use open
source frameworks can offer much more flexibility when it comes to providing
integrated functionality. As a developer, the author finds it much easier to provide
integrated solutions when there is access to the source code. Implementing
flexible solutions is made easier with the ability to work with and understand the
code bases rather than dealing with rigid proprietary methods of integration.

E-learning Systems Development. The previous sections have shown that the
development of e-learning systems requires attention be paid to the
technology, pedagogy and integration and should strive to produce systems that
can address the challenges within these areas. The remainder of this chapter
examines software development methodologies and principles that can support
adaptable e-learning systems development and then examines e-learning
software architecture and discusses its ability to accommodate these processes.

Software Development Methodologies and Principles

Developing software systems requires the management of all the processes
that are needed to gather requirements, including new ongoing and changing
requirements, and assuring that the results of the software conform to the
requirements. This is commonly called the software development process, or
software development life cycle. A project's success is highly dependent on the
management and implementation of these processes, and the methodology
needs to be established to provide guidelines and discipline to the development
process that can accurately identify and control requirements, ensure good
communication, timeliness and quality.
There are various ways that the activities can be structured and organized within a project, and selecting the appropriate methodology to define and perform these activities in a way that suits a particular software development endeavour can help with both productivity and quality of the system. The chosen methodology helps define the structure and organization of the activities and steps that need to be taken for the development of a software system, which includes activities from disciplines such as analysis and design modeling, implementation, testing, deployment, configuration management, project management, and environment setup.

There are several methodologies for these processes, each with advantages and disadvantages, depending on the type of software being developed. The most well-known and oldest of these methodologies is the Waterfall model, which was first proposed by Royce in 1970, which described the following seven phases of software development in order: requirements specification, design, construction, integration, testing and debugging, installation and maintenance (Royce, 1970). Since then, many people have written of the difficulties a sequential process presents. For example, Parnas & Clements (1986) wrote: “Many of the details only become known to us as we progress in the implementation. Some of the things that we learn invalidate our design and we must backtrack” (p. 251). These types of criticisms led to processes that are much more iterative in nature, starting with initially small versions of a system and grow the system through iterations as the requirements become more well known.

Many different methodologies have emerged that are iterative in nature, but
some prescribe principles and practices that are better suited than others for e-learning development. Choosing the appropriate software development methodology can be beneficial to developing e-learning systems by providing principles for developing systems that can adapt to new technology, meet the pedagogical needs of the educators, and enable flexible and efficient integration with technical infrastructure of an organization. Methodologies should be chosen that allow the processes to be managed in a manner that will help to address these challenges with principles and practices that are beneficial to e-learning.

Software development methodologies will also impact the structure, design, technology and implementation of the software architecture, which is why choosing the best software development methodologies to employ for developing e-learning systems is an important step to make before the architectural decisions can be made. The next section reviews the technological, pedagogical and integration challenges faced in e-learning systems development, and describes some principles and practices of software development methodology that could be sought to help address these problems.

**E-learning Systems.** Previous sections have outlined some challenges associated with e-learning systems development that the software development methodologies need to be able to address. There should be a focus by e-learning systems developers to work more closely with the educators on the development of rich pedagogical applications that give them the ability and freedom to implement their learning ideas, innovate and facilitate the learning based on the needs of the students. This is difficult and requires collaboration of knowledge between pedagogical experts and software developers:
It is too much to hope for the emergence of a new multi-skilled occupational profile encompassing both pedagogy and software development, although as e-learning becomes a more specialist field there are some signs that skills and knowledge are crossing occupational divides. More realistic and attainable is to strive for new development processes including, for example, co-design workshops and rapid application development allowing iterative processes of design and development. (Attwell, 2005, p. 358)

There is a complex interaction between the teachers and learners that must be observed and analyzed by technology developers with the aim to help teachers and learners develop a deeper understanding of how they use technology and ultimately make the pedagogy more effective. Caplan (2004) wrote: “it is not reasonable to believe that a high-calibre online course of instruction can be created by just one or two people. Quality courseware production requires a highly organized, concerted effort from many players” (p. 186). Course development teams on e-learning systems should be comprised of stakeholders from many different key roles such as authors, graphic designers, web developers, programmers, and instructional designers (Caplan, 2004). This approach should not be limited to the course development; the development of the entire e-learning system can benefit from close consultation between the developers of the software and the teachers, learners and other stakeholders. Ismail (2001) found that achieving this environment of collaboration is a difficult task and requires organizational planning and strategies for managing the interactions for e-learning that can be built into the development processes. He suggested an e-learning systems development framework where:

Designers should seek to understand the basic components of what constitutes an e-learning “ecosystem.” This systems framework is crucial in guiding the decisions relating to the choice and development
of each component in relation to the objectives outlined in the organizational e-learning strategy. The framework will specify a learning systems architecture for pedagogical development and systems integration. (Ismail, 2001, p. 331)

In addition to supporting collaboration, the software development methodology and principles for an e-learning project should allow for flexible systems that enable organizations to easily integrate e-learning into their system infrastructure and can also be adapted to accommodate changing technologies and standards. Davis (2004) stated: “curricula, online learning technologies and approaches evolve all the time, and therefore any real system must also be able to change constantly” (p. 98). The systems need to implement existing and emerging standards and be readily integrated with organizations’ systems infrastructure, which can be very difficult due to problems such as “unworkable policies and practices that you never knew existed, inadequate governance processes, administrative systems that might or might not be made to work with the new systems” (Davis, 2004, p. 98).

The author has examined and experienced many software development processes that employ different methodologies and principles, and will present two categories of principles that can be of benefit to e-learning in this regard: (a) agile software development methodology and (b) free/open source software.

Agile Software Development. Agile software development methodologies place a large emphasis on adapting to changing requirements. Agile software development strives to achieve the ability to adapt to change, and activities in the process are positioned with this in mind. The process expects change, and embraces these changes with principles that allow systems to become adaptable
(Martin, 2003). Agile processes focus on collaboration, soliciting continuous feedback from users and other stakeholders, and continually refining requirements by developing quick releases in response to changing requirements. One of the key advantages of adopting an agile process methodology is that the process places a large emphasis on adapting to unpredictable or rapidly changing requirements, striving to achieve the ability to adapt quickly to change.

In 2001 a group of independent software developers, consultants and industry experts met to outline principles and values that would strive to allow software teams to develop quickly and respond to change (Martin, 2003). This group formed the Agile Alliance and created a statement of values, which they called the Manifesto for Agile Software Development (Beck et al., 2001). The manifesto contains four basic values for agile processes to follow: “Individuals and interactions over processes and tools. Working software over comprehensive documentation. Customer collaboration over contract negotiation. Responding to change over following a plan” (p. 1). The full manifesto, as well as an accompanying set of principles is attached as Appendix C. The four statements in the manifesto provide a high level view of the principles of agile software development and can serve as valuable guides for development of adaptable e-learning systems.

*Individuals and interactions over processes and tools.* Martin (2003) argued that people are the most important ingredient of success. E-learning systems projects need to be built around motivated individuals who are given the environment and support they need, and then trusted them to get the job done.
The team is considered to be more important than the environment and it is advised to start small when it comes to choosing technologies upon which to build a system. In addition, often big, expensive development tools and architectural frameworks don't automatically help you do better; often they hinder more than they help (Martin, 2003).

*Working software over comprehensive documentation.* An e-learning system that uses working software as the primary measure of progress and delivers working software frequently, in short iterations that incrementally add new features allows the developers to satisfy the educators through early and continuous delivery of evaluatable features. Continously delivering working software for evaluation gains satisfaction from the educators that their needs are being responded to in a timely manner and can foster a productive relationship between developers and educators.

*Customer collaboration over contract negotiation.* The educators and developers must work together daily throughout the project because the most efficient and effective method of conveying information to and within a development team is face-to-face conversation (Martin, 2003). Frequent, regular feedback on the features of the working software as it is produced allows the developers to more easily prioritize needs of the educators while also allowing the educators to develop new understandings into what the software is capable of and provide meaningful feedback for enhancing the software.

*Responding to change over following a plan.* It is the ability to respond to change that is the author's primary goal in producing an adaptable e-learning systems architecture. The fundamental principles of agile processes are to
welcome changing requirements, even late in development and to harness change for the customer’s competitive advantage (Martin, 2003). The goal would be to create an agile process that allows educators to steer the pedagogical requirements as the system is developed. As e-learning technology advances, pedagogical practices can be created around the technology and then welcomed into the system by working with the educators to implement and enhance the pedagogy on an ongoing basis.

Developing an e-learning system that follows an agile process could help the system to respond to the pedagogical and technical needs of the teachers and learners by helping to bridge the gap between the developers and domain experts. An agile process would insist that developers work with the experts in the education domain to establish the pedagogical requirements, solicit continuous feedback as the system is developed incrementally and ensure that the tools are being developed such that they are productively contributing to the learning process. Educators would be able to identify the learning perspectives that they prefer and make informed decisions about which approaches the software should take to implement pedagogical practices by testing them out on the working systems as the features are implemented. Software developers would benefit because educators will be able to articulate more clearly what they need from an e-learning system if they can evaluate working software as it evolves. They will develop new ways to think about their learning activities if they can use and try the software and give direct feedback on the features as they are being developed.

Implementation of an agile process presents many challenges, including the
impact on the technological architectural decisions that need to be made that
would support such a process. An agile architecture requires continuous attention
and must enable designs to emerge that adapt to changing technology and
requirements.

**Free/Open Source Software.** This section discusses free/open source
software (FOSS), which has principles and practices that strive to achieve some
of the same goals as agile methodologies (Theunissen, Boake & Kourie, 2005),
as well as an open and free philosophy that promotes sharing of knowledge that
can beneficial to e-learning development.

According to the Open Source Initiative, a software license can be termed
open source if it: allows free redistribution without the need of an additional
license, has easy access to the source code, allows modifications and derived
works, allows distribution of modified software, has no discrimination against
persons, groups or fields of endeavour, is not specific to a product, does not
restrict other software and must be technology-neutral (Coar, 2006). Other terms
for FOSS include open source software (OSS) and free/libre and open source
software (FLOSS), which uses the French *libre* to emphasize the freedom of
open source to counter the problem that the free in FOSS is often misunderstood
as “no cost” (Wheeler, 2007).

It is the freedom philosophy of FOSS that is important in e-learning
systems, where there can be emphasis on academic freedom and the
collaborative nature of knowledge creation (Faber, 2002). Attwell (2004a) wrote,
“OSS has also contributed to a social recognition of the potential for sharing and
co-development of learning applications” (p. 3). The philosophy of FOSS mirrors
the collaborative open movement in academia, which Attwell (2004a) said is an encouraging social development: “the growing movement or open contents, the idea that resources can and should be shared. MIT gave the movement a great boost when they announced their Open Courseware initiative” (p. 4). Attwell (2005) expanded on this a subsequent article: “Open content parallels the ideas of open source in developing new license models for products and promoting the sharing and co-development of learning content” (p. 353). He lists many more reasons why FOSS has relevance to the education sector, citing such benefits as more flexibility for pedagogical approaches, supporting the exchange of ideas, and utilizing in-house human resources and learners that can become involved in the FOSS community to build upon the systems, and then concluded:

OSS has also contributed to a social recognition of the potential for sharing and co-development of learning applications. At the same time the emphasis on life long learning is driving awareness of the importance of different types of knowledge and of developing software to support wider forms and contexts of learning. (p. 358)

The openness of FOSS provides great benefits to e-learning systems development that desire flexibility and maintainability (Grob, Bensberg & Dewanto, 2004), which can make integration with organizational infrastructure easier from a technical perspective. Open source systems make it easier for programmers to integrate with open source tools of their own, which can be an advantage as Caplan described:

One clear advantage of code-based programming is that these tools are often open source; that is, they are created from freely available, stable code that encourages collaborative development. Commercial GUI software often requires less technical expertise to use than code programming, but it can be expensive, and the companies who publish these proprietary software programs update them often, rendering
earlier versions obsolete and constantly forcing developers who rely on them to purchase new versions. (2004, p. 191)

Making the decision to follow principles of FOSS has an immediate impact on the implementation of an e-learning architecture. An open e-learning system architecture would need to use tools, platforms and frameworks that conform to the principles, practices and licenses supported by the open source initiative.

**E-learning Systems Architecture**

This section reviews the current state of e-learning systems architecture, provides an overview of the general characteristics and then explores the evolution of architectures from monolithic designs to more modular systems. This is followed by a suggestion that e-learning systems architecture needs to rise to the demand for more flexibility and interoperability which leads to a discussion of the challenges and incompatibilities that architecture presents for agile development and open source principles.

As mentioned previously, e-learning platforms can be grouped into three categories: (a) proprietary solutions, (b) open source and (c) in-house solutions, which may or may not be also based on the open source software. Appendix A provides a partial listing of proprietary and open source LMSs and shows some of the technology used in their architectures. The architectural goals are generally purposed to provide specific LMS functionality for content creation, maintenance and delivery and student enrolment and management.

These systems usually started out using proprietary formats to manage the courses and focused on the delivery of content designed for a specific purpose. Initially, architectures were monolithic, had little support of interoperation, and did
not adapt easily as new Internet technology and demand for new pedagogical practices emerged due to their monolithic designs (Dagger et al., 2007). Ismail (2001) found that:

Products in this category do not address the need to develop and manage increasing volumes of content in smaller chunks by a larger group of content providers. Nor do they provide adequate mechanisms for maintaining consistent instructional presentation or adapting that content to the needs of learners. (p. 335)

The continued demand for personalized interoperable e-learning platforms has forced these systems to evolve to a more modular architecture (Dagger et al., 2007), and as designs have become more modularized and some of the standards have matured, it has become more feasible for platforms to integrate new functionality in their systems that promotes interoperability.

Proprietary Systems. In the proprietary sector there are emerging initiatives that provide software developers with hooks to tie third party software with the LMS (Dagger et al., 2007). As an example, Etesse (2004) explained Blackboard's Building Block architecture, which implements IMS specifications and “is designed to provide users the extensibility needed to quickly implement support for new and changing interoperability specifications” (p. 5). He goes on to conclude that “through Blackboard Building Blocks, Blackboard provides a flexible architecture that can take advantage of standards based-interoperability” (p. 8). Integration is supported through the IMS Enterprise Specification, which “allows for enterprise systems on campus to seamlessly interact with each other, for example allowing a SIS to send enrollment information to the Blackboard Learning System” (p. 6).
Integration Standards. The standards and specifications referenced in earlier sections of this document such as SCORM, IMS Content and Packaging and IMS Learning Design are designed to support sharing of learning content across different LMSs. The IMS Enterprise Specification, together with other standards and specifications listed in Appendix B, have emerged to help with the interoperation of functionality between the e-learning systems and the integration with other enterprise systems. These standards are far more complex as the goals of interoperation and integration requires a much wider range of information, context, control and semantics to be exchanged between systems (Dagger et al., 2007). The challenges that have already been discussed in this paper with the content sharing standards are even more daunting for the software architecture with the complexity and immaturity of these emerging interoperability and integration standards. It is not very feasible for an e-learning system to keep up with all the standards as they are developed, as evidenced by the fact that none of the vendors claim to do so. However, the architecture must be flexible enough to be able to quickly add standards support as it is needed, and be adaptable enough to provide solutions independent of the standards where necessary. Limiting the e-learning platform to the rigid boundaries defined by specific standards or frameworks is not an ideal option.

FOSS Systems. With projects such as Sakai, ATutor and Moodle leading the way, FOSS systems are also embracing and supporting standards to support interoperability, but they also offer a variety of other flexible features for enhancing and integrating the systems. The open source platforms are built on extensible frameworks that let implementors adjust and modify systems to suit
their needs more easily than the solutions provided by the proprietary vendors (Dagger et al., 2007). Some of the modular solutions to promote interoperability are similar in many ways in both the proprietary and FOSS systems, but having the source code available with the freedom inspect and extend it under open source license emphasizes the flexibility of FOSS and allows interoperability by eliminating complete reliance on the vendor supplied integration methods.

**Flexibility and Interoperability.** The evolution of e-learning has continued to demand more flexibility and interoperability as academic communities increase their expectations in this regard, placing more pressure on e-learning platforms to deliver these requirements. Attwell (2004a) predicted, “instead of having to install an LMS in the future educationalists will be able to select relatively thin learning applications which will do what they want and with the pedagogic application they wish to deploy” (p. 4). New guidelines and specifications are emerging to promote more service oriented architectures that will allow more interoperability and migrate organizations from single vendor solutions to platforms that allow selection of different services from various systems to meet their e-learning requirements.

Frameworks such as the E-Learning Framework (ELF), the IMS Abstract Framework, and the Open Knowledge Initiative (OKI) have defined some steps toward service-oriented e-learning platforms that can achieve this vision. Sakai is an example of an architecture that is based on OKI, using this service-oriented framework to “provide an integration layer that ensures portability of the tools and services across any environment” (Counterman et al., 2004, p. 3). Dagger et al. (2007) explain how these frameworks and platforms have emerged and how the
service-oriented approach might lead to integrating adaptive systems to generate a shared semantic view to include a wide variety of extra information about a user from a variety of different systems. Hussein & Khan (2005) showed how intelligent agents might be implemented in a service-oriented architecture that would “greatly increase interoperability, scalability, maintainability, flexibility, customization and interactivity in a heterogeneous e-learning environment” (p. 142). Xiaofei Liu, El Saddik & Georganas (2003) proposed a “multi-tiered component-oriented system architecture of LCMS to illustrate how to integrate Web Services into a J2EE platform” that would serve to “provide interoperability between systems on different platforms implemented by different technologies” (p. 720). This is far from an exhaustive list of the research in this area, but it serves as a sample of the direction that e-learning is going and the desire for e-learning to achieve higher levels of interoperation and flexibility.

The common theme throughout these proposed architectures is that implementing them is challenging because of new emerging standards and technology that is constantly undergoing changes and developments. These are all viable e-learning architectures that look to promote interoperability and flexibility, but the field of e-learning is still quite new and there is no certainty as to which architectural approaches will be the best fit for a given organization's e-learning needs. There are many decisions that need to be made for e-learning architecture, whether you are developing a system from scratch, or you are selecting a system that will need to be integrated with organizational infrastructure and be adapted to meet future pedagogical and technological needs.
Agile and Open Source Architecture. A software architecture dictates many aspects of a software design, some of which is specifically related to the purpose of the application, but more importantly it defines the technological frameworks and design patterns upon which the software is built that determines the structure of the system and determines how components will interact. The frameworks, patterns and component interaction architecture must feature characteristics that support the software development methodology and principles with which the software is being developed. Many architectural implementations can be unwieldy, complicated, and difficult to adapt to change, which makes implementation of agile development practices very difficult. Similarly, many architectures are built upon proprietary frameworks and platforms or require the use of proprietary tools and environments that are against the principles open source philosophies. An open, agile architecture must examine these principles from an architectural perspective to determine what fundamental characteristics of an architecture would best support the processes that follow these practices and principles.

Summary

In summary, this chapter reviewed the technical, pedagogical and integration challenges that e-learning systems developers are faced with, and explained the importance of software development methodology in developing e-learning systems to meet these challenges. Arguments were presented that showed the importance for e-learning systems to be adaptable to technological and pedagogical changes while providing flexibility and extensibility for
integration. Agile and open development principles were reviewed which could allow e-learning systems development to embrace the ever changing and improving pedagogical practices, as well as provide the needed flexibility for integration. The last section of this chapter reviewed e-learning architecture and provided some samples of the direction that e-learning architecture is heading in the future.

The literature reviewed in this chapter outlines the criteria that are used when evaluating architectural decisions in the remainder of this paper. The purpose of this paper is to promote open and agile methodologies as the primary solution for providing adaptability, flexibility and interoperability, and to evaluate and discuss the challenges that the open and agile principles might pose when making architectural decisions. The next chapter provides the methodology used for making these architectural decisions by examining architectural designs, patterns, frameworks and technologies which (a) are able to adapt to new technology and pedagogical requirements, (b) are extensible and flexible to allow integration with organizational systems, (c) can support the methodologies and principles of agile software development, and (d) concur with the philosophies of free/open source software principles.
CHAPTER III

METHODOLOGY

This chapter examines the challenges that a software architecture might present for open and agile development processes and explores desirable features that an open agile e-learning system architecture should possess in order to overcome these challenges. Design principles, patterns and agile practices are evaluated from an architectural perspective for their effectiveness and compatibility within agile and open development processes. This chapter offers insight into how architectural decisions can be made for e-learning systems that can support these processes.

Introduction

In Chapter 2, agile development methodology was introduced as a lightweight process that can address adaptability issues surrounding changing technology and pedagogical requirements. Additionally, the use of Free/Open Source Software (FOSS) was discussed and shown to help provide flexibility and extensibility to address integration issues in a software system. However, employing these practices can introduce additional architectural problems that must be addressed.

Agile software development principles discourage spending too much time on up front design, arguing that the future changing requirements will render much of this effort fruitless. This presents a challenge for defining architecture in an agile process, as it needs to be defined quickly and concisely, and be able to
adapt to changing requirements. Additionally, choosing FOSS over proprietary software places a constraint on the architecture by eliminating proprietary frameworks, systems and platforms as choices in the architectural process. The remainder of this chapter explores these issues and describes characteristics of software architecture that can be viewed as both agile and open.

**Agile Architecture**

An agile architecture must be able to adapt to new technology and pedagogical requirements and be supportive of the methodologies and principles of agile software development. This can be difficult, as there are many aspects of traditional software architecture and design that are simply incompatible with agile software practices. This can often lead to agile developers to simply ignore architectural issues altogether, which invariably leads to challenging problems. The remainder of this section discusses these challenges and explores agile design principles and practices that can be employed to implement agile architecture and overcome these challenges.

**Challenges.** The biggest challenge faced when trying to establish an agile architecture is that agile software development practices recommend that everything start small and evolve as needed, stressing that systems should not contain an overabundance of tools, systems or other architectural infrastructure until it is needed. For example, it is recommended that simple whiteboards and graph paper instead of expensive complex CASE tools, or flat-files should be tried instead of a database system until it is absolutely necessary to make the switch (Martin, 2003). However, employing such techniques can lead agile
developers to ignore architecture altogether, trusting that the architecture will simply emerge or evolve on its own accord, which many critics see as a glaring weakness of agile methodology (Fowler, 2004a). Fowler explains that agile methodology: “calls for the death of software design. Not just is much design activity ridiculed as 'Big Up Front Design', but such design techniques as the UML, flexible frameworks, and even patterns are deemphasized or downright ignored” (2004b, ¶ 1). This can certainly be problematic on an agile system of any significant size, and agile developers must consider architecture and make decisions that allow the design to be changeable and maintainable. Martin (2000) describes what can happen in an agile project if architecture is ignored:

> The program becomes a festering mass of code that the developers find increasingly hard to maintain. Eventually the sheer effort required to make even the simplest of changes to the application becomes so high that the engineers and front line managers cry for a redesign project. (p. 1)

These problems stem from a lack of understanding of the role of software architecture in an agile project. The development of e-learning systems cannot afford to suffer these problems; adaptability still must be planned from the outset and architectural decisions must be made that are supportive of the agile principles.

**Controversies.** Agile principles dictate that the architecture cannot be fully defined in advance, which can lead to controversy and confusion on agile projects when planning the architecture. Fowler (2004b) notes that agile methodology “challenges many of the common assumptions about software development. Of these one of the most controversial is its rejection of significant
effort in up-front design, in favor of a more evolutionary approach” (¶ 2). The software architecture should be a very important aspect of an agile development effort, but it must be approached differently than the traditional up front architectural design (Ambler, 2008a). Traditional architectural processes do not necessarily fit well with agile practices and different approaches to architecture need to be made. Fowler (2004b) explains further: “XP involves a lot of design, but does it in a different way than established software processes” (¶ 1). Lightweight agile methodologies need to be accompanied with lightweight development and design, resulting in architecture that can be readily refactored and adapted within agile processes (Tate, 2005a).

Agile processes must employ design principles, patterns and concepts that can be utilized to allow the software to be adaptable as well as provide an environment for the enabling agile practices of continuous integration, test driven development and refactoring to make evolutionary design plausible (Fowler, 2004b). The remainder of this chapter examines lightweight design principles, patterns, concepts and practices that can be employed in e-learning systems development to implement agile architecture within a lightweight process.

**Design Principles.** To develop an e-learning system while employing agile methodologies, the architectural design must be flexible to be able to evolve and adapt, as that is the key beneficial feature in choosing an agile methodology for e-learning systems. An agile architecture is not designed up front, but needs to be designed as the system progresses. This can be a dangerous proposition because if the system is progressively designed by aggregating ad-hoc design decisions, each addition to the design will make future code harder to alter. This
is a trap that many agile programs fall into, and in many ways this is not software
design at all and leads to a poor design as an end result. As the software
evolves, the design deteriorates and so does the software architecture’s ability to
effectively adapt to changes. Martin (2000) explains the deterioration of software
architecture as “symptoms of rotting design” (p. 3). He describes four symptoms
that indicate that a software design is difficult to change: (a) rigidity, the tendency
for software to be difficult to change, even in simple ways; (b) fragility, the
tendency of the software to break in many places every time it is changed; (c)
immobility, the inability to reuse software from other projects or from parts of the
same project; and (d) viscosity, when the design preserving methods are harder
to employ than the hacks and the development environment is slow and
inefficient. Most projects will attribute these symptoms to the problem of changing
requirements, but Martin argues: “If our designs are failing due to the constant
rain of changing requirements, it is our designs that are at fault. We must
somehow find a way to make our designs resilient to such changes and protect
them from rotting” (p. 4).

The symptoms of rotting design are due to issues of poor dependency
management between modules in the software architecture. The structure of
classes and packages in an agile architecture must keep the software application
flexible, robust, reusable, and developable (Martin, 2000). To achieve this, there
are principles and techniques for object oriented design such as package
coupling principles: (a) The Acyclic Dependencies Principle, the dependencies
between packages must not form cycles; (b) The Stable Dependencies Principle,
depend in the direction of stability; and (c) The Stable Abstractions Principle,
stable packages should be abstract packages (Martin, 2000). These principles can be used by the e-learning systems developers to calculate software package metrics that measure the quality of the software design as it progresses in an agile project, and are listed in Appendix D.

In addition to the package principles, there are class design principles that help manage the dependencies in an architecture, which introduce desirable traits of class inheritance. These principles of class inheritance should be followed closely in an e-learning system, which will have many hierarchical models in the education domain to categorize learning materials, students, teachers and delivery methods. The Open Closed Principle explains that a module should be open for extension but closed for modification, which simply means that modules must be written so that they can be extended without requiring them to be modified, while the Liskov Substitution Principle says that subclasses must be substitutable for their base classes (Martin, 2000).

One of the most important design principles in regards to agile architecture, is the Dependency Inversion Principle, which is also known as inversion of control. Martin explains that modules should “depend upon abstractions. Do not depend upon concretions” (2000, p. 12). He discussed this concept in more detail in a previous paper: “High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions” (1996, p. 6). He explains the importance of this principle as it relates to promoting loosely coupled adaptable architecture, and concluded:

The principle of dependency inversion is at the root of many of the
benefits claimed for object-oriented technology. Its proper application is necessary for the creation of reusable frameworks. It is also critically important for the construction of code that is resilient to change. And, since the abstractions and details are all isolated from each other, the code is much easier to maintain. (p. 11)

The package principles, software package metrics and the inversion of control principle can be used by agile architects as the foundation for an agile architecture that can avoid the symptoms of rotting design and provide a design that can evolve with a changing e-learning system environment.

**Design Patterns.** When following the design principles described above, many of the same structures re-appear from one project to the next. These repeated architectural structures are design patterns, and while design patterns are not new, they serve as very important techniques for establishing an agile architecture. The lightweight patterns introduced in this section are ones that agile developers have refined to promote loose coupling between modules and integrate services without forcing code into business logic or the domain model (Tate, 2005b), which is a desirable trait for an e-learning system design.

There are many patterns that promote design that conforms to the Dependency Inversion Principle, such as Abstract Server, and Abstract Factory, which allow module dependencies to use abstractions and isolate the concrete dependencies into one, and only one, place (Martin, 2000). The Abstract Server pattern describes how an abstract interface can be inserted into a dependency so that a module can depend upon the interface rather than the implementation. This allows different implementations of the interface to be inserted into the program with affecting the dependencies. The Abstract Factory provides a description of how such objects can be instantiated by isolating the creation of
objects, so that dependent objects that are coupled with an abstract interface do not need to perform the object instantiations. Patterns such as these have become important in designs of agile architecture to reduce coupling and dependencies, allowing more flexibility for the architecture to evolve.

The Dependency Inversion Principle should be a fundamental concern when it comes to agile design patterns. The Inversion of Control pattern achieves this by allowing for the assembly of components while separating their configuration from their use (Fowler, 2004a). While a traditional library would contain methods that are called by a program, the Inversion of Control pattern is generally implemented as a framework that will call methods within a program, thus inverting the control flow of the program. Dependency Injection is a specific style of inversion of control which uses assembler objects that inject appropriate implementations of dependencies into a program (Fowler, 2004a). The fundamental aspect of the pattern is that the program does not make any calls to create an object, but rather the dependent objects are sent to the program via setter methods, constructors or an interface method. These assemblers essentially wire the dependent components of an application together, which dramatically reduces the coupling within the application, allowing a great deal of flexibility for different implementations of components.

These design patterns, in particular Inversion of Control or Dependency Injection provide basis for an architectural foundation of an e-learning system that can be built upon with minimal dependency issues that often hamper the flexibility and adaptability of the software design.

**Lightweight Containers.** The Inversion of Control and Dependency Injection
patterns have inspired relatively new architectural structures called lightweight containers, which provide the framework and configuration that allow dependencies to be configured at runtime and inserted into a program using dependency injection. Instead of coding the assemblers yourself, you can rely on the framework to manage the assembler aspects of the dependent objects and call the injection methods in the code. The dependencies are managed with configuration files, eliminating any compile time dependency, which greatly increases the flexibility for implementation of the different components (Tate, 2005b). This increased flexibility can allow an e-learning systems design to introduce new components to implement new technologies and pedagogies with minimal impact on the rest of the system.

Domain Model. The value of using a domain model is its ability to handle complex business logic in a well organized way (Fowler, 2003). E-learning systems have increasingly complex and ever changing business rules that govern pedagogical practices, dictate course delivery, student assessment tools and learning styles, as well as monitor, authorize and organize the users of the system. Putting a domain model in an application involves inserting a layer of objects that model the business logic where each object represents something meaningful in the domain (Fowler, 2003). The objects in the domain model are separated from the rest of the system functionality so that you can concentrate on the complexity and focus on minimizing the coupling of this layer. This model must be able to be modified and tested easily, as it will become the core of the system's ability to adapt to the changing e-learning environment.

Data Mapper and Repository. The domain model of an e-learning system
will likely need to be integrated with legacy student information systems within an organization, which is well suited to the Data Mapper pattern that separates domain logic from data source (Fowler, 2003). The primary benefit of the pattern is that it allows data base schema and domain object models to evolve independently, as explained by Fowler:

This helps you in the code because you can understand and work with the domain objects without having to understand how they’re stored in the database. You can modify the business models or the database without having to alter either. With complicated mappings, particularly those involving existing databases, this is very valuable. (2003, p. 170)

Repository is another layer of abstraction that is added to the Data Mapper pattern that further isolates the domain objects from details of the data access. Repository is especially useful on large systems with multiple possible data sources (Fowler, 2003). E-learning systems may need to interface with many organizational data sources, and the Repository pattern allows the system domain model to appear as simple in-memory collections of objects. The underlaying location and storage of data is not exposed to client code, and the e-learning system can evolve independently of the data sources. The Repository data sources may be relational databases, or could also be different sources that are providing data to the domain model. This allows a tremendous amount of flexibility for the system to integrate with flat files, directory servers, messaging systems or unknown future technology in addition to the databases, with the Repository handling the access to the data, abstracting it from the domain. This is also beneficial during testing, when the repository data sources can be
replaced by in-memory objects to improve the speed and predictability of the testing.

**Agile Practices.** As has been previously discussed, there are many potential problems with an evolutionary architecture promoted by agile methods. Many argue that this type of design with ad-hoc design decisions causes the software to degrade and can’t possibly work. What is often overlooked are agile practices that can be employed to enable the architecture to evolve. Fowler (2004b) explains: “the criticisms stem from critics’ own experience where they didn’t do the enabling practices” (¶ 1). There are many recommended practices employed by agile methodology, three of which should be considered to be core for any e-learning systems development intent on implementing an adaptable e-learning system: (a) test driven development, (b) continuous integration, and (c) refactoring.

**Test Driven Development.** Test Driven Development describes a fundamental aspect of agile programming in which developers write tests before writing code. It is summarized as: “1. Quickly add a test. 2. Run all tests and see the new one fail. 3. Make a little change. 4. Run all tests and see them all succeed. 5. Refactor to remove duplication” (Beck, 2002, p. 1). Testing is automated, continuous and complete, which provides the safety mechanism that enables the rest of the agile process to be possible (Fowler, 2004b).

The architecture must take testability into account, and designs must consist of many highly cohesive, loosely coupled components, in order to facilitate test driven development (Beck, 2002). The architecture evolves with the tests and is kept as simple as possible to enable changes, Beck wrote: “By
coding only what you need for the tests and removing all duplication, you automatically get a design that is perfectly adapted to the current requirements and equally prepared for all future stories” (2002, p. 204). Design patterns that support testability such as the previously mentioned Inversion of Control and Repository emerge when coding with test driven development. These patterns can isolate code into testable modules that can substitute dependent modules with mock objects, which allows, for example, heavyweight components to be exchanged for speedier implementations for testing purposes.

**Continuous Integration.** Continuous integration is a set of agile software development practices where the work on a developing system is integrated frequently, usually multiple times per day. Each time an integration is made, the system is automatically built, tested and deployed to detect errors as quickly as possible (Fowler, 2006). The fundamental aspects of continuous integration, such as maintaining a single source repository, automating the build, integrating code every day, and automating the deployment. These practices result in always maintaining working, tested executable software as it is developed, which is one of the main goals of agile development. Keeping changes in sync, and identifying errors as they occur and maintaining a working system enables the architecture to be adaptable because changes can be made with the confidence that problems will be identified quickly (Fowler, 2006).

**Refactoring.** Refactoring is an agile practice that describes how to improve the design of a system in small steps by eliminating duplicate code, isolating the parts of code that need to change, defining interfaces and moving methods to improve cohesion (Beck, 2002). Fowler (1999) defined it as: “a change made to
the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior” (p. 53). Refactoring is a valuable tool that improves the design of the software on an ongoing basis, which is vital in agile methodology in order to prevent the architecture from degrading. Refactoring makes the software easier to understand, making it easier to maintain, and allows for more rapid development.

Refactoring is the key agile practice to enable an adaptable and maintainable architecture that can grow and evolve with the requirements. In the absence of a large amount of architecture and design being done up front, refactoring is the design of the program, done as the program is developed. The emphasis of the architecture should not be on achieving the correct design solution before development starts, but rather developing an architecture that can be easily refactored into a correct design as the requirements of the system change and grow. Reasonable and simple solutions should be sought in the up front design and refactored as the system grows and the problems become more well known, as Fowler explains:

Refactoring can lead to simpler designs without sacrificing flexibility. This makes the design process easier and less stressful. Once you have a broad sense of things that refactor easily, you don’t even think of the flexible solutions. You have the confidence to refactor if the time comes. You build the simplest thing that can possibly work. As for the flexible, complex design, most of the time you aren’t going to need it. (1999, p. 68)

In order to effectively refactor, you must have solid, automated and continuously integrated tests. A suite of automated tests gives developers confidence to ensure that the code functionality is not breaking as the result of a
refactoring. Refactoring code would be dangerous and time consuming if not accompanied with test driven development and continuous integration.

**Conclusion.** This section has explained agile architecture and how design principles and agile practices can be employed to implement an adaptable, lightweight e-learning systems architecture by providing loosely coupled, testable components. The next section explores open source software, which can be of tremendous benefit to agile software development as there are many tools available that are specifically geared towards agile practices such as continuous integration and test driven development. Agile team members are generally inclined to pride themselves on their ability to use these tools and libraries and these individuals will invariably support and promote their use (Theunissen, Boake & Kourie, 2005).

**Open Source Architecture**

Free/Open Source Software (FOSS) is increasingly providing tools, components and frameworks that can be used in software development infrastructure and architecture. The principles of FOSS and their benefits to e-learning systems integration and development were discussed in Chapter 2. This section examines how the decision to use FOSS impacts the implementation of an e-learning architecture and discusses the benefits that FOSS can provide to an agile e-learning architecture.

**Selecting FOSS Tools.** There are many FOSS tools, components and frameworks available for agile development, infrastructure and design. Agile architecture needs to be supported with tools for continuous integration, testing
and code analysis tools as well as system infrastructure such as application
servers and frameworks. The open source community is very active in all these
areas with many projects from large organizations such as Apache, Mozilla, Sun
and Red Hat contributing systems, servers and frameworks, as well as smaller
projects contributing tools and components on sites such as sourceforge.net.

Wheeler (2007) outlines many advantages for choosing FOSS (which he calls
OSS/FS) for software development:

- OSS/FS has significant market share in many markets, is often the
  most reliable software, and in many cases has the best performance.
- OSS/FS scales, both in problem size and project size. OSS/FS software
  often has far better security, perhaps due to the possibility of
  worldwide review. Total cost of ownership for OSS/FS is often far less
  than proprietary software, especially as the number of platforms
  increases. (Conclusions section, ¶ 1)

Searching for and evaluating FOSS that can support the project and
architecture can be a challenge because there are many open source projects to
choose from and it is not always easy to find the good ones. Many of them are
either just starting out or have stagnated and do not have very good quality
software. However, after having spent a tremendous amount of time evaluating
FOSS for use in software development projects, the author has found that it is
much easier to determine the value and quality of FOSS than proprietary
software. The basic steps for evaluating software, whether FOSS or proprietary,
are essentially the same: (a) identify candidates, (b) read existing reviews, (c)
briefly compare the leading programs’ attributes to your needs, and (d) perform
an in-depth analysis of the top candidates (Wheeler, 2008a). The key difference
is that FOSS programs have a great deal of publicly available information that
isn’t available for proprietary programs, and is generally uninfluenced by proprietary marketing. Not only do you have access to the program’s source code, but there are public forums for discussion and analysis of the software design, the project’s future direction, and on how well it’s working (Wheeler, 2008a). The defect tracking is public and can be searched to gauge the project’s quality and ability to respond to problems with the software. Also with FOSS, you can easily download and install the full versions of the software to try it out, as compared to trial versions of proprietary software that are often crippled or expire after a certain time. If you download and install FOSS, and then decide to use it in your project, there is nothing more you need to do, you can just continue to use it.

**FOSS Licenses.** The Open Source Initiative (OSI) has defined compliance criteria for open source licenses and has created an approval process to ensure that software labeled as free/open source conforms to these criteria. There are many OSI approved licenses, but there are differences between them that may need to be considered when evaluating the suitability of FOSS within a project. This section discusses some of the most popular FOSS licenses and explains some of these differences.

The MIT, BSD and Apache Licenses are three similar licenses that are the most straightforward in terms of their conditions. They are used in many open source projects, with Apache and BSD being two of the larger open source organizations with many licensed projects. These licenses are very flexible and compatible because they make the source code available without restriction, allow the code to be used in proprietary closed software without requiring that
those proprietary vendors distribute open source versions of their code (St. Laurent, 2004). In contrast, the Gnu General Public License (GPL) was created by the Free Software Foundation by open source advocates who do not want others to be able to distribute closed versions of the software. The GPL “explicitly requires that derivative works be distributed under the terms of the GPL and also that derivative works may only be permitted to be distributed under the terms of the license” (St. Laurent, 2004, p. 14). While this guarantees that the code will remain open, this is a much more restrictive license, and can cause incompatibilities with distributing the code as part of your software because if software is to be distributed that contains GPL code, then that software must be GPL compliant (Wheeler, 2008b). The Gnu Lesser General Public License (LGPL) is slightly less restrictive, in that it allows closed software to use and link to LGPL code without the closed software source code itself needing to be distributed, only the source code of the LGPL software itself needs to be made available (St. Laurent, 2004). The Mozilla License is similar in that it also requires source code to be distributed, however it does allow for the use of the code to be combined into larger works with other licenses without requiring the larger works themselves to be open source (St. Laurent, 2004).

It is important to note that the restrictions imposed by the GPL, LGPL and Mozilla licenses which require that the source code be made available are only applicable to software that is distributed to other parties. If you are using the code for an in-house project, you are free to use it however you wish, and there are no requirements that you make any source code available to anyone else. It is only if the software is redistributed to another party that the license requires the source
code to be made available to that party (Wheeler, 2008a). E-learning systems that are developed by an organization for their own internal use would be free to use FOSS in the project however they wish, but an organization that wants to sell or distribute the e-learning system to another organization would need to distribute under a license that is compatible with the FOSS licenses being used by the software.

**Agile, FOSS and E-Learning**

This essay has discussed the benefits that agile software development could have for e-learning, as well as benefits that could be found by using open source software. While both of these philosophies offer benefit, it is the combination of these two methodologies that the author argues can maximize the benefits available for e-learning systems development. The remainder of this essay examines e-learning systems architecture with a focus on the ability to develop systems with beneficial features that combine both FOSS and agile capability.

**Current E-Learning Systems**

Before discussing an e-learning architecture and design strategies in the next chapter, this section describes some of the existing e-learning systems currently available. There are many proprietary systems available, but the author will focus on describing those that are available under a FOSS license, while maintaining a focus on the architectural technology. Appendix A provides a partial listing of e-learning systems; this section briefly selects three of the most popular
open source e-learning systems from that listing: (a) ATutor, (b) Moodle and (c) Sakai, to describe aspects of current e-learning architecture that can be used in comparison to the prototype architecture presented in the next chapter.

**ATutor.** ATutor is a feature rich LMS developed by the Adaptive Technology Resource Centre of the University of Toronto. It was created with accessibility as a priority and claims to be the most accessible LMS available on the market allowing access to all learners, including those with disabilities who may be accessing the system using assistive technologies to participate fully in learning activities (Clark & Baggaley, 2004). ATutor is built upon a PHP based architecture and enables adaptability primarily by supporting the development of modules which can be plugged in to the system to add new features. ATutor provides a comprehensive module developer guide for module development to help organizations develop their own feature modules (“ATutor: Module Development Documentation,” n.d.). Modules are imported either through plugin hooks in the code, or newer versions of ATutor provide a module manager that allows system administrators to more easily import the modules from a web based administration console.

**Moodle.** Moodle is another open source PHP based system, which is very popular and has a significant user base with almost 50,000 registered sites using Moodle as their LMS (“Moodle Statistics,” n.d.). Like ATutor, Moodle also supports adaptability through the use of modular plugins (in fact the “M” in Moodle stands for Modular). The technical architecture in Moodle supports a variety of different types of modules with through their many plugin API (“Moodle: Developer Documentation,” n.d.).
Sakai. The Sakai project was started in 2004 as a collaboration between Stanford, Indiana, Michigan, MIT and Berkeley in an effort to build a system to be used by higher education without licensing software from commercial vendors (“The Sakai CLE,” n.d.). Sakai establishes a layered architecture and promotes adaptability through a service layer with standardized Open Service Interface Definitions (OSIDs) from OKI (Norton, 2007). The OSIDs are used to create services that are modular and portable across Sakai environments to create Sakai modules that are adapted to local system requirements.

The three systems discussed in this section have some good features for adaptability that could be used in an agile development environment, specifically the use of modules or plugins to develop new features. Agile development processes could focus on the development of modules within these systems to provide adaptability. However, relying solely on feature modules to provide adaptability limits it to the scope of the architecture’s supported module development. While you can add new features through modules, you cannot adapt the architecture itself to easily provide enhanced performance or support for new technologies for example. The fact that these systems are open source allows developers to go beyond module development and modify source code directly, however it can be very difficult to add features in this manner. In order to accomplish this type of adaptability, an architecture must be supportive of strategies that can further agile development. The next chapter will present such an architecture and compare it to current e-learning system technical architecture discussed in this chapter. ATutor and Moodle are PHP based systems, which would make them difficult to use for a meaningful comparison due to a lack of
tools, frameworks or containers that support the specific design and development concepts outlined in this essay. Sakai is based on a lightweight Java container architecture, and contains some design principles that resemble the prototype architecture presented by this paper. The next chapter will analyze the Sakai architecture in more detail, and then compare it to a new prototype architecture.

Conclusion

This chapter has described the characteristics of agile and open architectures by examining architectural designs, patterns, principles and tools that can be used for an implementation that supports these philosophies. Examining these characteristics has provided insight into how architectural design in an agile process can be approached and also outlines the role that FOSS can have in the architecture. With these characteristics in mind, current open source e-learning systems were described and the Sakai architecture was selected to be explored in more detail as part of the architectural analysis in the next chapter. The next chapter goes on to describe a model prototype architecture and tools that build upon the characteristics discussed in this chapter to provide a strategy that can enhance the current modular approach by providing a foundation that can better enable implementation of an agile e-learning system.
CHAPTER IV

ARCHITECTURE DESIGN AND STRATEGIES

This chapter proposes specific designs, frameworks and technologies built upon the characteristics presented in the previous chapter. Design models are presented as a template prototype architecture and strategies for implementing the models are discussed. Some specific open source frameworks and tools are proposed for the development of agile e-learning systems, and Sakai, and open source LMS, is evaluated in comparison to the models that have been created in this chapter.

Introduction

An architecture should describe the core elements of a system, a foundation upon which the system is built. Traditionally this involves extensive planning and design, however, in an agile process it should only describe a broad starting point for the architecture. There still needs to be an architecture defined up front for the agile system, but it should be lightly modeled and only need describe some of the base concepts upon which the architecture can be built. The architecture needs to be built as the system grows with the changing requirements, and as such it would not make sense to attempt to model an entire agile e-learning system architecture. Instead, the goal is to identify an architectural strategy that would foster agreement within an agile development team, and to do it swiftly without mounds of documentation (Ambler, 2008b).
Traditionally, architectures are modeled as detailed solutions to problems that are present in the requirements at the beginning of a project. Some agile methodologists refer to this as “big modeling up front”, which is done in conjunction with the up front requirements analysis to create a detailed system specification early in the life cycle of a project (Ambler, 2008b). This approach leads to serious technical decisions being made when very little is known about the problem domain, which becomes problematic if a chosen architectural model is difficult to adapt in the future. It may appear that everything in an up front model is going to work, but until it is proven with code and results in working software that satisfies some requirements, it can't be known for sure if the model is correct. Agile architectures are not completely defined and modeled up front, but need to evolve over time with working code and software that is adapted as needed.

Some agile developers do not do any up front architectural modeling at all, but the author believes that initial architectural modeling is important for scalability of large complex development effort. A technical vision is needed, but speculative aspects of the system that will have a tendency to change and render the models obsolete need not be modeled early on in development. An agile architectural model should create simple diagrams that explore the organizational aspects of the code, some initial entities and their relationships, and possibly some change cases to assess potential architecture-level requirements that the system may someday need to support (Ambler, 2008c). Ambler explains the goal:
“your initial architectural models will need to evolve as you learn more, but for now the goal is to get something that is just barely good enough so that your team can get going” (2008c, Initial Architecture Modeling section, ¶ 2).

The remainder of this section proposes high-level architectural models and frameworks that describe the concepts of a foundation of an agile architectural prototype that is layered in loosely coupled components for database interaction and systems integration.

**Layering.** Layering is a common architectural technique for breaking apart a complicated software system, logically organizing the system into layers where a higher layer can use the services of a lower layer, but the lower layer should be unaware of the higher layer. The concept of breaking down a system into layers has a number of important benefits for agile development, such as: (a) creates simplified understanding of of one layer without complicating it with details from another layer, (b) allows substitution of layers with alternative implementations, and (c) can minimize dependencies between layers (Fowler, 2003). The most difficult part of a layered architecture is determining how many layers the system should have, and designating the responsibility of each layer.

An agile architecture should begin by defining the layers at a high level to provide the organization of the modules in the system. These layers become the foundation of the design and should be well understood by developers to ensure the code is maintained within this framework as the system evolves.

**Logical Layered Architecture.** The logical layered architecture is modeled as a logical package diagram in Figure 1. The author finds UML package diagrams to be a useful construct to depict a high level overview of a layered architecture.
Each package represents a layer in the system and shows how the system is vertically layered and modularly organized. The diagram is composed only of packages and the dependencies that exist between them, and can be referenced when development begins to organize classes into packages and to determine the packages that should be analyzed with software package metrics.

The logical layered architecture provides a simple structure to build an e-learning system with designated responsibilities for each layer. The three principle layers of this architecture are the presentation, domain and data source layers. The presentation is responsible for giving the learners and educators
access to the system through a UI while the data source layer manages the integration of the various data systems. The domain layer is the isolation and encapsulation of the learning system itself, which includes specific user-driven requirements such as pedagogical work flows and student information tracking. The isolation of the domain layer is a key aspect of allowing the system to quickly adapt to these user requirements.

The layers must minimize dependencies to allow the system to remain adaptable. The logic encapsulated in the domain layer for has no dependency or knowledge of the technology used in the presentation layer or data source layer, and the data source layer has no dependencies into the system at all. These layers are joined by two mediating layers: the controller layer between the domain and presentation, and the data mapping layer between the domain and data source. The purpose of these mediating layers is to keep the domain layer as loosely coupled from the rest of the system as possible, and will be discussed in detail later in the chapter.

Presentation Layer. The presentation layer is responsible for showing information to the user and interpreting input commands. The user could be a human user accessing the system from a presentation of forms, such as HTTP requests with a web based client UI, a command line, thick client applications or mobile applications. The user of the system could also be an external system that access via an API presentation that enables integration or imports from external sources. There is never any business or application logic in the presentation layer; its sole purpose is to provide the user interface to the system.

Domain Layer. In the e-learning system, the domain layer encapsulates the
business logic and isolates the complexity of the learning systems within the layer. A domain model is isolated within this layer, and the classes within the domain model would be able to be adapted to the changing requirements of the domain while minimizing the dependencies with the other layers. A starting point for an e-learning domain model with classes that can be implemented as plain old Java objects (POJOs) is shown in Figure 2. This is not a comprehensive model, but a few simple domain classes to show how the business logic of e-learning can be organized in this architecture to isolate it from the rest of the system, minimizing its dependencies. The ability for the domain layer to remain loosely coupled from the rest of the system is the key aspect to this architecture's to allow organizations to evolve with changing pedagogical requirements.

Figure 2. Domain Model

The four classes: Assessment, Student, Course and Learning Object represent a starting point that can be built upon as requirements evolve to create a rich Domain Model that will provide adaptable implementation of changing e-learning requirements. The domain model has no dependencies on the controller and presentation layers, and the data mapping layer abstracts the actual data
sources. The essential aspect of allowing the domain model to be adaptable is also the most difficult part to implement; connections must be established with the controller and data mapping layers while minimizing the dependency issues associated with those connections.

**Controller Layer.** The presentation layer is dependent upon the domain layer and must be isolated from the parts of the domain model that are most susceptible to change. A controller layer is a mediator between presentation and domain that is added to the system to provide a clear API to the domain that can be stabilized so that changes to the domain model are abstracted from the presentation (Fowler, 2003). Any logic that is application or presentation specific is encapsulated within this layer so that the domain layer is unaffected by specific application implementations. This allows different applications, potentially running on disparate platforms and technologies to use the same domain logic. Different presentation platforms may also be able to re-use services, but would have the flexibility to extend controllers for specific needs of a specific presentation implementation without interfering with the domain model. Evans (2003) describes responsibility within this layer: “The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems. It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down” (p. 70).
The controller layer is expanded in Figure 3 to show an application controller that takes on the responsibility of calling the domain logic and determining the views that are to be displayed in the presentation layer. While the end-user applications can be isolated in the application controller, the controller layer also contains integration gateways which provide the mediator for external systems' access to the domain layer. The architecture allows for change by isolating these interfaces so UI and integration modules can be added or modified with minimal dependency on the domain model.

Data Mapping Layer. To act as a mediator and to manage the complexity between the domain and data source layers, a data mapping layer is added to the system. One of the biggest challenges with using a domain model is that the connection to persistent data sources can be complicated (Fowler, 2003). While
the data source layer manages any communication with databases or other data persistence mechanisms, a data mapping layer abstracts the domain layer from the data sources using data mapper and repository patterns as shown in Figure 4.

Figure 4. Data Mapping Layer

The complexity of the e-learning domain, combined with the fact that there will most likely be relational databases that need to be accessed from the system
makes data mapper pattern a good candidate for implementing the object/relational connections in this layer. The repository pattern adds another level of abstraction in the mapping layer to combine multiple, disparate data sources accessed by the system to provide a more object-oriented view of the data mapping layer, as Fowler explains: “Repository supports the objective of achieving a clean separation and one-way dependency between domain and data mapping layers” (2003, p. 323).

Figure 4 shows how the domain objects could access repositories in the data mapping layer to populate themselves from multiple data sources. The domain layer is completely decoupled from the actual data sources, the domain objects simply accesses a corresponding repository to retrieve data. The repositories are mapped to multiple data sources, with Student data retrieved from a student information system (SIS), the Learning Object data from a learning object repository (LOR), and Course data comes from a content management system (CMS). As data sources are changed, or new ones added, the mappings can be modified without impacting the business logic in the domain layer.

*Inversion of Control*. The data mapping layer abstracts the data sources and the controller layer abstracts access to the domain, and while this layering strategy allows the domain model to be the central point of the system, it is still coupled with services in these two mediating layers. For example, the dependencies between domain layer and data mapping layer in Figure 4 demonstrate that access to the repositories in the data mapping layer would need to be hardwired in the domain layer, resulting in a tight coupling between the
layers. The data mapping succeeds in abstracting the domain from the data sources, but each object in the domain is still coupled to a specific repository implementation.

Inversion of control allows for these dependencies to be specified by configurations and eliminates the tight coupling between the layers. The dependencies are resolved at run time by the configuration file, which an assembler uses to create the appropriate data sources and repositories. The coupling is removed from the code and placed into configuration, which a lightweight container framework uses to assemble the instances of the dependencies by injecting them where needed. The job of creating objects and setting the appropriate properties is passed from the application to a framework provided by the lightweight container (Tate & Gehtland, 2004).

For example, the domain model needs to access the repositories to retrieve and update the data, while the repositories need access to the data sources. An assembler reads a configuration file, creates the data sources and repositories, and injects the objects into the dependent locations. Figure 5 shows how the dependencies are now inverted, with the inversion of control assembler in place, the tight coupling between the data mapping and domain layers is eliminated.

![Figure 5. Inversion of Control Assembler](image)
The inversion of control assembler allows for decoupled domain objects that are easy to code, test, extend, and use. The same dependency injection concept can be used to manage dependencies between the application controllers and their views in the presentation layer, as well as any other services needed in the controller or data mapping layers now or as the system architecture evolves in the future.

Frameworks

The patterns and principles modeled in this architecture can be implemented with the help of two frameworks: The Spring lightweight container for inversion of control and Hibernate for object/relational data mapping.

Spring Lightweight Container. The Spring framework is a FOSS lightweight container that can provide inversion of control configuration to an application. Spring's inversion of control container is a central aspect of the framework that creates and initializes objects and calls initialization methods on dependent objects to wire an application's dependencies together (Tate & Gehtland, 2004).

Figure 6. Sample Spring Framework Configuration

```xml
<beans>
  <bean id="Student" class="domain.domainModel.Student">
    <property name="repository">
      <ref local="StudentRepository"/>
    </property>
  </bean>
  <bean id="StudentRepository"
       class="dataMapping.repositories.StudentRepository">
  </bean>
</beans>
```

A Spring configuration defines the objects as beans in an application context within an XML document. Figure 6 is an example of a partial Spring XML
configuration file that shows how the domain and repository dependencies could be specified as beans in a Spring application context. Spring would create the two specified objects, Student and StudentRepository, and then wire them together by injecting the StudentRepository into the repository field of the Student object based on this configuration.

The Spring lightweight container reduces complexity by providing the framework for assembling loosely coupled components via dependency injection which enables agility and improves testability. The management of the dependencies by the framework allows components, especially the domain model, to be developed and tested in isolation, which allows for easier refactoring to evolve the architecture within agile processes. Spring also provides support for data mapping frameworks, including Hibernate.

**Hibernate Data Mapping Persistence.** Hibernate is a FOSS project that provides a data mapping persistence framework for storing Java objects to a persistent relational database. The data mapping pattern is implemented by the framework so that the persistence is transparent and the domain model does not need to contain any persistence or transactional code. The domain model is mapped to a relational database schema via hibernate mapping configuration files, and then Hibernate is configured to find the database and JDBC driver (Tate & Gehtland, 2004).

Figure 7 shows how hibernate can be used as the data mapping layer in the architecture. Hibernate uses a Session object for each data source, which are configured by hibernate-cfg.xml files and created with a HibernateSession Factory. Each domain object is mapped by the hibernate mapping hbm.xml files,
and can then be queried through the Session objects. Spring can be used to create appropriate Session objects, which are configured by a Spring applicationContext.xml file, and then injected into the repository layer, providing complete transparency so that the domain and repositories have little dependency on the data persistence aware code in the mapping package.

Figure 7. Hibernate Data Mapping Layer

The three XML files allow the dependencies to be configured at runtime, allowing for loosely coupled components. The hibernate-cfg.xml files configure the data sources for each hibernate session. The hbm.xml files are the hibernate mappings that map the domain objects to the data base schema in the data source. Finally, the applicationContext.xml file is the Spring configuration that is used by the framework to determine which hibernate Session objects to create and inject into the repositories. Appendix E shows a simple example of the XML
configuration that would be required for the StudentRepository to access the SIS data source to extract student information into a Student domain object. Similar files would be required to wire the various domain objects to the appropriate repositories and data sources. The desired result is to have a domain layer that is decoupled from the data mapping layer, abstracting the data source access through the repositories, which are injected into the domain layer with Spring.

**Adaptable Architecture Strategies**

This section briefly discusses the strategies enabled by the architecture for future integration and possible change cases.

**Integration Strategy.** An important aspect to the e-learning architecture is the ability to integrate with other systems and data sources. The repository pattern, controller layer, and inversion of control pattern are the main tools for the integration strategy of this architecture. The repository pattern can be used to access data from multiple data sources while the integration gateways in the controller layer can be used by external systems to integrate with the e-learning system. Dependency injection minimizes the impact of dependencies and allows the integration of changeable components.

**Change Cases.** The most important aspect of an agile architecture is its ability to adapt to change. As such, a useful architectural model is a change case, which describes the potential architectural requirements changes that the system may need to support (Ambler, 2008c). Figure 8 provides two sample change cases, one describing a potential a technology change and the second a potential change to the e-learning domain.
The change cases describe the long-term viability of the architecture by thinking through the impact of these changes and ensuring that the system will be able to adapt. The impact section of these sample change cases describe how the architecture can accommodate these scenarios. We don't concern ourselves with the specifics at this point, but just ensure that some of the possible change cases have been thought through to ensure that the architecture would be able to adapt to these scenarios.

**Figure 8. Sample Change Cases**

<table>
<thead>
<tr>
<th>Change case</th>
<th>Delivery of learning materials and lessons will need to be delivered via mobile hand held devices.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood</td>
<td>Medium likelihood within two to three years, very likely within five years.</td>
</tr>
<tr>
<td>Impact</td>
<td>Medium. It is not known at this time what the technology for mobile devices will be able to deliver in two to three years time. Emergence of platforms such as Apple's iPhone and Google's Android have elevated the capability of these devices to utilize standard web based technology. It may be that new standards and protocols will emerge that will need to be developed for mobile e-learning, but it is equally as likely that new mobile devices will emerge that can utilize the existing web protocols and technology. Regardless, the architecture can accommodate the unknown nature of these changes in the presentation layer. New UI can be added in the presentation layer for the mobile devices, which may make use of existing application controllers. If new protocols need to be implemented for mobile access, a new application controller can be developed to implement these protocols and give access to the domain layer, without needing to make changes in domain layer.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change case</th>
<th>Student assessment methodology will need to be changed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood</td>
<td>High likelihood for this to be changing on an ongoing basis, due to previous experiences with changes in this area.</td>
</tr>
<tr>
<td>Impact</td>
<td>Low. The student assessment logic is encapsulated in the domain model, which is designed to be isolated and highly adaptable. The domain logic should be able to change and adjust to new pedagogical assessment techniques as these new requirements emerge.</td>
</tr>
</tbody>
</table>

**Physical Architecture**

The layers discussed thus far are logical; they divide the system organizationally, but do not necessarily determine the physical separation and distribution of the system, which can be done on any number of tiers. The
decision of where to run these layers is largely dependent upon the requirements that evolve in the agile development. Specific requirements and technology will determine how the software is best distributed. The logical architecture provides the flexibility of evaluating components of each layer in conjunction with the emerging requirements to make the best decision for the physical architecture of a specific application. For example, the system could be deployed as typical 3-tiered physical architecture, with the presentation layer running as a thin-client browser-based component, while the controller layer, domain and data mapping layers all run as components within an application server tier, and the data sources run on a third tier.

While a three-tiered physical architecture might be appropriate for much of the e-learning functionality, there is a great deal of flexibility within the layered architecture to distribute the components differently for other purposes. For example, all layers could be deployed on a single tier, with in-memory databases and an automated testing framework as the presentation layer for efficient automated testing of the system. Another example scenario would be a command line presentation component that runs on the same tier as the controller, domain and data mapping layers accessing remote data sources on a second tier. The command line application could provide useful administration features such as data loading or system configuration.

Regardless of the physical architecture scenario, the domain layer is reused without needing to make changes to the domain model. It can be packaged and distributed as a component in different physical architectures with different presentation layer components and access different database configurations.
without needing to make changes to the domain layer. This demonstrates the flexibility and adaptability of the architecture, and shows how new components could be added to support future services, integration, presentations or data sources as they are needed.

**Open Source Tools for Agile Development**

The primary goal of the architecture is adaptability, but agile development practices still must be employed to ensure the architecture remains maintainable. An agile development environment needs the infrastructure that supports the practices that enable the architecture grow while maintaining an adaptable design. This section outlines some important open source tools for test-driven development, continuous integration and software analysis metrics that can be used to help maintain the architecture and design of the system as it evolves.

**Test Driven Development.** JUnit is an open source test framework that enables automated unit test development. There is a remark from Martin Fowler, which used to be quoted on the junit.org website: “Never in the field of software development have so many owed so much to so few lines of code.” This emphasizes the importance of this tool for test-driven Java development, and also its simplicity. It provides tools to easily write test suites that can be run from the framework and to display testing results (Beck & Gamma, n.d.). JUnit can be integrated into virtually any Java development environment and has many plug-ins and extensions, such as HttpUnit, which automates web-based testing of http form submission, JavaScript, basic http authentication and cookies (Gold, 2002). DbUnit is a JUnit extension for database code that contains puts your database
into a known states between tests by exporting and importing data to and from XML datasets (“DbUnit-About DbUnit,” 2008).

An emerging area of test driven development is automated agile acceptance testing. These tests are different from unit tests in that they test functionality of the system, rather that individual units. They are designed to allow software developers and stakeholders to collaborate by creating tests together that ensure that the software does what it is expected to do. FitNesse is a lightweight, open source framework that allows acceptance tests to be collaboratively defined in a wiki format, and then can run the tests in an automated test environment (“What is Fitnesse?,” n.d.).

Another very useful addition to an automated test suite is a code coverage tool, which measures and reports how much code is being tested, and more importantly, identifies code that has not been tested. Cobertura is a FOSS tool that reports the test coverage of each class in the system; it can be integrated with various Java development environments and also has a health check feature that can be used in a continuous integration build to ensure that a code base is being tested to a desired level on each build (Doliner, 2006).

**Continuous Integration.** Cruise Control is a FOSS framework for implementing a continuous build process. It integrates easily with Java development environments and testing tools to provide a fully automated, tested and reproducible build (“CruiseControl,” n.d.). It can also be easily integrated with a version control system, such as the open source Subversion, to manage the ongoing development of code. CruiseControl can be configured to detect changes made in the subversion repository and automatically start a build and
test to provide immediate and continuous feedback for the software changes (“CruiseControl: Overview,” n.d.).

CruiseControl provides a consolidated view of each build that provides important information, such as build errors, test results, details of the files that have changed since the last build, and software metrics reports that allow you to monitor the maintainability of the software design at a glance after each build (“CruiseControl: Build Results JSP,”, n.d.).

Software Analysis and Metrics. The tools in this section provide analysis and metrics that allow for maintainability of the software design and architecture as it evolves in an agile project.

JDepend is an extremely useful open source tool for the maintainability of an evolutionary architecture. JDepend allows you to automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to effectively manage and control package dependencies (Clarkware Consulting Inc., 2008). JDepend can be integrated into a continuous integration environment for immediate feedback if thresholds of software package metrics are violated.

JavaNCSS is an open source static code analyzer that performs useful metrics that measure the complexity and maintainability of the code (Lee, n.d.). All the metrics listed in Appendix D are calculated by using a combination of JDepend and JavaNCSS. Integrating these tools with continuous integration reports the integrity of the package dependencies and monitors the maintainability of the design as it evolves through refactorings.

Checkstyle is an automated code analysis tool that provides checks to find duplicate code, class design problems and bug patterns (Burn, 2007). PMD and
FindBugs are similar tools that provide static analysis of source code and bytecode respectively. All of these tools can be integrated with CruiseControl to provide continuous builds with automatic static code reviews that can identify problem areas of code that should be considered for refactoring (Berg, 2008).

Using software metrics tools in a continuous integration environment is essential to agile architecture to ensure that the quality of the design is maintained as it evolves in terms of its extensibility, reusability, and maintainability.

**Sakai**

The architecture that has been presented in this chapter is similar in some respects to Sakai, an open source Java based LMS, but there are also many fundamental differences. This section analyzes and contrasts the Sakai architectural philosophies and design and poses some questions about the different challenges that would be faced if a software architecture such as Sakai was to be adopted in an agile development project.

There is a fundamental difference in the philosophies of the two designs which makes it difficult to compare the two. The focus of the architecture discussed in this chapter has been to provide adaptability within the design and code itself, which is a solution that is based on the assumption that the software is being developed in house by the organization to fulfill changes in the organization's requirements. Sakai is designed for use by any learning organization and needs to provide a large feature set of generic functionality, much of which many organizations will not use. Sakai also needs to provide the
ability for organizations to customize this functionality to meet their needs, without changing the design or code of the base system by providing plug in interfaces for adding functionality, which will be discussed in more detail later in this section. Sakai’s large feature set makes it a very large system with a complex architecture, which makes an in-depth analysis beyond the scope of this paper. It should be noted that the size and complexity of the system would in and of itself pose a challenge to agile processes, as compared to an in-house development project that could focus the features on what is needed by the organization, which is more aligned with an agile practice of avoiding unnecessary infrastructure until it is needed.

The high level analysis of Sakai provided in the remainder of this section serves to contrast the different architectural philosophies and emphasizes the challenges that adopting a large system such as Sakai would pose for agile processes.

Layers. Sakai uses an abstract layered architecture to implement a flexible system using Java, Dependency Injection, Spring and Hibernate. The Sakai abstract architecture describes layers that provide for separation for separation of application and presentation logic, shown in Table 1. The layers share some similarities, for example, the service layer is similar to a domain layer and the tools layer is similar in functionality to a controller layer.

One of the main differences is that the Sakai layered focus is primarily on the presentation, while the back-end is lumped into a single system layer: “The system is the server environment ... may include web servers, database servers, operating systems, file and resource repositories, enterprise and back office
systems, etc” (Norton, 2007, System Section). Sakai can use Hibernate to persist system objects to databases in the system, but it assumes that all data is being persist to Sakai data sources, and is not used for integration with other data sources. Any integration or interoperability in Sakai is not done in the system layer, but is done in the service layer, which provides an API and is moving towards promoting interoperability using standardized Open Service Interface Definitions (OSIDs) from OKI. Integration is also handled in the aggregator layer, which can combine output from other systems with the Sakai output (Norton, 2007).

Table 1. Sakai Abstract Architecture Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>Sakai is intended to be run as a client / server application pair with the client being a standard, off the shelf web browser the majority of the time.</td>
</tr>
<tr>
<td>Aggregator</td>
<td>The output of one or more Sakai application can be combined using an aggregator server application.</td>
</tr>
<tr>
<td>Presentation</td>
<td>The presentation layer combines data from a Sakai tool and a user interface description to create a mark up fragment which is aggregated before delivery to the client.</td>
</tr>
<tr>
<td>Tools</td>
<td>Tools are the applications which merge presentation logic with application logic contained in services, responding to user interface requests and events.</td>
</tr>
<tr>
<td>Services</td>
<td>The services layer is a collection of classes which manage data via a defined set of behaviors, and is accessible via a published Application Programming Interface (API).</td>
</tr>
<tr>
<td>System</td>
<td>The system is the server environment which the Sakai environment resides, plus any remote capabilities available to it.</td>
</tr>
</tbody>
</table>

Note. Descriptions are based on the layers and descriptions in Norton (2007).

Adaptability. The fundamental difference between Sakai and the models that are presented in this essay is the focus on the isolation of the domain model and the ability of the architecture to keep the domain layer uncoupled so that it is adaptable. While the service layer in Sakai is similar to the domain layer, it does
not employ a rich domain model, nor does it focus any efforts on keeping the domain logic adaptable. Sakai’s service layer promotes flexibility with the OSIDs and creates services that are modular and portable across Sakai environments; the focus is on producing re-usable service modules, while the focus of the models in this paper is on producing an adaptable isolated domain layer that allows for an architecture to evolve. Different applications can be aggregated and tools and services can be created as Sakai modules that are adapted to local system requirements, rather than adapting and changing the domain code directly.

**Agile Development.** The focus of this paper has been to provide a starting point template for an agile architecture, which could start small and evolve as requirements grow within a learning organization, while the Sakai architecture is implemented as a large feature rich LMS that provides generic e-learning functionality. Sakai can be customized by developing modules and tools that are integrated into systems using OSIDs. If the needs of the organization are large enough to make Sakai a good starting point for a system, it could prove to be worthwhile.

There would be many challenges if an organization were to adopt a large project like Sakai in an agile development environment. The scope of the system would be large right from the inception of the project, and would include much unneeded overhead. Development done by the organization would need to be maintained in a separate custom code base and would need to be merged when new versions of Sakai are developed. Many of the agile practices and principles outlined earlier in this paper may not be applicable in the Sakai code base, and
might also prove difficult to achieve in the custom code base. The agile approach to architecture would need be different, as the focus would need to be on providing custom modules, services, aggregators and tools that can be plugged into the Sakai architecture to meet the changing requirements of the organization, rather than on adapting the domain model and continuously updating the design to evolve the architecture to provide for changing requirements.

This section has discussed some of the similarities shared by Sakai and the template architecture proposed by this paper, showed the fundamental differences and introduced how these differences could impact an agile development project. Sakai is a large and very involved architecture and further analysis is beyond the scope of this paper.

Conclusion

The architecture modeled in this chapter presents a starting point that could be used as a template for an adaptable e-learning system design for use in an agile environment. In applying concepts of agile design principles and introducing an abstract layered architecture with dependency injection, an architectural model was presented that could be implemented in a lightweight container such as Spring. The architectural model is accompanied by strategies that demonstrate the adaptability of the architecture, and descriptions of open source tools that could be used in agile development of an e-learning system. Finally, the architecture was compared to Sakai to highlight the differences between the architectures and identify how the differences in architectures would impact an agile project.
The results presented in this chapter are not intended to provide a detailed architectural solution, however, the models and explanations provided insight into lightweight, agile architecture and highlighted, at a conceptual level, how it addresses some challenges common in e-learning systems development.
CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

This concluding chapter reviews the research problem, evaluates the architectural models and comments on the feasibility of the results being implementable as agile e-learning systems architecture. The conclusion reiterates the merits of agile and open software methodologies for e-learning systems development and provides recommendations and discussion in regards to implementation the architecture with agile and open development principles, and the potential consequences this would have on e-learning systems development.

E-learning Requirements Change

The research done in the second chapter of this essay provided evidence that e-learning systems development is susceptible to problems associated with requirements change, in particular changing technology, pedagogy and the increasing pressure to integrate e-learning systems with other systems. These changes can potentially degrade a software architecture as it needs to adapt in ways that an original design did not anticipate. Instead of addressing these issues directly with architectural solutions, software development methodologies were explored, specifically agile and open methodologies, to examine their ability to enable adaptability in e-learning systems development.

Agile and Open Software Development

The agile principles and practices examined in this paper provided a
methodology that can be employed to help address many of the challenges associated with changing requirements. The agile methods outlined a philosophy that accepts and welcomes changes in software development. The ability to do so starts with a set of agile principles, but these principles need to be supported by practices, tools and design principles. Similarly, there are many advantages to using open source software to address some of the issues associated with e-learning system development and integration, and these philosophies align well with similar emerging open philosophies of education and academia.

After examining agile and open principles of software development, the third chapter focused on the impact that these methodologies have on the software architecture and discussed some fundamentals of how an architectural design can be structured and implemented to accommodate the principles. One of the original points in this research is to develop an architecture by focusing the architectural decisions on the software development principles and practices that promote openness and adaptability, and to show the benefit of this approach for e-learning. By examining the agile and open methodologies and what impact they have on architectural decisions, original models were created that could deliver e-learning systems that would support the development practices associated with the methodologies.

**Agile and Open E-learning Systems Architecture**

The template architecture provided by the models in chapter 4 showed, at a high level, concepts that can be employed to accommodate agile practices and provide a loosely coupled adaptable architecture. The fundamental principles of
the architecture reduce problematic dependencies through abstraction and dependency injection, especially in areas that are likely to change in an e-learning systems, such as the domain model, data source integration, and presentation layers.

The architecture does not provide a complete e-learning systems solution, nor does it model any specific e-learning systems requirements, standards or application technology, but it does provide a template that can be referenced as a prototype example of an adaptable e-learning system that would allow an organization to start with a small set of requirements, develop their e-learning requirements in an agile manner and incrementally build a system as the requirements evolve and change.

This approach was contrasted to Sakai, which uses similar architectural technology, but is a large e-learning system that an organization could customize. This would require a different approach to the agile development, as compatible modules would need to be developed and customized that conform to the Sakai APIs and architecture. A thorough analysis of Sakai would be required to determine the feasibility of developing these modules in an agile development process, which is beyond the scope of this paper.

Recommendations

The strongest arguments in this paper are those that favour agile and open software development principles for e-learning systems and were presented in the second chapter. It is the author’s recommendation that e-learning systems development needs to develop strategies that deal with the pedagogical,
technical and integration issues that were discussed, and should strongly consider both agile and open source methodologies to do so. A software architecture that supports the practices employed by these methodologies is necessary for an e-learning development project to succeed in this regard, and the architectural models presented in this paper can be used as guidelines towards implementing a system while following these principles.

**Future Work**

While the architectural models presented in this paper provide a high level example, the author would be interested in evaluating this architecture more thoroughly and comparing it to other possible architectural solutions that might be implementable using agile and open practices. This could extend to an analysis that develops a working architectural prototype that could be used in implementation of an agile development iteration to identify areas of the design that might be problematic.

It would also be of interest to examine existing open source e-learning systems, especially Sakai, and evaluate the relative merits and drawbacks in relation to agile software development. In conducting such an analysis, the author would want to determine:

- Are there automated test suites with unit and agile acceptance tests?
- What percentage of code coverage do the tests provide?
- What are the values of the various software metrics outlined in Appendix D?
- What interfaces are provided for integration?
- Are the interfaces part of a standard?
● What mechanisms or frameworks are used for data persistence?
● Are there well defined layers, components or modules?

Evaluating an e-learning system by answering these questions would aid in determining the feasibility of using agile software development practices by determining the maintainability, scalability and adaptability of the design and architecture.
REFERENCES


CruiseControl: Build Results JSP. (n.d.). Retrieved July 14, 2008 from
http://cruisecontrol.sourceforge.net/reporting/jsp/index.html

http://cruisecontrol.sourceforge.net/overview.html


Elloumi (Eds.), *Theory and Practice of Online Learning* (pp. 97-114).
http://cde.athabascau.ca/online_book/


http://cobertura.sourceforge.net/introduction.html

Compliance and Compatibility – Building the Architecture to Support Future
http://www.blackboard.com/docs/AS/Blackboard_Whitepaper_Standards.pdf

Evans, E. (2003). *Domain-Driven Design: Tacking Complexity In the Heart of

Faber, B. D. (2002). Educational models and open source: resisting the
proprietary university. In SIGDOC ’02: *Proceedings of the 20th annual
international conference on Computer documentation* (pp. 31-38). New
York: ACM Press.


http://www.campussource.de/aktuelles/docs/Bensberg_Paper_ITI04_1.pdf


96


APPENDIX A

TECHNICAL ARCHITECTURES OF E-LEARNING SYSTEMS

A brief overview of some of the architectural technologies such as frameworks, databases, programming languages and systems in use by some selected Learning Management Systems is shown in Table A1.

Table A1. E-learning Systems Architectural Technologies

<table>
<thead>
<tr>
<th>Open Source</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ATutor (<a href="http://www.atutor.ca">http://www.atutor.ca</a>)</td>
<td>PHP, Apache, MySQL</td>
</tr>
<tr>
<td>Claroline (<a href="http://www.claroline.net">http://www.claroline.net</a>)</td>
<td>PHP, MySQL</td>
</tr>
<tr>
<td>Dokeos (<a href="http://www.dokeos.com">http://www.dokeos.com</a>)</td>
<td>PHP, MySQL, LDAP</td>
</tr>
<tr>
<td>Moodle (<a href="http://moodle.org">http://moodle.org</a>)</td>
<td>PHP, ADOdb Database access library, LDAP</td>
</tr>
<tr>
<td>OLAT (<a href="http://www.olat.org">http://www.olat.org</a>)</td>
<td>Java, Tomcat, Hibernate, Spring, YAML Framework</td>
</tr>
<tr>
<td>Sakai (<a href="http://www.sakaiproject.org">http://www.sakaiproject.org</a>)</td>
<td>Java, Tomcat, Spring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proprietary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ANGEL (<a href="http://www.angellearning.com">http://www.angellearning.com</a>)</td>
<td>MS SQL, Active Directory, ASP.NET, IIS</td>
</tr>
<tr>
<td>Blackboard (<a href="http://www.blackboard.com">http://www.blackboard.com</a>)</td>
<td>Java, Oracle, MS SQL</td>
</tr>
<tr>
<td>Desire2Learn (<a href="http://www.desire2learn.com">http://www.desire2learn.com</a>)</td>
<td>MS SQL, Active Directory, ASP.NET, IIS</td>
</tr>
</tbody>
</table>
APPENDIX B

E-LEARNING STANDARDS AND SPECIFICATIONS

Table A2 explains some of the important standards and specifications that have emerged in e-learning and provides links to the organizations' websites where the documented specifications can be found.

Table A2. E-learning Standards and Specifications

<table>
<thead>
<tr>
<th>IEEE Learning Technology Standards Committee (LTSC) Standards (<a href="http://ieeeltsc.org/">http://ieeeltsc.org/</a>)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Schema Definition Language Binding for Learning Object Metadata (LOM)</strong></td>
</tr>
<tr>
<td>This standard defines a conceptual data schema that defines the structure of a metadata instance for a learning object. This allows for interoperability and the exchange of LOM instances between various systems (Institute of Electrical and Electronics Engineers [IEEE], 2005).</td>
</tr>
<tr>
<td><strong>XML Schema Binding for Data Model for Content Object Communication</strong></td>
</tr>
<tr>
<td>This standard defines an XML schema definition language binding of the learning object metadata (LOM) data model. The purpose of this Standard is to allow the creation of LOM instances in XML and the exchange of LOM XML between systems (IEEE, 2006).</td>
</tr>
<tr>
<td><strong>ECMAScript Application Programming Interface for Content to Runtime Services Communication</strong></td>
</tr>
<tr>
<td>This standard described an ECMAScript API for the communication of information between content and an LMS. The purpose of this standard is to build consensus around, resolve ambiguities, and correct defects in existing specifications for an ECMAScript API for exchanging data between learning-related content and an LMS (IEEE, 2004).</td>
</tr>
<tr>
<td><strong>Learning Technology Systems Architecture (LTSA)</strong></td>
</tr>
<tr>
<td>This standard specifies a high-level architecture for information technology-supported learning, education, and training systems that describes the high-level system design and the components of these systems (IEEE, 2003).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IMS Global Learning Consortium (IMS GLC) Specifications (<a href="http://www.imsglobal.org">http://www.imsglobal.org</a>)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content Packaging</strong></td>
</tr>
<tr>
<td>The IMS Content Packaging Specification provides the functionality to describe and package learning materials, such as an individual course or a collection of courses, into interoperable, distributable packages. Content Packaging addresses the description, structure, and location of online learning materials and the definition of some particular content types (IMS Global Learning Consortium [IMS GLC], 2004a).</td>
</tr>
<tr>
<td><strong>Learning Design</strong></td>
</tr>
<tr>
<td>The IMS Learning Design specification supports the use of a wide range of pedagogies in online learning. Rather than attempting to capture the specifics of many pedagogies, it provides a generic and flexible language (IMS GLC, 2003).</td>
</tr>
<tr>
<td><strong>IMS Global Learning Consortium (IMS GLC) Specifications</strong> (<a href="http://www.imsglobal.org">http://www.imsglobal.org</a>)</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Enterprise Services</strong></td>
</tr>
<tr>
<td><strong>Learner Information Package</strong></td>
</tr>
<tr>
<td><strong>Question &amp; Test Interoperability</strong></td>
</tr>
<tr>
<td><strong>Advanced Distributed Learning (ADL)</strong> (<a href="http://www.adlnet.gov">http://www.adlnet.gov</a>)</td>
</tr>
<tr>
<td><strong>Sharable Content Object Reference Model (SCORM)</strong></td>
</tr>
<tr>
<td><strong>Open Knowledge Initiative (OKI) Specifications</strong> (<a href="http://okiproject.org/">http://okiproject.org/</a>)</td>
</tr>
<tr>
<td><strong>Open Service Interface Definitions (OSIDs)</strong></td>
</tr>
</tbody>
</table>
APPENDIX C

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools  
Working software over comprehensive documentation  
Customer collaboration over contract negotiation  
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler

James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick

Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.

Agile Principles

• Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
• Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
• Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
• Business people and developers must work together daily throughout the project.
• Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
• The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
• Working software is the primary measure of progress. Agile processes promote sustainable development.
• The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
• Continuous attention to technical excellence and good design enhances agility.
• Simplicity—the art of maximizing the amount of work not done—is essential.
• The best architectures, requirements, and designs emerge from self-organizing teams.
• At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

(Beck et. al, 2001)
APPENDIX D

SOFTWARE PACKAGE METRICS

Table A3 explains some common software package metrics that can be tracked in agile development to ensure that the architecture continues to be maintainable as it evolves.

Table A3. Software Package Metrics

<table>
<thead>
<tr>
<th>Software Package Metrics, as described by Martin (2003, p. 282), and calculated by JDepend.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(Ca) Afferent Couplings</strong></td>
<td>Calculated as the number of classes from other packages that depend on classes within the subject package.</td>
</tr>
<tr>
<td><strong>(Ce) Efferent Couplings</strong></td>
<td>Calculated as the number of classes in other packages that the classes in the subject package depend on.</td>
</tr>
<tr>
<td><strong>(A) Abstractness</strong></td>
<td>Calculated as the ratio of the number of abstract classes (or interfaces) in package to the total number of classes (and interfaces) in the package. The range of this metric is 0 to 1.</td>
</tr>
<tr>
<td><strong>(I) Instability</strong></td>
<td>Calculated as the ratio of efferent coupling to total coupling. ( I = \frac{Ce}{Ce + Ca} ).</td>
</tr>
<tr>
<td><strong>(D) Distance from the Main Sequence</strong></td>
<td>The main sequence is idealized by the line ( A + I = 1 ). The formula for D calculates the distance of any particular package from the main sequence; the closer to 0 the better.</td>
</tr>
</tbody>
</table>

**Additional Relevant Metrics provided by JDepend.**

| Package Dependency Cycles | All packages dependencies that intersect a dependency cycle are calculated and reported. |
| Number of Classes and Interfaces | The number of concrete and abstract classes (and interfaces) in the package are calculated. |

**Additional Relevant Metrics provided by JavaNCSS.**

| NCSS | Non Commented Source Statements. Calculates the number of lines of code, per method, per class and per package. |
| Classes per Package | Number of classes in each package, and the average classes per package for a project. |
| Functions per Class | Number of functions (methods) in each class, and the average methods per class for a project. |
| CCN | Cyclomatic Complexity Number. For each method, measures the number of linearly independent paths to indicate a method’s complexity. |
APPENDIX E

SPRING / HIBERNATE XML CONFIGURATION

Figure A1. Sample Spring Application Context

Filename: applicationContext.xml

Purpose: Spring configuration file that determines the objects that Spring needs to create, and where they should be injected. The excerpt shown here configures Spring to create a Hibernate Session for the SIS data source. The Session is created using a Spring Factory that is injected with properties from the sis-hibernate-cfg.xml (Figure A2) Hibernate configuration, and the Student.hbm.xml (Figure A3) Hibernate mapping. Spring will inject the SISSession into the StudentRepository Session property and the StudentRepository into the Student domain object.

```xml
<beans>
  <!-- SISSession object configured by Spring using the Spring Hibernate SessionFactory, which is injected with properties from the designated hibernate configuration file and mapping file. -->
  <bean id="SISSession"
       class="org.springframework.orm.hibernate.LocalSessionFactoryBean"/>
  <property name="hibernateProperties">
    <value>sis-hibernate-config.xml</value>
  </property>
  <property name="mappingResources">
    <list>
      <value>Student.hbm.xml</value>
    </list>
  </property>

  <!-- StudentRepository is injected with a SISSession object -->
  <bean id="StudentRepository"
       class="dataMapping.repositories.StudentRepository">
    <property name="Session">
      <ref bean="SISSession"/>
    </property>
  </bean>

  <!-- Student domain object is injected with a StudentRepository object -->
  <bean id="Student" class="domain.domainModel.Student">
    <property name="repository">
      <ref local="StudentRepository"/>
    </property>
  </bean>
</beans>
```
### Figure A2. Sample Hibernate Session Configuration

**Filename:** sis-.hibernate-cfg.xml  
**Purpose:** Hibernate configuration file to provide the data source connection information to the Hibernate Session objects. This example excerpt shows configuration for a PostgreSQL SIS DB.

```xml
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.url">URL to SIS DB here</property>
        <property name="connection.username">SIS DB username</property>
        <property name="connection.password">SIS DB password</property>
        <!-- Other properties that can be set for the datasource connection -->
        <property name="hbm2ddl.auto">update</property>
        <property name="connection.pool_size">20</property>
        <property name="hibernate.jdbc.batch_size">20</property>
    </session-factory>
</hibernate-configuration>
```

### Figure A3. Sample Hibernate Mapping File

**Filename:** Student.hbm.xml  
**Purpose:** Hibernate mappings configuration file provides the mapping of a database schema to the domain objects. This excerpt shows an example of mapping from a Student domain object to a STUDENTS table in a SIS DB. The example assumes that the Student domain object contains properties: studentID, birthDate, firstName, lastName, and that the STUDENTS table contains columns: student_id, birth_date, first_name, last_name.

```xml
<hibernate-mapping>
    <class name="domain.domainModel.Student" table="STUDENTS">
        <id name="studentID" column="student_id" />
        <property name="birthDate" type="date" column="birth_date" />
        <property name="firstName" column="first_name" />
        <property name="lastName" column="last_name" />
    </class>
</hibernate-mapping>
```