CATALYST
Volume 4, No. 1 (November 2009), 35-44

An Architecture Proposal for Academic Software in Adventist Universities

Omar Otoniel Soto Romero and Germán Harvey Alférez Salinas

Omar Soto is a computer science engineer and project manager of the finance system at Montemorelos University, Mexico, and is currently doing an MSc in Computer Science at Montemorelos Univeristy.

Harvey Alferez is a computer science engineer and lecturer at the undergraduate and graduate levels as well as Co-ordinator of the Research Department at the Faculty of Engineering and Technology, Montemorelos University, Mexico. He is also an adjunct research associate at Asia-Pacific International University.

Abstract

The Seventh-day Adventist Church (SDA) has a large number of universities world-wide. These institutions have in common some relative urgencies to establish standard software to facilitate their academic processes. Some of them buy academic systems which do not satisfy their needs completely. Another option they have is to contract outsourcing companies to develop software to fulfil theirs needs. However, this option is expensive and may require a lot of time to materialise. With a repository of academic software components, every Adventist university will be able to develop its own academic software in a fast way, with high quality and complete requirements.

Introduction

Around the globe, different efforts have been made to build repositories of components in such a way components could be acquired from a common library to be implemented in any software system (Bertoa & Vallecillo, n.d; Henninger, 1996; González, 2004).

In order to understand the concept of software components, it is a good idea to give a practical example with Lego blocks. Lego consists of colourful interlocking plastic bricks and accompanying array of gears, mini figures, and various other parts. A Lego brick can be assembled and connected in many ways, to construct a wide diversity of items like: vehicles, buildings, and even working robots. Anything constructed can be taken apart again, and the pieces used to make other objects (Alférez, 2009b; Wikipedia, n.d.).

Most Lego pieces have two basic components: studs on top and tubes inside (see Figure 1). In software components, studs can be compared with export interfaces because they offer services and information through them. Tubes in Lego pieces can be compared with import interfaces because software components often require services and information from other components (Alférez, 2009a). Component-based software engineering produces real value to business while reducing costs and time in the development effort (Bertoa & Vallecillo, n.d; González, 2004).

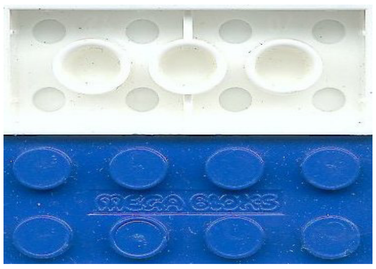


Figure 1. Lego Studs and Tubes

Once developed and tested, software components can be stored in a common place called 'repository of components'. A repository is a storage site for objects of some sort (Hagen, 2007). In other words, a repository stores information about an organisation's assets. It can store information, support multiple ways of looking at the same data, store in-depth documentation, support for versioning and change control, generate name conventions, etc. (Mullins, 2002).

Seventh-day Adventist universities (see: Department of Education of the Seventh-day Adventist Church, 2008) form a great study field to implement a repository of academic components. Universities with a reduced budget for software development will be blessed in such a way that they will be able to build their own software reusing high quality components. Institutions with more resources will support the repository of components with their own components, so it will grow dynamically.

The software architecture plays a critical role in the design of the repository of academic components. An architecture can be seen as the game box in Figure 2 (Alférez, 2009a). It explicitly defines the shapes of the figures that can be inserted in the holes. However, it does not define the colour of each component, allowing the insertion of, for example, triangles of different colours.

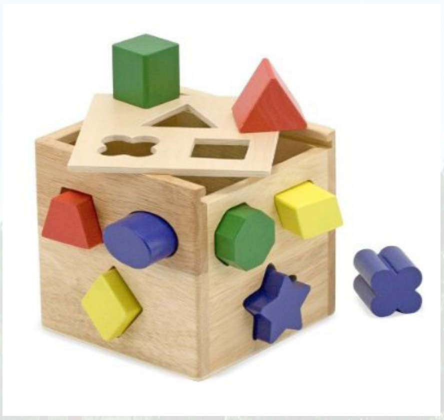


Figure 2. Software Architecture as a Game Box

The following paragraphs compare architecture-related concepts with a game box:

1. *Interfaces:* They are bridges to connect different components. The bridges are inside the game box to let figures connect among themselves.
2. *Components:* Each figure that can be inserted into the cube represents a component. A component can be defined as 'an independently deliverable piece of functionality providing access to its services through interfaces' (Brown, 2000). Components are independent units that can be assembled to build larger systems (Kroll & Macisaac, 2006).
3. *Architecture:* The cube itself is the architecture (Alférez, 2009a). It is easy to comprehend how components depend on a well-designed architecture to be able to fit in the right place and play the right role with the correct functionality. In others words, an architecture defines quality attributes that the software system will satisfy (Bass, Clements & Kazman, 2003).
4. *Repository of Components:* It is a place where different architectures (game boxes) can be stored, shared and used in different software products (Alférez, 2009a).

Quality attributes are such a critical success element in software systems (and of course in software architectures), that Microsoft Research has created an entire new operating system based only in one quality attribute: dependability (Hunt, n.d.).

It is not enough to build software-intensive systems to perform the functions they were designed to do. They must also be accurate, reliable, predictable, secure, and quick to support the volume of use expected (Brown, 2000). These quality attributes are the basis of software architecture.

According to a study made by the Software Engineering Institute (SEI), 2005 has emerged as 'The Golden Age of Software Architecture' (Shaw & Clements, 2006). It means that research on software architecture is mature enough with tools and researchers world-wide. It is the time to tell IT people in the Adventist Church to immerse in this knowledge to build software systems.

The remainder of this paper is structured as follows: the benefits of creating and using software architectures and components; the SEI's Attribute-Driven Design method (ADD) which helps to determine the critical quality attributes for the repository of academic components for Adventist universities; and the ADD method to create the architecture for the repository of academic software components for Adventist universities. Conclusions are given in the final section.

Benefits of Software Architectures and Components

Even when the software industry has been around for a while and software engineering has evolved over the past years, there is a lack of control and predictability in many software projects. Surveys show the difficulties that organisations have in predicting software costs, in identifying suitable target technologies to use during software development and maintenance activities, and in gaining visibility and control throughout the software lifecycle (Brown, 2000). The answer to these problems is found in well-defined software architectures.

Important institutions have done extensive research to determine how important software architecture is in quality software systems (Bergey et al., 2009). They have found that there are non-functional requirements such as security, reliability, disponibility, modifiability, availability, performance, usability etc. that can be reached with well-defined and managed architectures. In other words, architecture is critical to the realization of many quality attributes in a system (Bass et al., 2003).

Software components are created according to defined architectures. The component-based software development approach has intrinsic advantages such as: to increase software quality, especially evolvability and maintability; to increase the productivity of software development through the reuse of building blocks that have been developed in earlier efforts or by other parties; to use existing components rather than developing components from scratch; and to enable concurrent development of components, shortening the development time, and thus time to market (Vliet, 2008).

Even when designing and implementing reusable components is 10% to 25% more expensive in average than in non-reusable components, advantages are superior (Hamilton, 1999).

Attribute-Driven Design

A detailed analysis must be done to successfully determine which quality attributes are critical for the repository of academic components for Adventist universities. This analysis can be made by following the steps defined by the SEI's method called Attribute-Driven Design (ADD) (Bass et al., 2003). ADD defines a software architecture that bases the decomposition process on the quality attributes the software has to fulfil. The ADD is composed of the following steps:

1. *Choose the module to decompose:* The decomposition typically starts with the system, which is then decomposed into subsystems, which are further decomposed into sub-modules.
2. *Refine the module according to the following steps:*
   a. Choose the architectural drivers from the set of concrete quality scenarios and functional requirements. Architectural drivers are the combination of functional and quality requirements that 'shape' the architecture or the particular module under consideration. The drivers will be found among the top-priority requirements for the module.
   b. Choose an architectural pattern that satisfies the architectural drivers.
   c. Instantiate modules and allocate functionality from the use cases and represent using multiple views.
   d. Define interfaces of the child modules. An interface of a module shows the services and properties provided and required.
   e. Verify and refine use cases and quality scenarios and make them constraints for the child modules
3. *Repeat the steps above for every module that needs further decomposition.*

Add Method Application in the Repository of Academic Software Components for Adventist Universities

Because of the short available space in this document, the following sections present the results of the first iteration when applying the ADD method on the proposed repository of academic software components for Adventist universities:

1. *Module to decompose:* The repository of academic software components for Adventist universities.
2. *Refine the module:*

   a. Choose the architectural drivers from the set of concrete quality scenarios and functional requirements: This analysis can start with a set of motivations to build a repository of academic software components for Adventist universities. It is a set of them (Curtis, 2009):

   - **Efficiency:** To produce more in less time for less money as a result of having a collection of starting points for creative works.
   - **Consistency:** To get a predictable experience in design efforts as a result of applying components and leverage past design decisions.
   - **Memory:** A component library implies a central and common destination to record and refer

to all design decision.

- **Portability:** To produce artifacts within a common framework that can be transitioned, reused, and shared.
- **Vocabulary:** To establish and promote common understanding.
- **Authority:** To provide a more formal and credible resource to make design decisions, prioritize efforts, and refer to conventions in an open and accessible way
- **Predictability:** To have a starting point to approximate to breadth and impacts, and discuss with others in a concrete way.
- **Collaboration:** Components can be a way to trigger conversations, share knowledge, and learn together.

From the motivations defined above, the following *architectural drivers* were identified for the repository of academic software components for Adventist universities:

- The repository of academic software components must be always available (7x24) for developers in Adventist universities.
- The repository must have a well-designed search capability to let developers find a specific component according to their current needs.
- The repository must let browse components by categories, tags, download statistics, ratings, etc.
- The repository must have version management capabilities.
- The repository must let browse components by version.
- Users of the repository must be able to upload and download components according to their privileges and access policies.
- The repository will store components that can be applied to the Model-View-Controller (MVC) pattern. Therefore, any uploaded component must fit into one of the three main layers proposed by this pattern.
- Access to the repository must be correctly validated by security levels according to the kind of user who is requesting access. The hierarchy of user roles is: administrators, architects, recruiters, mentors, developers, and guests. Users of the repository must be able to upload and download components according to their privileges.

b. Choose an architectural pattern that satisfies the architectural drivers: Appendix 1 summarises the mapping between the architectural drivers that were found in step 2.a with tactics to reach the quality attributes.

Table 1 summarises the mappings between the architectural tactics that were found in Appendix 1 with the architectural patterns that fulfil the quality attributes. Figure 3 shows the initial architecture of the repository of academic software components.

Architectural Tactics	Architectural Patterns
Redundancy and echo/ping	Distributed computing: The execution of application logic occurs in multiple locations (Wood & Olson, 2003).
Separate user interface from the rest of the application	MVC Model: In this architecture pattern, the application is divided in three layers called Model, View and Controller. The *model* maintains the application data and the business rules. The *view* has to understand the particulars of the display technology. The *controller* translates user interaction with the view into actions to be handled by the model (Prem et al., 2003).
Authenticate, authorise, data confidentiality and data integrity	- Credentials and authentication: Access control credentials are special pieces of secret information that entitle a user. This may be a password (Wallingford, 2005). - Policy-Based Security Model: Specifies in detail the rights of a user to access system resources (Litwak, 1999). - Access Control Lists (ACL): Provides a basic level of security for accessing a network (Paquet, 2009).

Table 1: Architectural Patterns to Fulfil the Architectural Tactics

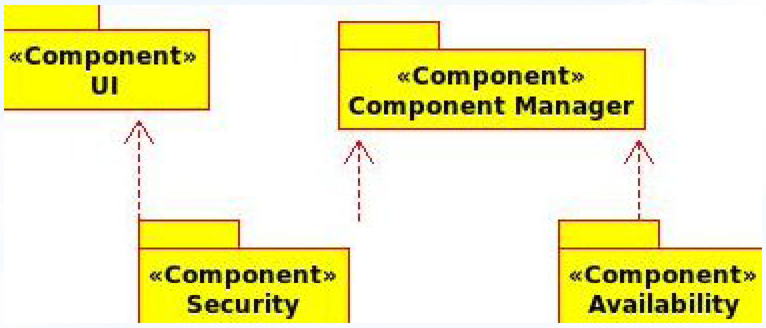


Figure 3. Initial Architecture Model

In Figure 3 there are four architecture components. Each one of them represents certain functional and non-functional attributes:

- **Security:** The whole security functionality is embedded here.
- **Availability:** This module guarantees that the repository of software components will always be available for all developers in Adventist universities.
- **Component Manager:** It has the whole functionality and business logic to allow users to upload/download academic components, update the version of every component, generate releases, etc.
- **UI:** The user interface.

c. Instantiate modules and allocate functionality from the use cases and represent using multiple views: Figure 4 shows the use case model which delimits the context of the repository of academic components for Adventist universities. Appendix 2 shows how functionality, represented by use cases, is allocated in the components instantiated in Figure 5.

Each stakeholder has one or more views of the architecture. A view is responsible to show the information that fulfils one or more viewpoints (Clements et al., 2002). Appendix 3 defines stakeholders' concerns about the repository of academic software components.

After the first iteration of the ADD, the initial architecture model has suffered some changes as shown in Figure 5. ADD steps must be followed iteration by iteration until the architecture model is detailed enough and every stakeholder is able to understand the architecture.

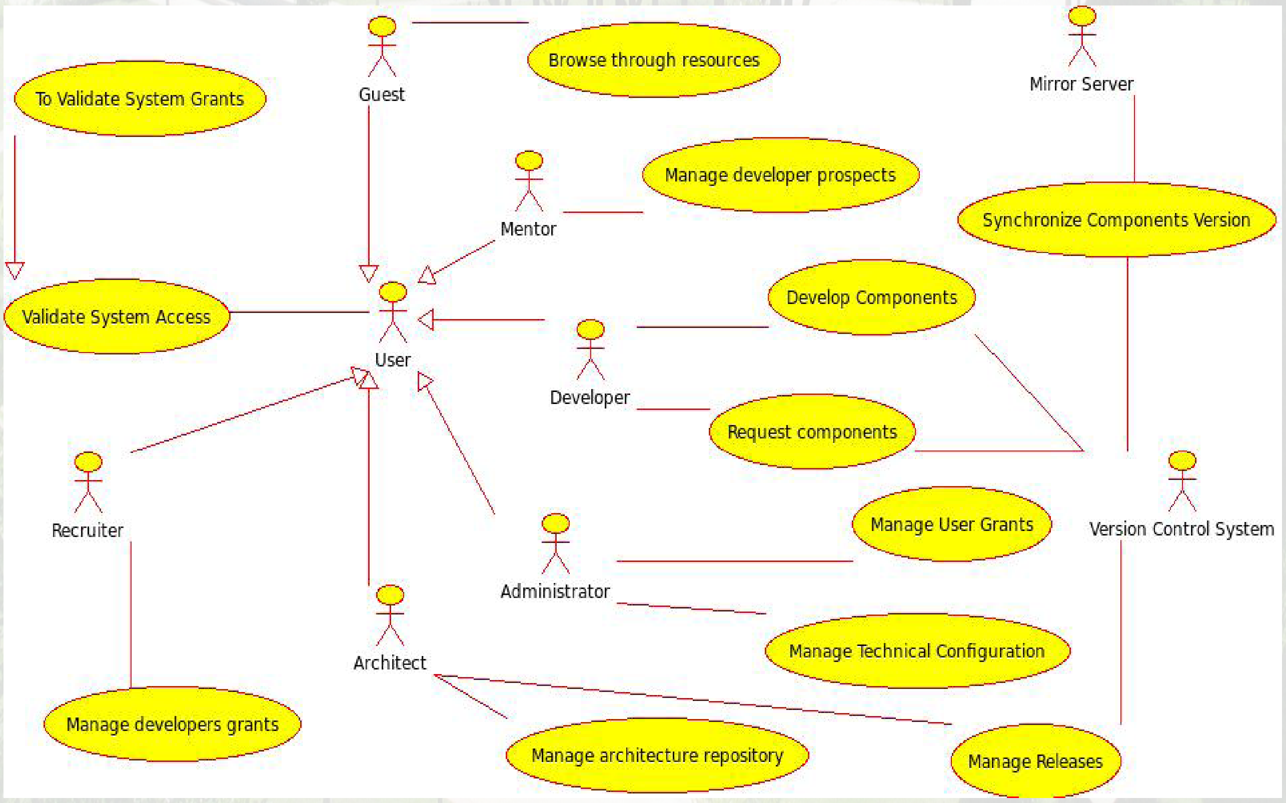


Figure 4. Use Case Model for the Repository of Academic Components

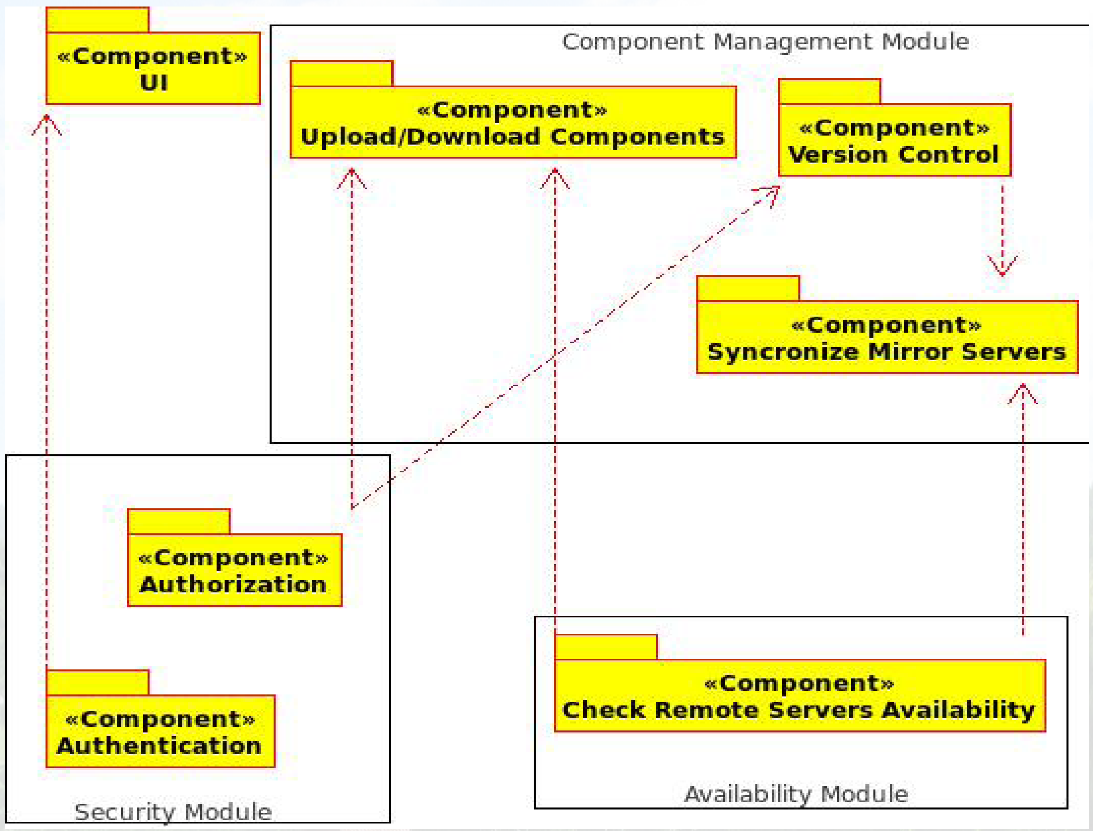

«Component» UI
Component Management Module
«Component» Upload/Download Components
«Component» Version Control
«Component» Syncronize Mirror Servers
«Component» Authorization
«Component» Authentication
Security Module
«Component» Check Remote Servers Availability
Availability Module

Figure 5. Resulting Architecture in the First ADD Iteration

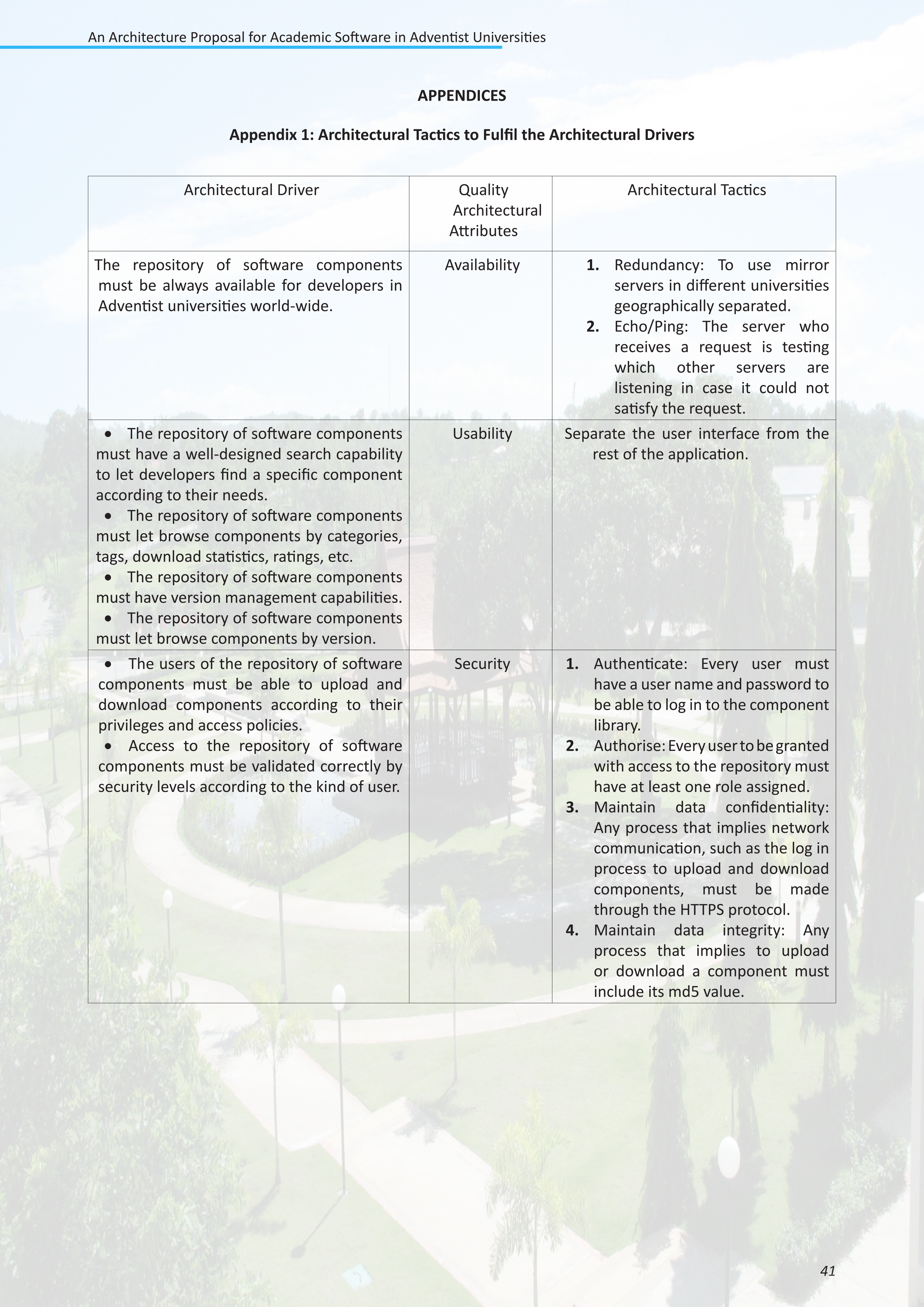
Conclusions

Management of software architectures is not just an academic matter. It is a well-defined software development technique that must be implemented before any other activities in software development. This paper has shown that an architecture is solidly founded on the critical goals and business objectives, and a system is strongly tied to its architecture.

The architecture guarantees that quality attributes will be preserved through the development cycle to reach the final goal: to create high quality software to fulfil the stakeholders' requirements and that can evolve with the organisational needs.

This paper showed the architecture of a repository of academic software components for Adventist universities. This repository will be a great blessing to all Adventist universities, not only because it will promote the creation of new software, but because it will be a great research field. Besides, this repository will promote distributed teams to work together world-wide in components to fulfil the current needs in our institutions.

Finally, the proposed repository is a starting point to promote high quality standards in the software development process and in academic products in all Adventist universities.



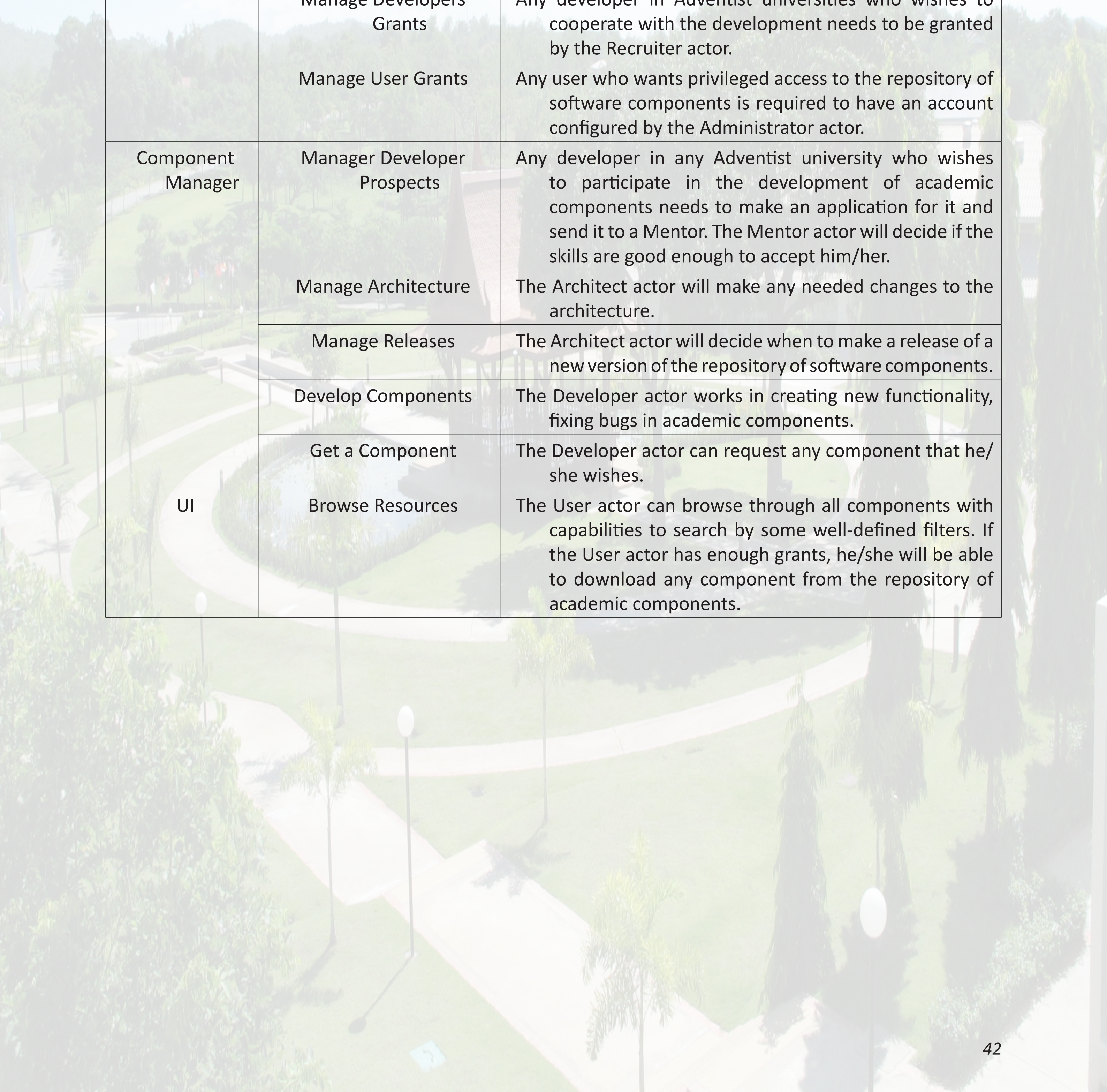
APPENDICES

Appendix 1: Architectural Tactics to Fulfil the Architectural Drivers

Architectural Driver	Quality Architectural Attributes	Architectural Tactics
The repository of software components must be always available for developers in Adventist universities world-wide.	Availability	**1.** Redundancy: To use mirror servers in different universities geographically separated. **2.** Echo/Ping: The server who receives a request is testing which other servers are listening in case it could not satisfy the request.
• The repository of software components must have a well-designed search capability to let developers find a specific component according to their needs. • The repository of software components must let browse components by categories, tags, download statistics, ratings, etc. • The repository of software components must have version management capabilities. • The repository of software components must let browse components by version.	Usability	Separate the user interface from the rest of the application.
• The users of the repository of software components must be able to upload and download components according to their privileges and access policies. • Access to the repository of software components must be validated correctly by security levels according to the kind of user.	Security	**1.** Authenticate: Every user must have a user name and password to be able to log in to the component library. **2.** Authorise: Every user to be granted with access to the repository must have at least one role assigned. **3.** Maintain data confidentiality: Any process that implies network communication, such as the log in process to upload and download components, must be made through the HTTPS protocol. **4.** Maintain data integrity: Any process that implies to upload or download a component must include its md5 value.

Appendix 2: Functionality of the Architectural Modules

Architectural Module	Use-Case	Brief Use-Case Description
Availability	Synchronise Components' Versions	To synchronise the latest version of the repository of software components in every mirror server in Adventist universities around the world.
Security	Validate System Access	Every user must be identified with a valid user name and password to get access to the repository of software components.
	Validate System Grants	Every user who has been authorised in the repository must have enough grants to be able to download/ upload and develop components.
	Manage Developers Grants	Any developer in Adventist universities who wishes to cooperate with the development needs to be granted by the Recruiter actor.
	Manage User Grants	Any user who wants privileged access to the repository of software components is required to have an account configured by the Administrator actor.
Component Manager	Manager Developer Prospects	Any developer in any Adventist university who wishes to participate in the development of academic components needs to make an application for it and send it to a Mentor. The Mentor actor will decide if the skills are good enough to accept him/her.
	Manage Architecture	The Architect actor will make any needed changes to the architecture.
	Manage Releases	The Architect actor will decide when to make a release of a new version of the repository of software components.
	Develop Components	The Developer actor works in creating new functionality, fixing bugs in academic components.
	Get a Component	The Developer actor can request any component that he/ she wishes.
UI	Browse Resources	The User actor can browse through all components with capabilities to search by some well-defined filters. If the User actor has enough grants, he/she will be able to download any component from the repository of academic components.



Appendix 3: Levels of Detail Stakeholders Require for Each Architecture View

	Module Views		*C&C Views*	*Allocation*	*Other*
Stakeholders	Layered View	Decomposition View	Client-Server View	Work Assignment View	Security View
Administrator	d		d		d
Architecture	d	d	d		d
Recruiter	s	s	o	d	
Mentor	d	d	o	s	
Developer	d	d	o		
Guest					
'd' – detailed; 's' – some details; 'o' – overview; 'x' – any detail					

Works Cited

Alférez, G..H.
2009a 'Ideas Related to AdventistForge and Software Integration in the Seventh-day Adventist Church'. Technical Report of the Faculty of Engineering and Technology, Montemorelos University (COMP-021-2009).

2009b 'The Power of Free and Open Source Software'. Presented at the Global Internet Evangelism Network Forum (GiEN) 2009, Orlando, FL. http://gien.adventist.org/schedule

Bass, L., Clements, P., and Kazman, R.
2003 'Software Architecture in Practice'. (Second Ed.). Addison-Wesley Professional.

Bergey, J., et al.
2009 'U.S. Army Workshop on Exploring Enterprise, System of Systems, System, and Software Architectures'. Software Engineering Institute, Carnegie Mellon.

Bertoa, M. F., and Vallecillo, A.
(n.d.) 'Atributos de Calidad Para Componentes COTS'. Retrieved 14 October 2009 from: www.lcc.uma.es/~av/Publicaciones/02/AtributosCalidadCOTS.pdf

Brown, A. W.
2000 *Large-Scale, Component-Based Development*. Prentice Hall.

Clements, P., et al.
2002 *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional.

Curtis, N.
2009 *Modular Web Design: Creating Reusable Components for User Experience Design and Documentation*. New Riders.

Department of Education of the Seventh-day Adventist Church.
2008 'Seventh-day Adventist Schools, Colleges and Universities'. Retrieved 15 October 2009 from: http://education.gc.adventist.org/colleges.html.

González, R.
2004 'Repositorio de Metadatos de Componentes de Software Para PYMES Colombianas'. Retrieved 14 October 2009 from: http://pegasus.javeriana.edu.co/~comp/.

Hagen, W. V.
2007 *Ubuntu® Linux® Bible*. John Wiley & Sons.

Hamilton, M.
1999 *Software Development: Building Reliable Systems*. Prentice Hall.

Henninger, S.
1996 'Supporting the Construction and Evolution of Component Repositories'. *ICSE* 96, 279–288.

Hunt, G.
(n.d.) 'Episode 88: The Singularity Research OS with Galen Hunt | Software Engineering Radio'. Software Engineering Radio. Retrieved 15 October 2009 from: http://www.se- radio.net/podcast/2008-03/episode-88-singularity-research-os-galen-hunt.

Kroll, P., and Macisaac, B.
2006 *Agility and Discipline Made Easy: Practices from OpenUP and RUP*. Addison-Wesley Professional.

Litwak, K.
1999 PURE Java™ 2. Sams.

Mullins, C. S.
2002 *Database Administration: The Complete Guide to Practices and Procedures*. Addison-Wesley Professional.

Paquet, C.
2009 *Implementing Cisco IOS Network Security (IINS): (CCNA Security exam 640-553)* (Authorized Self-Study Guide). Cisco Press.

Prem, J., et al.
2003 *BEA® WebLogic Platform 7*. Sams.

Shaw, M., & Clements, P.
2006 'The Golden Age of Software Architecture: A Comprehensive Survey. Institute for Software Research International'. Carnegie Mellon University, Pittsburgh, PA, USA.

Vliet, H. V.
2008 *Software Engineering: Principles and Practice*. John Wiley & Sons.

Wallingford, T.
2005 *Switching to VoIP.* First Ed. O'Reilly.

Wikipedia
(n.d.) 'Lego'. Retrieved on October 14, 2009, from http://en.wikipedia.org/wiki/Lego.

Wood, B., and Olson, J. D.
2003 *PowerBuilder® 9: Internet and Distributed Application Development*. Sams.