Chapter I

INTRODUCTION
CHAPTER I- INTRODUCTION

1.1 GENERAL INTRODUCTION

The increased involvement of Information and Communication Technology (ICT) in all aspects of education has enabled the teaching-learning process with better educational outcomes. The proliferation of internet has created huge potential for e-learning systems to offer flexibilities to learners to access the digital content independent of time, space, and proximity (Miller, & Hutchens, 2009). A Learning Management System (LMS) is an e-learning platform that manages different educational and administrative aspects of an online teaching-learning process. It facilitates delivery of learning materials, sharing of resources, monitoring and guiding learning, administration of users’ activities, and assessment and evaluation of learners’ performances within an e-learning environment. In last two decades, LMSs have become a standard tool for running training programs both in academia (universities, schools, etc.) and in industry (employee training programs, end user’s training etc.). From a technical standpoint, an LMS is primarily a server-based software program that interfaces with a database containing information about learners, courses and content. However, the constant and rapid growth of LMS software and their worldwide acceptance in educational institutes and organizations has generated an increasing demand for different features and functionalities within LMS, which makes LMS a complex system. For any successful implementation of LMS, we have to consider several factors to compare and evaluate different approaches and techniques during all the Software Engineering (SE) process (Rego et al., 2007). Therefore, it has become necessary to extend and adapt the conventional methods of SE in order to analyze, design, implement and maintain LMS applications (Sarasa, 2013).

However, selecting appropriate SE methodology for LMS development is a challenge for software engineers for different reasons. For example, individual requirements of diverse stakeholders – learner, teacher, author, institutional management, and accreditation bodies, etc., are to be handled effectively in the Software Development Life Cycle (SDLC) (Gupta et al., 2013). Such requirements often use multiple abstraction levels and can have hidden conflict of interests. Depending solely on Natural Language (NL) for the specification of requirements and guidelines indulges problems like mix of abstraction level and
traceability of the requirements between levels. Moreover, Compliance of a new LMS with learning technology standards involves verification of the design artifact with NL written conformance rules. Since software design use models like Unified Modeling Language (UML), Data Flow Diagram (DFD), Entity Relationship (ERD), the challenge for conformance testing is to model the conformance rules in accordance with the design model. In addition, factors that influence some of the major quality attributes like correctness, scalability, conformance to standards, and reduction of costs or time of LMS are largely different from the factors that influence quality of common business-oriented software applications and information systems. Unfortunately, most of the SE approaches for LMS development typically follow same methodologies as used for non-educational software systems (Calvo, 2003). Consequently, LMS developments often result in delays, incomplete software and faulty modules, and even could fail to meet the objectives of educational institutes. Therefore, there is a considerable scope for LMS development processes to be improved, especially if different SE methodologies are combined to supplement each other in the development process.

This thesis endeavors to find out a suitable SE methodology for LMS development. This chapter presents an overview of the LMS domain and some of the SE methods and tools used in this work. This chapter spans over two sections. In the first section, we discuss about different modules of LMS and their features, and some of the e-Learning standards applicable for LMS. In the second section, we first explore the importance of RE, software design, requirement verification and software quality in the context of the LMS development. Then, we briefly describe some of the SE methods used in this work, namely Domain modeling using Conceptual Class diagram and Ontology, Conceptual Graph, and Vienna Development Method-Specification Language (VDM-SL).
CHAPTER I- INTRODUCTION

SECTION–A

1.2 LEARNING MANAGEMENT SYSTEM (LMS)

LMS is an e-learning software for delivering, tracking and managing online training and education (Oneto et al., 2009). Several vendors have come up with different LMS products over years. Variety of LMSs ranges from systems managing entire teaching-learning process to few web pages for displaying e-learning contents over the Internet. LMS not only serves the aspired students, in many instances, corporate training departments purchase LMS to facilitate training programs for its employees. LMSs can be broadly classified into two categories: propriety (commercial) and open source. Some of the popular propriety LMS software are Blackboard (2010), Joomla (Rahmel, 2009), and Saba Learning Suite (2010). Eckstein (2010) has shown in a review that, most of the proprietary software are based on Microsoft .NET and/or Java technologies whereas most of the open source systems are based on Linux, Apache, PHP and MYSQL (LAMP). Some of the popular open source LMS are Atutor (2009), Moodle (2009), ILIAS (2009), Sakai Project (2010), and Claroline (Bennett, 2011).

Design and development LMS is a wide area and its success lies in the correct blending of different strategies from educational theories, cognitive theories, pedagogies, instructional theories and most importantly- software technology. This thesis focuses on the SE approaches of LMS development with primary focus on RE and design methodologies. Therefore, it is prerequisite to identify features of the core modules of LMS, applicable learning technology standards on LMS, and compare how different features are available within the popular LMS software. We have considered some of the popular open source LMS software like Atutor (2009), Moodle (2009), Sakai (2010) and ILIAS (2009) to identify the core modules.

1.3 LMS MODULES

LMS has come across a long path since the early days of displaying only static course contents. Most of the contemporary LMSs are web-based applications that
support dynamic course creation, dynamic content delivery, interactive communication between users, learners’ feedback based adaptation, and personalized learning. Despite of having varied features, most of LMSs have a common set of modules, as followed:

1.3.1 **Authoring:** This module supports creating, editing and delivery of learning contents by authors. It includes two different types of authoring options. The first one allows offline creation of contents using author’s preferred editor (Word processor, Power point, pdf, Dreamweaver, FrontPage, Flash etc.) and then online uploading and integration of the contents. The other approach is completely online that facilitates form-based authoring for content creation. Authors can search the existing repository and re-use any learning content or can set links to existing resources. Authoring modules of many LMS have additional option to develop learning content in multiple languages.

Some of the common features of authoring module -

- Creation, modification, and deletion of Learning Contents
- Uploading and linking of Learning Contents
- Searching learning content repository using metadata and keywords
- Support for authoring in multiple languages
- Support for import/export of learning materials

1.3.2 **Personalized learning:** This module allows learner for a customized view of the learning contents based on her personal choices and preferences in different issues like content displaying, language, learning style, level of hardness, assessment technique etc. Learner can set the pace, style and level (basic or advanced) of learning. The LMS then presents the course content in a customized way to the learner.

Some of the common features of personalized learning -

- Learner centric learning (individualized)
- Setting preferences of the pace, style, and level
- Support to achieve individual learning goals
• Support for tailoring learning plans and content delivery that fit individual learner
• Suggestions based on learner’s performance

1.3.3 Content management: This module controls the storing and structuring of learning contents in the form of Reusable Learning Objects (RLO), within a database called Learning Object (LO) repository. It manages revisions, modifications, and ensures that content is indexed for easy retrieval and reuse. An RLO is a piece of digital instructional material, which is modular, independent, non-sequential and transportable and can be used for various educational purposes, in different settings. Therefore, management of metadata of LO is particularly important for this module.

Some of the common features of Content Management -

• Providing searchable library of reusable content
• Manage contents storing and version controlling
• Launch and track contents under course
• Collaboration with Synchronous Learning Tools
• Managing Metadata

1.3.4 Course management: This module manages creation, deletion and updating of courses. A course is formed by assembling a collection of relevant RLO with a single learning objective. It is a sequence of multiple learning objects that are bundled together. Sometimes courses are also considered as a collection of lessons where a lesson can consist of multiple learning objects (Balatsoukas, 2008). The selection of appropriate course for and by a learner is guided by either the administrator or a system agent. A course catalog should contain all the relevant information about the course e.g. level, duration, schedule, prerequisites, dependencies etc.

Commonly a course management module has the following features-

• Enables learners to access catalogue offerings, register and enroll in the courses.
• Support learners to select course from the course catalogue in accordance with the competency level of the learner.
• Sends notifications for changes made to the schedule
• Defines the prerequisites and dependencies for a course

1.3.5 Learner profile: This module maintains a database to store user-profile and competency level, which is used to provide personalized learning experiences to the learner. It includes two types of learner record, editable – such as personal preferences and non-editable – such as observed behavior of the learner. This module usually involves a system agent that monitors the learning sessions and keeps record of learners’ progress and failures. It also tracks and records different activities of learners like search-query, upload-downloads, feedback etc., within the course.

The following features are common in learner profiling:

• Tracks learners’ activities and progress throughout the courses
• Stores and maintains learner records in the database
• Tracks all queries, feedback, assignments, and all upload/download history by a learner
• Manages learner’s performance information over times

1.3.6 Assessment: Assessments are integral part of LMS that provides value to the learners overall learning experience. Assessment is important at the start of the course (Pre-assessment) as well as at the end of the course (Post-assessment). Assessment module provides a built-in quiz tool, which can be used to create different types of tests like MCQ, list matching, fill-in-the-blank questions. There are two types of assessment techniques – summative and formative. The goal of summative assessment is to evaluate student’s learning at the end of an instructional unit by comparing it against some standard or benchmark. On the other hand, the goal of formative assessment is to monitor student’s learning to provide ongoing feedback that can be used by instructors to improve their teaching and by students to improve their learning.
CHAPTER I- INTRODUCTION

This module commonly has the following features-

- Pre-assessments enable learners to study prerequisites for learning the course
- Post-assessments reports learner’s progress and the status of course completion
- Formative assessment for learners progress and teachers improvement
- Summative assessment for evaluation of learner’s performance

1.3.7 Administration: This module is responsible for the overall administration of the LMS. Sometimes some of the functionalities of this module are made automated by using system agents, especially in the case of intelligent Learning Management Systems (iLMS). Administrative tasks include authentication of users, managing security, analyzing learners’ performance and setting policies.

Some of the common features Administration module:

- Authentication of users, courses, resources and the registrations
- Security and configuration management
- Learners’ performance analysis
- Setting policies for content delivery, monitoring and evaluation
- An administrator is a super user of the LMS having privilege for creating, modifying, removing, assigning and monitoring almost everything within the LMS

1.3.8 Reporting tools: This module provides useful summary information about learner’s behavior, recorded during the learning process, to the administrator and instructors. Typical reports include information about learners’ progress and assessments, learning path, learners’ interaction.

Common reports of this module are-

- Report on learning path and learning activities in each learning session.
- Report on learners’ progress and assessments.
- Report on usage of course catalogues and the contents of the system.
• Report on learners participation in chatting and blogging

1.3.9 Communication: An LMS relies heavily on the communication among its users—learner to learner or teacher to learner. Therefore, communication tools come as an integral part of LMS. Communication within an LMS can be classified into two types – synchronous and asynchronous. Below we list examples of each type-

Asynchronous communication includes
   • Mail
   • File exchange
   • Discussion forum
   • Blogging
   • Notifications

Synchronous communication includes
   • Text/video chat
   • Video conferencing
   • Application sharing
   • Virtual Classroom

1.4 E-LEARNING STANDARDS

E-learning standards refer to a common set of rules to enable interoperability between applications, and to provide uniform communication guidelines for the design, development, and delivery of learning objects (Uta, 2007). E-learning systems like LMS, if developed without following any standards, often lead to the fragmentation of incompatible technologies. As a result, we get systems that cannot cooperate and content that cannot be reused (Varlamis & Apostolakis, 2006). Although Internet has provided a cost effective infrastructure to facilitating delivery of LMS, without standards, sharing and integrating e-Learning contents between vendors and tools would be a costly and time-consuming process. Therefore, one major concern of LMS development is its compliance to e-learning standards. While some technical standards are intended to ensure that components of e-learning are inter-operable and re-usable. Other
standards are focused on content—providing a common means of cataloging content so that user can easily find it and make content accessible to learners. Some of the established standards in e-learning are Aviation Industry CBT (AICC), Instructional Management System (IMS), Sharable Content Object Reference Model (SCORM) and Learning Technology System Architecture (LTSA). These standards not only help to ensure e-learning effectiveness but also promote different qualities like interoperability, reusability, manageability, accessibility and durability of the e-learning software.

Different standards address different aspects of e-learning. AICC proposes a Communication Protocol (HACP) to facilitate communication between the course content and the LMS. SCORM focuses on a convenient package procedure, which makes uploading courses to the LMS as simple as uploading a .zip file. SCORM-conformant courses communicate with an LMS by calling methods of a JavaScript object using an API adapter. The IMS standard focuses on bridging the gap between high level and machine interpretable descriptions of the learning system with help of an Educational Modeling Language (IMSGC, 2003), which typically describes the entire pedagogical methodology of a course. On the other hand, LTSA standard specifies a framework for higher-level abstraction of the system. LTSA provides a guideline for developing system architecture of new LMSs. However, all the above standards usually fit to the needs of specific purposes in the development process and no single standard is adequate to support all the aspects of LMS development (Friesen, 2005). In this work, we have restrained our focus to SCORM and LTSA.

1.4.1 Sharable Content Object Reference Model (SCORM)

SCORM (ADL, 2004) is developed by the Advanced Distributed Learning (ADL) group collaboratively with other standards bodies, government entities, and educational organizations around the world as an initiative for e-learning standards. The worldwide adoption of SCORM by the software industry and academic institutions has made it most popular standard for learning contents and packaging. At its simplest, it is a model that references a set of interrelated technical specifications and guidelines designed to meet high level requirements for learning contents and systems. A Shareable Content Object (SCO) is a
collection of one or more learning assets that represent a single learning resource. A SCO represents the lowest level of granularity of a learning resource that is tracked by an LMS. SCORM describes a Content Aggregation Model (CAM) and a Run-Time Environment (RTE) for LOs to support adaptive presentation of content based on different criteria such as learner objectives, preferences, and performance. RTE is responsible for communication between SCO and the LMS.

SCORM sees the LMS as a web-based application for delivering instruction. It does so under the assumption that anything that can be delivered by the web can be easily integrated with the instructional settings with help of accessibility and network communications. Thus, selection of the web itself as a universal delivery medium eliminates much of the development work once needed to consider for the technology platform. By building upon existing web standards and infrastructures, SCORM frees developers to focus on effective learning strategies.

SCORM specification is a collection of three related sub specifications, which are content packaging, runtime and sequencing.

**Content Packaging:** The Content Packaging section specifies how content should be packaged and described. It is primarily based on XML technology. SCORM specifies that content should be packaged in a self-contained directory or a ZIP file called a Package Interchange File (PIF). The PIF must always contain an XML file named imsmanifest.xml (the “manifest file”) at the root. SCOs are combined into a tree structure that represents a course. The manifest contains an XML representation of the tree, information about how to launch each SCO and (optionally) metadata that describes the course and its parts.

**Run-Time:** The Run-Time section specifies how content should be launched and how it communicates with the LMS. It is primarily based on JavaScript technology. The run-time specification states that the LMS should launch content in a web browser, either in a new window or in a frameset. Once the content is launched, it locates a JavaScript API provided by the system to exchange data within the LMS. A list of data elements is available that can be written to and read from the LMS. Some examples are – status of the SCO (completed, passed, failed, etc), the score the learner achieved, a bookmark to track the learner’s location, and the total amount of time the learner spent in the SCO.
Sequencing: The Sequencing section specifies how the learner can navigate between parts of the course (SCOs). Sequencing rules are represented by XML within the course’s manifest. It determines which navigational controls the LMS should provide to the user (previous/next buttons, a navigable table of contents, etc). It randomly selects a different subset of available SCOs to be delivered on each new attempt (to enable test banking, for instance).

SCORM technical standard is focused on the following important software qualities:

i). Availability: Makes educational content available for remote locations.

ii). Adaptability: Permits the orders to become adapted to each individuals and organizations.

iii). Economic: Decreases the time and costs of production of the learning contents

iv). Durability: Educational content should not have need to redesign, reprogramming or reconfiguration to be updated with technology evolution.

v). Interoperability: Educational content developed in one location with one set of tools can be used in another place with other tool sets.

vi). Reusability: A part of the educational content can be used in several applications or texts.

1.4.2 Learning Technology System Architecture (LTSA)

LTSA specifies a high level architecture for a wide range of systems such as LMS, Computer Based Training (CBT), Computer Assisted Instruction, Intelligent Tutoring System (ITS), etc. commonly known as learning technology (Farance & Tonkel, 2003). It provides a framework for understanding existing and future systems and promotes interoperability and portability by identifying critical system interfaces. Figure 1.1 depicts a schematic representation of LTSA framework.
The five-layered LTSA a standard specifies a high-level architecture, which is pedagogically neutral, content-neutral, culturally neutral, and platform-neutral. Each level is more detailed than its previous abstract level along with Level-I to Level-V. Below we provide a brief description of the levels.

LTSA layer I:

This section specifies learner’s interaction with environment i.e., how learner interacts with content, teacher and other learners as part of a learning experience. This level concerns with learner's acquisition, transfer, exchange, formulation, discovery, etc. of knowledge and information through interaction with the environment.

LTSA layer II:

This section deals with Human-Centered/Pervasive features. From an information technology perspective, human learners impose significant design constraints because of their strengths and weaknesses. From the learner's perspective, she may perceive the learning technology as a tool to pursue her own learning strategy. Considering human nature has pervasive effect on system design, this layer narrates an iterative process for learning.

Loop

Multimedia delivery to learner
CHAPTER I- INTRODUCTION

Feedback on assessment and corrective actions taken

Records Database: Learner’s performance history

Directs different learning styles and strategies based on knowledge library

Interaction between Learner and system to negotiate on learning styles

End

LTSA layer III:

This section deals with identification of the components of the system. It describes the processes, flows, and stores of the systems architecture. Processes are described in terms of boundaries, inputs, functionalities, and outputs. Flows are described in terms of connectivity (one-way, two-way, static connections, dynamic connections, etc.) and the type of information across the flow. Stores are described by the type of information stored, accessed by search, retrieval, and updating methods.

The LTSA system components are:

- Processes: Learning, Evaluation, System Coach, Delivery
- Stores: Records Database, Knowledge Library
- Flows: Behavior, Assessment, Performance (past, present, future), Indexes (query, content, and locator), Learning Content, Multimedia, Learning Style

LTSA layer IV:

This section identifies different perspectives of stakeholders in learning technology systems. The main aim of this section is to show how each perspective is relevant, and included within the framework. Each perspective is represented by a subset of the LTSA components.

The following stakeholders’ perspectives are identified in the LTSA:

- Author’s Perspective
- Learner’s Perspective
CHAPTER I- INTRODUCTION

➢ Teacher’s Perspective

LTSA layer V:

This section identifies the main operational components that are common to many learning technology systems – such as API, coding, protocol, interchange specification, process, store (databases), information flow, and human interface.

It describes operational and system components have the following properties:

➢ One operational component (e.g., HTTP) might be used by several system components (e.g., records database, knowledge library)
➢ One system component (e.g., delivery) might use several operational components (e.g., HTTP, HTML, MPEG)

LTSA Standard Conformance:

Any LMS design that wants to conform to LTSA standard must fill a form ‘Pro-Forma Implementation Conformance Statement’ to claim conformance to the LTSA Standard (Farance and Tonkel, 2003). The conformance entails that a conforming implementation must identify all the specified LTSA components. The LTSA framework is used as a reference to measure conformance of the new LMS with LTSA, by identifying the following entities, processes, and flows within the new system:

➢ Identification of LTSA learner entity
➢ Identification of LTSA behavior data flow from learner entity to assessment
➢ Identification of LTSA evaluation process
➢ Identification of LTSA assessment data flow from evaluation to coach
➢ Identification of LTSA learner information data flow between evaluation and learner records
➢ Identification of LTSA learner records data store
➢ Identification of LTSA learner information data flow from learner records to coach
➢ Identification of LTSA learner information data flow from coach to learner records
CHAPTER I- INTRODUCTION

- Identification of LTSA learning preferences between learner entity and coach
- Identification of LTSA coach process
- Identification of LTSA query control flow from coach to learning resources
- Identification of LTSA learning resources data store
- Identification of LTSA catalog information data flow from learning resources to coach
- Identification of LTSA locator data flow from coach to delivery
- Identification of LTSA delivery process
- Identification of LTSA locator control flow from delivery to learning resources
- Identification of LTSA learning content data flow from learning resources to delivery
- Identification of LTSA interaction context data flow from delivery to evaluation
- Identification of LTSA multimedia data flow from delivery to learner entity

1.5 SOME OF THE POPULAR LMS SOFTWARE

There are many open source LMS software currently available that are being adopted by different Institutions to provide online courses. Below we discuss and compare features of three popular open source LMS software – ATutor (2009), Moodle (2009), and ILIAS (2009).

1.5.1 ATutor

ATutor is an open source web-based LMS software developed and maintained by the Adaptive Technology Resource Centre (ATRC) at the University of Toronto. It supports institutional management including content creation, learner’s activity management and tracking learning objectives in a social networking environment, using various online communicative tools. ATutor provides good documentation, ease of installation, and strong potential for development. It is one of the few LMS to support LO repositories. ATutor scores highly for openness and is written in a modular format. Although the user interface may not seem intuitive to many,
the overall functionality is good wide open and modular, and the development is committed to standards. ATutor is very strong on standards and can import external content in IMS/SCORM format. It has many features that are rated highly for usability, including accessibility for learners with disabilities. The system is also install-friendly and receptive to new language versions.

1.5.2 Moodle
Moodle is an open source LMS software designed using known pedagogical principles to help the educators to create effective online learning communities. Moodle is one of the most user-friendly and flexible open source courseware products available. It has excellent documentation, strong support for security and administration, and evolves towards IMS/SCORM standards. The key to Moodle is that it is developed with both pedagogy and technology in mind. Moodle has a robust development and user community. Great with languages although some development may be needed for robust handling enhanced tracking features. Still, this program receives a high recommendation. Moodle's architecture sets an excellent foundation, following good practices of low coupling and high cohesion, which the other LMSs fail to achieve. This supports a system that is simple, flexible and effective, and easily accessible to developers.

1.5.3 ILIAS
ILIAS is an open source LMS for developing, sharing and organizing knowledge and education and training activities. At its core, it supports learning content management (SCORM 2004 compliance) and tools for collaboration, communication, evaluation and assessment. A general characteristic of ILIAS is the concept of a Personal Desktop and Repository. While the Repository contains all content, courses, and other materials (structured in categories and described by metadata), the Personal Desktop is the individual workspace of each learner, author, tutor, and administrator. ILIAS is an object-oriented LMS software. In ILIAS, each module is implemented as a class, which exposes an arbitrary interface. However, ILIAS has a complex architecture with tight coupling that is hard to work with and debug.

Below we present two tables, Table 1.1 and Table 1.2, which portray a comparison of different features of these three open source LMS vendors.
Table 1.1: Comparing some generic features of the LMS vendors

<table>
<thead>
<tr>
<th>Issues</th>
<th>Atutor</th>
<th>ILIAS</th>
<th>Moodle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>No modularity</td>
<td>Complex architecture; Tight Coupling</td>
<td>Simple Architecture; high cohesion, low coupling;</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>Good support for multiple application server and data server</td>
<td>Support for multiple application server; not multiple database servers; caching is not possible</td>
<td>High scalability with multiple application server and data server; caching is supported partially</td>
</tr>
<tr>
<td><strong>Adaptive presentation</strong></td>
<td>Styles can be altered by pre-existing themes; No support for layout resizing</td>
<td>Limited support for layout resizing; user’s themes are allowed</td>
<td>Good support for customizable header-footer with centralized theme control</td>
</tr>
<tr>
<td><strong>Authorization</strong></td>
<td>Limited roles; hard-core implemented, cannot be extended</td>
<td>Flexible support for roles using role template and also user interface</td>
<td>Limited roles; hard-core implemented, cannot be extended</td>
</tr>
<tr>
<td><strong>Authoring</strong></td>
<td>Only teachers can post</td>
<td>Students and teachers can post</td>
<td>All types of user can post; content editing by teachers and authors</td>
</tr>
<tr>
<td><strong>Interoperability</strong></td>
<td>Cross browser support; standard database and flat file backup;</td>
<td>Weak cross browser support; standard database and flat file backup; good in Linux platform, poor in Windows</td>
<td>Wide portability, cross browser support; standard database and flat file backup; XML based backup</td>
</tr>
<tr>
<td><strong>Standard Support</strong></td>
<td>IMS (export only)</td>
<td>IMS (import only) and SCORM</td>
<td>IMS (Full) and SCORM</td>
</tr>
</tbody>
</table>
### Table 1.2: Comparing different communication methods of the LMS vendors

<table>
<thead>
<tr>
<th>Software</th>
<th>Whiteboard/ Video Services</th>
<th>Discussion Forum</th>
<th>File exchange/ Internal Mail</th>
<th>Online Journal Mail</th>
<th>Real Time Chat</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATutor</strong></td>
<td>A Java based instant messaging and white board tool, AComm, is used in ATutor. It has full keyboard functionality, allowing users to type text description within the white board.</td>
<td>ATutor provides room for users to develop a network of contacts, set up a network profile, join interest groups, discuss and share pictures.</td>
<td>Inbox available for users to send and receive private mails from other users. Sent messages are saved and have provision for messages to be exported.</td>
<td>Instructors can upload and manage course related files. A pop up file manager opens alongside the content editor or test questions editors and enables files to be linked with ease into content or test items.</td>
<td>ATutor chat tool is based on XMPP protocol that supports efficient data transfer. It facilitates one to one messaging and multi user chat among course members.</td>
</tr>
<tr>
<td><strong>ILIAS</strong></td>
<td>ILIAS has no whiteboard feature. However, it supports online video conference feature that instructors and students use to exchange ideas and information.</td>
<td>Discussion forums are available over web for ILIAS users worldwide to exchange information and views on the LMS.</td>
<td>It has an internal email system available. Mails can be sent to individuals or a group of participants.</td>
<td>It offers multiple options for importing and creating content for journal. It provides platform for creating and publishing of the journal.</td>
<td>It offers a JAVA based chat system that has to be installed before use.</td>
</tr>
</tbody>
</table>
CHAPTER I- INTRODUCTION

<table>
<thead>
<tr>
<th>Software</th>
<th>Whiteboard/Video Services</th>
<th>Discussion Forum</th>
<th>File exchange/Internal Mail</th>
<th>Online Journal Mail</th>
<th>Real Time Chat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moodle</td>
<td>It has wonderful features like Skype whiteboard. This interactive feature can add learners and instructors dynamically at a virtual classroom.</td>
<td>There are four active forum types within the Moodle community for helping students and teachers exchange ideas.</td>
<td>It provides an easy way for teachers to present materials to their students. Uploaded files require compatible software to open it at the client end.</td>
<td>It has a journal module available, that provides a text area where students can type in and upload. It can also be revisited and updated.</td>
<td>Real time chat is available enabling participants to have a real time synchronous discussion in a Moodle course.</td>
</tr>
</tbody>
</table>

SECTION-B

1.6 SOFTWARE ENGINEERING METHODOLOGIES

The importance of SE in the development of different types of LMS has increased with the advancement of the LMSs from being mainly research projects to mature software products (Martens & Harrer, 2006). From the perspective of SE, LMS development must carefully handle some challenging issues. One major challenge is to avoid conflict of interest between the requirements of diverse set of users like student, teacher, author, administrator etc. – e.g. some learner prefer to study online because they want or need individual flexibility but the administrator/coordinator/teachers promote collaborative learning, because they believe it improves learning performance and solve problems faster. Another challenge comes from blending of instructional theories, learning theories, pedagogies, and learning technology standards within the software’s functional features. Because of such blending, requirements are often stated in mixed level of abstractions,
which makes them less sustainable and traceable. For example, ensuring conformance of an LMS design elements to LTSA rules (standard, written at a higher level of abstraction using natural language) is a challenging job. Similarly, establishing traceability of the requirements between higher-level abstraction at requirements specification and lower level abstraction at software design is a challenging task that entails a systematic, preferably model based approach. On the other hand, multiple methods are required to model different aspects of LMS requirements. For example, Users and their interaction with the system can be easily modeled with UML diagrams at a high level of abstraction, whereas functional modeling can be more precisely detailed with introduction of formal methods like VDM-SL. An Architectural Design (AD) can bridge the gap between abstractions of requirement specification and detailing of design artifact. While UML is the de-facto standard in detailed design, it can also be used to model software architecture. Alternatively, software architecture can be modeled using any specialized modeling tool like ACME. However, use of multiple methods (graphical, formal, and semi-formal) in requirement analysis, architectural design and detailed design, brings in many new challenges to the development process. Particularly, ensuring traceability of the requirements at different levels and checking consistency between design components become harder jobs. Below, we give an overview of the RE, AD, and detailed design phases with focus on the challenges of LMS development.

1.7 REQUIREMENT ENGINEERING

The Requirement Engineering (RE) process plays a pivotal role in ensuring the overall success of SE. Correct identification of the requirements and understanding requirements from different perspectives – e.g. Stakeholders, Functional modules, and Quality attributes – is critical to the success of LMS development (Maguire, 2002). Hence, it is important to classify/group/map the requirements into these perspectives for a better understanding of their impact on the system and to trace the requirements in the design and implementation phases. In addition, a systematic RE approach is required for analyzing software requirements at different levels of abstraction, which allows for the traceability of
the requirements between different levels, and makes them verifiable early in the project.

Normally, in LMS development, elicitation and specification of requirements from different perspectives and at different abstraction levels are usually practiced in NL. However, a model based requirement analysis, with systematic illustration of requirements, alleviates the design phase. The advantage of building such model over the specification written in NL is that the later can avoid ambiguities better than NL sentences and it alleviates the verification of the design against the requirements specification. UML is the most popular modeling language of RE in a wide range of software applications due to its higher understandability and the ease of use. UML is semi formal in nature and uses graphical and text-based notations, which makes it convenient to use but at the same time, it compromises with the level of preciseness in defining the requirements. Conversely using a formal modeling technique such as VDM can model a precise and complete view of the system’s requirements with the help of VDM-SL. However, using VDM-SL alone to model all types of requirements at every level of abstraction is not a feasible solution as it is laborious, expensive and has hardly any tool support. Moreover, it results in a complex specification, which is not only difficult to analysis but also requirements are hard to trace between different levels of abstraction. Therefore, integrated use of formal and semi-formal methods is more useful in RE than using any one method. This would also make it easier for applying formal methods in design and verification of the requirements, which come next in the development phase.

1.8 SOFTWARE DESIGN

Software design is an iterative process through which requirements are translated into a blueprint for constructing the software (Pressman, 2005). A software design document narrates detailing of the software data structures, architecture, interfaces, and components. The design process begins at a high level of abstraction and subsequent refinements lead to design representations at much lower levels of abstraction. According to classical Software life cycle process (ISO95b), (Tremblay, 2001), software design phase consists of two major
activities that fit between software requirements analysis, and software coding and implementation: i) software architectural design and ii) software detailed design.

1.8.1 Software architecture design

Software architecture design is quickly emerging as one of the main artifacts of software development (Mikkonen et al. 2004). The IEEE Standard 1471 (Jen & Lee, 2000) defines architecture as: “the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”. Software architectures are essential engineering artifacts used in the early design process of software development. From the perspective of SE, it deals with the design, study and description of the structure of software systems. This structure is usually composed of a set of components and their relationships via connecting pathways. It represents a global view of the system by specifying various aspects of a system - gross organization into components, protocols for communication and data access, functionality of design elements, etc. - at a high-level of abstraction (Mateescu, 2004). Therefore, software architectures are particularly important for modeling complex systems. Software architecture design is also helpful for developers to understand the design issues involved, reducing the risk factor (Capilla & Martinez, 2004). Throughout the development process, software architecture can serve as a baseline against which the various stakeholders analyze, understand, build their decisions, and evaluate the software (Gloahec et al., 2010). Software architecture design, representing system's initial outlines, supports the analysis and evaluation of system's quality in the early phase of development.

The main advantages of AD are the following (Ratcliffe et al., 2004):

- It provides a framework to satisfy the high-level requirements
- It provides basis for the detail design and the implementation of the system
- It supports the analysis of the quality of the design with the architecture
- It promotes reusability
CHAPTER I- INTRODUCTION

- It facilitates the communication among stakeholders

1.8.1.1 Architectural design techniques

There are three main streams of AD techniques: ad-hoc, UML and ADL. At times, new notations for AD of a domain have been proposed on ad-hoc basis to keep things simple and to avoid overburden complexity of standardized notations in that specific domain (Schmidt, 2008). However, ad-hoc notations lack formality, preventing architectural descriptions from being analyzed for consistency or completeness and for being traced back and forward to actual implementations. UML has a widespread acceptability among the practitioners for modeling software system at different phases of development. Using UML to describe software architectures could bring lower development cost, better tool support and interoperability, as well as lower training costs (Goulão & Abreu, 2003). UML2.0 in particular is supported with some new constructs for design architectural elements. The component-connector style is commonly used for an abstract view of the AD. Components are used to describe the physical, deployable pieces of a system. UML components can expose their functionality through interfaces and ports. UML connectors use roles to participate in interactions as varied as procedure call, event broadcast, database queries, and pipes (Garlan et al. 2002& 2000(a)). However, UML lacks from sufficient expressiveness to produce a formal description of the architecture. On the other hand, Architecture description languages (ADL), such as Acme (Garlan 2000(b)), C2 (Medvidovic et al., 1996), Darwin (Magee et al., 1995), Rapide (Luckham, 1995), and Wright (Allen, 1997) are specialized formal languages for supporting modeling and reasoning of software architectures. They also support verification and simulation of software architecture models (Navarcik, 2005). Among the non-proprietary ADLs, Acme is pioneer for several reasons. It is a mature, general purpose, second generation ADL that uses all the common architectural constructs. It is also an architectural interchange language with the goal of providing a framework to move information between different ADLs (Shaw & Clements, 2006).
1.8.1.2 Software architecture for LMS

LMS vendors are competing these days in a race of implementing as much functionality as possible, the design and development of LMS is largely focused on satisfying certain functional requirements, overlooking and underestimating the quality requirements (Avgeriou et al., 2003). Since quality attributes in a system depend profoundly on its architecture, methodology related software AD is equally important in the development process (Avgeriou et al., 2003). AN LMS architecture description is a visual, holistic view of the system at a higher level of abstraction. In order to evaluate the quality attributes of the architecture, an ADL based description of the system is useful to check for conformance with a standard. Certain quality features such as performance, availability, portability, and usability of LMS can be easily achieved if the design of the system conforms to the recommendation of LTSA standard. LTSA specifies a checklist, known as conformance rules (Farance & Tonkel, 2003), which must be satisfied by an implementing LMS to claim conformance with LTSA. The set of conformance rules entails identification of different entities and flows within the system (section 4.2). Since LTSA framework specifies a high-level description about the system, intuitively a high-level design artifact like software architectural design is more suitable for the conformance checking than a detailed design. Therefore, the challenge to SE methodology is how LMS architecture should be designed in order to ensure conformity with LTSA standard.

1.8.2 Software Detailed Design

In the design of the system, ideally the AD is followed by the detailed design. It is the process of refining and expanding the preliminary design of a system or component to the extent where the design is sufficiently complete and ready for coding (ISO/IEC 24765) (Tremblay, 2001). In common practice, software detailed design process is a seamless transformation of the software requirements, as described in the SRS document, into a form suitable for coding and implementation. The output of the designing phase is a software design artifact that demonstrates how to fulfill the requirements and guides the implementation of the software. UML is the de facto method in software design among the practitioners. UML notations are widely accepted and used for visualizing models
in different phases of developments, from early abstraction of requirements to
detail design of them (Booch, 2005 and Rayner, 2005). UML diagrams are also
useful in communication among different stakeholders. The detailed design
process elaborates the use case diagrams of RE phase into UML class diagram,
Activity Diagram, Sequence Diagram, etc. Similarly, a formal design of the
system (using VDM-SL) extends the requirements written in VDM-SL in the
specification by a more detailed-description in the design phase. Two important
aspects of the design verification are to check consistency and continuity among
different UML diagrams of the design artifact, and to ensure traceability of the
requirements between SRS and design artifact.

1.8.3 Software interface design

The popularity of internet has made the users so convenient with the web-based
software that LMS users these days are not attracted only by a rich set of
functional features, instead their choice for selecting an LMS is highly influenced
by the presentation part of the software, i.e. how the user’s interface looks like.
The interface design for web-based LMS is a complex task due to the event driven
nature of such applications. Tasks like sequencing of activities and invoking the
functionalities cannot be predefined since the order of their occurrence within the
system is determined by the users of the application. The common challenge for
SE is to incorporate the interface requirements within the functional design by
mapping them with the functional requirements. Moreover, some of the interface
requirements can appear as an integral part of the functional requirements. For
example, a requirement for a dropdown list or an Ajax based suggestion in a text
box is inseparably attached with the input method of the functional requirement.
In addition, this type of compound requirements (interface-requirement attached
with a functional requirement) often has different parts of a requirement stated at
different levels of abstraction, which makes them even harder to include in the
design. However, interface requirements, if handled separately from functional
requirements, bring many hazards to the later integration stages. It may lead to
problem related to traceability, inconsistency, and missing interfaces. Therefore,
interface requirements must be included within the detailed design.
1.9 REQUIREMENTS VERIFICATION

Requirement Verification is the process of confirming that the design and development of the software product addresses to the documented requirements. Such verification can be performed in various ways like simple checklist, reviews and inspections, prototyping, model-based (formal) verification, or any combination of these. The verification process spans throughout the SDLC – early design, detailed design, and implementation – to find out any unaddressed requirement, which can result in rework and significant cost overruns. The verification process involves checking the requirements specification and the successive refinements descending from it – the product design, detailed design, code, database, and implementation in order to keep these refinements consistent with the requirements specification. Thus, in general the verification activities begin in the design phase and conclude with the acceptance test on the end product.

Formal methods have been successfully used for the verification of software systems over years (Fuxman, 2003). However, commonly formal methods for verification are used in the later phases of the development process, e.g., at the detailed design and coding phases. Incompatibility between the constructs of formal specification languages and other conventional methods (semi-formal and ad-hoc) of requirements specification is one of the main reasons for that. However, using formal methods in RE and design phase, enables model based checking as a viable solution to perform requirement verification at early stages of the design process (Hoyos, 2012).

1.10 SOFTWARE QUALITY ATTRIBUTES IN LMS

Next, we discuss about how some of the requirements for LMS influence the quality attributes.

**Correctness:** Correctness of LMS can be defined as the adherence of the software to the requirements specification. Ensuring traceability of the requirements in the development process implies that no requirement is left unaddressed. LMS users should be able to interact with the learning content using the software. The
displayed content should always come from a repository of verified and trusted learning resources and should be reusable in different courses. The search engine should find the contents on a valid input query. The evaluation process should be able to assess learners’ performances correctly. Questions should be set dynamically and randomly form a question bank that covers the entire course. There should be a feedback system, which could be used to rectify incorrect content and features of the system.

**Reliability:** Reliability measures the level of risk and the likelihood of potential failures of the software. Correctness intuitively increases reliability. The learning contents should be properly sequenced and linked, so that the learner can navigate smoothly through the learning path. The omission of dead links and unsupported media increases reliability. The support of minimum software and hardware requirements should be satisfied. Flexibility of the LMS to update courses based on the feedback from the users also increases reliability.

**Robustness:** Robustness is defined as the ability of a software product to cope with unusual situation. The system should be able to behave reasonably in an unexpected circumstance like failure at client or server end. The session management and maintaining log files are crucial in this regard because it helps the system to identify the learner’s last state of action so that it can again start over the session from that same point. Since the learning content should come from the repository, it is possible to restore a learning session instead of restarting from the beginning.

**Portability:** It refers to how well the software can adopt to a changing environment. The LMS software should be portable from one system to another. The role of XML based metadata management is immensely important in this regard. Learning contents should come in a package with a manifestation file in order to ease portability.

**Usability:** Usability is the ability of a software to offer its interfaces in a user friendly and elegant way. The learning system should be easy to use by its users. A proper sequence of topics with correct linking is the prime objective. The learner should be able to search the right content. The learner should be able to
download and the author should be able to upload the contents easily. Users should have interface to interact with teachers and other learners.

**Reusability:** In general, reusability refers to the use of existing assets in some form within the software product or development process. In LMS, assets are mainly the learning contents that should be reused in different courses. Reusable Learning Object (RLO) can be reused or referenced including multimedia content, instructional content, software, data, or events for any e-Learning software (Duval, 2002). The contents should be always verified and required metadata should be stored before publishing so that it could be reused. Learning contents should come in packages which are easier to export and import across different courses.

**Maintainability:** Maintainability is the ability of software to adapt to changes, improve over time, and correct the bugs through maintenance. The LMS software should be easy to maintain, proper packaging and use of metadata are crucial in this regard. Modular architecture of the LMS with high cohesion and low coupling among the modules promotes maintainability of the software. Learners’ feedback on her experience in the learning path along with all the interactions and communications between the users should be properly recorded in order to increase maintainability.

**Availability:** Availability of software refers to the ability to keep functioning despite of problems. From the learners' point of view, contents should be always accessible irrespective of the situations. It means that, learning content should be properly displayed under any environment that has adequate multimedia support. Proper linking and synchronization among the different contents increases availability of the software. The individual personalization (Lu, 2004) feature also increases availability. The contents should be always available when searched, even with bad queries (wrong spelling in query). Responsive web page technology enhances availability by adjusting the display according to the user’s device e.g. laptop, smart phones, PDA etc.

**Interoperability:** It is the ability of a system to work with external systems by communicating and exchanging information without any special effort on the part of the user. The learning contents should be sharable to different LMS software
that follows the same standardization process. The metadata is used for exchanging information between the systems.

**Conformance with standard:** The LMS software should conform to some of the established standards in e-Learning. For example, the learning content and its packaging should comply with SCORM while the overall architecture should comply with specifications of LTSA.

### 1.11 SOME OF THE METHODS/TOOLS FOR REQUIREMENT ENGINEERING AND DESIGN

Formal methods of SE are mainly used in the requirement analysis, design, and testing phases of the developing system. Despite of its tremendous advantages on correctness and stability of the system, application of mathematical methods in the development and verification of software is very labor intensive, and thus expensive (Plat et. al., 1992). Therefore, it is not feasible to rely only on formal methods for all aspects of the development; instead, it is more cost effective to support the conventional development methods (graphical and semi-formal) of SDLC with the assimilation of different formal methods for different purposes. In the context of LMS development, the pertinent issues are – how to model the natural language statements at different abstraction levels, how to guide the design and development with the domain knowledge to avoid ambiguity and assumptions, how to give a formal description of the functional requirements at a higher level of abstraction, etc. Therefore, below we discuss some of popular SE tools that are commonly used to handle such issues – namely, Domain model (DM) using Conceptual Class Diagram and Domain Ontology, modeling NL written requirements with Conceptual Graph and describing functionalities with VDM-SL.

#### 1.11.1 Domain Model (DM)

Before software can be designed, we must know its requirements and requirements can be expressed properly only after we understand the domain (Bjorner, 2006). Therefore, domain knowledge plays an important role in RE and software design phases. Lack of domain knowledge leads to poor requirement
CHAPTER I- INTRODUCTION

specification and design artifact, and as a result, the developed software reflects low quality (Kaiya & Saeki, 2006). DM describes all aspects of the real world that are relevant for the design and development of software in the specified area. One of the prime objectives of domain modeling is to make all the required information about the domain readily available at the RE and software design phase (Bjorner, 2006). It provides a framework for a full analysis of requirements, which is independent of the technology of implementation. Typically, knowledge of a domain evolves collections of abstract concepts identified at different levels of abstraction (Prieto-Diaz, 1990).

The success of a DM depends on the correct identification of the components of the domain that are used for future detailing and reusability. This identification is carried out by analyzing and classifying the key components of similar systems on that domain (Meekel et al., 1997). An efficient DM therefore specifies the basic components of the domain, their organizations, understanding of the relationships among these components, and finally represents this understandings in a useful way, probably using a graphical, semi-formal, or formal method. Below we discuss two popular methods of domain modeling namely Conceptual Class Diagram and Ontology.

1.11.1.1 Conceptual class diagram

A conceptual class represents a high-level description of a real world concept. A conceptual class diagram depicts abstractions of concepts from the problem domain using classes, attributes and associations. It is a variation of UML class diagram without the use of methods. Since DM represents abstraction at a higher level where actual software classes with specific responsibilities (methods) are not defined, conceptual class diagrams are better suited than core UML class diagram for domain modeling. Conceptual class diagrams are commonly used for requirement analysis by abstracting the domain entities (Larman, 2005). Domain experts generally pull out the class abstractions from the available descriptions of the system. In conceptual class diagrams, domain entities are represented by rectangular boxes, which are joined by associations represented with connecting lines that shows the name and cardinality of the relationship. An association shows semantic relationship between two or more classes. For a conceptual model
with ‘n’ classes, there can be \( n^*(n-1) \) possible associations. Each association describes a role and multiplicity. Multiplicity are shown using symbols 0,1, and * (for many). E.g.–

- 1..* means one or many
- 0..1 means optional one
- 1..4 means a range from 1 to 4

Figure 1.2 illustrates the conceptual class diagram for the well-known Point of Selling (POS) System [adopted from (Ryder, 2013)].

![Conceptual class diagram](image)

Figure 1.2: Conceptual class diagram [adopted from: (Ryder, 2013)]

1.1.1.2 Domain ontology

Ontology is a formal representation of definition of the types, properties, and interrelationships of the entities that exists for a particular domain. Besides conceptual class diagram, domain ontology is another popular tool for domain modeling. Domain ontology is a lightweight formal method that defines a common vocabulary and relationships of the entities to aid the developers with useful information about the domain. It enables reuse of domain knowledge in a seamless way by making explicit domain assumptions. Ontological analysis
clarifies the structure of the knowledge base and helps to identify the semantic components that are involved in understanding discourse in that domain (Chandrasekaran et al., 1999). Most of the domain ontology descriptions are represented by either tree or graph, where nodes are units of knowledge (called concepts) and edges show the different types of relationships between nodes (Kaiya et al., 2010). Different categories for ontology are available based on their degree of expressivity (Roussey et al., 2011). The category of Information Ontology [Figure 1.3] is commonly represented with hierarchical relationships among the nodes in a tree structure, which is easily modifiable and scalable, or with relationship described in first order predicate logic. For modeling the requirements at different levels of abstraction, an ontology-based detailing of the entities, concepts, processes and objects in a hierarchical manner is required for ensuring consistency between the models.

![Ontology description](Adopted from: (Reimer, 2013)]

1.11.2 Conceptual graph:

Conceptual graph is a graph representation for logic based on the semantic networks of Artificial Intelligence (AI) and the existential graph theory (Sowa J.F., 1998). Conceptual graphs are popularly used techniques for reasoning, representing, and analyzing natural language semantics. From the SE perspectives, it is a model-based technique to represent NL requirements in a formal way (Delugach & Lampkin, 2001). Conceptual graph diagrams are capable to model requirements at different abstraction levels. A conceptual graph diagram can represent the relationship between two concepts without knowing the
attributes of the abstractions; hence, it becomes an automatic choice for any evolutionary requirement analysis model. Another major advantage of using conceptual graph diagram is that it can be reformulated into predicate logic using Lambda expressions. In the context of conceptual graph-theory, a lambda expression represents a conceptual graph with referents to the concepts in the graph are replaceable variables. When more than one referent has been replaced by a $\lambda$, we often use other symbols, such as $?x$, $?y$, $?z$, etc.

Lambda expressions in conceptual graph are used in two situations:

1. When defining new concept types, and
2. When defining new relation types.

For example, a NL written requirement can be represented graphically in conceptual graph as in Figure 1.4 with the equivalent Lambda expression is shown below

**Requirement:** Teacher should assesses student by examination

![Conceptual graph representing a requirement](image)

**Lambda expression:**

\[
[\text{Concept: } ?\text{Teacher}] \leftarrow \text{(agnt)} \leftarrow [\text{Concept: } ?\text{assessment}] \rightarrow \text{(obj)} \rightarrow [\text{Concept: } ?\text{Student}] \land \ [\text{Concept: } ?\text{assessment}] \rightarrow \text{(inst)} \rightarrow [\text{Concept: } ?\text{Examination}]
\]

The above example reflects a higher-level abstraction of the requirement. It shows relationship between four concepts using the relationships agent (agnt), instrument (inst), and object (obj). The lambda expression at this level shows all the nodes like teacher, student, examination, and assess as members of the class 'concept'.
In a more detailed representation, some of them like Teacher or Student itself can appear as a class.

1.11.3 VDM-SL

The introduction of formal description of requirements in the specification enhances preciseness and correctness of requirements at the requirement analysis and software design phases. The Vienna Development Method – Specification Language (VDM-SL) is a well-established formalizing tool for requirements specification (Fitzgerald et al., 2005). VDM-SL follows a mathematical model, rooted in simple algebraic theory and predicate logic. It can model system’s behavior at any level of abstraction, as required in the specification. A specification written in VDM-SL is a document that primarily contains type definitions, state variables, values, functions, and operations. VDM-SL expresses system’s behavior as logic expressions by using operations and functions. Operations differ from functions in the ability to accessing state variables; as operations may read or write to state variables, whereas functions may not.

A sample VDM-SL description structure looks like:

```
types
Some Type = ..... 
values
constantName : ConstantType = someValue
state SystemName of
attribute_1 : Type
: 
attribute_n : Type
inv mk-SystemName(i_1:Type, ..., i_n:Type) Δ Expression(i_1, ..., i_n)
init mk-SystemName(i_1:Type, ..., i_n:Type) Δ Expression(i_1, ..., i_n)
end
functions
specification of functions ..... 
operations
specification of operations ..... 
```
Operations are defined by the combination of pre and post conditions. A pre-condition is an expression over the input variables representing the restrictions that are assumed to hold on the inputs. Similarly, a post-condition is an expression that must be satisfied to achieve the output. Below we describe a requirement first in NL and then it is described formally in VDM-SL. For example VDM-SL for the requirement – “A decrement operation must decrease the temperature of the system by 1 unit, only if its current value is greater than 10” – would be:

decrement () Δ

\text{ext wr temp: Z}

pre temp > 10

post temp = temp - 1

1.12 STRUCTURE OF THE THESIS

The structure of this thesis is as followed:

Chapter I narrates an overview of the LMS domain and SE methodologies. Different LMS modules, Learning technology standards, and comparative analysis on features of some open source popular LMS software are described in the LMS-overview section. In the SE methodologies section, we give an overview on the RE phase and design phase, with focus on challenges of LMS development process. We also analyze software quality attributes in the context of LMS. Finally, we briefly describe some SE tools and techniques like domain model, conceptual class diagram, ontology, conceptual graph, and VDM-SL.

Chapter II presents the review work for this thesis. We explore issues related to SE methodologies for LMS development. Use of formal methods in collaboration with semi-formal and graphical methods in RE and Software design phases, with their pros and cons, are discussed. Techniques for traceability of the requirements at different levels of abstraction are reviewed. We also examine the scope of checking conformance of the high-level design of LMS with the learning technology standards. A discussion on how LMS could be enriched with some
additional features is also presented. Finally, we brief our findings and set the objectives of this research work.

Chapter III illustrates our work on the requirement engineering for LMS. A set of forty-two requirements is identified and analyzed. Mapping of the requirements with stakeholders and software quality attributes are depicted. A pertinent framework to incorporate different levels of abstraction in the specification is introduced. A technique for ensuring traceability of the requirements between different levels of abstraction is also described.

Chapter IV illustrates an AD of LMS that provides a high-level design of the system. Then this high-level design is checked for conformance with the LTSA standard. This chapter introduces a new meta-model based on Acme style for UML and exhibit how LMS architecture should be designed so that we can test its conformance with LTSA. A measure for the conformance level to understand the extent, to which an AD artifact conforms to the LTSA standard, is also presented.

Chapter V depicts our work on the detailed designing of LMS. It proposes a methodology to guide the SRS into an implementable design. In order to ensure consistency and traceability of the requirements, UML modeling is supported by usage of formal methods like VDM-SL and conceptual graph. The proposed verification technique is based on a three dimensional verification matrix and the traceability is ensured by matching the equivalence between conceptual graphs produced at the RE phase and the design phase respectively.

Chapter VI presents a systematic approach for analysis and design of the interface requirements that uses formal methods in a uniform way for functional as well as interface requirements. We propose some add-ons to VDM-SL syntaxes to cover the interface-requirements of a web-based application. A Finite State Machine (FSM) based verification model is built from the design to test the design with the SRS.

Chapter VII describes some new techniques to incorporate some additional features into conventional LMS. We proposed a method for developing scaffolding to support personalized learning in synchronous medium. We propose multiple approaches to construct learning path for learners- using Frequent Pattern
(FP) generation technique, using a new data structure for a tree of connected terms, and using Ant Colony Optimization Technique. The proposed adaptive features of content, sequencing, and navigations are illustrated with a prototype LMS implementation, named CompTutor.