Software Production Process Using Open Source Software

Javad K. HESHMATI ✥ 30-4-2006 ✥ 32 Pages

Dixite
Av. Louise 179, P.B.: 3, 1050 Brussels ✥ `www.dixite.com`

Preface

Briefly describes the Software Production Process based on *Open Source Software* resources. It ends with a real case study in which open source software tools, libraries and middleware are used as the building blocks of the project.

How to Read This Document

Chapter 1 gives a short background information about Dixite and Open Source.

Chapter 2 describes *roles* involved during the project life cycle as well as the produces *deliverables*

Chapter 3 discusses in detail the *project life cycle*. Each step of the cycle is divided to the following section:

- *Approach*: explains the methodology, standards and best practices that are used
- *Roles and Deliverables*: indicates the roles involved and deliverables produced by the cycle step
- *Tools and Resources*: provides software tools and resources that are used by the team.

Chapter 4 provides a real case study in which open source software tools, libraries and middleware are used as the building blocks of the project.

CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

At Dixite, we use a *process* to produce our software product. Moreover, we are commited to Open Source Software (OSS) standards, tools, libraries and middleware. By using this process in combination with OSS based resources, we make sure that our software product is

- **Correct** it behaves according to the requirement specification
- **Reliable** that is our client can depend on it.
- **Robust** it behaves reasonably even in circumstances that were not anticipated in the requirement specification. For example, when it encounters incorrect input data.
- **Maintainable** which allows for modifications and improvements with a limited amount of work.
- **Cost Effective** which means that the total cost of ownership is at a reasonable price.

In this document, we present our process and describe how we ensure the above qualities[1]. In each step of the process, a brief explanation of our *approach* is given. Where appropriate, a list of OSS, *tools and resources* are also provided. In the Last section of this document, we provide a Case Study (see section 4.1) in which we outline our process and the usage of OSS in our solution.

[1]Quality Control is also applied at each step of the process. Please refer to Dixite Quality Control Process Document for more information.

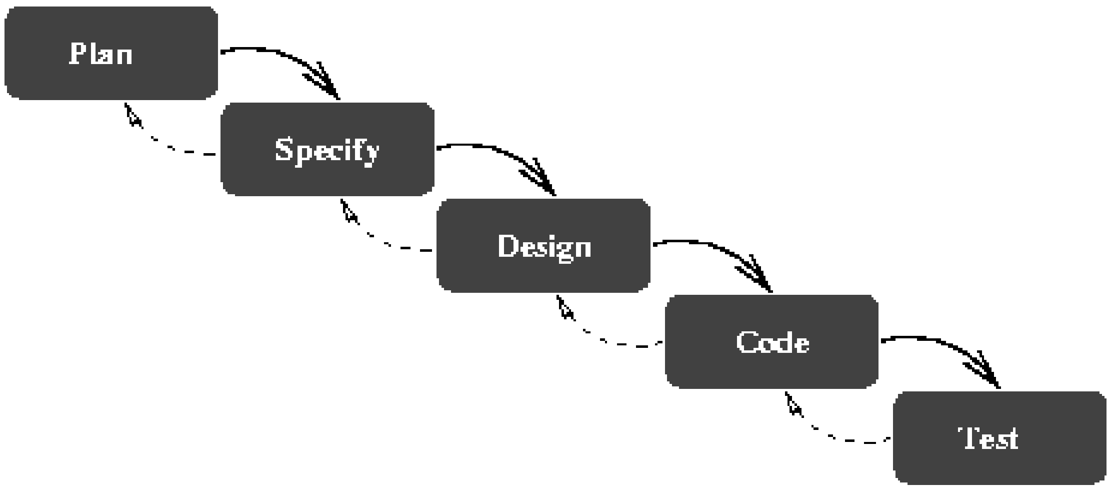

Figure 1.1: Software Production Life Cycle

1.0.1 Why Using Open Source Software and Resources ?

Total cost of ownership for OSS is often far less than proprietary software, especially as the number of platforms increases. OSS does not impose license management costs and avoids nearly all licensing litigation risks. OSS is often the most reliable software, and in many cases has the best performance. OSS often has far better security, perhaps due to the possibility of worldwide review.

CHAPTER 2

Roles and Deliverables

2.1 Roles, Skills and Techniques

This section describes each role along with its required skills and techniques.

2.1.1 Business Analyst

An experienced person in the business domain. Knows how the business operates and can answer questions about how things are done; what is stable; what is changing; what is essentially attached to each concept and term.

- *Skills*: knowledgeable in the operation of the business and its plans
- *Techniques*: none related to designing the new system

2.1.2 Requirement Analyst

Knows enough of the business to examine the users' requirements statements and enough of the technology to ask for alternative requirements when solutions looks too hard.

- *Skills*: communications skills, thoroughness
- *Techniques*: domain object modeling (find the nouns, reduce them, check business rules, apply cardinality rules)

2.1.3 Software Architect

Knows how to design the system as a whole. Responsible for major subdivisions and interfaces within the system, performance targets, ensuring functioning of the overall system. Works with the project manager (2.1.4) to prioritize and organize the project plan. Does high-level design.

- *Skills*: ability to evaluate the entire system in his or her charge; ability to make global and detailed technical decisions

- *Techniques*: System analysis and modeling, performance modeling and estimations

2.1.4 Project Manager

Knows how to gather and integrate information from all stakeholders in the project (project sponsors, architects, developers, testers, etc.) and put it together into a workable plan. Knows how to fend off "feature creep" and other hazards of running the project. Responsible for the process, with input from the other roles.

- *Skills*: motivation, observation, communication and planning
- *Techniques*: project estimation, management by team building and spirit

2.1.5 Lead Designer

Knows how to create frameworks and toolkits, knows the difference between strong and weak designs, can monitor and coach other developers without demoralizing them. Reasonably good communication skills. Designs subsystems, applying design techniques to the requirements. Typically but not always the best programmer on the team.

- *Skills*: framework design, class design, communicating with new designers, programming
- *Techniques*: domain modeling, design by client interface, design by theory building or design by intuition

2.1.6 Programmer

Knows how to design a complete and programmed set of classes from requirements and a sketched design, good programming skills.

- *Skills and Techniques*: same as the *lead designer*

2.1.7 Technical Writer

Creates the external documentation, such as the class, screen, and test specifications, and first draft user manual.

- *Skills and Techniques*: good at writing, familiar with the domain topics

2.1.8 Tester

Knows how to create and run test cases, given either the requirements document or the screen specifications. Creates system test suits that can be run repeatedly (regression tests).

- *Skills*: creating test suites
- *Techniques*: white box and black box test creation, disturbance tests , and test recovery

2.1.9 User Interface Designer

Knows how to create easy-to-use user interfaces. Know the U (UI) standards. Enforces simplicity and consistency in the UI design. May have special training in gathering feedback from users on the interface.

- *Skills*: able to learn users' work needs and habits, and able to evaluate and test the user-interface design
- *Techniques*: user-centered design and questionnaire-based testing

2.2 Deliverables

This section describes the main deliverables produces by a project life cycle. It is possible for some of the deliverables to be started and undergoing change at the same time.

2.2.1 Project Plan and Milestones

2.2.2 User Requirements Document

This includes the system purpose; the use cases, the business rules and relationships that must be preserved in the design; usability and performance requirements; and definitions of needed interfaces to other systems or subsystems. This document is produced by the *business analyst* and the *technical writer*. The purpose of this document is to communicate with the project sponsors, the user community, the external test team, and the function team over time.

2.2.3 User Interface Design Document

This document initially includes a description of the user metaphor, each screen's purpose, and the navigation among the screens. Over time, the details of each each screen added, along with details of error conditions (if any). This is produced by the *UI designer*, *Requirement Analyst* and *technical writer*.

2.2.4 Architectural Design Document

This initially includes the major partitioning of the system into subsystems, each with its purpose, responsibilities and interfaces. This is produces by the *software architect*.

2.2.5 Detailed Design Document

2.2.6 User Manuals

Describes how the users will use various parts of the system. It is produced by the entire team, reviewed by the users, and used for external test and to prepare training materials.

2.2.7 Code

This comprises the source code and the compiled, bound code for delivery.

2.2.8 Test Cases

These are the regression tests applied to every class, subsystem and total system.

2.2.9 Test and Observation Reports

A report that includes observation reports collected during testing.

CHAPTER 3

Project Life Cycle

3.1 Planning

3.1.1 Approach

Our project managers decide what objectives are to be achieved, what resources are required to achieve the objectives, how and when the resources are to be acquired. In the planning task we basically determine the flow of information, people and products within Dixite. We plan over and over as we progress. Of course, we revise our plans if necessary. Our project managers verify the project staff performances against project plans and take corrective actions when deviations occur. We use *Gantt Charts* for several purposes, including scheduling, budgeting and resource planning. We also use *XP - Extreme Programming* (Please refer to *Extreme Programming Explained* Beck) principles for overall project management.

3.1.2 Roles and Deliverables

- *Roles*: Software Architect and Project Manager
- *Deliverables*: Project Plan and Milestones

3.1.3 Tools and Resources

Software and Utilities

Apache Maven is used to split the project tasks and allocate resources.

XP website is consulted on a regular basis to keep up to date with the best practices of project management.

PMI website is consulted on a regular basis to keep up to date with the best practices of project management.

Books, Guides and Templates

Planning Extreme Programming Kent Beck.

Fundamentals Of Software Engineering Chezzi.

The Unified Software Development Process Booch The Unified Software Development Process.

3.2 Requirement Analysis and Specification

3.2.1 Approach

Understanding business objectives is essential. If requirement is misapplied it can become a drain rather than a gain for the client. In this phase our engineers identify and document the exact requirement for the system. To establish the requirements clearly and precisely, we use Unified Modeling Language (UML) use cases to document the requirements. Please refer to *The Unified Modeling Language User Guide* Booch The Unified Modeling Language User Guide. This phase produces *user manuals* and *system test plans*. Where appropriate, a *prototype* is also prepared so that the client may verify the functional requirements.

3.2.2 Roles and Deliverables

- *Roles*: Requirement Analyst and Business Analyst
- *Deliverables*: User Requirements Document, User Interface Design Document and Test Cases

3.2.3 Tools and Resources

Software and Utilities

CVS is used as document version managment tool.

DocBook and LaTeX are used as document production framework for technical writing.

Argo UML software is used to prepare requirements' use cases.

UML Resources is consulted on a regular basis to access up to-date UML specification and resources.

Books, Guides and Templates

Extreme Programming Installed Installed.

The Unified Modeling Language User Guide Booch The Unified Modeling Language User Guide.

UML Distilled Fowler.

UML In A Nutshell Albir.

User Requirement Document (URD) Template Heshmati User Requirement Document (URD) Template.

Software Requirement Document (SRD) Template Heshmati Software Requirement Document (SRD) Template.

The LaTeXCompanion Goossens/Samarin.

DocBook: The Definitive Guide Norman Walsh.

3.3 Design and Specification

3.3.1 Approach

Once the requirements for the system have been documented (refer to section 3.2), our engineers design a software system to meet them. We split the design phase into two sub-phases: *Architectural Design* and *Detailed Design*. In the *Architectural Design* we deal with the overall module organization and structure whereas in the *Detailed Design* we refine the *Architectural Design* by designing each module in details. Again we use UML (Please refer to *The Unified Modeling Language User Guide* Booch The Unified Modeling Language User Guide) to specify Arcitectural Design Document (ADD) and Detialed Design Document (DDD). Moreover, we practice the principles described in Extreme Programming (XP) (please refer to Beck) to make sure that we'll produce an open and cost-effective design.

3.3.2 Roles and Deliverables

- *Roles*: Software Architect, Lead Designer and User Interface Designer
- *Deliverables*: Architectural Design Document and Detailed Design Document

3.3.3 Tools and Resources

Software and Utilities

CVS is used as document management version control system.

Java Doc Tool, LaTeX and DocBook are used as document production framework for technical writing.

Argo UML software is used to prepare requirements' use cases.

Xfig is used to perform high level design drawings.

XAE and Xeena software are used to design XML schemas.

Books, Guides and Templates

The LaTeXCompanion Goossens/Samarin.

DocBook: The Definitive Guide Norman Walsh.

The Unified Modeling Language User Guide Booch The Unified Modeling Language User Guide.

UML Distilled Fowler.

UML In A Nutshell Albir.

Architectural Design Document (ADD) Template Heshmati Architectural Design Document (ADD) Template.

3.4 Implementation and Unit Testing

3.4.1 Approach

In this phase we produce the actual code that will be tested, packaged and delivered as the running system. The other phases (refer to section 3.2 and section 3.3) may also produce code, such as prototypes, library modules and header files but these are for use by our engineers. Individual modules developed in this phase are also unit tested before delivered to the next phase. To code the system software efficiently, we rely upon our extensive experience in addition to guides available from industry leaders.

3.4.2 Core Practices

Frequent Releases

At the end of every development iteration we plan a small release. This makes the development progress more visible and tangible. To release frequently and reliably we use our *customized built tools*.

Test Driven Development

Our programmers define and implement *unit tests* for every required feature of the system. These *programmer tests* , or *unit tests* are put together, and every time any programmer release any code to the repository, these *unit tests* must run correctly. This help out our programmers detect any possible side effect right away.

Design Improvement

While in the development phase our engineers constantly improve the system design by applying *refactoring* which emphasize on continuous design improvement.

Coding Standard

Our programmers follow a common coding standard which makes the overal software more readable and therefore maintainable.

3.4.3 Roles and Deliverables

- *Roles*: Lead Designer and Programmer
- *Deliverables*: Code

3.4.4 Tools and Resources

Operating System (OS)

We mainly develop and deploy on GNU/Linux. In particular: Debian and Redhat .

Selection Criteria When the choice of the OS is left to us, we choose the OS taking into account the following criteria:

- Reliability
- Scalability
- Performance
- Cost of hardware and software
- Availability of software
- Cost of training staff

Software and Utilities

Depending on the case at hand we use various programming languages. Some of the tools and utilities are dependent on the programming language. Therefore this section is split into:

Section 3.4.4 lists common software tools that maybe used regardless of the choice of the programming language.

Section 3.4.4 lists tools, Integrated Development Environment (IDE), servers and middleware that maybe utilized while implementing in Java.

Section 3.4.4 lists utilities and runtime environment used while developing in Perl and/or Hypertext Preprocessor (PHP).

Section 3.4.4 lists software utilities, IDE and database middleware that are used when coding Structured Query Language (SQL).

Common

Ant is used as the build tool.

CVS (Concurrent Versions System) is used as document management version control system.

Emacs is used as the general programming editor.

GNU/Bash shell and its utility programs are used as the general development and build environment.

LDAP Browser Editor (LBE) and Visual LDAP are used as the general Light Directory Access Protocol (LDAP) clients for browsing and editing.

Vim is used for manipulating configuration files.

Coding in Java

Apache Tomcatis used as the servlet container and application server.

Eclipse is used as the Java Integrated Development and debug Environment.

Open LDAP is used as the LDAP directory server.

Java 2 Platform, Standard Edition and Java 2 Platform, Enterprise Edition is used for the Java 2 Platform, Enterprise Edition (J2EE) and Java 2 Platform, Standard Edition (J2SE) compilations and runtime environment.

JBoss is used as the J2EE server.

JUnit framework is used to perform unit testing.

Jinsight is used to perform profiling and code optimization.

Argo UML software is used to keep the software design and the software code synchronized.

JEdit is also used as the Java Integrated Development and debug Environment.

Coding in Perl/PHP

Active Perl is used as Perl runtime environment on Windows platform.

Apache Web Server is used as the HyperText Transfer Protocol (HTTP) server.

PHP 4 is used as the PHP runtime environment.

Perl 5.6 is used as Perl runtime environment on GNU/Linux and Unix platforms.

Coding in SQL

MySQL Database software is used as the Relational Database Management System (RDBMS) when a free and light relational repository is preferred.

Books, Guides and Templates

Software Release and Build Guide (RBG) Heshmati Software Release and Build Guide (RBG).

Design Patterns Gamma.

Enterprise JavaBeans (3rd Edition) Monson-Haefel.

Effective Java Programming Language Guide Bloch.

Thinking in Java (2nd Edition) Eckel.

Refactoring Improving The Design of Existing Code Fowler.

The Java Language Specification Gosling.

Programming Pearls (2nd Edition) Bentley.

MySQL DuBois.

Java Design Patterns Cooper.

Programming Perl Wall.

Code Complete: A Practical Handbook of Software Construction of Software Construction.

The Practice of Programming Kernighan/Pike.

Understanding And Deploying LDAP Directory Servers Hows.

GNU Emacs Cameron

Linux Administration Handbook Evi Nemeth/Boggs

3.5 System Integration and Testing

3.5.1 Approach

All the modules that have been developed before (see section 3.4) and unit tested indivitually are put together in this phase and tested as a whole system. Tests are performed based on the system test plan produced in the Requirement Analysis phase (refer to section 3.2).

3.5.2 Roles and Deliverables

- *Roles*: Tester
- *Deliverables*: Test and Observation Reports

3.5.3 Tools and Resources

Software and Utilities

JUnit framework is used to perform unit testing.

Bugzilla is used to perform effective bug reporting and tracking.

Books, Guides and Templates

Fundamentals Of Software Engineering Chezzi

Software Release and Build Guide (RBG) Heshmati Software Release and Build Guide (RBG)

3.6 Delivery and Maintenance

3.6.1 Approach

Once the system passes all the required tests specified in the test plan (refer to section 3.2.1), it is packaged and delivered. At this stage the system enters the maintenance phase. Any modification made to the system after initial delivery are usually attributed to this phase.

3.6.2 Roles and Deliverables

- *Roles*: Project Manager. In case of change requests, other roles defined in section 2.1 maybe needed.
- *Deliverables*: User Manuals. In case of change requests, other deliverables defined in section 2.2 maybe delivered.

3.6.3 Tools

Software and Utilities

Bugzilla is used to register and followup the client *change requests* and requested features.

Books and Guides

Extreme Programming Explained Beck.

Software Release and Build Guide (RBG) Heshmati Software Release and Build Guide (RBG)

Software Installation Template (SIT) Heshmati Software Installation Template (SIT)

CHAPTER 4

Case Study

4.1 CURIA II Project

4.1.1 Client

Publication Office of the European Communities (OPOCE) and the European Court of Justice.

4.1.2 Problem

To provide *indexation* and *search* engines to index and query the jurisprudence text produced by the European Court of Justice. The system is code named as *CURIA II* and should support:

- Complex search queries including Fuzzy and Proximity.
- Term highlighting.
- Different file formats: HyperText Markup Language (HTML), Potable Document Format (PDF) and Extensible Markup Language (XML).
- Various Operating Systems: Windows NT, Windows 2000 and MacOS X.
- Internationalization, several European languages including Greek.

4.1.3 Solution

After evaluating the existing tools, libraries, middleware and their major characteristics, strengths, and weaknesses, we put together an Open Source based Solution shown in figure 4.1 on the following page. Moreover, we setup a development and staging environment using Open Source Software, as shown in figure 4.2 on page 23.

Software Architecture

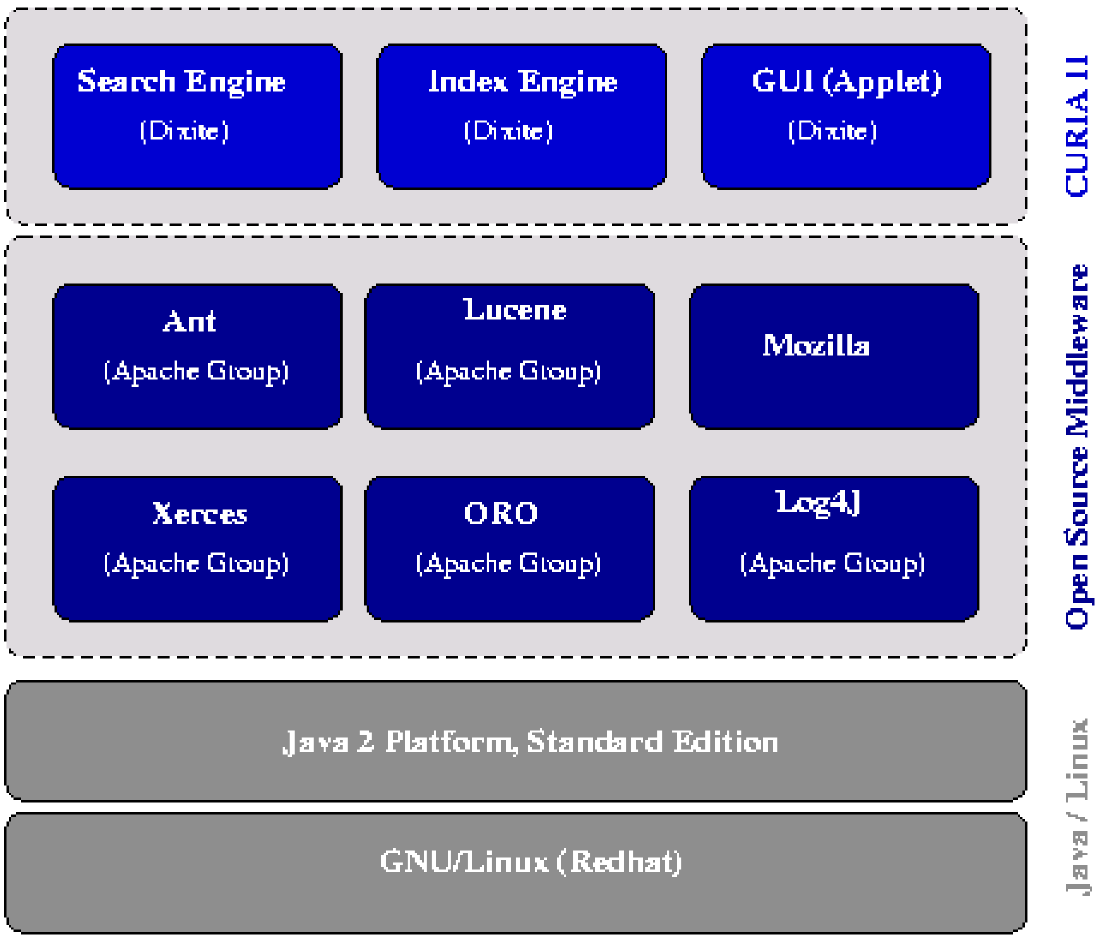

Figure 4.1: Software Architecture View

CURIA II Application To fulfill the requirement, we chose a modular design split into three main modules: Index Engine, Search Engine and a GUI Applet.

- *Index Engine* based on *Apache Lucene* we implemented the required index engine. It generates the *index* database that contains a compiled version of documents and is optimized for quick lookup for a list of documents. Indexation is performed based on a set of files in PDF, HTML and XML formats.
- *Search Engine* we used *Apache Xerces*, *Apache Lucene* to parse and build the required *search queries*. Moreover, the *Apache ORO* package was used to perform the required *Term Highlights*.

- *GUI Applet* the *J2SE* was used to implement and sign the Graphical User Interface (GUI) applet which runs in *Mozilla* and is used as the search engine front end. Using Mozilla as the Applet container, we made sure that the application behavior is consistent across various operating systems.
- *Internationalization* we used the *Internationalization* feature of the J2SE to support various European languages.
- *Error Management* to achieve a systematic error and log management, we used the *Apache Log4J* package.

Open Source Software and Middleware

- Ant is a Java-based build tool.
- Lucene is a high-performance, full-featured text search engine written entirely in Java.
- Mozilla is an open-source web browser, designed for standards compliance, performance and portability.
- Xerces is a high performance, fully compliant XML parser.
- ORO is a set of text-processing Java classes that provide Perl5 compatible regular expressions.
- Log4J is a log and error management framework.

Java Runtime on GNU/Linux The application was developed using J2SE and *Eclipse* on GNU/Linux, the Redhat distribution.

Development Environment and Tools

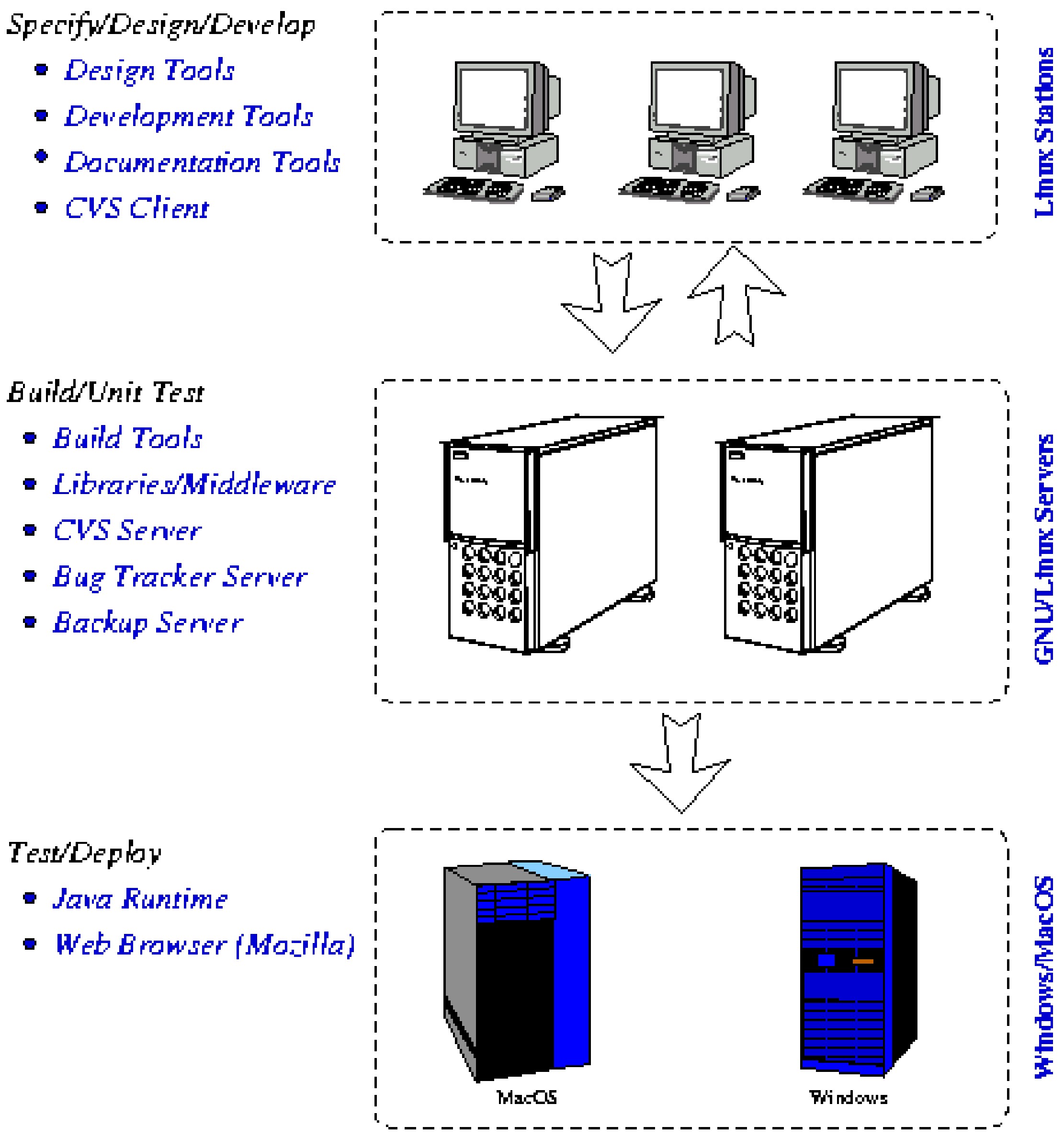

Figure 4.2: Development, Staging and Test Environment

Design Tools An UML driven approach was chosen in which we used ArgoUML to do the UML diagrams.

Development Tools

- GNU/Linux, the Redhat distribution was selected as the main development platform,
- Sun J2SE 1.4 and Eclipse were used for coding the application.
- CVS Client was used to keep the source code synchronized with the main CVS server.
- LaTeX and DocBoook were used for Specification and User documentation.

Build Tools

- CVS was used as the main repository of sources.
- Ant was used as the build tool.

Libraries and MiddleWare See section 4.1.3.

CVS Server Concurrent Versions System (CVS) server and client were used as the Version Control system.

Bug Tracker Server Bugzilla was used as the Bug Tracking system.

Backup Server Amanda server was used as the backup server.

4.1.4 Conclusion

During the life cycle of the CURIA II project, we experienced that using *open source software* is a reasonable (or even superior, depending on the case in hand) approach to using their proprietary competition. Therefore, the usage of OSS should always be considered when developing software.

APPENDIX A

Quick Reference

Table below lists Dixite guides and templates that should be used at various steps of the software production process. Refer to the *References* section to lookup *Internal Resources Ref.*

Section	Title	Internal Resources Ref.
3.1	Planning	
3.2	Requirement Analysis and Specification	Heshmati User Requirement Document (URD) Template and Heshmati Software Requirement Document (SRD) Template
3.3	Designand Specification	Heshmati Architectural Design Document (ADD) Template
3.4	Implementation and Unit Testing	Heshmati Software Release and Build Guide (RBG)
3.5	System Integration and Testing	Heshmati Software Release and Build Guide (RBG)
3.6	Delivery and Maintenance	Heshmati Software Release and Build Guide (RBG) and Heshmati Software Installation Template (SIT)

continued ...

Table A.1: Quick Reference to Dixite Guides and Templates

Section	Title	Internal Resources Ref.

Table A.1: *Quick Reference to Dixite Guides and Templates*

APPENDIX B

Document Information

B.1 Document History

Revision	Date	Modified Sections	Modification
1.0a	Jun 5, 2002	Document Creation	All
1.0b	Nov 14, 2002	Appendix A was created, and sections: 3.6, 3.3, 3.4, 3.5, 3.2 were modified.	Updated the *Books, Guides and Templates* sub-sections.
1.0b1	Nov 15, 2002	Appendix A	Added a new paragraph.
1.0b2	May 9, 2003	Section 4.1	Add a new chapter.

Table B.1: *Document Revision History*

APPENDIX C

Acronyms

ADD Arcitectural Design Document

CVS Concurrent Versions System

DDD Detialed Design Document

GUI Graphical User Interface

HTML HyperText Markup Language is the lingua franca for publishing on the World Wide Web. Having gone through several stages of evolution, today's HTML has a wide range of features reflecting the needs of a very diverse and international community wishing to make information available on the Web.

HTTP HyperText Transfer Protocol The underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

IDE Integrated Development Environment

J2EE Java 2 Platform, Enterprise Edition

J2SE Java 2 Platform, Standard Edition

LDAP Light Directory Access Protocol

OS Operating System

OSS Open Source Software

PDF Potable Document Format

PHP Hypertext Preprocessor is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

PL/SQL Procedural Language extension to SQL Oracle's Procedural Language extension to SQL. PL/SQL's language syntax, structure and data types are similar to that of ADA. The language includes object oriented programming techniques such as encapsulation, function overloading, information hiding (all but inheritance), (. . .)

PMI Project Management Institute

RDBMS Relational Database Management System

RGB Software Release and Build Guide

SQL Structured Query Language A standardised query language for requesting information from a database.

SRD Software Requirement Document

UML Unified Modeling Language

UI User Interface

URD User Requirement Document

VM Virtual Machine A self-contained operating environment that behaves as if it is a separate computer. For example, Java applets run in a Java virtual machine (VM) that has no access to the host operating system.

XML Extensible Markup LanguageThe Extensible Markup Language (XML) is the universal format for structured documents and data on the Web.

XP Extreme Programming Extreme Programming, or XP, is a lightweight discipline of software development based on principles of simplicity, communication, feedback, and courage.

SA Software Architect

PM Project Manager

XSL eXtensible Stylesheet Language is a language for expressing stylesheets.

BIBLIOGRAPHY

Albir: UML In A Nutshell. O'REILLY, 1988, ISBN 1–56592–448–7

Beck: The Extreme Programming Explained. Software Development, 2000, ISBN 0–201–61641–6

Bentley, Jon: Programming Pearls (2nd Edition). Addison-Wesley, 1999, ISBN 0201657880

Bloch, Joshua: Effective Java Programming Language Guide. Addison-Wesley, 2001, ISBN 0201310058

Booch, Jacobson, Rumbaurgh: The Unified Modeling Language User Guide. Addison-Wesley, 1999, Object Technology, ISBN 0–201–57168–4

Booch, Jacobson, Rumbaurgh: The Unified Software Development Process. Addison-Wesley, 1999, Object Technology, ISBN 0–201–57169–2

Cameron, Rosenblatt, Raymond: GNU Emacs. O'REILLY, 1996, ISBN 1–56592–152–6

Chezzi, Jazayeri, Mandrioli: Fundamentals Of Software Engineering. Prentice-Hall, 1991, ISBN 0–13–818204

Cooper: Java Design Patterns. Addison-Wesley, 2000, ISBN 0–201–48539–7

DuBois: MySQL. New Riders, 2000, ISBN 0–7357–0921–1

Eckel, Bruce: Thinking in Java (2nd Edition). Prentice-Hall, 2000, ISBN 0130273635

Evi Nemeth, Garth Snyder, Trent R. Hein/Boggs, Adam: Linux Administration Handbook. Prentice-Hall, 2002, ISBN 0130084662

Fowler: REFACTORING Improving The Design of Existing Code. Addison-Wesley, 1999, ISBN 0–201–48567–2

Fowler, Scott: UML Distilled. Addison-Wesley, 1988, ISBN 0–201–32563–2

Gamma, Helm, Johnson Vlissides: Design Patterns. Addison-Wesley, 1996, ISBN 0–201–63361–2

Goossens, Mittelbach/Samarin: The Latex Companion. Addison-Wesley, 1994, An excellent reference, ISBN 0–201–54199–8

Gosling, Joy, Steele Bracha: The Java Language Specification. Second Edition edition. Addison-Wesley, 2000, ISBN 0–201–31008–2

Heshmati, Javad K.: Software Release and Build Guide (RBG). Av. Lousie 179, 1050 Brussels Belgium: Dixite, Available on Dixite intranet website.

Heshmati, Javad K.: Architectural Design Document (ADD) Template. Dixite Internal Template, 10 2002, Available on Dixite intranet website.

Heshmati, Javad K.: Software Installation Template (SIT). Dixite Internal Template, 10 2002, Available on Dixite intranet website.

Heshmati, Javad K.: Software Requirement Document (SRD) Template. Dixite Internal Template, 10 2002, Available on Dixite intranet website.

Heshmati, Javad K.: User Requirement Document (URD) Template. Dixite Internal Template, 10 2002, Available on Dixite intranet website.

Hows, Smith, Good: Understanding And Deploying LDAP Directory Servers. MACMILLAN TECHNICAL PUBLISHING, 1999, ISBN 0–57870–070–1

Installed, Extreme Programming: Ron Jeffries, Ann Anderson, Chet Hendrickson. Addison-Wesley, 2000, ISBN 0201708426

Kent Beck, Martin Fowler: Planning Extreme Programming. Addison-Wesley, 2000, ISBN 0201710919

Kernighan, Brian W./Pike, Rob: The Practice of Programming. Addison-Wesley, 1999, ISBN 0–201–61586–X

Monson-Haefel, Richard: Enterprise JavaBeans (3rd Edition). O'REILLY, 2001, ISBN 0596002262

Norman Walsh, Leonard Muellner: DocBook: The Definitive Guide. O'REILLY, 1999, ISBN 1–56592–580–7

Software Construction, Code Complete: A Practical Handbook of: Steve C McConnell. Microsoft Press, 1993, ISBN 1556154844

Wall, Christiansen, Schwartz: Programming Perl. O'REILLY, 1996, ISBN 1–56592–149–6

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.