Software Development of the Automated Law Review Submission System

by

David Bendix Gottlieb

Submitted to the Department of Civil and Environmental Engineering on May 7, 2004, in partial fulfillment of the requirements for the degree of Master of Engineering in Civil and Environmental Engineering

Abstract

Organizing article data and information as well as streamlining communications are the most important goals of any submission system. The submission process used by law professors and law review journals severely fails to meet those goals. The Automated Law Review Submission System (ALRSS) has been developed in order to meet those goals. Many potential ALRSS users were interviewed and a rigorous set of requirements was developed. ALRSS successfully provides an easily accessible, central document and information repository, and also would nearly eliminate the use of phone, fax, mail or personal email. Furthermore, the system enables editors of law reviews to control the workflow of the review process. With further development, ALRSS will compete commercially with ExpressO™ in the law article community and strive to operate as effectively as MonsterTrak™ and LSDASTM do in their respective communities. The long term goal of ALRSS is to function as a general submission system that can be customized for a specific submission processes.

Thesis Supervisor: Dr. George A. Kocur
Title: Senior Lecturer, Department of Civil and Environmental Engineering
Acknowledgments

The development of ALRSS as well as the completion of this thesis would not have been possible without the help and time of a few people. I would first like to thank my advisor, Dr. George Kocur, for providing me with useful guidelines, helpful hints, necessary software, and dedicating large portions of his time to reading this thesis and critiquing the project. Next, I want to thank my group members, Colleen O’Shea and Patricia Crumley. Working together with you two made this project significantly easier. Two other people that deserve recognition are my parents, Alice and Allan Gottlieb. Thank you so much for always making time to review my thesis and provide useful suggestions at any hour of the day.

Finally, and most importantly, I want to thank my fiancé, Sarah Simmons. You were always there for me throughout the whole process. Whether it was proof reading, listening to my ideas or motivating me to complete the thesis, I could always count on you to provide the support and patience I needed. This project would not have been possible without you.
Contents

1 Introduction and Problem Statement .......................................................... 13
   1.1 Problem Statement .............................................................................. 13
   1.2 Purpose ......................................................................................... 14
   1.3 Thesis Roadmap ............................................................................... 15

2 Literature Review ....................................................................................... 17
   2.1 ExpressO™ .................................................................................. 17
      2.1.1 Account Creation and Article Submission ...................................... 18
      2.1.2 Payments .............................................................................. 19
   2.2 ALRSS vs. Expresso™ ................................................................. 22
   2.3 Monstertrak™ ............................................................................. 22
   2.4 LSDAS™ ...................................................................................... 26
   2.5 Comparing ALRSS ........................................................................ 28

3 Program Design and Choice of Technologies ......................................... 31
   3.1 Program Design ............................................................................ 32
      3.1.1 UserVariable ......................................................................... 32
      3.1.2 ALRSSArticle ....................................................................... 35
      3.1.3 ALRSS .................................................................................. 35
      3.1.4 ALRSSLawAuthor .................................................................. 36
      3.1.5 ALRSSLawReviewEditor ......................................................... 37
      3.1.6 ALRSSLawReviewReviewer ....................................................... 38
      3.1.7 ALRSSAdministrator ............................................................... 38
List of Figures

3-1 Class Diagram for ALRSS ............................... 33
3-2 Data Model for ALRSS ................................. 34
4-1 Transition for Logging In as a Law Author .................. 49
4-2 Administrator page for enabling accounts .................... 51
4-3 Page where Reviewers view assigned articles ................ 53
4-4 Workflow Page for a Law Review Editor .................... 56
4-5 Page where Authors can view submissions of their articles ...... 58
4-6 Article Submission Page .................................. 61
6-1 State Diagram of ALRSS .................................. 76
List of Tables

2.1 ExpressO™ Open Account Plan savings compared to other submission methods (based on 30 electronic and 10 hard copy submissions) .... 21
2.2 ExpressO™ Complete Prepaid Plan savings compared to other submission methods (based on 30 electronic and 10 hard copy submissions) 21
2.3 Monstertrak™ Single Job Postings prices .............................. 25
2.4 Monstertrak™ Volume Discount prices .................................... 25
2.5 Monstertrak™ Resume Screening prices .................................... 25
2.6 Comparing ALRSS to the other submission systems mentioned in the chapter .......................................................... 29

5.1 Function-Point Multipliers, Adapted from [10] ....................... 68
5.2 Function Point Totals for ALRSS .......................................... 68
5.3 Number of Lines of Code for ALRSS ................................. 68
5.4 Efficient Estimation Schedules for Business Products (Adapted from [13]) ............................................................... 68
5.5 Estimated and Actual Development Schedule ...................... 69
Chapter 1

Introduction and Problem Statement

Even with the enormous advances we have recently witnessed in Internet and web technologies, a number of systems and processes continue to rely on older technologies, such as personally generated emails, phone, fax and postal letters. For many of these systems to operate, several individuals need to participate frequently, which leads to potentially large inefficiencies, especially when the users involved are geographically dispersed, often separated by several time zones. Furthermore, such systems often require a single user to transmit nearly the same information to multiple recipients, in essence manually replicating data, a task that should be automated.

1.1 Problem Statement

One system that still uses outdated technologies is the law review submission process. In this system, law authors submit each article to multiple law reviews (about 60 on average)\(^1\) in hopes of getting their article published in the highest ranked law review. Unlike the situation in scientific publications, not only can law authors submit articles to multiple reviews, but when a law review accepts an article, the author is given a time period in which the professor may ask other (presumably higher ranking) law

---

\(^1\)Personal communication with Martha Davis, Law Professor, Northeastern University
reviews for an expedited decision. Law review rankings are roughly the same as those of the host law school.\footnote{Law school rankings can be found in \textit{US News and World Report.}} Finally, authors decide which accepting law review publishes their papers.

The current law review submission system uses older technologies to perform the vast majority of the communication. Articles are faxed, emailed or even mailed from each professor to tens of law reviews. Questions that arise must be resolved by using personal email or calling the other party. For this system to work, there needs to be people answering the phone and picking up the faxes as well as being able to read and respond to personal email. In the setting of law professors and law review editors, those conditions are not easily satisfied. Law reviews consist entirely of law students who have other, sometimes more important, priorities than the law review. Coordinating times where both professors and law review members have access to a phone or fax (as well as the time to use them) is not an easy task. As submission deadlines approach, this coordination can prove to be even more challenging since many law professors would be trying to reach the same law review members.

Another problem with the system, which cannot be overlooked, is that all of the data and information needs to be manually replicated: The same article needs to be sent separately to each law review. In the best case, law review members need access to their personal email or the computer on which the article is saved in order to read the article. In other cases, the law review member needs to have the physical article that was faxed or mailed to them in order to read it. Ultimately, these restraints lower the efficiency of the system when compared to one in which the article is stored in a secure, but universally accessible, repository.

\section{1.2 Purpose}

The Automated Law Review Submission System was designed to solve the law article submission system problem just described. The primary strategy used by ALRSS is to centralize all data and information and automate most of the communication.
and processing. With this system in place a law review author would create an account once and then, for each article, upload the body, abstract and cover letter, and finally select the law reviews to which the article is to be submitted. The article, together with appropriate notification is then delivered automatically to all specified law reviews. At any time authors can view the status of their article (e.g., submitted, received, accepted, rejected). Authors can also review which law reviews are considering their article. ALRSS provides users continual access to this information from any commonly used browser (e.g. newer versions of Internet Explorer, Netscape and Mozilla). In addition, ALRSS simplifies the process of requesting expedited reviews. Once a paper is accepted, authors are notified and presented with a list of law reviews for which the system can automatically request expedited reviews.

An important issue that ALRSS must address is security. ALRSS stores many documents and much information about law authors and law reviews. Furthermore, users need to log into the system and some of them employ usernames and passwords that they use for other purposes (such as email servers, personal computers or even credit cards)! The information that ALRSS requires is not sensitive, however, it needs to be protected from unauthorized access. Since ALRSS supports a browser interface, there exists a large potential for unauthorized access if security is not sufficiently addressed.

While ALRSS was designed for a specific problem, it is a robust system that can be modified for almost any type of submission system with a known and controlled set of users.

1.3 Thesis Roadmap

The rest of the thesis is divided into five chapters. Chapter two is a literature review of existing work in the area of submission systems. The primary focus of that section is on ExpressO™, which is a competitor to ALRSS. The literature review also discusses two other commercial submission systems, Monstertrak™, and LSDAS™. Monstertrak™ is used by companies and job seekers to post jobs, upload resumes,
cover letters and transcripts as well as schedule interviews. LSDAS™ is employed by prospective law students and law schools to submit and receive applications all parts associated with the application. Chapter three discusses the software design of ALRSS and the technologies used, including the class diagrams employed. The subsequent chapter details the software implementation. The focus is primarily on the mechanics of coding the methods as well as on the communication between the software layers in ALRSS (such as the application and database layers). That same chapter also describe any unique and interesting code developed. Chapter five presents the results of the software implementation including web page transitions, user feedback and measured data such as average page load time. Chapter six presents the detailed conclusions of the ALRSS project and suggests possible future development. It goes into depth on the positives and negatives of the ALRSS software development process. In addition, the section focuses on the successes of ALRSS. The biggest question to be answered is whether ALRSS has a future beyond this project.
Chapter 2

Literature Review

2.1 ExpressO™

Currently, there is a system similar to ALRSS called ExpressO™, which has been operating since August 2003 and has handled over 7500 submissions [18]. Like ALRSS, ExpressO™ is a centralized submission system that streamlines the communications between law professors and law reviews. It performs all the necessary, and basic tasks required for the submission of law articles. As the ExpressO™ website advertises, authors can submit their article by:

1. logging in and entering information such as their name and university affiliation (if any);

2. perusing over 300 law reviews, which can be searched by name or subject;

3. choosing law reviews for article submission; and

4. uploading a cover letter, Microsoft Word article and an optional cover letter and clicking the “Deliver” button [18].

ExpressO™ enables an author to

1. request expedited reviews or withdraw manuscripts from law reviews;

---

1Developed by The Berkeley Electronic Press http://law.bepress.com/expreso
2. determine which law reviews are not accepting new submissions due to full volumes; and

3. submit an article to more law reviews after the initial upload [18].

2.1.1 Account Creation and Article Submission

Creating an account with Expresso™ is rather simple. All that is required is for a user to enter their first and last name, email address and university affiliation (if no affiliation, only the word “none” is needed) into the web form. The user then clicks the submit button, and the information in the form is transferred to the system. A few seconds later, the system responds by sending an initial Expresso™ password to the email address supplied by the user. Users are free to change their password at any time.

Once logged in, a user is presented with the first 50 out of 388 registered law reviews, sorted alphabetically. There are buttons on the page to navigate to the other groups of 50 law reviews, a text box to enter a search phrase and a list of search categories. Next, users select the law reviews to which their article should be submitted. The system then presents a list of the law reviews selected and total cost for submitting an article to all the reviews. At this point, a user can choose to add more reviews before continuing to the upload section.

In the upload section, a user enters and uploads all necessary parts of an article, namely the title, abstract (either in plain text or HTML), and category, the article itself, a cover letter and an optional CV. The article cover letter, CV, and abstract must be in Microsoft Word format and the Expresso™ FAQ page offers users advice on how to convert other word processing programs (e.g. WordPerfect) to Microsoft Word.

Finally, a user is directed to a page to confirm submission details and enter payment options. A more in depth discussion of Expresso™ payments is in Section 2.1.2. A user can either pay individually (and possibly get reimbursed by their uni-

---

2Expresso™ FAQ page: http://law.bepress.com/expresso/faq.html
versity) or through an institutional account [19]. When the user clicks the “Finish” button, Expresso™ will submit the articles electronically to the selected law reviews and, if necessary, alert the Expresso™ staff to print the article and mail it to any selected law reviews that do not support electronic submissions. A confirmation email is automatically sent to the user from ExpressO™ and some law reviews will send their own confirmation that they received the author’s article [18].

2.1.2 Payments

ExpressO™ is not a free service and collects revenue from the law authors. In general, ExpressO™ charges $2 for every electronic submission and $5 for hard copy submissions, which are submitted only when a law review does not accept electronic submissions. ExpressO™ does give new users $8 worth of free submissions.

As mentioned in Section 2.1.1, a user has a choice of either paying individually or through an institutional account [19]. If a user chooses to pay individually, then they are asked to enter their credit card information before submitting an article to ExpressO™. ExpressO™ also provides an alternate means of payment that encourages a law school to have all of their law authors use ExpressO™, called an “Institutional Account citeinstitutional.”

There are two types of Institutional Accounts: an Open Account Plan and a Complete Prepaid Plan [19]. In an open account, a law school may set a credit limit on how much a law author can spend on submitting an article. The law school receives a 15% discount on all their author’s submissions and is billed quarterly. With the Complete Prepaid Plan, a law school pays an annual fee of $2000 and receives unlimited electronic submissions. There is still a $5 charge for all hard copy submissions and the law school gets billed quarterly for those charges [19]. To setup an Institutional Account, a law school needs to fill out a form on the ExpressO™ website and fax the form back to ExpressO™. Table 2.1 and Table 2.2 detail some of the savings gained from using ExpressO™ Institutional Accounts as compared to Individual ExpressO™, regular mail, and FedEx™.3 The implementation of payments

---

3Assumes 100 page articles, submitted to 30 law reviews electronically and 10 via print-and-mail.
in ALRSS will be discussed in Chapter 4.
<table>
<thead>
<tr>
<th>Articles</th>
<th>Individual ExpressOTM</th>
<th>Mail</th>
<th>FedExTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>$3,390</td>
<td>$45,790</td>
<td>$95,790 - $175,790</td>
</tr>
<tr>
<td>150</td>
<td>$2,542.50</td>
<td>$34,342.50</td>
<td>71,842.50 - $131,842.50</td>
</tr>
<tr>
<td>100</td>
<td>$1,695</td>
<td>$22,895</td>
<td>$47,895 - $87,895</td>
</tr>
<tr>
<td>75</td>
<td>$1,275.25</td>
<td>$17,169.50</td>
<td>$35,919.25 - $65,919.25</td>
</tr>
<tr>
<td>50</td>
<td>$847.50</td>
<td>$11,447.50</td>
<td>$23,947.50 - $43,947.50</td>
</tr>
<tr>
<td>40</td>
<td>$678</td>
<td>$9,158</td>
<td>$19,158 - $35,158</td>
</tr>
<tr>
<td>30</td>
<td>$508.50</td>
<td>$6,868.50</td>
<td>$14,368.50 - $26,368.50</td>
</tr>
<tr>
<td>20</td>
<td>$339</td>
<td>$4,579</td>
<td>$9,579 - $17,579</td>
</tr>
<tr>
<td>15</td>
<td>$254.25</td>
<td>$3,434.25</td>
<td>$7,184.25 - $13,184.25</td>
</tr>
<tr>
<td>10</td>
<td>$169.50</td>
<td>$2,289.50</td>
<td>$4,789.50 - $8,789.50</td>
</tr>
<tr>
<td>5</td>
<td>$84.75</td>
<td>$1,144.75</td>
<td>$2,394.75 - $4,394.75</td>
</tr>
</tbody>
</table>

Table 2.1: ExpressOTM Open Account Plan savings compared to other submission methods (based on 30 electronic and 10 hard copy submissions)

<table>
<thead>
<tr>
<th>Articles</th>
<th>Individual ExpressOTM</th>
<th>Mail</th>
<th>FedExTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>$20,600</td>
<td>$63,000</td>
<td>$113,000 - $193,000</td>
</tr>
<tr>
<td>150</td>
<td>$14,950</td>
<td>$46,750</td>
<td>$84,250 - $144,250</td>
</tr>
<tr>
<td>100</td>
<td>$9,300</td>
<td>$30,500</td>
<td>$55,500 - $95,500</td>
</tr>
<tr>
<td>75</td>
<td>$6,475</td>
<td>$22,375</td>
<td>$41,125 - $71,125</td>
</tr>
<tr>
<td>50</td>
<td>$3,650</td>
<td>$14,250</td>
<td>$26,750 - $46,750</td>
</tr>
<tr>
<td>40</td>
<td>$2,520</td>
<td>$11,000</td>
<td>$21,000 - $37,000</td>
</tr>
<tr>
<td>30</td>
<td>$1,390</td>
<td>$7,750</td>
<td>$15,250 - $27,250</td>
</tr>
<tr>
<td>20</td>
<td>$260</td>
<td>$4,500</td>
<td>$9,500 - $17,500</td>
</tr>
<tr>
<td>15</td>
<td>(-$305)</td>
<td>$2,875</td>
<td>$6,625 - $12,625</td>
</tr>
<tr>
<td>10</td>
<td>(-$870)</td>
<td>$1,250</td>
<td>$3,750 - $7,750</td>
</tr>
<tr>
<td>5</td>
<td>(-$1435)</td>
<td>($375)</td>
<td>$875 - $2,875</td>
</tr>
</tbody>
</table>

Table 2.2: ExpressOTM Complete Prepaid Plan savings compared to other submission methods (based on 30 electronic and 10 hard copy submissions)
2.2 ALRSS vs. Expresso™

When compared to Expresso™, ALRSS offers two significant advantages. In terms of software, it appears that Expresso™ uses older web development technology than ALRSS. ALRSS consists of 7 webpages and is developed in a compiled, strongly-typed, object oriented language while Expresso™ appears to have more HTML pages and uses an interpreted scripting language. This makes ALRSS more scalable and easier to maintain. The distinction between scripting and compiled languages as well as between object oriented and non-object oriented languages is rigously treated in Chapter 3.

What really separates ALRSS from Expresso™ is the service ALRSS provides to law reviews. When a law author submits an article to Expresso™, the only service Expresso™ offers to the law reviews is the (electronic or hard copy) submission. While this is a vast improvement in the submission process, ALRSS offers more to the law reviews. One of the key components implemented in ALRSS is to give a law review the ability to control the inner workflow of that particular law review. Law review editors can view all the submitted articles, assign and reassign articles to reviewers, set internal deadlines and update the status of an article all from one webpage! The details of the workflow component of ALRSS are thoroughly covered in chapters 3 and 4. In addition, ALRSS has a marketing scheme that is designed to compete with Expresso™ by providing law authors with a cheaper alternative to Expresso™. ALRSS will also be free of charge for all registered users. The business plan of ALRSS is detailed in [8].

2.3 Monstertrak™

Monstertrak™ is an example of a successful central submission systems. Monstertrak™ is used by job applicants, employers and university career centers. In the most basic

---

4From navigating the website, one can see “cgi-bin” in the URL, which implies that a scripting language is used.

5Job Postings http://www.monstertrak.com
use, employers post jobs on the Monstertrak™ website and perspective employees can search through jobs by different categories, such as industry and state. An applicant interested in the job descriptions submits his resume to Monstertrak™, which presents it to the job poster whenever that company logs in. All subsequent communication between the company and the perspective employee is done directly.

Monstertrak provides an excellent and custom service for university recruiting called InterviewTrak™ [14]. The entire process from searching posted jobs through first round interviews is done completely within the InterviewTrak™ system. InterviewTrak™ has three distinct types of users: students, employers, and college staff (career centers).

For students, the process begins with creating a user profile. Information such as name, email address, major and graduation year is required. Monstertrak™ uses a student’s major and graduation year to determine whether or not the student is eligible for a particular job. Once a student completes a profile, they can upload up to 10 resumes, one cover letter, one transcript, and one supplemental information file. All potential jobs require a resume submission and some require a cover letter, transcript, and/or supplemental information in addition to the resume. After a student has uploaded the necessary documents, they can search through the list of full-time jobs and internships targeted for university students. When students find a position of interest for which they meet the requirements, they can submit a resume (and any other required information) to that company for interview consideration. If a company is interested in a student, the company will pre-select that student for a first-round interview and the student will be able to pick an interview time from a list of available slots. The interviews are usually conducted on-campus. After the first-round interview, the company and student contact one another directly. InterviewTrak™ is free for student use.

An employer's role in the system is to post jobs, review submitted resumes, and select students for interviews. When an employer posts a job, they usually give a brief description of the company (including geographic locations), the specific role posted, the qualifications for the job, a person to contact for further questions, and a
resume submission deadline. Once the resume submission deadline passes, employers
review all the resumes submitted and select a certain number of students for first-
round interviews. The number of students selected is completely determined by the
employer. Again, any further communication with a student beyond the first-round
interview is done outside Monstertrak™.

The role of the college staff is to be a bridge between a student and an employer. A
college staff user of Monstertrak™ helps companies schedule on-campus information
sessions, alerts students to companies interested in on-campus recruiting, and provides
on-campus interview locations. Purchasing InterviewTrak significantly simplifies all of
those tasks for college staffs. In addition MonsterTrak™ provides free and unlimited
on-site training and technical support [14]. The annual cost for a college staff is $2,875
annually.

Just like ExpressO™, Monstertrak™ is not a free service. Monstertrak™ gen-
erates revenue from employers and college staffs. Employers can purchase the follow-
ing three Monstertrak™ services: Single Job Postings, Volume Discounts, Resume
Screening. The prices for these services are shown in Tables 2.3, 2.4, and 2.5 [14].
<table>
<thead>
<tr>
<th>Number of Schools</th>
<th>Pricing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 11</td>
<td>$25.00 per school</td>
</tr>
<tr>
<td>12 - 99</td>
<td>$295.00 total</td>
</tr>
<tr>
<td>100+ (Nationwide)</td>
<td>$395.00 total</td>
</tr>
</tbody>
</table>

Table 2.3: Monstertrak™ Single Job Postings prices

<table>
<thead>
<tr>
<th>Number of Jobs</th>
<th>Pricing (per job)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>$395.00</td>
</tr>
<tr>
<td>5 - 9</td>
<td>$206.00</td>
</tr>
<tr>
<td>10 - 14</td>
<td>$182.00</td>
</tr>
<tr>
<td>15 - 29</td>
<td>$173.00</td>
</tr>
<tr>
<td>30 - 59</td>
<td>$154.00</td>
</tr>
<tr>
<td>60 - 99</td>
<td>$120.00</td>
</tr>
<tr>
<td>100 - 199</td>
<td>$104.00</td>
</tr>
<tr>
<td>200 - 299</td>
<td>$90.00</td>
</tr>
<tr>
<td>300 - 599</td>
<td>$85.00</td>
</tr>
<tr>
<td>600+</td>
<td>$80.00</td>
</tr>
</tbody>
</table>

Table 2.4: Monstertrak™ Volume Discount prices

<table>
<thead>
<tr>
<th>Duration</th>
<th>Pricing (per user)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Days w/job listing</td>
<td>$100.00</td>
</tr>
<tr>
<td>2 weeks (introductory)</td>
<td>$400.00</td>
</tr>
<tr>
<td>3 Months</td>
<td>$1,800.00</td>
</tr>
</tbody>
</table>

Table 2.5: Monstertrak™ Resume Screening prices
2.4 LSDAS™

Another highly successful, web-based commercial submission system is the Law School Data Assembly Service (LSDAS), which is operated by the Law School Admission Council (LSAC). LSDAS is used by perspective law students while applying to law schools. According to the LSDAS website, “Almost all ABA-approved law schools and several non-ABA-approved schools require that their applicants register for the Law School Data Assembly Service (LSDAS) [7].” Currently, there are 187 law schools that are approved by the American Bar Association (ABA) [1]. In 2003, 99,500 people applied to ABA-approved law schools [6] and the number is projected to increase for 2004. The sheer number of applicants is already a large task for a system to handle. If you compound that with the fact that most law schools have similar admission deadlines, LSDAS has to be able to handle a significant number of near-concurrent submissions.

The service LSDAS provides is gathering all the necessary non-school specific application materials, preparing a report, and sending that report to all the law schools a perspective student chooses. A LSDAS report includes:

1. An undergraduate academic summary.

2. Copies of all undergraduate, graduate, and law school transcripts (which the respective colleges and/or universities send directly to LSDAS and contain an official seal).

3. LSAT (Law School Admission Test) scores and writing samples (obtained through LSAC, which organizes both LSAT and LSDAS).

4. Copies of Letters of Recommendation, if processed by LSAC.

Any information LSDAS has for a perspective students remains on file until five years after the student’s last LSAT [7].

The process for having a LSDAS report sent is quite simple. The first step is to create an online account with LSDAS. The information required is similar to that
required on college and graduate school applications (e.g. name, address, date of birth, social security number, etc.). The registration also allows the user to pick a username and password. Once an account is created, the user pays a $99 registration fee that includes sending a report to one law school. Next, the user has the option of requesting LSDAS to send law reports to other law schools at $10 per report. It is important to note that the student need not specify all relevant law schools at this time. Any reports requested after the initial registration are $12 per report. Another interesting fact about the process is that the LSDAS reports are not sent until the law schools request the report. LSDAS sends law schools both electronic and hard copies of the report.

LSDAS is clearly an extremely useful for perspective students. Instead of filling out multiple forms that request the same data, the students just fill out the form once and select to which law schools they want their information sent. Also, LSDAS’s website allows students to view the status of the requested documents and whether a law school has requested a LSDAS report for the student. This way, students can recognize early in the process if a form did not get sent or was lost and can fix the situation early in the admissions process when law schools are not at their application volume peak.

It is very impressive that almost all of the ABA-approved and several non-ABA-approved law schools require perspective students to use LSDAS. It is quite difficult (and usually rare) for a large group, consisting of many parties, to agree to a set of standards. An important question to ask is “why would law schools all agree to require LSDAS reports?” A cynical reply would be that they receive a payment from LSDAS for every report sent. A more realistic response is that LSDAS drastically simplifies the process for the law schools, thereby lowering their costs. If the law schools did not use LSDAS, they would have to modify their current application webpages to accept the information that LSDAS supplies. Modifying the webpages would not be too difficult, instead, the most difficult part would be keeping track of what a student has and has not completed. The information that LSDAS collects, such as official LSAT scores, transcripts, and letters of recommendation, usually do not arrive at the
same time. Without LSDAS, law schools would have to receive and process all of these documents as they arrive. However, LSDAS reduces the law school’s effort by collecting and processing all of this information at their site and then sending a single report to the law schools. The only effort required by the law school is to correctly file that single report. Another benefit to using LSDAS is that LSDAS has already organized all the data in a manner that is easy for admission readers to comprehend. This saves law schools the trouble of creating and refining their own reports.

2.5 Comparing A LRSS

While A LRSS and ExpressOTM serve the same community, A LRSS is modeled closer to Monstertrak™ and LSDAS. The most significant software aspects of ExpressOTM focus on law authors. ExpressO’s™ software dramatically consolidates the efforts of a law author but does not provide that much help to law reviews. Conversely, Monstertrak’s™ InterviewTrak, significantly eases the recruiting process for students, employers, and college staffs. In addition, InterviewTrak is very reasonably priced. While LSDAS provides a greater service to perspective law students than to law schools, the help it provides to law schools cannot go unnoticed. Considering the huge number of applicants and that application information needs to come from multiple sources, the consolidation of the application data into a single report from one source is almost as valuable to law schools as InterviewTrak is to employers and A LRSS workflow is to law reviews. What will separate A LRSS from ExpressOTM is the ability to provide law authors with a simple, streamlined process that is similar to Expresso™ and give law reviews workflow software that is as effective, easy to use and well supported as InterviewTrak in addition to providing similar simplicity to that of LSDAS. Table 2.6 provides a summary of how A LRSS compares to the other submission systems mentioned in the chapter. In the table, submitter refers to people who submit information (i.e. law authors and students) while receiver is the body that receives the submitted information (i.e. law reviews, employers, and law schools).
<table>
<thead>
<tr>
<th></th>
<th>ALRSS</th>
<th>Expresso&lt;sup&gt;TM&lt;/sup&gt;</th>
<th>Monstertrak&lt;sup&gt;TM&lt;/sup&gt;</th>
<th>LSDAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralizes Data</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Software for Submitter</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Software for Receiver</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Noticeable Submitter Effort</td>
<td>Decreases</td>
<td>Decreases</td>
<td>Decreases</td>
<td>Decreases</td>
</tr>
<tr>
<td>Noticeable Receiver Effort</td>
<td>Decreases</td>
<td>No Change</td>
<td>Decrease</td>
<td>Decreases</td>
</tr>
<tr>
<td>Submitter Cost</td>
<td>Decreases</td>
<td>Decreases</td>
<td>No Change</td>
<td>Increases</td>
</tr>
<tr>
<td>Receiver Cost</td>
<td>Decreases</td>
<td>No Change</td>
<td>Decreases</td>
<td>Decreases</td>
</tr>
</tbody>
</table>

Table 2.6: Comparing ALRSS to the other submission systems mentioned in the chapter
Chapter 3

Program Design and Choice of Technologies

The most critical aspect of any software project is the software design. An inadequate design is one of the major reasons why software projects run severely over-schedule and are of mediocre quality. About 50% of the effort spent on ALRSS went towards developing a solid program design, data model [8], and user manual [17]. The program design is represented by the class diagram shown in Figure 3-1. All three of the documents were generated from comments and suggestions made by many law professors and law review members. The current class diagram has 12 classes and 73 methods.

While the class diagram and data model (shown in Figure 3-2) of ALRSS are very similar, there are a two discrepancies worth mentioning. First, there are four tables that appear in the data model but are not visible in the class diagram. These tables are University, Subject, Article.State, and Author.Article. The first three tables are domain tables while the last is an intermediate table. Domain tables consist of a single column and are used to provide referential integrity when another table that references the domain table is updated. For example, the University table has one column, UniversityID, which is a foreign key in the Authors table. When an author is added or updated, a value for UniversityID is entered. Before the value for UniversityID is inserted in the Author table, the database will make sure the
value for UniversityID exists in the University table. If it does, then the value will be inserted into the Author table. Intermediate tables, such as Author_Article, are used when there exists a many-to-many relationship between two database tables. Without Author_Article, which is the intermediate table between the Authors and Article tables, it would be impossible to select a unique Author/Article pair.

While domain tables are useful in designing databases, they are not useful as classes in in the application. Making a domain table into its own class would add complexity to the system while not improving the functionality. Therefore, in the class model and the actual implementation, the domain tables are left out. Intermediate tables are not implemented and do not appear in the class model for the same reason as domain tables. Creating the extra class for the intermediate table would not enhance the implementation. The intermediate table is acknowledged in the implementation whenever a SQL query is made that involves joining the Article and Authors tables.

Another small discrepancy between the data model and class diagram is between some of the values in the Users table and the UserVariable class respectively. All of the columns in the Users table directly appear in the UserVariable class with the exception of username and role. However, these columns translate to the variables currentUser and currentRole. The rationale for the name change is that both currentUser and currentRole are used frequently within the ALRSS implementation and naming the variables this way is clearer to other people reading the code.

3.1 Program Design

The subsequent subsections will describe the roles and responsibilities of each class in ALRSS. Chapter 4 covers the details of how the classes are implemented and how they interface with one another.

3.1.1 UserVariable

UserVariable is one of the two helper classes in the system. The primary function of UserVariable is to hold static variables and methods used by all of the main classes.
Figure 3-1: Class Diagram for ALRSS
Figure 3-2: Data Model for ALRSS
Variables common to all classes include those that interface with the database, such as the SQL connection, command, reader and transaction. Other variables contained in this class relate to users such as their username and role in the system. The class also stores variables for error messages to display when users try to access information inappropriately. There is one method called checkConnection() that is responsible for making sure that the connection to the database is defined (supplies the server, username, password, and database name variables) and that the connection is not in use.

3.1.2 ALRSSArticle

ALRSSArticle is the other helper class. The sole purpose of this class is to display the Microsoft Word documents retrieved from the database. Users of the ALRSS, ALRSSLawAuthor, ALRSSLawReviewEditor, and ALRSSLawReviewReviewer classes may need to view an article. In order to accomplish this task, the class in question will pass to the ALRSSArticle class the necessary information, such as the ArticleID, ReviewID, and what type of document needs to be retrieved (i.e. Article, Abstract, or Cover Letter) [8]. The ALRSSArticle class is the only one whose behavior depends upon the browser used. With newer versions of Internet Explorer (tested on version 6.0), ALRSSArticle will display the Microsoft Word document in the browser itself. However, in the newer versions of Netscape (tested on 7.1 and 7.0) and Mozilla (tested on 1.5 and 1.6), the browsers will ask the user if they want to open the document with the default application (most likely Microsoft Word) or save the file to disk.

3.1.3 ALRSS

ALRSS is the only class that can be accessed anonymously. It acts as the gateway to all of the other main classes. From the ALRSS class, an anonymous user can perform six functions. In order to access other pages, users need to authenticate themselves. The ALRSS class has variables and methods that permit a user to enter his username and password. The class will then query the database and not only check whether
the username and password pair is correct, but also retrieve the role of the user in the system. The ALRSS class uses the username and role to correctly redirect the user.

A user also has the option of registering either a Law Author or Law Review Editor account. The class reads in the data entered by the user and processes the information accordingly. Since an anonymous user can register an account, all accounts registered must be enabled by an ALRSS Administrator. Enabling a law author account requires an ALRSS Administrator to confirm that a law author exists at the university they are claiming. To enable an initial law review editor account, an ALRSS administrator must confirm that the law review entered exists at the university it claims and that the person registering the account is indeed an editor of the law review. ALRSS administrators only need to enable the initial law review editor account because it has been decided that any law review editor can create additional law review editor and reviewer accounts for a given law review [17].

An interesting service provided by the ALRSS class is the ability for anonymous users to either search abstracts by category or employ full-text searches. The search results consist of the title, links to the abstract and author's contact information, and in the case of full-text searching, a ranking of the search phrase to the abstract text. The ranking is an integer between 0 and 1000, though displaying the rank as a percentage is an alternative solution. The ALRSS class offers anonymous users two other options. Anyone is able to anonymously or non-anonymously send the ALRSS development team suggestions, errors or complaints. The ALRSS class retrieves the user-entered information and automatically generates an email to all the members of the ALRSS development team containing the information. Finally, all users have the ability to view a page containing information about ALRSS and its developers.

3.1.4 ALRSSLawAuthor

ALRSSLawAuthor is used by law authors for article submissions. When law authors are ready to submit an article, they use this class to browse the list of law reviews that are currently accepting new submissions and choose to which their article should
be submitted. When submitting an article, law authors are also required to submit an abstract and cover letter. Law authors also use ALRSSLawAuthor to view all their articles in the ALRSS system as well as check the status of their articles at the law reviews to which they are submitted.

Once a law author's article is accepted by a law review, the author will use this class to request expedited reviews from other law reviews. At any point, authors can agree to publication in a law review that has accepted their article. This action will automatically withdraw their article from all the other law reviews currently reviewing it. The ALRSSLawAuthor class is also used to submit different versions of the article. Law authors are allowed to submit a new version of their article to a law review if that law review has accepted it. Finally, the class is used by law authors to change their password and update their contact information.

3.1.5 ALRSSLawReviewEditor

ALRSSLawReviewEditor contains the workflow software discussed in Chapter 2.1 that separates ALRSS from ExpressO™. This class gives law review editors the ability to assign and reassign articles to law review reviewers as well as set deadlines for when the articles need to be reviewed by the assigned editor and reviewer. Law review editors can also use this class to update the status of each submitted article. Editors are the only type of users that can accept and deny articles. Law review editors are able to update the submission and copyright requirements for the law review.

Law review editors also have the ability to perform full-text and category searches. When editors perform searches, they search the actual article, whereas anonymous users were only allowed to search abstracts.

ALRSSLawReviewEditor is responsible for managing user accounts for each law review. For a given law review, any editor can create and delete law review editor and law review reviewer accounts. In addition, a law review editor can update the contact information for the law review. Finally, law review editors have the ability to change their own contact information as well as their password.
3.1.6 ALRSSLawReviewReviewer

ALRSSLawReviewReviewer is one of the simplest classes. This class is used by reviewers to read articles that an editor has assigned to them and make comments about the article. When a reviewer logs on, the class displays the article’s title, which links to the actual article, and the personal deadline the law review editor has given the reviewer to review the article. The class also enables the reviewer to change their password and update their contact information.

3.1.7 ALRSSAdministrator

ALRSSAdministrator is another simple main class. The sole function provided by the class is to enable all law author and initial law review editor and reviewer accounts. For law authors, the class displays the author’s full name, username, and university affiliation (if the author is not affiliated with a university, then the affiliation is “none”). For law review editors and reviewers, the administrator is able to view the editor’s full name, username, university affiliation, (which cannot be “none” since all law reviews are associated with a university) and law review name. The ALRSSAdministrator class has four methods that allow an administrator to enable a single account (either law author, law review editor, or law review reviewer), enable all of the disabled accounts, disable a single account, disable all of the enabled accounts. Again, accounts are only enabled once an administrator verifies that the law author or law review editor is valid.

3.2 Choice of Technologies

Choosing the right development technologies for ALRSS was a critical step. In particular, it was crucial to choose technologies that interface well with one another. In addition, knowing the needs for your system and price of the software is almost as important as the performance of the technologies. The development technologies can be broken down into three areas: development language, web server, and database.
3.2.1 Development Language

ASP.NET was the language chosen for code development. ASP.NET is part of the .NET environment, which contains a rich class library of over 3,400 classes and is designed to interface well with Microsoft products. In web development, programmers have the choice of using interpreted or compiled languages. Interpreted languages, such as Perl (Practical Extraction and Report Language) and PHP (PHP Hypertext Preprocessor), are common in CGI (Common Gateway Interface) and Apache web servers. Compiled languages, such as Java, VB.NET (Visual Basic.NET), and C#, are common in web servers running J2EE and ASP.NET.

Interpreted, or scripting languages are usually very easy to learn and use. They are not strongly-typed and there are usually many ways to write programs that accomplish the same task. Some interpreted languages (such as Perl and PHP) have powerful class libraries and are very useful for small systems and small scripts. However, because these languages are not strongly-typed and do not have many syntactical constructs, they pose some problems in building large, scalable systems. Because interpreted languages have loose syntax, it is very easy for programmers to write code that is extremely hard for another programmer to read and understand (this type of code is often referred to as spaghetti code). In addition, since the interpreted languages are not strongly-typed, it is possible that unrelated variables can be used together and produce unknown effects. When developing large, scalable systems, it is critical that programmers are able to write code that can be read and integrated by other programmers.

Alternatively, compiled languages are often used primarily in large, scalable systems and execute faster than interpreted languages. Complied languages are faster than executed languages because the code is compiled into machine code. Most compiled languages used in web development, such as Java, VB.NET and C#, are strongly-typed, object-oriented languages. A strongly-typed language allows the execution of the program to be more safe and predictable. An object-oriented language results in code that is easier to read, understand and maintain. In addition, object-
oriented languages all have the concept of inheritance, which is the idea that objects can inherit methods and properties from base classes. Most compiled languages only support single inheritance, though C++ supports multiple inheritance. The concepts of objects and inheritance simplify the process of modeling they system in a class diagram. As mentioned earlier in the chapter, a solid and clear design is a crucial step when developing a large, scalable system.

Instead of ASP.NET, ALRSS could have been developed in the J2EE environment. However, the ALRSS development team thought that ASP.NET would be an easier development environment that also provided a rich set of additional web controls. Data shows that ASP.NET is a more scalable and productive development environment. “If a unit of work costs 10 cents on .NET platform, that same unit of work will probably cost 50 cents to a dollar on J2EE/Unix [23].” In addition, “the .NET Framework includes a huge system libraries (called Namespaces) combined with the unique development tool (Visual Studio .NET) dramatically reduce development time and thus cost. J2EEs best development tool IBMs WebSphere is out of competition [23].” In a productivity test on the same benchmark (PetShop 2.0, developed by The Middlware Company), developing the application in J2EE using Middleware required 14,004 lines of code, while only 2,096 lines of code was needed to develop the application in ASP.NET using Visual Studio.NET [23].

Despite the Middleware Company’s test, J2EE is the most common development environment for large web sites and there are compelling arguments for choosing J2EE over ASP.NET. J2EE is being marketed by an entire industry and is a proven platform, while .NET is a rewrite and subject to non-trivial risks associated with that. J2EE also lets developers take advantage of existing hardware and provides platform neutrality, including Windows. In addition, J2EE lets developers use any operating system he or she prefers, such as Windows, UNIX, or mainframe. Finally, “J2EE is a more advanced programming model, appropriate for well-trained developers who want to build more advanced object models and take advantage of performance features. [20]” Since ALRSS uses only Microsoft products, platform neutrality is not as important. Furthermore, the superiority of Visual Studio.NET as an IDE is more
important than an advanced programming model.

ASP.NET can be developed in over 30 different programming languages, such as C#, VB.NET, C++ and Java. However, C# and VB.NET are designed specifically for ASP.NET and therefore, are the most common languages used in ASP.NET. In addition, C# is “the only language designed from the ground-up for the .NET Framework”, which is one of the reasons why the ALRSS development team uses it to develop the ALRSS code [22]. C# is also very similar in structure to Java and C++ (which are the programming languages most commonly used in industry). This increases the probability that other programmers will be able to understand the code and be able to extend the system.

3.2.2 Web Server

ALRSS uses Microsoft Internet Information Services (IIS) 5.0 as its server on Windows XP Professional operating system (Version 6.0 of IIS is available, but it is only recommended for Windows 2003). The alternative to IIS 5.0 is the Apache Web Server. Apache can be run on both Linux and Windows operating systems, but is designed primarily for Linux. Both Apache and IIS are easily configurable. While IIS can be configured using a Graphical User Interface (GUI), Apache is configured from the command line (though there are tools being developed for GUI configuration, such as Comanche). We have read that, “installing Apache is a snap, and it can be done by any competent webmaster in a few minutes. [15]” Apache and IIS deliver similar performance and functionality. However, according to the most recent Netcraft survey, 67.21% of all servers run Apache and 20.88% run some version of IIS [16]. Two important questions to consider are “why is Apache used significantly more than IIS” and “if Apache is the preferred server, why does ALRSS use IIS?”

In response to the first question, we note that one factor causing administrators to choose Apache over IIS is that both Linux and Apache are free, but Windows software and licenses are not (IIS comes free with Windows). Probably the most important consideration is the security issues IIS 5.0 had. For most companies, it is an expensive task to patch all of their web servers every time a security update
is needed. Compared to Apache, IIS has needed noticeably more updates (the last Apache update was noticed and fixed in January 1997 [15]). In addition, most of the IIS security holes “grant more general access to the webserver system. Crackers can breach the network security and gain access to file systems and other permission to execute commands. [15]” Another flaw of IIS when compared to Apache is that IIS components run with superuser (root) permissions, while Apache components run as a non-privileged user. Users with root permissions have the ability to “access, change, and delete any file anywhere on the system. [15]” According to Scott Culp, security program manager at Microsoft, in Redmond, Washington, “There is a problem with IIS...We’ve just had too many vulnerabilities affecting IIS...We recognize the need to do a better job of making it secure [15].” Microsoft has responded to the security issues in IIS 6.0 and “there’s very little differentiation between the most recent versions [of] both applications [3].” Even with a more stable version of IIS, Apache might still continue its market domination because switching operating systems and software is a very expensive and time consuming task for most companies. In addition, Linux and Apache are much cheaper than Windows 2003, and using IIS locks users into using Microsoft products, which might be a problem for certain companies.

Despite all the issues presented in the previous paragraph, ALRSS chooses to IIS as its web server. Since the code for ALRSS is developed in ASP.NET, IIS is a better choice since it interfaces very well with ASP.NET.

3.2.3 Database

ALRSS uses the Microsoft SQL Server 2000 database. The primary reason for this choice is that Microsoft SQL Server 2000 is “an excellent choice if Windows is, and will continue to be, the operating system on which the application will run [4].” SQL Server 2000 also supports a full-text indexing of Microsoft Word Documents, which is used by ALRSS to search submitted articles. Alternative RDBMSs (Relational Database Management Systems) to SQL Server 2000 are Oracle 9.0, IBM DB2, and MySQL 4.0. MySQL is Open Source and is free under the Open Source license and is available at $495 for a commercial license. MySQL is very fast, but is mainly
designed for smaller-scaled applications. Oracle and DB2 are good choices for large scale systems that need to be run on multiple platforms. Oracle also has many built-in bells and whistles for database management, however, Oracle is difficult to tune to Windows operating systems. The criteria used by the ALRSS development team for RDBMS selection are: easy to setup and use in Windows XP, pass the ACID test for transactions, provide reliable and fast service for roughly 5000 transactions a day, and easily interfaces with ASP.NET. Only Microsoft SQL Server 2000 satisfies these criteria.
Chapter 4

Software Implementation

ALRSS has an interesting software implementation that takes advantage of the rich class library provided by the .NET framework and the unique features of ASP.NET. The four essential features of ASP.NET that are frequently used by ALRSS are code-behind, user controls (which are a form of macro processing as described below, and are denoted with .ascx extensions rather than .aspx), dynamically displaying content in ASP.NET Panels, and DataBinding to ASP.NET controls [21].

The present chapter begins with a section on the overall implementation strategy, explaining our use of the ASP.NET framework and the advantages thereby attained. We then devote one section to each of the six web pages with which the user interacts and its associated C# class.

4.1 Overall Implementation Strategy

ALRSS makes significant use of user controls, which enable the developer to encapsulate a variety of visual objects (radio buttons, drop down lists, text input boxes, etc.) into one object that can subsequently be placed on a number of pages. This encapsulation can result in significant code savings and enables ALRSS to be a highly modular and scalable system. For example, the ALRSS Law Author page contains 9 user controls and requires only 60 lines of html for its implementation (excluding the definitions of the user controls). We estimate that were user controls not used the
page would require about 2500 lines. In larger systems where the same control is used multiple times on a single page, the savings can be even more dramatic. Naturally, if the ALRSS Law Author page was extended to have increased functionality, the lines of code would increase. However, this would only be an additional 6 lines of code for each additional user control placed on the page. As with other modular programming techniques, user controls ease maintenance by pinpointing the code sections that must be examined to update various entities. Finally, it must be noted that user controls can respond to ASP.NET and HTML events (see [21] and [22] for an explanation of events).

ALRSS also employs Panels, which act as placeholders in a page for the subsequent loading of controls. By dynamically choosing which control to load based on the user’s history and state (e.g., login name), the system can customize the display to the current situation. As mentioned above, the ALRSS code actually has but six web pages. However, users of ALRSS see about 50 different pages. Under the covers, ALRSS code (written in C# as ASP.NET code-behind modules) chooses and loads various user controls to give the page the desired appearance.

Rather than place the user controls on the web page directly, ALRSS follows normal ASP.NET practice and instead constructs the web page out of Panels into which the user controls are placed. Although ALRSS has only six web pages, it appears to its users to have approximately 50. This is due the extensive deployment of dynamically generated content. For example, when a user visits an ALRSS page, the C# code-behind file dynamically sets the Visible property of each Panel to either true or false, thereby controlling the perceived contents of the page. The total number of panels in ALRSS is 44.

The final ASP.NET feature used in ALRSS is DataBinding, which gives on-line access to a stored database thereby enabling the visible contents of web pages to contain values obtained at runtime from the database and also supports a web interface to database queries and updates.
4.2 The Front Page

The ALRSS class is the front page of the system. It serves as the gateway to the other pages and is the only page that can be accessed by an anonymous user. In order to access any other page, users must first log in. They are then redirected to another page based on their role within the system. The front page also permits users create a Law Author or Law Review account, read information about ALRSS, search for article titles and abstracts based on a full-text search or category, and send comments to the ALRSS development team.

Based on the requirements for ALRSS [17], all passwords stored in the database must be encrypted. Both logging in as well as creating accounts require encrypting plain-text passwords before checking the password against the database or inserting the password into the database. If a user forgets his or her password, it needs to be retrieved from the database and then decrypted before it can be emailed to the user.

Encrypting passwords is one interesting implementation found in ALRSS. The encryption algorithm used is DES, the United States Data Encryption Standard [21]. DES is a symmetric encryption algorithm, which means that both encryption and decryption is done with the same private key. Asymmetric encryption algorithms such as RSA use a private key to encrypt data and a public key (that is available to everyone) to decrypt. ALRSS uses a symmetric algorithm because it is faster and shorter than asymmetric algorithms. Furthermore, since both encryption and decryption are done within the system, therefore, a public key is unnecessary. More information on DES and RSA can be found at http://www.itl.nist.gov/fipspubs/fip46-2.htm and http://www.rsasecurity.com/rsalabs/rsa_algorithm/index.html respectively.

The Encrypt and Decrypt methods are very similar and work in the reverse of one another. For both methods, two byte arrays of length 8 are required. One array contains the secret key in its first seven bytes and the other is an initialization vector. In addition, the plain-text password, which is an argument to the Encrypt method, is also converted into a byte array. Next, objects are created to perform the actual encryption. First, a DES object of type, DESCryptoServiceProvider, is instantiated
and then its method, `CreateEncryptor(byte [] key, byte [] ivector)`, is used to create the encryption transform [5]. Then, a CryptoStream object is created and its Write() method performs the encryption and writes the result to a MemoryStream. Finally, the contents of the MemoryStream are converted from a byte[] to a string and returned.

Another code sample to examine is the DBAuthenticate procedure, a noteworthy example of a SQL stored procedure, the use of which is known to increase the performance and maintainability of an application [21]. We discuss SQL stored procedures further in Chapter 6. DBAuthenticate takes two arguments, the username and the encrypted password. The procedure first executes a SELECT statement containing the username to obtain from the database the user's role, encrypted password and whether or not the account is enabled. If the passwords match and the account is enabled, DBAuthenticate redirects the user to the appropriate page. Otherwise, the user is redirected to the front page and an appropriate error message is displayed. Figure 4-1 shows the transition of an author logging in and being redirected to the law author page.

4.3 The Administrator Page

ALRSS provides Administrators with their necessary functions. As discussed in Section 3.1.7, they need to enable and disable law author, law review editor and law review reviewer accounts. In addition, it is convenient for Administrators to view usernames and (encrypted) passwords from a web browser, rather than requiring them to search the database directly. The administrator functionality is almost entirely implemented as an ASP.NET DataGrid, one of the DataBindings described above. "The DataGrid control is the most feature-rich (and most complicated) control included with the ASP.NET framework...it has several advanced features, such as support for sorting...which make it unique [21]."

The ALRSS Administrator class makes use of two types of DataGrid columns: BoundColumn and ButtonColumn. The former display data directly from the database,
Figure 4-1: Transition for Logging In as a Law Author
while the latter can perform user-defined actions when clicked. A DataGrid object is filled by initializing its DataSource property and calling the DataBind() method. The DataSource is set to the DataReader returned from a standard SQL SELECT statement. Database data is bound to the appropriate BoundColumn by examining the DataField property which is set to match the appropriate column name returned by the SQL query. ButtonColumn click events are distinguished from one another through the CommandName property of the DataGrid object. The CommandName property is uniquely set by the developer. A description of events raised by the DataGrid control can be found in [21].

The Administrator class has three DataGrids, one for law authors, another for law review editors and a third for law review reviewers. Each of them have five BoundColumns: username, password, full name, role and university affiliation. Law review editors and reviewers have a sixth column for their law review name. In addition, all three roles have three ButtonColumns for enabling accounts, disabling accounts, and displaying the decrypted password. Figure 4-2 displays the page ALRSS administrators use to enable accounts.

4.4 The Article Page

The Article class can be used to read articles, abstracts and cover letters of law authors as well as submission and copyright requirements for law reviews. Article is implemented so that the type of document to retrieve is encoded in the Params collection of the HTTPRequest class. In order to view the submission and copyright requirements, Article needs the Review ID of the desired Law Review. To view an article, abstract or cover letter, the Article ID must be supplied. If a user is using Internet Explorer, then the Microsoft Word document will be displayed within the browser. Otherwise, a Save As dialog box will appear and a user chooses the location to save the article.
Figure 4-2: Administrator page for enabling accounts
4.5 The Law Reviewer Page

Law Review referees use the Law Reviewer Page to view assigned articles and deadlines and post comments. The implementation of this page also allows reviewers to change their passwords and addresses. These changes are accomplished through simple SQL Updates.

The most significant software implementation within the Law Reviewer class is the ArticlesPage user control. This page permits reviewers to post and update comments for assigned articles. They can also see to which editor they report and the review deadline set by that editor. All of this information is contained in a DataGrid. This DataGrid supports sorting as well as ButtonColumns to view cover letters, abstracts, manuscripts and comments as well as upload new comments. When reviewers want to upload comments, they click on the “Edit” link and a file upload control appears below the DataGrid. Next, they browse the file system for their comments document and then upload the document to the comments column in the Article_Reviewer table.

ASP.NET makes sorting a DataGrid on any BoundColumn very simple: the AllowSorting attribute of the DataGrid and the SortCommand property must be set to the appropriate values. The headers of sortable columns appear as hyperlinks, which when clicked fire the SortCommand event. This event is accomplished by essentially re-binding the DataGrid to the output of a sort-by SQL command. A sample ArticlesPage can be found in Figure 4-3.

4.6 The Law Editor Class

The Law Editor class is one of the more complicated implementations. Law review editors can change their password and update their addresses just as law reviewers can. The implementation for these functions is very similar to those described in Section 4.5. The only significant difference is that editors have the power to designate whether or not they are the contact editor, i.e. the editor whose address is returned when an author queries a law review. In ALRSS, a law review may only have one
Figure 4-3: Page where Reviewers view assigned articles
contact editor. When an editor becomes the contact editor, the previous contact editor’s information is overwritten.

Another feature implemented in the Law Editor class is uploading new submission and copyright requirements for the law review. In order to implement this feature, the code needs a control that can browse the user’s file system and retrieve the selected file. This can be accomplished through the HTML control:

```html
<input id="fileUpload" runat="server" type="file"></input>
```

When the editor clicks the Upload button, the program will read the data from the posted file and put its contents in a byte[]. The document is then uploaded to the database using a standard SQL UPDATE. SQL Server 2000 supports a column type called **Image**, which is designed to hold binary documents (such as Microsoft Word). The byte[] created from reading the posted file is the value that is actually stored in the database.

Another function law review editors can perform is the addition of new law review editors and reviewers. The implementation of this feature is very similar to the creation of law author and law review accounts described in Section 4.2. Editors use a radio button to choose whether they are adding another editor or reviewer, enter all the other pertinent information and then the data is uploaded to the database using multiple SQL INSERTs.

Implementing workflow in the Law Editor class is what separates ALRSS from ExpressO™. When editors go to the workflow section, they can view all the articles submitted to the law review, the assigned editors and reviewers, deadlines for reviewers to review article as well as deadlines for when the law review needs to review the article, the comments reviewers have made on an article, and the current status of the article. All of this information is contained in a **DataGrid**. The data grid has six sortable columns: the article’s title, assigned editor and reviewer, internal and personal deadlines, and the article’s status within the law review.

There is one **ButtonColumn** that allows editors to control the workflow. When the link in the button column is clicked, a panel appears that consists of nine
DropDownLists, a Submit button and Cancel button. The drop down lists are data bound with all the editors for the review, all the reviewers for a review, the month, day, and year (only the current and next year) to set for the internal deadline, the month, day, and year for the personal deadline, and all the possible StatuslD1s (these values can be found in Appendix A.2 and the expected transitions in Figure 6-1). Editors can use the drop down lists to control the workflow for any of the articles. When the Submit button is then pressed, the database is updated to reflect the editor’s changes, and the data grid has its DataBind() method called again. The Cancel button, simply closes the panel containing the drop down list and buttons. An example workflow page is shown in Figure 4-4. The article with the title “Harvard Law Review Paper” is the article selected in this example.

It would appear more logical for the workflow ButtonColumn to be replaced with the type of EditColumn described in Section 4.5, and when the Edit link is clicked, the drop down lists would appear in the corresponding data grid columns. However this task is extremely difficult and complex. In order to have non-text boxes appear in editable columns, TemplateColumns need to be used instead of BoundColumns. [21] However, data binding to these template columns is not possible given the current implementation of the workflow section. Moving the drop down lists outside of the data grid provides a cleaner and more efficient implementation that is just as effective. Workflow is a significant topic in ALRSS and is thoroughly covered in [17].

4.7 The Law Author Page

The Law Author class is the most detailed implementation in ALRSS. Just like law review editors and reviewers, law authors are able to change their password and update their address using the procedure described in Section 4.5. However, law authors also need to view the submission and copyright requirements for every law review, submit articles to law reviews, view their articles currently in ALRSS and the status or each article, withdraw or accept articles, request expedited reviews, and submit new versions. While ALRSS is designed to improve the law review submission pro-
Figure 4-4: Workflow Page for a Law Review Editor
cess for both law authors and law reviews, it is imperative that law authors receive
greater functionality. The majority of the law authors are law professors whose pro-
motion and tenure depends largely on the quality of the law reviews that accept their
manuscripts. Therefore, ALRSS needs to operate in such a fashion that authors can
concentrate on writing their articles and not worry about the details of submitting
their article.

One necessary feature of the law author class is the ability to view all the sub-
mission and copyright requirements for each law review. To view the requirements, a
law author just has to go to the requirements section and an alphabetical list of law
review names and links to their requirements documents are displayed in a DataGrid.
The data binding that occurs is similar to that discussed in Section 4.3. A similar
implementation is used to view the articles that an author has in ALRSS. When an
author visits this sections, his or her articles are listed in a DataGrid with links to
the actual article, abstract, and cover letter.

An important feature of the Law Author class is the ability for authors to view the
status of their articles at each law review, withdraw articles, and accept publication
offers. Like most of the implementations, the data in this section is bound to a
DataGrid. This particular data grid enables authors to view the article title, law
review name, and article status. Authors also have the ability to sort the data grid
by those same columns. The sorting implementation is discussed in Section 4.5. This
particular implementation allows authors to accept or withdraw a specific article. An
author can withdraw an article at any point by clicking the Withdraw link in the row
of the selected article. However, if an author withdraws an article from a particular
review, that article is considered to be deleted from the system because law reviews
will stop reviewing the article and there is no way to transition from the withdrawn
state [17]. When an author has decided to accept publication in a specific law review,
he or she can click the Accept link, which will change the status of the article for
that specific law review to “Final Version”. For every other law review in which the
article was submitted, the status is changed to “Withdrawn.” Figure ?? shows a
sample view submissions page.
Automated Law Review Submission System

Submitted Manuscripts

<table>
<thead>
<tr>
<th>Title</th>
<th>Law Review</th>
<th>Manuscript Status</th>
<th>Withdraw Manuscript</th>
<th>Accept Publication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case On MIT Law Review</td>
<td>Final Version</td>
<td>Withdraw</td>
<td>Accept</td>
<td></td>
</tr>
<tr>
<td>Staff Staffed Law Review Submitted to ALRE</td>
<td>Withdraw</td>
<td>Accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Staff Staffed Law Review Submitted to ALRE</td>
<td>Withdraw</td>
<td>Accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Staff Staffed Law Review Submitted to ALRE</td>
<td>Withdraw</td>
<td>Accept</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case On Harvard Law Review</td>
<td>Withdraw</td>
<td>Accept</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Once you accept publication, your manuscript will be withdrawn from all other Law Reviews.

If your manuscript is withdrawn, it cannot be viewed by other law reviews and will be deleted from the system.

Figure 4-5: Page where Authors can view submissions of their articles
Two additional features of the Law Author class is the ability of authors to request an expedited reviews of their article once it has been accepted by at least one law review. Requesting expedited reviews begin by querying the Article.Status table in the database for all law reviews where the ArticleID is the identification number matching the current article and the StatusID is either “Accepted with Changes Required” or “Accepted with No Changes Required.” If the number of law reviews returned is greater than zero, then authors are able to request expedited reviews.

Authors eligible to request an expedited review are presented with a check box list of all other law reviews to which the article has been submitted. In addition, there are three drop down lists at the top of the page for the author to set the deadline by which the other law reviews need to perform the expedited review. After the author has set the deadline and chosen the law reviews from which expedited reviews are requested, he or she submits the information to the system. For all the law reviews selected, the StatusID of the article at those reviews will be changed to “Request Expedited Review” and the expedited_review_deadline_requested_by_author column will be set to the deadline selected by the drop down lists.

If an author’s article has been accepted by a law review, the author is then allowed to upload a newer version of his or her article to the law review. Determining whether an author can submit a newer version of his or her article is implemented almost identical to determining whether an author can request expedited reviews. If an author is allowed to submit a new version of his or her article, he or she is presented with three HTML inputs (with the type attribute set to “File”) to upload new versions of the article, abstract and cover letter to the Version table, which changes the StatusID of the article in the accepting law review to “Version.” Articles with StatusIDs of “Version” are not eligible for further versioning. In order for a new version to be submitted to the same law review, the law review would have to change the StatusID of the article to “Accepted with Changes Required.”

The most important implementation in the Law Author class is the code for submitting articles to law reviews. Authors must first use the three HTML inputs to select their article, abstract and cover letter. Next, they need to enter the article
title and choose a category for their article. The possible categories are contained in a drop down list that is data bound to the Subject database table. Finally, authors need to select the law reviews to which they want their articles submitted. This proves to be the most interesting implementation of the class.

At ALRSS’s peak, it will have approximately 390 law reviews to which authors can submit articles. ALRSS has to allow authors to submit their articles to each law review exactly once, however, authors need to be able to submit articles to different reviews at different times. To satisfy all the requirements and make it easy for users to scan through all 390 law reviews, the reviews are loaded dynamically into the page as a check box list that is divided into alphabetical sections. Before the controls are loaded, a SQL query is issued to determine which law reviews have an article under review for the current user. These law reviews are loaded into the list but are disabled and have the message “Already Submitted to this Review.” In addition, when law reviews are loaded into the list, a hyperlink appears next to the law review name. The link retrieves the submission and copyright requirements for that particular law review. A sample submission page is shown in Figure 4-6.

After the author clicks the submit button, multiple SQL INSERT statements are issued, which inserts data into the database tables ArticleAuthor, Article, Article_Status, and Article_Reviewer. It is necessary to point out that when an article is submitted to a law review, ALRSS automatically assigns an editor and reviewer for that article. Not assigning an editor and reviewer would break the Workflow and ArticlesPage implementations described in Sections 4.6 and 4.5 respectively.

One common theme throughout the ALRSS implementation is bringing the power of a relational database and the convenience of a web interface to its users. The implementation allows users to transparently execute complex and powerful SQL queries and updates. In addition, ALRSS contains a workflow implementation that separates itself from the competition. To obtain any of the code for ALRSS, please send email to davegot@alum.mit.edu detailing the request.
Figure 4-6: Article Submission Page
Chapter 5

Results

The success of a software application can be measured by three factors: cost, schedule, and quality. Staying under budget, meeting the estimated release date, and delivering the application that conforms to the agreed upon requirements is essential for business applications. Furthermore, these three factors are not always independent. If a software project is significantly overrunning its schedule, then the stress of getting it delivered as soon as possible often results in increased cost (through overtime payments, contractors, and/or worker turnover) and decreased quality (requirements skipped or not tested thoroughly).

Based on these three factors, ALRSS was a success. ALRSS was completed on time and within budget as well as meeting the requirements stated at the beginning of the project. As with most software applications, there are possibilities for improvement, which are discussed in Chapter 6. The topic of requirements and how ALRSS met them is thoroughly discussed in [17]. The budget for ALRSS allowed for the purchase of software. The development team chose Microsoft Visual Studio .NET, Microsoft SQL Server 2000, and Microsoft Visio 2003. The rest of this chapter will focus on completing the application within the schedule estimate.

Deriving a schedule estimate for a software project is a crucial step in the software development process. Despite its importance, many organizations make “seat-of-the-pants guesses” and most projects “overshoot their estimated schedules by anywhere from 25 to 100 percent [11].” Drastic schedule overruns usually result in revenue
and job losses. Creating a good schedule estimate is a difficult task. However, there is a reasonable, process available to create an accurate development schedule. The process consists of three steps: ** Estimating the product size** (number of lines of code or function points), **Estimating the effort** (man-months), and **Estimating the schedule** (calendar months) [13].

### 5.1 Estimating Product Size

Estimating the size of a product is the most difficult step (and is usually skipped). If the product is similar to an existing product with the organization, then one can estimate the size of the new product as a percentage of the size of the similar product. When a product is being developed for the first time within an organization (as opposed to an upgrade), size estimations have to be made based only on requirements and design. In this case, size estimations can be made using an algorithmic approach, such as function points, or size-estimation software. However, Steve McConnell notes that “most software programs and some of the algorithmic approaches require that you calibrate the estimation practice to your environment before you use it. Accurate measurement of historical projects is key to long-term success [13].”

ALRSS was a unique product for the development team. Therefore, we needed to make size estimates based on our requirements [17] and software design [8]. We decided to use function points (an algorithmic approach) as the means of product size estimation. There are numerous methods for counting function points; the one we used is close to the “1984 IBM Method [10].” In this method, function points are based on the number and complexity of: **inputs, outputs, inquiries, logical internal files, and external interface files.** These terms are defined in [13] and summarized in Table 5.1. The resulting function point count forms the *unadjusted function-point total*, which is then multiplied by the *influence multiplier* to create the *adjusted function-point total*. The influence multiplier is based on 14 factors that influence a program and ranges in value from .65 to 1.35 [13]. ALRSS has an influence multiplier of 1, therefore, the unadjusted and adjusted function-point totals are equal.
For experienced software development/management teams, function points are sufficient for estimating the size of a product. However, function point are not a meaningful description of product size to most people in the software industry. For these people, number of lines of code is the standard product size measurement. Fortunately, there exists conversions between function points and lines of code. Furthermore, function points are not useful if they are not accompanied by the development language. For example, 50 function points in macro assembler would take about 3 or 4 times as long to implement as 50 function points in C++ [13]. As mentioned in Sections 3.2.1 and 3.2.3, ARLSS uses C++ and SQL Server 2000. This translates into 50 lines of code per function point and 30 lines of code for every database table. ARLSS was estimated to have 231 function points and 15 database tables, which results in an estimated 12,000 lines of code. The specific calculations can be found in Tables 5.2 and 5.3.

5.2 Estimating Effort

Once the size estimate is complete, the next step of the process is to estimate the effort required to develop the product. The effort is measured in man-months, which is essential when deciding how many people are required to develop the product. There are a variety of approaches to calculate the effort estimate from the size estimate including: estimation software, conversion tables, historical data, and complex algorithms (as used by COCOMO [2]).

The effort estimation for ARLSS was 5 months, which was calculated using a table that converts lines of code to man-months. The specific table used was based on an efficient schedule that assumed: talent from top 25%, low turnover, competent management, staff available when needed, minor requirements changes ( 5%), and that the tools used are effective [13]. Using the efficient schedule table, a best-fit quadratic equation was calculated to represent the relationship between lines of code and man-months. From the equation, 12,000 lines of code corresponds to 5 man-months. More information on this particular table as well as other tables can be
5.3 Estimating Schedule

The estimated time to develop the product can be calculated from the estimated effort. As with estimating size and effort, there are many different approaches, including estimation software, historical data, conversion tables, and equations. We computed the schedule length using two equations, the Software Schedule Equation \[ SM = 3.0 \cdot MM^{1/3} \] (5.1)
and Jones’ First-Order Estimation Practice [12]
\[ SM = FP^a \] (5.2)
and Table 5.4. In both equations, \( SM \) is the schedule length measured in months. In the first equations, \( MM \) is the effort measured in man-months and \( a \), which is determined by the type and quality of the product being developed, is obtained from a table in [12], For ALRSS, \( a = 0.42 \). Using these two equations, the estimated schedule length was 5.12 months and 9.83 months respectively. From Table 5.4, the estimated schedule length was 4.9. A closer look into the derivation of each equation revealed that the Software Schedule Equation was more realistic for ALRSS. Jones’ equation is based on an analysis of several thousand software projects and we felt that ALRSS was developed under more favorable conditions than most of the projects used in Jones’ analysis. In addition, Jones’ equation provides a rough schedule estimate that is more of a reality check [13]. Therefore, we believed that the Software Schedule Equation was a better estimate than Jones’. Furthermore, the Software Schedule Equation and Table 5.4 return similar values, which strengthens the position that the estimated schedule should be around 5 months.
5.4 Actual Schedule Results

As mentioned earlier, the development of ALRSS was completed within most of the schedule estimates. The project contained 10,600 lines of code, the overall effort was 10 man-months, and the development length was 4 months. When counting lines of code, only blank lines and comments generated by Visual Studio.NET were omitted. The only value outside the estimate was the overall effort. Based on the estimates for effort and schedule length, only one person would have been needed to develop ALRSS (5 man-months / 5.12 months = 1 person). With 3 people developing ALRSS, the project was completed faster, but the total effort expended was increased. We believe this is because the project size did not justify 3 developers for minimal overall effort. However, in order to meet the tight time constraint, a less efficient deployment of 3 developers was needed. A summary of the results can be found in Table 5.5. It is important to note that ALRSS has neither gone through the exhaustive quality assurance testing period nor contains the extensive documentation as would a commercial product. The second release of ALRSS will have undergone a rigorous testing period and will be well documented.

5.5 Installation

Currently, ALRSS resides on a single web server. If ALRSS needed to be moved to a different web server, the installation would have to be done manually. However, only a few changes would be required in order to create an install program. ALRSS is configured to run specifically on its current webserver. The code would have to be modified so that any code designed for the specific web server, would now apply to any Windows-based web server running SQL Server 2000\(^1\). Next, all the files used in ALRSS would need to be bundled together (e.g. a zip file) and include an install script. The install script would then copy all the files to the appropriate directories. In addition, a README file would be included to help users with the installation.

\(^{1}\)ASP.NET only runs on Windows machines and ALRSS is only designed to work with the SQL Server 2000 database
<table>
<thead>
<tr>
<th>Program Characteristics</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Number of inputs</td>
<td>x 3</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>x 4</td>
</tr>
<tr>
<td>Inquiries</td>
<td>x 3</td>
</tr>
<tr>
<td>Logical internal files</td>
<td>x 7</td>
</tr>
<tr>
<td>External interface files</td>
<td>x 5</td>
</tr>
</tbody>
</table>

Table 5.1: Function-Point Multipliers, Adapted from [10]

<table>
<thead>
<tr>
<th>Program Characteristics</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Number of inputs</td>
<td>24x3=72</td>
</tr>
<tr>
<td>Number of outputs</td>
<td>6x4=24</td>
</tr>
<tr>
<td>Inquiries</td>
<td>21x3=63</td>
</tr>
<tr>
<td>Logical internal files</td>
<td>0x7=0</td>
</tr>
<tr>
<td>External interface files</td>
<td>0x5=0</td>
</tr>
<tr>
<td>Unadjusted function-point total</td>
<td></td>
</tr>
<tr>
<td>Influence multiplier</td>
<td></td>
</tr>
<tr>
<td>Adjusted function-point total</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Function Point Totals for ALRSS

<table>
<thead>
<tr>
<th>Function Points</th>
<th>Language</th>
<th>Multiplier</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP.NET Application</td>
<td>231</td>
<td>C#</td>
<td>50</td>
</tr>
<tr>
<td>Database</td>
<td>15</td>
<td>T-SQL</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Number of Lines of Code for ALRSS

<table>
<thead>
<tr>
<th>System Size (lines of code)</th>
<th>Schedule (months)</th>
<th>Effort (man-months)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>4.9</td>
<td>5</td>
</tr>
<tr>
<td>15,000</td>
<td>5.8</td>
<td>8</td>
</tr>
<tr>
<td>20,000</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>25,000</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>30,000</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>35,000</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>40,000</td>
<td>9</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 5.4: Efficient Estimation Schedules for Business Products (Adapted from [13])
<table>
<thead>
<tr>
<th></th>
<th>Estimated</th>
<th>Actual</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (lines of code)</td>
<td>12,000</td>
<td>10,600</td>
<td>-11.67</td>
</tr>
<tr>
<td>Effort (man-months)</td>
<td>5</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Schedule (months)</td>
<td>5.12</td>
<td>4</td>
<td>-22.4</td>
</tr>
</tbody>
</table>

Table 5.5: Estimated and Actual Development Schedule
Chapter 6

Future Work and Conclusions

After most products are initially released, there is a period where developers can review and enhance the existing code as well as add new features based on user feedback. Upon the completion of ALRSS, the development team reviewed the code base as well as comments received from initial users of the system. Based on the information gathered, we have generated several improvements to be implemented in the next release of ALRSS.

6.1 Code Rewrites

In the next release, all SQL queries that appear in C# code would be moved to the database as Stored Procedures. Stored procedures increase performance and maintainability in ASP.NET applications. SQL Server optimizes queries issued to the database and stores them in a compiled state, which execute faster than when they are submitted as SQL statements in C# code. “The SQL managed provider also talks to SQL Server in its own language, which shrinks the front-to-back time required to get a query through and data back to your client [9].” Furthermore, stored procedures separate database code from ASP.NET code. Changing the procedure’s implementation be transparent to the application, providing an equivalent value is returned. Finally, moving SQL queries to stored procedures makes the ASP.NET code more readable.
Another change that will appear in the next ALRSS release will be the relocation of field validators to the database. Currently, data entered into text boxes and file uploads are checked by an ASP.NET validator before the data is submitted to the database. ALRSS has many inputs that require validation and the implementation of all the validators makes the ASP.NET code hard to understand. Furthermore, validation within ASP.NET requires more communication between the client and the server, which affects performance. By providing rules within the database, the input data can be sent to the database, which will perform the validation. The disadvantage would be that ASP.NET provides better formatting for errors caught by its validators. However, the increased code readability and performance is more important than the aesthetic loss.

6.2 Small-Scale Additions

Based on user feedback, we believe that ALRSS could benefit and be more competitive with a few additional features. The first enhancement to be included is the ability for users to search law reviews by category. These categories are the same as those used when searching manuscripts (see Chapter 4). Law reviews have a many-to-many relationship with the categories. This means that a single law review can be in many categories and each category can contain many law reviews. Currently, law reviews and categories are represented by the Law-Review and Subject tables respectively. Implementing this feature would require a change to the data model. To satisfy the many-to-many relationship, an intermediate table would need to be created between the Law-Review and Subject tables. This intermediate table would have a composite key consisting of the primary keys from the Law-Review and Subject tables. Therefore, knowing the review name and category would create a unique pair. This enhancement will permit a user to ascertain all the categories in which a law review belongs and/or all the law reviews contained in a certain category.

Another important feature that will be added is a safeguard against law review editors transitioning manuscripts to states that do not conform to ALRSS's state
diagram (see Figure 6-1). When an editor uses the workflow section, he or she has
the ability to set the status of the manuscript to any status contained in the drop
down list (see Figure 4-4). It is possible that editors can accidentally change the
article status to an unintended state that could cause confusion and/or problems (e.g.
setting the article status to “Withdrawn” instead of “Version”, which could cause the
manuscript to eventually be deleted from ALRSS a daemon that runs periodically). In
the next version of ALRSS, we will code the state diagram into the system. Whenever,
an editor attempts to change the status of a manuscript, there will be a method that
accepts the current status and new status as arguments and returns whether the
state diagram permits the transition. If an illegal transition is about to be made, the
system will warn the editor about the transition and ask if he or she wants to proceed.
In the future, it is possible that illegal transitions will not be allowed in ALRSS.

6.3 Complex Additions

In order to better compete with ExpressO™, ALRSS will need to support issue
numbers in the next release. Most law reviews publish papers multiple times a year
and not at the same frequency. ALRSS has to obtain the issue dates for each law
review either by directly contacting the law review or providing a form in the system
for law reviews to submit their issue dates. The data model would also be affected by
supporting issue numbers. The Article_Status table would need a column to hold the
issue number for which the manuscript was submitted. Also, the Law_Review table
would need a column that contains the current issue number. In supporting issues,
editors would need to be able to control workflow for each issue. One possibility would
be to add another column to the workflow DataGrid for issue number and allow the
column to be sorted by issue number. However, the volume of manuscripts submitted
to each issue would likely be too much for one page. A more realistic approach
would be to create separate workflow DataGrids for each issue and dynamically show
only the one the editor chooses to view. Similarly, the articles page (see Section
4.5) would also have to support issues in the same fashion as the editor’s workflow
section. ALRSS would also be required to guarantee that the manuscripts submitted are for the current issue of each law review. In addition, if the issue is full, authors should either be prevented from submitting a manuscript to that review (which is the ExpressOTM policy) or submit the manuscript to the next issue. ALRSS would most likely implement the latter, which we believe is superior, unlike ExpressOTM.

A final implementation plan for the next ALRSS release is a ranking of law reviews that is based on ALRSS statistics. ALRSS could keep track of the number of manuscripts submitted to each review. Reviews that receive the most manuscripts can be viewed as more desirable than others (though not always). Another useful statistic is the acceptance rate of a law review (the number of manuscripts accepted divided by the number submitted). Usually, the most selective law reviews are the best (colleges are sometimes ranked this way). Another useful, although difficult, data to collect and measure is which law reviews are chosen by law authors over other law reviews. This can be measured through the number of expedited requests a law review receives. Since expedited reviews can only occur when a law author is accepted by at least one law review, it is necessary to keep track of the accepting law review in addition to the law reviews that get the expedited review requests. Law authors want their manuscripts accepted at the most prestigious law review in order to ensure tenure and promote their status within the law community. Measuring these last few statistics accurately would most likely lead to a very good ranking of the law reviews because the authors care the most about the quality of the law review and will accept publication in the highest ranked law review.

6.4 Conclusion

ALRSS was designed as a solution to the law article submission system problem. As implemented, ALRSS satisfies the conditions to function as a submission system for law articles. Furthermore, ALRSS needed to provide features that would enable it to enter and compete in an area with a pre-existing application. The workflow and articles sections for law review editors and reviewers respectively, distinguish
ALRSS from ExpressO™ and should generate interest from all members of the law article community. Finally, we have outlined potential and reasonable improvements to ALRSS that could be performed by the current development team or any other team that might take over development.
Figure 6-1: State Diagram of ARLSS
Appendix A

Class Members

A.1 SubjectID Members

1. Administrative Law
2. Child Law
3. Civil Procedure
4. Constitutional Law
5. Construction Law
6. Contracts
7. Corporate
8. Criminal Law Procedure
9. Elder Law
10. Evidence
11. Family Law
12. Labor Law
13. Law and Economics
14. Litigation
15. Maritime and Admiralty Law
17. Property Law-Real Property
18. Property Law-Real Estate Transactions
19. Tax Law
20. Torts
21. Trusts and Estates

A.2 Article State Members

1. Submitted to ALRSS
2. Received by ALRSS
3. Not Received by Law Review
4. Received by Law Review
5. Under Review
6. Request Expedited Review
7. Under Expedited Review
8. Denied
9. Withdrawn
10. Accepted with No Changes Required
11. Accepted with Changes Required
12. Version
13. Final Version
Bibliography


