Modular Application Framework for Web Applications

by Aaron Halfaker

902 5th Ave S.
Virginia, MN 55792

aaron.halfaker@gmail.com

Abstract

This paper argues that good object-oriented practices have been almost entirely ignored in the traditional development of web applications. Most of these applications are incompatible with some of the most praised development and design techniques like modularization, abstraction and unit testing. The purpose of this paper is to outline a modular framework that brings good design practices to web applications in a manageable way.

This paper proposes a modular framework that uses readily available technologies such as XML, XSL, MySQL and PHP. The author suggests this system will alleviate the common problems encountered when web applications are developed and maintained. Included is a description of testing and implementation of this modular framework in the form of a used textbook database and a description of the future research plans related to this framework.

Introduction

The complicated nature of web applications has been addressed consistently by the development of server side languages (like PHP, JSP and ASP) with the addition of new tools, libraries and language structures. It seems, though, that good object-oriented practices have been almost entirely ignored when web applications are developed (Wallace, 2000). Most web applications are incompatible with some of the most praised development and design techniques like modularization, abstraction and unit testing. The purpose of this paper is to outline an application development framework whose purpose is to bring good object-oriented (OO) design practices to web applications in a manageable way.

This framework is constructed using tools that are already available and familiar to most web developers including XML, XSL, MySQL and PHP. These tools were used both selected to decrease the learning curve and to take advantage of, arguably, the most powerful languages that are freely available.

This paper outlines the individual problems with traditional web development, including the limited development style, difficulty of structured testing and accelerated software entropy. It will then explain why the layer process of the modular framework is an apt solution to the proposed problems.

Problem Definition

Obfuscated Design and Development

Single mind design style

One of the frustrating factors of traditional web development is that the developers must be familiar with the entire system while they write code. Design of a system is done by one person or a team of people who meticulously iron out all details of the system before development begins. This design style can lead to serious problems in even basic applications because it is impossible to plan for the *entire* system before development begins.

It is for this reason that large amounts of code are often thrown away and rewritten. Architectural changes either lead to a complete redesign of large parts of the system or the failure of the project.

Logical mixture

Most web applications are developed with the inclusion of several languages in the same document. The use of these languages is required because they perform the most basic functionality that is common among web applications: connection to a database and presentation of HTML to a web browser.

SQL – Most web applications are developed with the intent to connect with a database. Structured Query Language (SQL) is the most popular language used to create, modify and retrieve data from relational database management systems(RDBMS, the most commonly used database system ("SQL", n.d.). The server side language accesses the database by passing SQL strings to the RDBMS server and receiving the resulting data. These strings are built within the server side application language and this causes the languages to be intermixed.

HTML – This is the presentation language used by most web applications to create a user interface. It is commonly written within the server side languages in all areas of the code. Most commonly, HTML is output from within conditionals, before and after loops and most anywhere the developer can see a benefit from outputting HTML during presentation.

Server Side Language – Most commonly ASP, PHP or JSP, the server side language serves as glue between database (SQL) and presentation (HTML). In this paper, I will refer to its most common uses as the application layer (AL). Its purpose is to perform calculations and organize data to output to a web browser.

The mixture of these languages requires the developers to not only be familiar with the workings of all languages and systems in order to work with a single script, but also they have to be aware of the effects that their changes to one part of a system may have to another.

Application and presentation mixture

Most web applications are designed with the presentation (HTML) mixed within the application code (server side language). There are tools designed to separate HTML from a server side language, such as the template include system used in the PHP Bulletin Board (“phpBB.com”, n.d.). However, this system uses a series of server-side includes in place of every instance HTML would need to be output.

Even the template based systems that are designed to separate the HTML from the application, like phpBB's, fall short as they do not allow the presentation designer full control of the display of the data inside the presentation (“phpBB”, n.d.). For instance, if I were to rewrite a template file for one of the tables on the page, I may be able to affect the way it is shaped and positioned, but I would not be able to affect the order in which it appears in the resulting HTML document. This is a serious issue that limits the designer of the web application's user interface.

Testing

Managing input and output

In traditional web development, the most basic level of input and output that can be tested is an entire script. This makes testing a difficult task that is only done after the entire script is completed. To further increase the difficulty of debugging a script, the output is solely based on the presentation of the script in a web browser. The lack of standard compliance in today's browsers makes it difficult to predict the visual results of the output of a script. The actual text/HTML output is all that can be tested confidently, because the rest of the process is up to the user's unpredictable browser (Wallace, 2000).

Why doesn't testing happen?

There are many reasons that good testing practices are not applied to most web applications. Standard techniques like unit testing are difficult to implement. Documentation may need to be formatted different for each script. This is because of the lack of proper use of good OO techniques to separate logical units and because documentation is interspersed throughout the code. All in all, documentation and testing of a web application is difficult and non-intuitive. It is not much of a surprise that it is poorly implemented.

Maintenance

Difficult feature addition

Adding features to a traditional web application is not a menial task. Usually when one makes a change to a system, that change has to be reflected in several other scripts. If it changes the way the core application works, the application will most likely have to be redesigned. Most often, the designer will try to stay as close to the previous application base as possible in order to limit the changes that have to be made.

Frequently, the designer or developer that has been given the task has long forgotten about how the system works or was not involved with its initial development in the first place. This situation results in a lot of time that has to be spent learning the inner workings of the program whose behavior is probably not well documented (Beck, 1999).

Once the new functionality is added to the system, it will most likely go undocumented. If the developer does not have a grasp for the entire system and they aren't sure that the documentation is up-to-date, they will most likely leave their changes with the most minimal documentation if any. If documentation is to be kept up to date, there is no way of documenting only the individual part of the system that they changed, but instead the maintainer must change documentation for all affected scripts.

Limited documentation

For any programmer, the importance of documentation should go without saying. Good documentation not only facilitates the maintenance of code and increases an application’s lifetime, it often can increase the quality of the code to which it applies (Beck, 1999).

Documentation of a web application is often limited to some comments available in the source code. PHPBB, one of the most popular open source web applications, does not include documentation for any modules other than instructions about how to write new ones. Each script is only required to have a comment that contains the date of the creation of the script and its last update (“phpBB”, n.d.).

One reason behind the lack of documentation is the non-intuitive nature of attaching code to its explanation at any more than the most basic level, such as including comments in correspondence with their line of code. Even these comments are often inconsistent in any project that more than one developer has contributed to.

Deep bugs

In a traditional web application, initial testing only begins once an entire script is written. This leads to the discovery of many initial bugs. Thousands of lines of code in a script can be written before testing then takes place. This combined with the lack of debugging tools makes it difficult to discover coding defects. Also, maintenance and repair of one bug most often leads to other bugs buried deep within multiple scripts, because interfaces between units of the system are not commonly strictly defined.

Solution Requirements

This section contains criteria based on the problem definition outlined above that will be used to analyze the modular framework and to assess its qualifications as a solution.

The system must be modularized.

Feasibility of simultaneous development

Developers must be able to work independently based on strictly defined interfaces. The major layers of the application (database interaction, application and presentation) should work independently, and therefore, be able to be developed simultaneously. The layers of an application will use these interfaces to standardize their interaction.

Separation of logic and languages

Languages and their corresponding logic must be separated into modules. SQL is designed to retrieve data from a database. Server side languages are designed to organize and calculate data. HTML is designed to present data in a user-friendly way. Each of these languages are responsible for different layers of a web application and must bust be separated accordingly.

Seamless changes to user interface

Beyond modularization of the system, the user interface must be interchangeable with no modifications done to the application. The presentation layer must have absolutely no limits, beyond those of the presentation language itself, to style and order the data to be presented within it.

Testing must be integrated into development

Ease the introduction of modern testing techniques

The framework must make modern techniques like unit testing feasible. The units of testable code must be small enough to limit the amounts of defects that can develop before a the unit can be tested (Beck, n.d.). This requires good OO design with interfaces explicitly drawn.

Tie documentation and testing closely

A Design-by-Contract style system must be in place in order to tie documentation to the testing.

Documentation must be written to strictly define the operations of each unit, then testing must take place for each unit based on that definition. Above all, testing must become an essential part of the development process (“Design”, n.d.). Therefore, testing can be done through the development lifecycle, including bug repair and feature addition.

Maintenance must be a simplified

Ease the addition of features

Feature addition and other changes to the architecture must require minimal amounts of changes in the overall code. In order to achieve this, the system must be modularized so new features would require redesign of a module as a worst case scenario rather than the entire system. A developer must be able to confidently make minor additions to a working system confidently without understanding the entire system’s design.

Increase the amount and quality of documentation

Documentation must include, but not be limited to, a detailed API that strictly details the behavior and requirements of each object. This API must not only be available to the developer from the source code, but also be converted into an easy-to-read format outside of the source code files such as PDF, HTML, or CHM (Microsoft Compressed HTML Help files).

This documentation should be as easy as possible to maintain and alter from the source code. Preferably, it should exist in only the source code and be extracted to other formats so that only one copy needs to be maintained.

Separate modules for testing purposes

Since there are many hurdles to cross when finding defects in a server side language, such as a lack of proper debugging tools, the results of execution must be testable at each layer. For example, the results of a database query should be extractable in a raw format for development purposes.

The Layered Process

The layers in the Modular Application Framework are designed to progressively limit and calculate the data required by a user accessing a script. Information trickles down through the system beginning in the database management system and the database interface layer. This data is calculated and organized by the application layer. It is then, finally, formed into a presentable web page by the presentation layer. This process is outlined in Figure 1.

1. The RDBMS contains the vast amount of data that represents the memory of the application. It can store, manipulate and return this data.
2. The Database Interface Layer (DIL) contains the logic to perform queries against the database

that are required by the Application Layer.
3. The Application Layer (AL) organizes and performs the necessary calls to the DIL, calculates the data required by the current script and forms it into an XML document.
4. The Presentation Layer (PL) supplies an XSL document containing the information required to properly format the XML document into the required web page.
5. The formatted web page is sent to the user's web browser by the web server.

Database Interaction Layer

Summary

The DIL is designed for the specific purpose of managing the data necessary for the application. It contains all logic necessary to connect to, select from, insert into and update the database as necessary for the application. This layer's purpose is to separate the AL as far as possible from the database and SQL required to use it.

The Database Plugin works as an abstraction layer to the database. In this plugin, are two simple classes whose purpose is to mask the specific functionality used to connect to a particular database. This allows different plugins to be written that offer connectivity to different database systems without having to change any other code within the application.

The Table Manipulator is the largest part of the DIL. Its main purpose is to create and format SQL code to be executed against the database. The Table Manipulator's interface is characterized by member functions and produces output by returning Result objects.

Interface

Input
The DIL works with the database plug-in to connect to the RDBMS to interact with the database. This plug-in masks the database being used in the simplest way possible. A generic database class is formed to hold the database specific logic required by the system. This allows different plug-ins to be written for different databases. The rest of the application will not know which plug-in is being used and therefore, databases could be interchanged as seamlessly as possible.

Output
This layer's output interface consists of a standard class structure for each table, called a Table Manipulator. Each Table Manipulator has a standard set of member methods and variables that are inherited from the Table Manipulator interface. Some manipulators have expanded capabilities. These capabilities are most often member functions that are specific to the table that their object represents.

Standard methods for every Table Manipulator are add, clear, get, rm (remove), search and update. These allow basic access to each table and are available for use with every table's Table Manipulator.

An example of extended functionality is the add_from_reg() method found within the user Table Manipulator. This method is used to add a record from the registration table to the user table. This is commonly used functionality for the user table, but not common for any other table. Hence it exists as an extended functionality specifically for the user table.

Application Layer

Summary

The application layer manages calls made to scripts on the web server, including user authorization and formatting and calculation of data. Each Script is a subclass of PageGenerator, a class designed to generate an XML document representing the data necessary to view a web page.

After the Script creates the XML document, the Transformer object is called. The purpose of the Transformer is to find the XSL document that is appropriate for the current script. This XSL document is then executed against the generated XML document. The result is an XHTML document that contains the data from the generated XML document and is formatted into a web page by the XSL document. An appropriate formula would look like:

XML Document + XSL Document = HTML Document

Interface

Input
The application layer relies on two inputs, the result objects returned by the DIL and the XSL document contained in the PL. It is the AL's responsibility to use these two inputs to produce a well formatted HTML document

Output
The application layer produces one important output, the HTML Document. This is the document is returned to the user's browser by the web server. It represents a visual interface with which the user can interact. If the AL is doing it's job, this web page will allow the user to make calls for information, navigate the website and update information within the database.

Presentation Layer

Summary

The purpose of the presentation layer is to provide a means for formatting the pure data (in XML form) that has been created by the AL. It achieves this objective by using XSL stylesheets that have been developed to take advantage of the data returned by the AL.

This layer separates responsibility for the user-interface design from the AL (which usually caries this burden in more traditional applications). This layer is incredibly effective because XSL stylesheets were expressly designed for the purpose of transforming XML documents.

This layer is very unlike the others, because it is not accessed by a programming interface. Although there is limited programming capabilities inherent within the XSL stylesheets, an interface did not need to be designed since the XSL translator itself is designed to accept raw XSL.

Interface

Input
This interface has no required input as it draws its information from stylesheets written by designers.

Output
This interface's outputs are the XSL stylesheets that are used to generate the user interface and the media required by that interface. The XSL stylesheets are passed to the XSL transformer within the AL, while the media is passed directly to the user's browser.

The Modular Application Framework, as a Solution

Modularized System

Simultaneous development

As its name suggests, the Modular Application Framework's initial objective was to separate a web application into modules. These modules are designed to do a very specific job as independently as possible with their only interaction being between their various interfaces.

There are two major interfaces that have to be accounted for before development can begin. The database interaction interface (Table Manipulator) and the presentation interface (XML Document). Once these interfaces can be decided upon by developers, development of their individual function can be left up to the individual designer and his/her module. Although all pieces must be in place in a minimal sense before a functional system can come together, development can take place independently and simultaneously.

Logical Separation

The framework is designed to separate languages and their purposes into layers. This is meant to limit certain areas of an application for specific expertise.

For instance, a developer who specializes in database interaction and SQL can design and develop the database manipulation layer. That developer would only have to understand a small amount of a server side language in order to encapsulate the necessary logic.

A developer who specializes in data organization and calculations can design and develop the application layer. This developer would only have to know the server side language well (which would normally be required) and know only how to satisfy the presentation interface requirements with XML. This developer would not write any XML, but would rather build a multiway tree, a common data structure, that would be converted to XML for use by the presentation layer.

A designer who specializes in the user interface could work independently from all languages except XSL and XML. The XML is simply provided by the application layer, so the user interface designer needs only a rudimentary knowledge of XML. The XSL will be written by the designer in order to format the user interface properly.

Because of this separation, a developer working within each layer of the application only needs to know the language that is required for that layer and a little bit about how his/her interfaces work. It is only required that one developer understand the capabilities of all languages in order to aid in the design of the framework's interfaces. This developer would need to be a jack-of-all trades but a master of none since the developers in each layer of the application can supply the detailed knowledge and skills.

Absolute separation of presentation from application

Through the use of XSL Transformations, this framework has the unique ability to separate the user interface of an application almost entirely from its inner workings. The XSL "skin" developer must only know of the nature of the XML document that will be accessed in order to write a functional interface. This allows the developer the freedom to mold information in any fashion that he/she sees fit.

This also allows more than one of these skins to be developed separately and switched between seamlessly. The application only needs to know where to find the proper XSL document. It may otherwise be completely unaware of the user interface.

Testing

Modern testing techniques

The Modular Application Framework is specifically designed with the idea of abstraction in mind, in that the goal of the framework is to handle complexity by breaking the system into its simplest parts and strictly defining their interactions.

Because of this careful abstraction throughout the system, testing is a simple and intuitive process. Since the framework requires the intended behavior of all objects to be strictly documented, testing the objects and systems proves to be a simple matter. All a tester must do is write a test to make sure a method/object meets the requirements that have been defined for it in the documentation.

Interdependent documentation and testing

In the Modular Application Framework, it is required that a developer write the documentation for an object or method before making it work for two important reasons. First, it has been proven to help developers plan their work before they begin (Beck, 1999). Second, it lays down the ground rules for testing before code is written. This has proven valuable to the process because it requires the developer consider the module's testability during the development process.

This, of course, has the convenient side-effect of making documentation required even before the code is written. Once this documentation is in place, it can always be referenced. Even if a developer were to leave a module half-finished, another developer could complete the module from the original developer's documentation.

Maintenance

Easy addition of features

The framework is built on a structure of standard interfaces. Because these interfaces can be built on, but not limited, they open the way for feature addition by later developers. Since code is encapsulated and re-used, adding a new functionality like adding an additional field to a query would be a matter of editing 1 or 2 lines in 3 files at most.

After a feature is added, one unit test should need to be changing. All related tests should then be able to run. This makes spotting regression errors a quick and easy process. If the tests that run before fail, a developer will know exactly where a defect is by which tests fail.

Single edit documentation

The PHPDoc documentation style (based on Javadoc) that is used in the Modular Application Framework requires only one version of the documentation to exist in the source code of an application. That documentation can later be extracted to form a navigable representation in HTML, CHM or PDF by an open source application called PHPDocumentor.

Documentation is easy to update and maintain because it exists in only one place, the source code, from which it can be extracted. Changes can be made to the documentation for a specific object, method or module in the same file as the source code. This puts the documentation in the easiest place possible for the developer to update and future developers to find.

Simplified testing and bug fixing

The modular style of the framework makes potentially deep bugs shallower. This is because it allows you to break the application apart into fully functional modules. The unit tests for these modules can then be run at increasingly simpler levels to flush out the coding defect.

Each layer of the framework has special functionality that allows a developer to examine the raw output of work at that level. For instance, if the developer believes that the cause of a bug is in the application layer, the presentation layer can be removed and the XML output of the application layer can be examined.

An exception system is also required by the system to force a defensive programming style. This system of exceptions allows more detailed error information to either be displayed or logged invisibly and allows the application to recover as gracefully as possible. This exception system should make bugs easier to locate, track and recover from.

Modular Application Framework

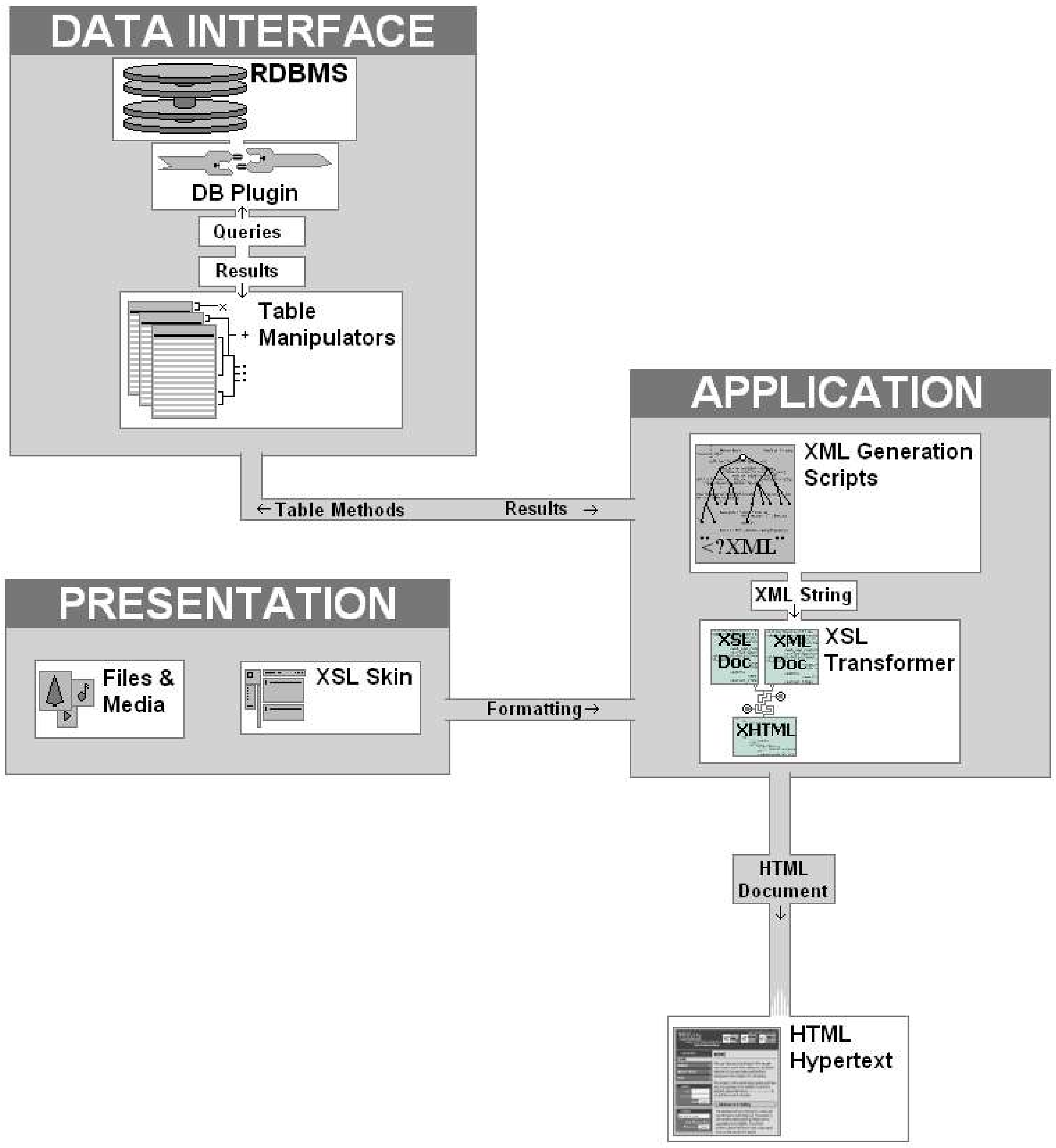

Figure 1. Layers of the modular framework

Future Work

Currently, the system has been implemented in the form of a used textbook database for use at The College of St Scholastica. The system has been developed and tested in a LAMP (Linux, Apache, MySQL and PHP) environment. It has not been released to the public yet, but is nearly complete and functional. It is currently working properly on the modular framework and serving web pages. The documentation for this system is available at
http://woot.css.edu/half/online/developing_projects/bookdb_map/.

What is next?

Modular/extensible configuration files

The current system of configuration files is limited and not user friendly or human readable. The implementation of an XML-based configuration system could alleviate this issue. It will most likely be an addition to the system in the near future.

Quantifiable comparison

This is the most important next step for acceptance as a viable alternative to traditional web development. Due to time and funding, it was not possible to generate quantitative data about the maintenance cycle and cost of the modular framework.

Such a comparison would require considerable data about traditional development and development within this framework over a multitude of applications and ability levels of developers. To date, only qualitative evidence has been collected based upon the extensive experience of the researcher.

Continued refinement of best practices

The system must continue to evolve because while it is an improvement over traditional development, it is far from a perfect implementation. Improvements can be made to the method in which the presentation layer works. Documentation standards to map this specific system need to be explored. There currently has been no exploration done into the advantages or disadvantages of documentation of the entire system and interactions between layers.

Definitions of Terms

Abstraction

A mechanism or practice that attempts to reduce detail so that one can focus on few concepts at one time. This practice effectively separates module or unit of logic from its interface (wikipedia.org) .

Browser

A tool used to access web pages over the internet. Common examples are Firefox, Internet Explorer and Netscape.

CSS

An abbreviation for Cascading Style Sheets. This language is used in conjunction with HTML documents. It contains formatting information for the display properties in a web page.

Design-by-Contract

A methodology for designing computer software that requires that a strict contract be written through documentation and then code developed to satisfy that contract.

HTML

An abbreviation for Hypertext Markup Language. This language is used to create and present web pages on the internet.

Javascript

A language whose logic is executed by the client's browser. This means that the computer that is used to access the web page is responsible for executing the Javascript code. This language's most common use is to interact directly with the web page once it has already been delivered.

OO

An abbreviation for Object-Oriented. A programming paradigm that is based around separating code into logical units referred to as objects.

RDBMS

An abbreviation for Relational Database Management System. An RDBMS is a type of database management system that is organized and accessed according to the relationships between data values. It is the most common form of database systems used today.

Server-Side Language

A language whose logic is contained in and executed on the server before or during the loading of a web page. This language is most commonly used to generate the HTML required for a web page. Common examples are ASP, PHP, JSP, Perl(CGI) and ColdFusion.

SQL

An abbreviation for Structured Query Language. This is a language that is used to interact with a relational database management system. It allows functionality to search for, append, delete and alter records contained within the database management system.

Unit Testing

A testing methodology that relies on strictly testing the behavior of units of code at every level possible. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing provides a strict, written contract that the piece of code must satisfy(Unit).

XML

An abbreviation for Extensible Markup Language. This language that is specifically designed to describe many different kinds of data. Its primary purpose is the sharing of data across systems.

XSL

An abbreviation for Extensible Stylesheet Language. A language which that's purpose is to describe how XML documents are to be formatted or transformed.

References

Abstraction. (n.d.). Retrieved March 22, 2006, from http://en.wikipedia.org/wiki/Abstraction_ (programming).

Beck, K. (1999). *Extreme Programming explained: Embrace change.* Boston, MA:Addison-Wesley Professional.

Beck, K. (n.d.). Simple smalltalk testing: With patterns. Retrieved Febuary 22, 2006, from http://www.xprogramming.com/testfram.htm.

Design by contract. (n.d.). Retrieved March 10, 2006 from http://en.wikipedia.org/wiki/Design_by_contract.

Introduction to XSL. (n.d.). Retrueved March 22, 2006, from http://www.w3schools.com/xsl/xsl_intro.asp.

phpBB. (n.d.). Retrieved March 22, 2006, from http://en.wikipedia.org/wiki/PhpBB.

phpBB.com. (n.d.). Retrieved March 22, 2006, from http://www.phpbb.com/.

Unit Test. (n.d.). Retrieved March 22, 2006, from http://en.wikipedia.org/wiki/Unit_test.

Wallace, N. (2000, March 8). Design Patterns in Web Programming. Retrieved March 23, 2006, from http://www.e-gineer.com/v1/articles/design-patterns-in-web-programming.htm.