# *60-311 Term Project*

A Sample Report

Submitted By: xxx1 YYY1 and xxx2 YYY2

## BACKGROUND

Communication has been evolving since the beginning of time.  Inventions such as the phone, television and radio have made it one of the most dynamic fields on earth.  Over the last decade the Internet has emerged as the new type of communication.  Internet Technology can no longer be touted as the wave of the future - it is the wave of today.

Everything from the entertainment industry to the business industry is quickly veering towards this new form of communication.  It is the fastest growing medium in the world.  It was born in the mid-eighties and today it is used by millions of people.  People ranging from those employed by large conglomerate Corporations to five year-old children.

Businesses are scurrying to use the Internet to tap into the fastest growing client base in the world.  People use it to do research, meet people and find information.  It has truly made the world into a Global Community.  Experts predict that it will eventually be the main form of communication.  It will most likely replace traditional televisions and telephones.

The most popular method of information exchange over the Internet is through the World Wide Web.  Information is transferred through the WWW using a text-formatting scheme called Hyper Text Markup Language (HTML).  HTML is a platform independent method used to produce, format and display documents on the Internet.  HTML has been on the forefront of the Internet Explosion for the last decade or so.  It has expanded to allow almost any media to be viewed and transferred.  It has become the new evolution of communication.

# 1. MARKET INVESTIGATION

## 1.1 Overview

### *1.1.1. Introduction*

R-S Production Solutions has been intensely involved in Web Development for the last two years. During that time it has become increasingly obvious that a large percentage of the HTML files accessible on the Internet are erroneous. Some common problems are:

- Unmatched open and close tags
- Invalid tags
- Improperly formatted structures (tables, frames, etc)
- Invalid links
- Broken pictures

R-S Production Solutions has also recognized that many Web Developers are unaware of browser inconsistencies, and platform dependencies. This becomes obvious to anyone browsing the Internet. All too often viewing a URL with Netscape shows an extremely different page than viewing the same URL in Microsoft Internet Explorer.

In a surprisingly high number of cases, the aforementioned problems will cause HTML documents to be displayed differently then their authors intended. These issues lead to a tremendous amount of frustration and problems maneuvering through the Internet. This causes people to be skeptical about using and developing on the Internet. Many Businesses see the frequency of problems in HTML documents and lose confidence in the Internet. This is a serious obstacle that prevents the Internet from becoming a credible medium on which to do business.

R-S Production Solutions believes that a software system can be created to help reduce the abundance of errors and inconsistencies that are found on the Internet. A program that parses a HTML document recognizing and displaying any errors or possible issues would help web designers create and maintain more efficient HTML documents.

### *1.1.2. Target Market:*

There are three main groups of people that could be targeted as stakeholders in the HTML Verification System. These groups are defined by HTML coding experience.

- Web Designers with extensive HTML experience.
- Semi-Experienced Web Designers with limited HTML experience.
- People with very limited or no HTML experience.

By considering all possible types of HTML Designers, the target market includes as many people as possible. R-S Solutions is a small company and as such we lack the funds necessary to effectively produce or market a National product. To counteract this, we have defined a long-term plan that will eventually lead to a national or worldwide release of the HTML Verification System.

The first step is to produce and market our HTML Verification System within the local IT industry of Newfoundland and Labrador. This will allow accrual of profit and provide feedback on our product.

The second stage consists of using the feedback and profit from Stage 1 to improve our product and eventually produce a system sufficient for National release.

(*This report considers the short-term plan of producing and marketing this system for the local IT Industry)*

## 1.2 Market Survey

### *1.2.1 Introduction*

To make this system suitable for a wide range of users, it is essential to target the three types of stakeholders mentioned in Section 1.1. To obtain feedback from these stakeholders we surveyed 50 people. This survey outlines how the local IT industry would react to the HTML Verification System. The people included in this survey were:

- Personnel from xwave solutions
- Personnel from Web Express
- Personnel from Memorial University of Newfoundland
- Several Independent Freelance Web Designers.

xwave solutions is the trend setter in the local IT industry. They set the standards which other IT companies strive to achieve. Interest from xwave solutions would certainly invoke interest from other IT companies. xwave also has offices in Halifax, Calgary, Ontario, and Dallas and this could provide a means of national exposure. To obtain accurate feedback from xwave solutions, we surveyed 3 people from upper management and 7 people who work in Web Development.

Web Express is a local company that designs web systems. Since this company concentrates solely on HTML they would find a HTML Verification System useful. Our survey included 10 web designers employed by Web Express.

Memorial University of Newfoundland has produced many of the specialists in the local IT industry will continue to produce the specialists of tomorrow. Faculties like IT, Computer Science, Business and Nutrition all incorporate HTML into their curriculum. M.U.N. also hosts a large number of web pages designed by students and staff. To obtain feedback from the University we surveyed 4 Technical Support staff, 4 web page designers employed by the University and 2 Computer Science Professors.

Freelance Web Designers are the people who would use our program at home. This is potentially the biggest market. Feedback from this group would provide insight on how useful the public would find our product. To obtain this feedback we surveyed 20 Freelance Web Developers.

### *1.2.2. The Market Survey*

The survey that was distributed is attached in Appendix A at the end of this document. The results of this survey were compiled and considered by staff at R-S Solutions.

### *1.2.3. Potential Markets based on Survey Results*

Based on the 50 distributed surveys it was evident that there is a need for a high quality HTML Verification System in the local IT Industry.

- 64% of those surveyed had used a HTML Verification System before.
- 98% said they would use a high quality HTML Verification System if it were available.
- 80% said they would suggest a high quality HTML Verification System to others.

The survey also provided insight on what functionality was desired in a HTML Verification System. Following is a list of some key requirements obtained from the survey:

- The system should provide local access and not require online access.
- The system should perform comprehensive tests.
- The system should allow scheduling of verification sessions.
- The system should produce detailed reports.
- The system should be easily configured.

These results play a large role in configuring the necessary functionality of the HTML Verification System. This is discussed further in section 2.2 of this document.

### *1.2.4. Competitive Factors*

There are a large number of systems that offer HTML verification. Many of these services are actually free and online. Users can use these online systems to check HTML code for errors. This is a huge competitive factor. Some of the more popular packages are:

| Product | Description |
|---|---|
| Website Garage | Many HTML  Tests |
| NetMechanic | Link Check,And More |
| W3C HTML Validator | Check HTML Syntax |
| SiteInspector | HTML and more |
| WebTech's Validation Service | HTML Syntax Checker |
| Dr. Watson | Gentler HTML Verification |
| Green Pac | Checks for Broken Links |

These systems are fairly useful, but they contain many shortcomings. Some of these glaring shortcomings include:

- These systems perform only a few of the operations requested in the market survey.
- These systems use a very simplistic style of HTML checking, and can only report missing or invalid tags. They fail to report inconsistencies and subtle errors.
- They lack detailed browser dependence information.
- They lack scheduling routines.
- They require online access.
- They require that a html file be transferred over a connection, which could cause security issues.
- They do not support batch checking.
- They require registration and personal information to use. Things like email addresses and other info. These email addresses are often used for commercial purposes.
- They do not produce useful and accurate reports.

By creating a product that avoids these shortcomings while offering many innovative and useful services, R-S Solutions could easily overcome its competitors. Most web designers realize how easy it is to make an error while creating a HTML document, and have been looking for a true verification system which could help them to reduce or eliminate these errors. The market survey shows that the market is clearly unsatisfied with the current HTML Verification Systems on the market.

R-S Solutions firmly believes that a Web Developer who cares about his/her code will bypass primitive free services, and use a high-quality HTML Verification System to ensure that there code is error free and efficient.

## 1.3 Cost Factors

*1.3.1 Introduction*

The design of the HTML Verification System will consist of development and testing stages. It also requires the use of certain hardware and facilities. The total cost of development would have to incorporate all of these factors. Following is a list of each of the cost factors included in the development of this system.

*1.3.2. Coding and Programming*

The cost of development will consist of the wages of the developers. Using Expert Judgement of several programmers who have developed HTML parsing software systems, it seems that development of this system will take approximately 4 man months. Assuming the average $50000 per annum wage for a programmer, approximate coding cost will be $17000.

*1.3.3 Testing and Debugging*

Testing this system will be relatively easy and inexpensive. With the wealth of HTML documents available on the Internet, test data generation is not an issue. After a version of this software is developed, programmers will switch their efforts to testing and evaluation.

Using the Expert Judgement of developers with similar experience, it seems that testing and debugging will require a total of approximately 1 man month. Using a salary of $50000 per annum for programmers, testing and debugging will cost approximately $4200. Section 3.7 contains a detailed discussion on this testing process.

*1.3.4 Hardware and Facilities*

The software will be developed for the Windows 95/98 Operating System using Visual C++. The two programmers on staff would require two sufficiently powerful computers, two Visual C++ Compilers, proper software development apparatus and CASE tools. This equipment would cost approximately $6648 (See Appendix B for a summary of this cost).

Since R-S Solutions is a small company it operates out of a small office in St. John's, Newfoundland. The cost of the office space, and other necessary equipment for 3 months would be approximately $6080.

Combined, the approximate cost for hardware and facilities would be approximately $12728. For a more detailed description of the above costs, see Appendix B attached at the end of this document.

*1.3.5 Summary of Development Costs*

| Item | Cost |
|---|---|
| Manpower (8 Months) | $17000 |
| Manpower (1 Month) | $4200 |
| Facilities and Hardware | $12728 |
| **Total Cost (Approx)** | **$34000** |

Combining the cost of coding, testing, debugging, hardware and facilities the total cost of this system will be approximately $34000. For a staff of 2 programmers, it would require approximately 2 months to code and 1 month to test.

### *1.3.6 Feasibility Summary*

The survey clearly shows that a market does exist for a HTML Verification System. HTML Developers currently use free online services that perform primitive error checking. It is possible to take advantage of this market by producing a superior system that will be relatively inexpensive.

It must be re-emphasized that R-S Solutions is not placing an emphasis on obtaining profit through the local IT industry. The long-term plan is to use the local industry to create enough revenue and feedback to produce a system that can be marketed Nationally. Therefore, we are satisfied to market this product at a very inexpensive price.

If we market this product at $40.00, then we would have to sell approximately 850 copies of the HTML Verification System to recover development costs. Any sales over and above this goal would produce profit. Considering the IT market in Newfoundland is growing every day, this seems like a very reasonable goal.

Inevitably the success of the HTML Verification System would depend on its final quality. A high quality system would sell well, and word of mouth would be very influential. A medium or low quality system would not be viable to develop.

# 2. REQUIREMENTS ENGINEERING

## 2.1 Introduction

The largest risk associated with this project is that an insufficient market would make it economically unfeasible to produce. The feasibility of this product is directly dependent on its quality. It is essential to produce a quality system that users would be comfortable and satisfied with.

To enhance the quality and usefulness of the HTML Verification System, it will be developed by refining, and adhering to certain requirements. These requirements will define the functionality of the final state of the system. Developing and satisfying sufficient requirements will ensure that the final product sufficiently meets the needs and demands of customers.

## 2.2 Requirement Process

### *2.2.1 Requirements Analysis*

A comprehensive HTML Verification System should perform a set of tests that consider a wide range of issues. This function is the backbone of the entire system. The definition and specification of this requirement is fairly straight forward, but additional system requirements are not as obvious.

Several possible stakeholders for this system were identified in Section 1.2. Each stakeholder has his or her own set of requirements for this system. To ensure that the HTML Verification System adheres to a large customer base, it is essential to consider the needs of each of these stakeholders. The possible stakeholders for this system were:

- Web Designers with extensive HTML experience.
- Semi-Experienced Web Designers with limited HTML experience.
- People with very limited or no HTML experience.

To ensure that the system is suitable for each stakeholder, viewpoint-oriented analysis was used to determine its requirements. By using a Viewpoint Analysis, we can define the needs and desires of each stakeholder and then focus on satisfying these needs. Two major sources of information were considered in defining these needs:

- ***The results from the market survey.*** The Market Survey provided insight from Freelance Web Designers and IT professionals containing what they would desire in a HTML Verification System.
- ***R-S Solutions' previous HTML Coding and Internet experience.*** R-S Solutions is comprised of staff that are very experienced Web Designers as well as staff that is inexperienced in HTML design.

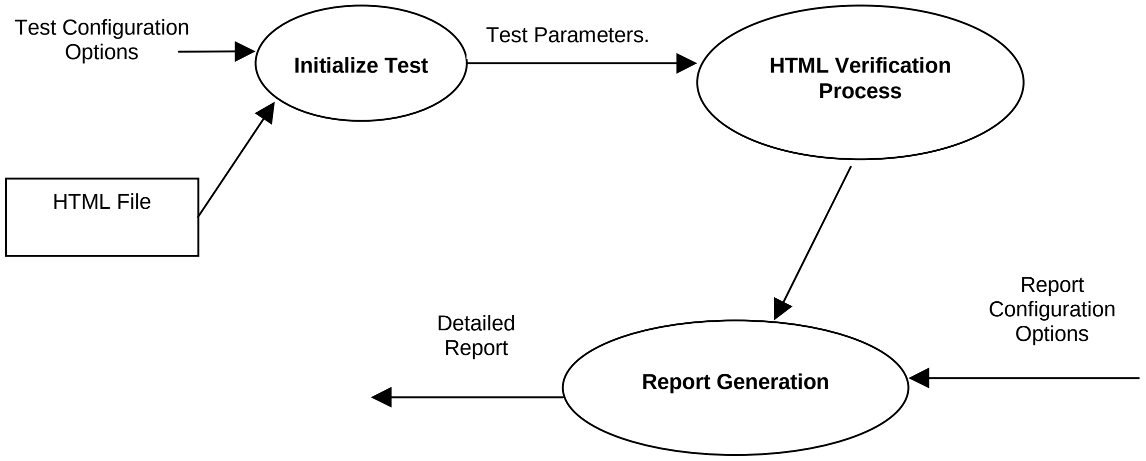
These two sources of information produced functionality suitable for all stakeholders. The HTML Verification System should provide capabilities for

- Verifying HTML code
- Configuring verifications and reports
- Producing Detailed Reports
- Scheduling Utilities

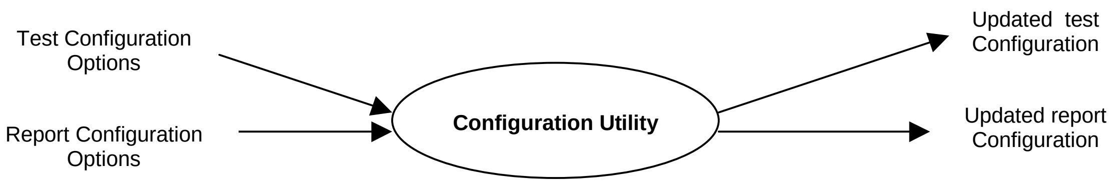
Adhering to this functionality should ensure that the final system complies with the needs of a wide range of users. This should expand the possible market, and increase the chances of financial feasibility in the HTML Verification System project.

Each of these functional requirements can be depicted with a Data Flow Diagram.

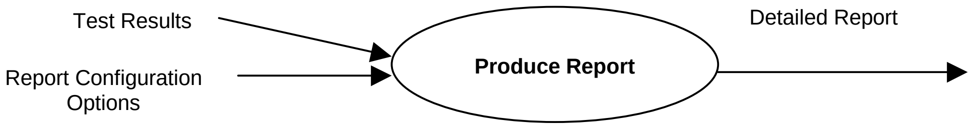
Verifying HTML Code


Test Configuration Options
Initialize Test
Test Parameters.
HTML Verification Process
HTML File
Detailed Report
Report Generation
Report Configuration Options

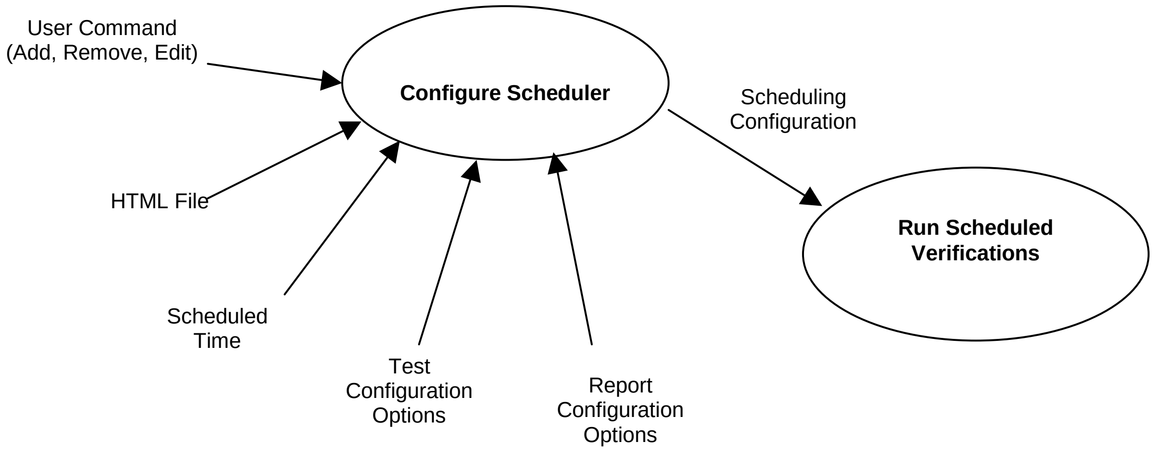
Configuration verifications and reports:


Test Configuration Options
Report Configuration Options
Configuration Utility
Updated test Configuration
Updated report Configuration

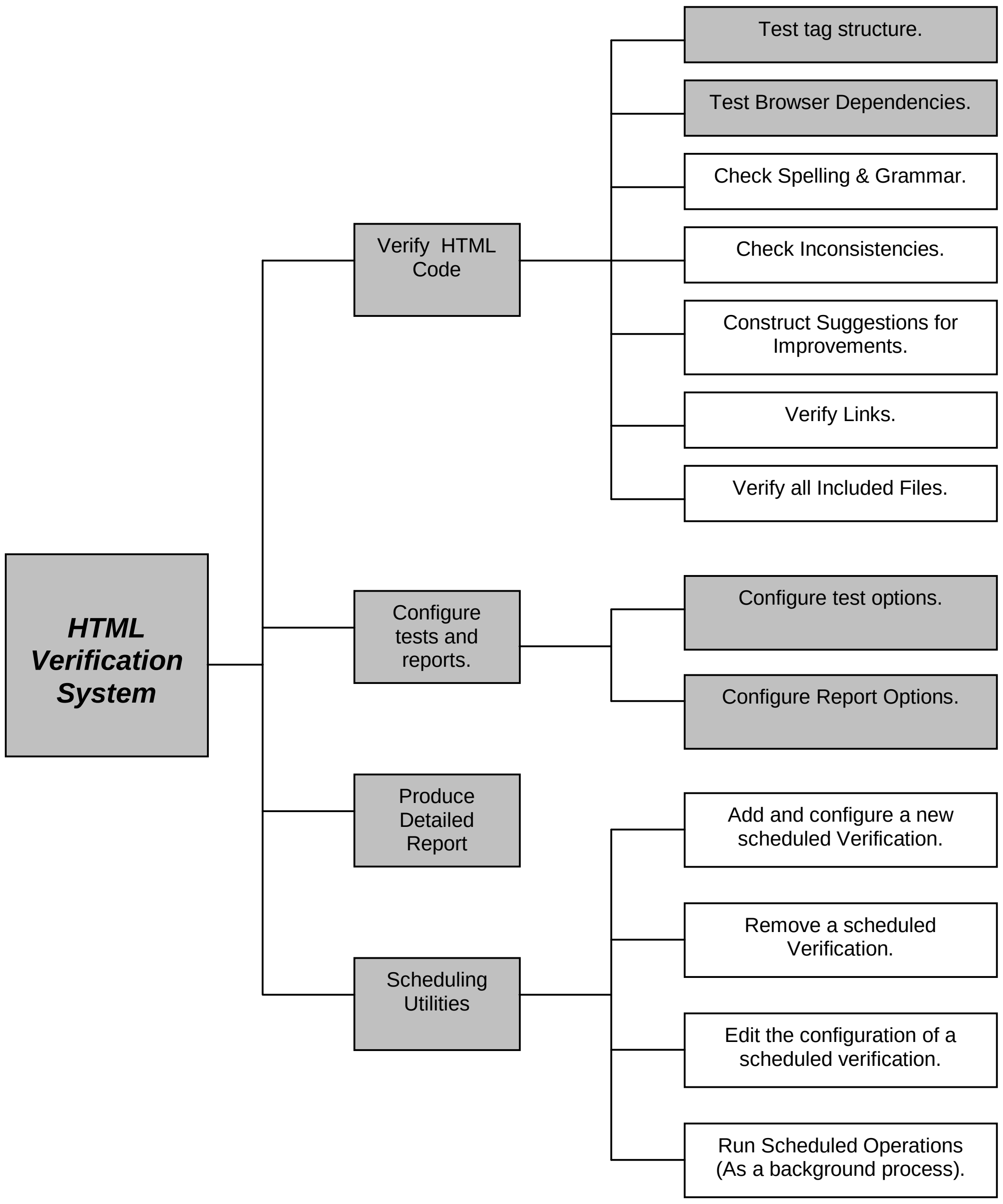
Producing Detailed Reports:



Scheduling Utilities:


User Command (Add, Remove, Edit)
HTML File
Scheduled Time
Test Configuration Options
Report Configuration Options
Configure Scheduler
Scheduling Configuration
Run Scheduled Verifications

The previous DFDs (Data flow diagrams) outline the main functions of this system. The total functionality of the system can be illustrated by using the functional model below.


HTML Verification System
Verify HTML Code
Configure tests and reports.
Produce Detailed Report
Scheduling Utilities
Test tag structure.
Test Browser Dependencies.
Check Spelling & Grammar.
Check Inconsistencies.
Construct Suggestions for Improvements.
Verify Links.
Verify all Included Files.
Configure test options.
Configure Report Options.
Add and configure a new scheduled Verification.
Remove a scheduled Verification.
Edit the configuration of a scheduled verification.
Run Scheduled Operations (As a background process).

*These Diagrams represent the functional requirements of the system. They provide an abstract picture of what the system does. They DO NOT represent the design or detail of system implementation. These topics are detailed later in this document.*

Further decomposition of the shaded boxes is possible, but not helpful. Any further decomposition of this model would make it too detailed. The given decomposition is sufficient for outlining the main functionality of the HTML Verification System. For example, the '*Test tag*' box could be further decomposed into the sub functions:

- Test for matching open and close tags
- Test for proper table structure
- Test for invalid tags
- Test for ambiguous tags
- Test for meaningless tags

In addition to the previously mentioned functional requirements, there were several non-functional requirements identified in the analysis process.  These are discussed in Section 2.3.3 of this document.

### *2.2.2. Requirement Definitions*

Requirement Analysis produced several functional and non-functional requirements for the HTML Verification System.  This section will further define and specify these requirements.

#### 2.2.2.1 Functional Requirement Definitions

The four main functions of the HTML Verification System were described in section 2.2.1.

- Verifying HTML code
- Configuring verifications and reports
- Producing Detailed Reports
- Scheduling Utilities

Following is a complete and consistent set of definitions for these requirements.

> *Each of the following definitions are presented in common forms.  These forms ensure consistency and they make the requirements easy to understand.  This layout allows requirements to be easily changed.  It also allows users to easily see the main functionality of the final system.  More detailed specifications of specific functions are attached in Appendix C of this document.*

## *Requirement 1*

**Function:** Testing and Verification of HTML Code

**Description:**

**The Verification Process will search through HTML code and detect any errors, inconsistencies or suggestions.** It will allow a user to choose a HTML file and select which tests to perform on this file. The tests that it performs on the chosen HTML file will be a subset of the following tests.

- Search for proper tag structure (According to HTML Version 1.1)
- Search for missing tags
- Search for invalid tags (those not in HTML Version 1.1)
- Search for ambiguous and misused tags
- Check for spelling and grammatical errors
- Check for invalid or incorrect links (Formed incorrectly, or invalid addresses)
- Check for invalid embedded files (missing or corrupt files)
- Check for possible browser dependency issues.

Tests will also create possible suggestions on how to improve the quality of the HTML code.

Before a test is performed, a user is given the option to select default test options. If he/she chooses not to use default test settings, they are provided with a configuration utility (See Requirement 2). These choices made in this configuration utility apply only to the test of the HTML file chosen by the user.

**Rationale:**
This function is the backbone of the system. It performs the main operation of the software - to Verify the correctness of HTML files.

**Sub-functions:**
- Test for proper tag structure. (Specification 1.1)
- Test for browser dependency issues. (Specification 1.2)
- Check spelling and grammar. (Specification 1.3)
- Test for inconsistencies in code. (Specification 1.4)
- Construct suggestions on how to improve HTML document. (Specification 1.5)
- Check all contained links. (Specification 1.6)
- Verify all pictures, sounds and included files on page. (Specification 1.7)

**Specification:**
- Each sub function has a detailed specification in Appendix C of this document.

## *Requirement 2*

**Function:** Configuration Utility.

**Description:**

**This will allow a user to configure the HTML Verification System**. The configuration utility will allow a user to configure the following:

- The subset of tests to perform on a HTML file. (See Requirement 1).
- The file format used to store the report. (See Requirement 3).
- The options and results to include in the Report (See Requirement 3).
- The location used to store the generated report file (See Requirement 3).

If the Configuration Utility is used independent of any HTML file test (See Requirement 1), the settings chosen are to be saved and used as the default settings for the system.

If the Configuration Utility is used for a specific test (See Requirement 1) then the choices affect that test only.

**Rationale:**
This function will allow a user to configure the HTML Verification system to conform to their needs. This allows individual tests to use separate configurations. It does not force a standard set of options to be enforced with every test. It also allows the possibility of creating a standard set of default options, which users can use. These default options will allow a user to perform quick tests without having to worry about using any configuration utility.

**Sub-functions:**
- Configure test options. (Specification 2.1)
- Configure report options. (Specification 2.2)

**Specification:**
- Each sub function has a detailed specification in Appendix C of this document.

## *Requirement 3*

**Function:** Produce Detailed Test Result Document

**Description:**

**This function will examine the results of the HTML verification (See Requirement 1), and produce a high quality document containing a detailed description of any errors, inconsistencies or suggestions that were discovered during the HTML test.** This document will conform to the configuration chosen by the user (See Requirement 2). The contents of this report will contain results of the verification and a description of the reasoning behind them.

**Rationale:**
This function will allow a user to produce a report that will summarize the results of a verification of a specific file.

**Sub-functions:**
- Produce detailed report. (Specification 3.1)

**Specification:**
- Each sub function has a detailed specification in Appendix C of this document.

## *Requirement 4*

**Function:** Scheduling Utility

**Description:**

**This function will allow users to schedule when HTML documents will be verified and configure previously scheduled verifications.** Using the Scheduling Manager will allow a user to add or delete scheduled tests. A scheduled test will include the following information:

- An associated html file
- Scheduled time of verification
- Configuration options (See Requirement 2).

Users will also have the ability to edit an already scheduled test. They can alter which html file will be tested, the time of verification and the configuration options for the test (See Requirement 2).

Users should be able to schedule batch (Simultaneous) and single verifications.

**Rationale:**
This function allows a user to schedule verifications. It allows them to ensure that HTML files will be verified at a certain time. This creates much flexibility, as the user isn't required to personally begin a verification. It also allows a user to maintain the current schedule by allowing each scheduled verification to be edited to conform to the users desires.

**Sub-functions:**

- Run Schedule. (Specification 4.1)
- Add a schedule verification(s). (Specification 4.2)
- Remove a scheduled verification. (Specification 4.3)
- Edit an already scheduled verification. (Specification 4.4)

**Specification:**
- Each sub function has a detailed specification in Appendix C of this document.

*2.2.2.2 Non-Functional Requirements*

It is essential for this system to conform to user's needs and demands. Requirements Analysis produced the following non-functional requirements.

- **The System Must be Easy to Use:** Users should be able to learn this system very quickly. It should be fairly intuitive. The reports should be clear and precise, and not overly technical.
- **The System Should not be overly technical:** The information produced by the system should be understandable by anyone with general HTML knowledge. Details should be written in a very understandable fashion.
- **The System Must be Fast and not Memory Intensive:** A HTML verification system should not be a large, intense system. The Scheduling Operations should run as a background process, and not draw too many resources away from other software.
- **The Scheduling System Must be Easy to Learn and Use:** The scheduling routines must not be complicated. Changes in scheduling should be easily implemented.
- **The System should be Distributed in an Efficient Manner:** It should be easy for the software to be transferred and installed. The most likely distribution is with CD-ROMs.
- **Configuration of the System Should be Easy and Efficient:** Configuration routines should not be overly complicated. It should be very easy to view or alter the configuration information of the system.

## 2.3 Prototype

*2.3.1 Introduction*

The market survey (Section 1.2) and previous HTML coding experience has produced a set of system requirements (Section 2.2) that will be used to develop a prototype. This prototype will then be used as a tool to refine and improve the system requirements.

The process of developing a prototype involves four main steps.

- The prototype objectives must be determined. (2.3.2)
- The prototype functionality must be defined. (2.3.3)
- The prototype must be developed. (2.3.4)
- The prototype must be tested and evaluated. (2.3.5)

This process is iterative. The four steps are repeated and the prototype is refined until evaluation determines that the prototype has reached a satisfactory state.

*If at any point in this process user feedback becomes negative, then the entire project will have to be re-evaluated. It is essential that users are satisfied with this system. If feedback does become negative, the potential market for this project would not be large enough to make it economically feasible, and the entire project would have to be reconsidered.*

*2.3.2 Prototype Objectives*

Evolutionary prototyping of the HTML Verification System is based on the idea of developing an initial implementation based on the requirements in Section 2.2, exposing this implementation to user comment and refining it through further stages until an adequate system has been developed. Using an Evolutionary Prototype will:

- Produce the final system using continuous feedback from users.
- Allow the system to be developed and delivered quickly.
- Help developers refine their understanding of what customers want from the system.
- Help refine the requirements of the system.
- Discover any functionality that should be included in the system.
- Discover any important performance issues that may have been overlooked.
- Provide a means to demonstrate the different uses of the HTML Verification System.
- Provide an excellent marketing tool.

A problem with evolutionary prototyping is that when the system becomes outdated it usually has to be completely rewritten. This would shorten the lifetime and reusability of many systems. Developing efficient object-oriented code will eliminate this problem. Creating coherent and cohesive objects will ensure that they can be reused and easily altered. This will make updating and manipulating the system very easy and greatly enhance system evolution and lifetime.

### *2.3.3 Prototype Functionality*

The final version of the evolutionary prototype will be the commercial version of the HTML Verification System. It is essential that the prototype satisfies all of the functional and non-functional requirements described in Section 2.2. It is likely that the prototyping process will refine these requirements, but unlikely that it would change the basic functionality of the program. Therefore, the functionality of the prototype will likely be exactly the same as the functionality described in Section 2.2.

### *2.3.4 Prototype Development and Design*

The prototype will be developed in Visual C++, the production language of the final system. A team of qualified developers will initially create a system adhering to the requirements in section 2.2 of this document. This prototype will be continuously refined and altered using user feedback. Since the product of evolutionary prototyping is the final system, the phrases prototype development and system development are interchangeable. The final prototype is the final system and hence prototype design is the system design.

*A detailed discussion on Prototype/System Development and design is included in Section 3.0 of this document.*

### *2.3.5 Testing and Evaluation*

Testing is perhaps the most important evolutionary prototyping stage. Testing an evolutionary prototype decides if the design process is completed, or if it needs to be further refined. If testing discovers problems, then the prototype will need to be altered, and the prototyping cycle must continue. Testing of the HTML Verification System Prototype will include two main stages.

**Developer Testing:** When development of the prototype is complete and the functional and non-functional requirements seem to be met, developers will be given 7 days to perform intensive tests. Developers will use a White-box method of testing. Path testing will be employed to test as many paths of execution as possible. This will include HTML files that would fail each of the tests detailed in the requirements. Developers will also use a wide variety of HTML files to ensure that all of the non-functional requirements are met. An example of a HTML file that could be used as test cases is attached in Appendix D of this document.

Testing by developers will confirm if the prototype actually does satisfy all of the required functional and non-functional requirements.  This will also provide an opportunity for the developers to identify any new requirements that should be included.  Once all developers are in agreement that the prototype meets all of the necessary requirements and does not lack any necessary functionality, the protocol is labeled as a 'Protocol Release'.

**Stakeholder Testing:** When a 'Protocol Release' is produced it will be distributed to 10 software testers.  These software testers will be chosen to ensure that each type of stakeholder (Section 1.1.2) has at least three representatives.  More specifically software testers will include:

- At least 3 very experienced HTML programmers
- At least 3 somewhat experienced HTML programmers
- At least 3 inexperienced HTML programmers.

These testers will have their names included in the product's documentation and will receive free copies of the final version of the HTML Verification System.

Once given a version of a 'Protocol Release', the software testers and the developers will be given 14 days to evaluate and test the software.  They will perform black box testing, since they are unaware of the implementation details of the system.  Software testers will be encouraged to test a wide variety of HTML files.  They will be encouraged to try to 'trick' the software into performing erroneously.  It will be suggested that they try large and small files, as well as correct and incorrect files.  They will have the opportunity to test any functionality they would like.  This will expose the software to a wider range of tests.

After their 14-day testing period, software testers will be given an opportunity to suggest any new functional or non-functional requirements, as well as any changes to existing functional or non-functional requirements.

Once all feedback is obtained, developers will evaluate each suggestion as 'useful' or 'not useful'. A suggestion will be deemed as 'useful' if it is determined that multiple users of the software system would benefit from it and no users would be offended or annoyed by it.

If the suggestion is evaluated to be 'useful', then developers will alter the requirements and the prototype accordingly.  The process will then return to the Developer Testing stage.

**End of Testing:** When the software testers and developers cannot make any 'useful' suggestions for modifications to the prototype, the 'Prototype Release' will be evaluated as 'Production Ready'.  When a protocol is evaluated as 'Production Ready' then it is completed and prepared for release.

# 3.0 Software and System Design

## 3.1 Software / Prototype Design

### *3.1.1 Introduction*

It is essential for the HTML Verification System/Prototype to be reliable, robust, maintainable and efficient. The HTML Verification System will be designed in an Object Oriented fashion. It is important for its design to produce high cohesion and loose coupling. This will achieve high maintainability. The extensive testing procedures detailed in Section 2.3.4 ensure that the system will be reliable and efficient.

Software Design is a challenging process. It requires creativity and experience. There is no way to ensure a good design, but there are several steps that can help. Experience shows that the adhering to the following steps improves the quality of an Object Oriented System Design:

- Architectural Design and Control Design.
- Object Identification
- Object Inheritance
- Object Aggregation and Interaction.
- Object Interface Design.
- Algorithm Design.
- User Interface Design.

The design of the HTML Verification System should follow these seven steps of design. The following sections detail the development of the HTML Verification System and outlines the steps listed above.

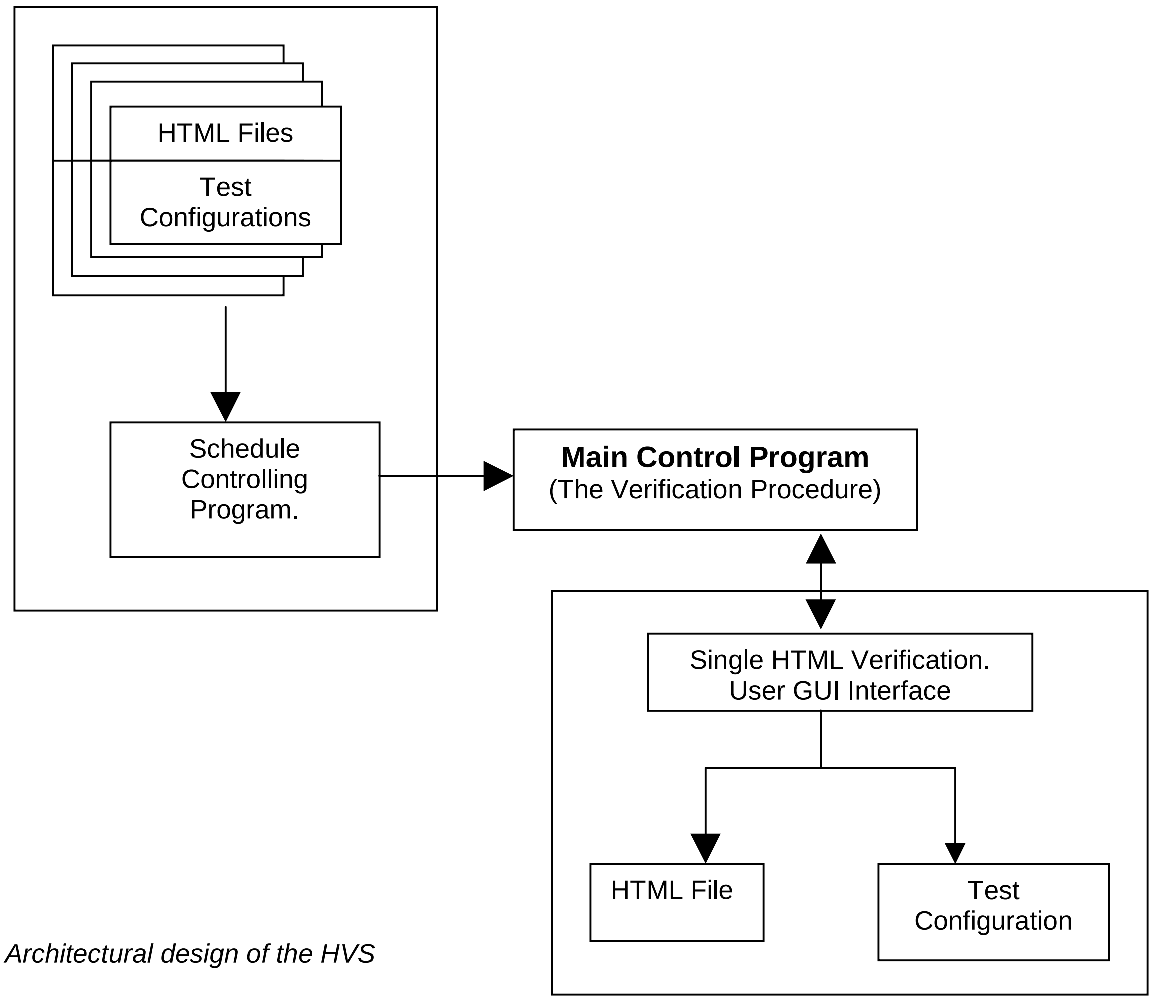
### 3.1.2 Architectural and Control Design

This system is composed of two main sub-systems. One is the main program, which allows a user to verify a single HTML file and create a report. Another is the Scheduling Program that runs scheduled verifications. These sub-systems execute independently, but are controlled by the main program. This situation is depicted in the architectural model on the following page.

From this model we can see how the two main sub-systems interact with each other. The program that runs single HTML Verifications obtains a HTML file and Test Configuration Information (from the user) and runs a test using the testing capabilities of the main program. The schedule processor runs as a background process. It uses a group of HTML files each with an associated test configuration. The Main Control Program controls both of these programs and offers the Verification and other system procedures.

The HTML Verification System uses a form of Centralized Control. The two main programs are controlled from within a Main Control Program. The Main Program will in fact:

- **Control the basic Single HTML Verification Program.** This process will obtain input from a user and perform a test on a single HTML file.
- **Control the Scheduling Program.** The Scheduling Program will run as a background process and will control the scheduled Verifications. The Scheduling program is a sub-system that will control all of the actions of the HTML Verification System’s scheduling routines.


HTML Files
Test Configurations
Schedule Controlling Program.
Main Control Program
(The Verification Procedure)
Single HTML Verification. User GUI Interface
HTML File
Test Configuration

*Architectural design of the HVS*

3.1.3 Object Identification

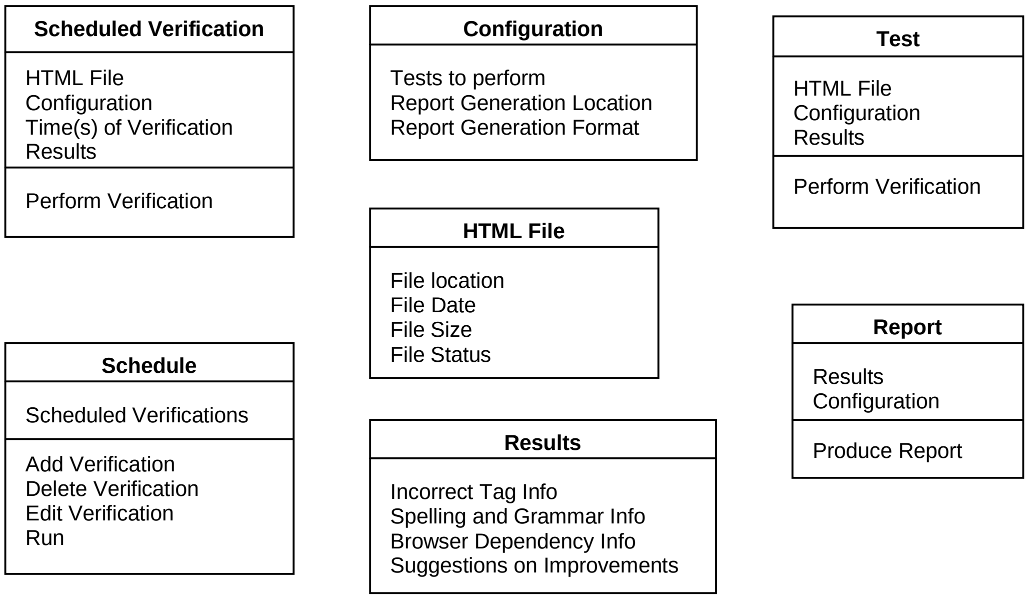
Object oriented design requires carefully defined and separated objects. Each object has to have a properly designed set of operations and attributes. These objects will interact with each other to create the functionality of this system. Careful analysis of the description of the HTML Verification System and the requirements described in section 2.2 suggests the formation of seven main objects. The first three objects will only serve as information containers and will have no public operations. The remaining four will contain the functionality necessary to create the system. Encapsulating this functionality and information into separate objects will increase the ease of evolution and maintenance.

*Containers:*

- **HTML File Object:** This is the object to be tested. It will include such things as the HTML code and other file information.
- **Configuration Object:** This is an object that will contain the configuration information of a test. It will contain information like what tests to perform, where to store the report, what file type to use for the report, etc.
- **Results Object:** This object will contain the results of the test on a 'HTML file' object. It will contain incorrect tag information, spelling or grammatical mistakes and any suggestions on improvements.

*Functional Objects*

- **Test Object:** This object will contain a 'Configuration' object, a 'HTML file' object and a 'Results' object. It will perform the test as defined by the 'Configuration object' on the 'HTML file' object and create a proper 'Results' object.
- **Report Object:** This object will contain a 'Results' object and a 'Configuration' object. It will provide an operation to produce a report detailing its results. The contents of this report will comply with the options in the 'Configuration' object.
- **Scheduled Verification Object:** This is an object that contains a 'HTML File' object and a 'Configuration' object. It performs a test on the HTML file according to its configuration and produces a 'report' object. Its main operation is to perform the test.
- **Schedule Object:** This object contains all of the Scheduled Verifications. It contains the operations add, edit and delete as well as operations required to run scheduled Verifications.



*This is a summary of the main objects of the HTML Verification system. It should be noted that these diagrams produce a rather abstract view of the system. In the actual implementation combinations of smaller more detailed objects may be required to create these seven main objects. The development of these objects is left to the developer's discretion.*

### 3.1.4 Object Inheritance

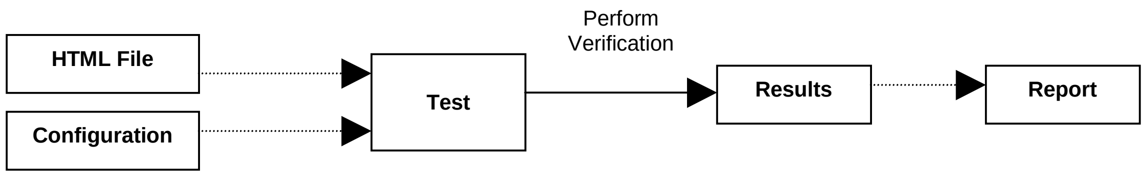
From the diagrams above, it is clear that the Scheduled Verification object could be inherited from the Test object. The Scheduled verification object has the added attribute of a Time(s) of verification. It shares all other attributes and operations with the Test object.

### 3.1.5 Object Aggregation and Interaction.

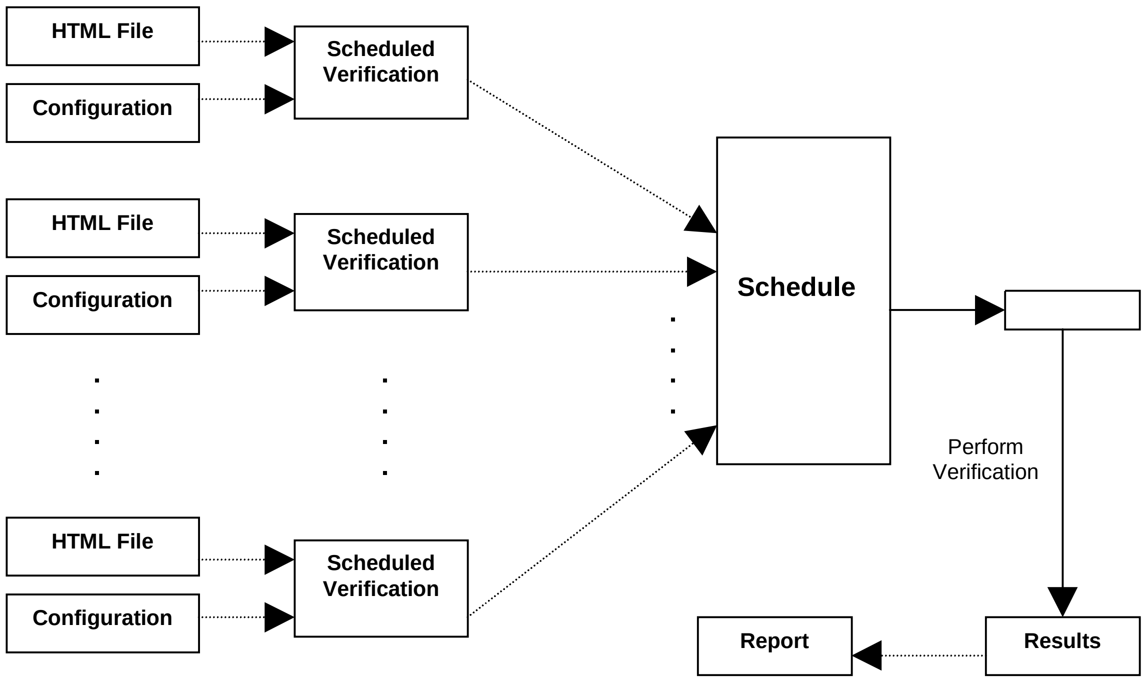
The architectural model included in Section 3.1.2 describes a system containing a Main Program that controls a Scheduling Program and a Single Verification Program. These programs execute by using instances of the objects described in Section 3.1.3 and allowing these objects to call operations on other objects. Some objects in the HTML Verification System are actually composed of two or more instances of objects. For example, a test object contains a 'HTML file' object and a 'Configuration' object. The composition of objects from other objects is known as aggregation. The Aggregation and Interaction of objects in the HTML Verification System can be depicted in the following diagrams:

*In the diagrams that follow, rectangles represent objects. Dotted directed lines represent aggregation (The object at the back of the arrow is used by the object at the head of the arrow). The solid directed lines represent operations on objects. The name of the operation is located next to the solid directed lines. The product from this operation is at the head of the solid arrow.*

***A Single HTML Verification producing a detailed report:***



***The Scheduling Process producing a detailed report for each scheduled verification:***



*3.1.6 Object Interface Design*

The sub-systems and functionality of the HTML Verification System are accomplished through the use of objects. Each object has certain contents and operations. It is through the use of these contents and operations that the desired functionality of the system is achieved.

It is essential that objects are independent of each other (low coupling). It is also essential that each object perform a set of closely related operations (high cohesion). These two factors are the key behind quality OO design. The content and operations of the objects defines the cohesiveness and coherence of the final system.

The interface specification of each object contains a detailed description of its interface. This interface determines how objects interact with each other, and ultimately how a system is produced. Interface specifications (written in C++) for each of the objects introduced in Section 3.3 are included on the following pages.

Several non-standard types are used in the following object specifications.

```
test_config, grammer_results, report_config, spelling_results,  time_for_Verification,
browserinfo_results, tag_results,  suggestions_results, time_for_Verification
```

Viewing these types in the following specifications, it becomes obvious what data they represent. For clarity of this document their implementation is left to the developer.

HTML File Object:

```
class HTML_File {

public:
       //Constructor and Destructor
       HTML_File(char* FileName, char* Location);
       ~HTML_File();

       // The following routines provide access to information
       char* getFileName();
       char* getFileLocation();
       File  getFileHandle();

private:
       char* File_Location;
       char* File_Name;

}
```

Results Object:

```
class Results {

public:
       //Constructor and Destructor
       Results(tag_results taginfo,
               grammer_results grammerinfo,
               spelling_results spellinginfo,
               browserinfo_results browserinfo,
               suggestions_results suggestions)
       ~Results();

       // The following routines provide access to information
       tag_results gettaginfo();
       grammer_results getgrammerinfo();
       spelling_results getspellinginfo();
       browserinfo_results getbrowserinfo();
       suggest ions_results getsuggestions();

Private:
       Tag_results taginfo;
       Grammer_results grammerinfo;
       Spelling_results spellinginfo;
       Browserinfo_results browserinfo;
       Suggestions_results suggestions;

}
```

Configuration Object:

```
Class Configuration {

Public:
       //Constructor and Destructor
       Configuration(test_config tests,
                     char* ReportLocation,
                     report_config ReportFormat);
       ~Configuration();

       // The following routines provide access to information
       test_config getSelectedTests();
       char* getLocationForReport();
       report_config  getReportFormat();

Private:
       Test_config tests;
       Char* ReportLocation;
       Report_config ReportFormat;
}
```

Report Object:

```
Class Report {

Public:
       //Constructor and Destructor
       Report(Results R,
              Configuration C);
       ~Report ();

       // The routine to produce the report
       void produce_report();

Private:
       Results results;
       Configuration config;
}
```

Schedule:

```
Class Scheduler {

Public:
       //Constructor and Destructor
       Scheduler(Sverifications sv);
       ~Scheduler();

       // The routines to use the scheduler
       void addVerification();
       void removeVerification();
       void editVerification();
       void run();

Private:
       // A definition of a list of scheduled verifications
       typedef Sverifications scheduled_ver*;

       // The list of scheduled verifications
       Sverifications svs;
}
```

Test Object:

```
class Test {

public:
       //Constructor and Destructor
       Test(HTML_File F,
            Configuration C);
       ~Test();

       // The routines to verify a file and produce report
       void Verify();

Private:
       // The following routines help to perform the validation
       // and produce the report
       Results DoValidate();
       Report CreateReport(Results Res);
       Void CreateReport(Report Rep);

       HTML_File html_file;
       Configuration config;
       Results results;

}
```

Scheduled Verification Object (Inheritance from Test):

```
class Scheduled_Verification : public Test {

public:
       //Constructor and Destructor
       Scheduled_Verification(HTML_File F,
                              Configuration C,
                             Time_for_Verification T);
       ~Scheduled_Verification();

Private:

       typedef scheduled_ver RECORD OF
              Test test;
              Time_for_Verification time;
              END;

       scheduled_Ver sv;

}
```

### *3.1.7 Algorithm Design*

There are two main algorithms that encapsulate the functionality of the HTML Verification System. They are the scheduling and single test HTML Verifications. Algorithms for these two operations would conform to:

Scheduling Algorithm (Algorithm 1)

```
-   BEGIN SCHEDULE OPERATION
-   WHILE (TRUE) DO
    -   Let X = the Schedule
    -   FOR EACH Scheduled Verification, Y, in the X DO
        -   IF ( Y has a Verification time of NOW ) THEN
            -   Open Y
            -   Verify Y
            -   Set Z = results of Verifying Y
            -   Produce Report R base on Z
        - END IF
    -   END FOR
-   END WHILE
-   END SCHEDULE OPERATION
```

Single Verification Algorithm (Algorithm 2)

```
-   BEGIN SINGLE VERIFICATION
       -Let X = a user identified HTML file
       -Create a HTML Object to represent X
       -Create a Configuration Object Y to store configuration options
       -Create a new Test object Z to store X and Y
       -Let Results = the results of the verify operation of Z
       -Create a Report Object using Z and Y.
       -Produce a Report by using the operation in Z.
-   END SINGLE VERIFICATION
```

It is interesting to note that these algorithms use the functionality found in the objects described in Section 3.3.  These objects include four operations.

- Run Scheduled Verification
- Produce Report
- Verify File
- Run

The Run Schedule algorithm is simply the process that executes the schedule algorithm.

Run Schedule (Algorithm 3)

```
-   BEGIN Run Schedule
-   Call Scheduling Algorithm (Algorithm 1)
-   END Run Schedule
```

The Produce Report Algorithm simply prints the results of a test according to the configuration:

Produce Report (Algorithm 4)

```
-   BEGIN Produce Report
    -   Let V = a HTML Verification
    -   Let C = the Configuration associated with V
    -   Let X = Results of V
    -   FOR EACH Detail Y in X
        -   Print Y According to C
    -   END FOR
END Run Schedule
```

The Verify File algorithm is the algorithm that checks the HTML Document. This is the most important algorithm of the entire system. Great detail should be taken when implementing this algorithm.

Verify File (Algorithm 5)

```
-   BEGIN Verify File
    -   Let X = contents of a HTML File
    -   Check to see that each open tag in X has a close tag.
    -   Check that all tags are valid
    -   Check for ambiguous tags
    -   Check for unnecessary tags
    -   Check for inconsistent tags
    -   Check for incorrect use of tags
    -   Check the spelling and grammar in X
    -   Check for Browser Dependencies in X
    -   Create Suggestions on how to improve X.
    -   Store all of the above Results.
-   END Verify File
```

Since the scheduled verification object is inherited from the test object, is obtains the verify operation from the Test object. Therefore, the algorithm for this is identical to algorithm 5.

### *3.1.8 User Interface Design*

In today's software market a product needs to have a easy to use, efficient user-interface to be successful. The HTML Verification System will use a GUI interface similar to the standard set by Windows 95. It will use a standard set of menus (File, Edit, Help, etc) and commands to increase user familiarity. It will also use the Windows method of browsing files. Since C++ has built in functions to produce a windows standard interface, this will ease development. In using this interface we will ensure that the following properties will be maintained:

**User familiarity:** The interface should use terms familiar to the user.

**Consistency:** Commands and menus should be designed to incorporate the same format throughout the entire system. In this manner user learning time will be reduced, as a result of knowledge learnt in one application being applicable in other parts of the system.

**Minimal surprise:** The interface should be designed so that comparable actions will have comparable effects.

**Recoverability:** Facilities should be included to allow users to recover from their mistakes. These facilities should either be in the form of an undo facility or prompting to confirm any destructive action that is requested.

**User guidance:** Built in help facilities should be integrated with the system and should provide different levels of advice.

The interface will contain clearly displayed buttons with obvious functionality. The testing stage will also help refine our graphical user interface.

### 3.2 Design Evolution

An important advantage to object oriented design is that it simplifies the problem of making changes to the design.  Since objects only interact through their interfaces (Section 3.1.5), the implementation details of objects do not affect each other.  This makes changing objects very simple.  Since objects are loosely coupled, it is also easy to implement and incorporate new objects.

The process of changing the design of this system will be fairly simple.  Developers from R-S Solutions will continue to evaluate the usefulness of the HTML Verification System after production.  If any changes are deemed 'necessary' by the developers, they will be implemented.

Given the interfaces and implementations of the system objects, developers familiar with OOD can easily make necessary changes.  By considering the architecture and design of the system, developers can change the implementation details without concern.

A list of all changes will be maintained in a database for future reference.  If a substantial amount of changes occur the product may be re-released as a new version.  This evolution process will help R-S Solutions implement several future plans.

## 4.0 Future Plans

R-S Solutions plans to provide the HTML Verification System to IT companies in Newfoundland.  There will be a small fee associated with its use, and we plan to use the feedback of our local IT industry to produce an even more efficient system in the future.  When we are confident and comfortable with the quality of our system and its success in the local market, we intend to market this system nationally and internationally.

If this product can create substantial profit in the local IT Industry, there are a plethora of possible options that can be included in future updates.  Possible updates that R-S Solutions are currently investigating are:

- **ASP Verification (VBScript, Javascript, Perl, etc).**
  It would be interesting to have a product that could check included scripts for errors.  This could find errors that would not be produced until the page is created.  Since there is a growing amount of ASP pages on the Internet today, this seems like a natural progression of this system.
- **HTML Verification / Auto Correction System.**
  It would be useful if this system could fix the mistakes it finds.  Perhaps it could remove incorrect links, or replace invalid photos or sounds.  It could insert missing tags, and remove unnecessary tags.  This would be a very useful for HTML programmers and bring the HTML Verification System to a whole new level.
- **Verification and Correction of files at Runtime.**
  This system could be extended to monitor a server and verify a HTML file each time it is accessed.  It could incorporate the correction routines and fix any problems whenever a HTML file is accessed by a browser.  This could prevent any broken links or images from appearing on the page.  This would create a more efficient web hosting system.

There are many other possible changes that could be made to increase the functionality and usefulness of this system.  These changes provide a bright future for the HTML Verification System.  R-S Solutions realizes this possibility, and with innovative and efficient development plan to make it a reality.

## Appendix A

# HTML Verification System Survey

**1. How would you rate you HTML Coding experience?** (*Check the most suitable answer*)

|  |  |
|---|---|
| __ No experience | __ Very Limited Experience |
| __ Limited Experience | __ Extensive Experience |

**2. Have you used an HTML Verification System before?** *(Check one)*

__YES __ NO

**3. If you selected yes in question 2 please list the system(s) that you have used.**

**4. How would you rate your experience with the Verification Systems that you have used?** *(Check 1)*

__Very Poor __ Poor __ Average __ Good __ Very Good

**5. What shortcomings do you feel are evident in these HTML Verification Systems?**

**6. What services would you desire in a HTML Verification System?**

**7. Would you consider purchasing a high quality HTML Verification System if it was available at a reasonable price?** *(Check 1)*

__ Yes __ No

**8. Would you recommend a high quality HTML Verification System to a friend?** *(Check 1)*

__ Yes __ No

## Appendix B

## Project Costing Details

**Hardware/Software:**

- two PC's (Pentium III class) @ $ 2,070.00 ea. $ 4,140.00

- software needed (two copies ea):
  Visual C++ compliers, Microsoft Word,
  Microsoft Internet Explorer, Netscape,
  Windows '98, HTML editors,
  CASE tools @ $ 700.00/system $ 1,400.00

- maintenance and depreciation (20% of total cost) $ 1,108.00

**Travel/Training:**

- None.

**Effort:**

- Application Development (4 man months @ $4,200.00/month) $16,800.00
- Testing and Debugging (1 man month @ $4,200.00/month) $ 4,200.00

**Facilities:**

- Office space located in St. John's - 600 square feet should be appropriate for a 2 person office, with heat and light included (4 months @ $ 1,000.00/month) $4,000.00

**Phone and Internet Charges:**

- As the target market for this product is local there will be no necessary long distance calls. $0

- Assume an average cost/month of $100 dollars for phone service.
  (4 months @ $ 100.00/month) $ 400.00

- Installation of cable modems as well as monthly internet service fees.
  (2 computers @ $ 840.00/computer) $ 1,680.00

# Appendix C

## *Specification 1.1*

**Function:** Test for proper tag structure.

**Description:** This function will test HTML Code for the following:

- Matching open and close tags.
- Invalid tags.
- Ambiguous tags.
- Unnecessary tags.
- Incorrectly used tags.

It will perform the tests according to those requested through input.

**Input:**
- A HTML Document, X.
- A list, L, of tests to perform.

**Output:**
- A list, R, of results from each test performed.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- The HTML file must be readable
- L must contain a valid list of tests.

**Post Conditions:**
- All HTML code in X has been examined.
- Each test in L should be performed on X.
- R must contain only the results of the tests in L..

**Side-Effects:**
- None

## *Specification 1.2*

**Function:** Test for browser dependency issues.

**Description:** This function will test HTML Code for any browser dependency issues. It will discover if the browser will be seen any differently when viewed in Microsoft Internet Explorer and Netscape. It will perform this test in accordance with the configuration given as input.

**Input:**
- A HTML Document, X.
- A list, L, of tests to perform.

**Output:**
- A list, R, of all browser dependency issues in X and where they occur in X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- The HTML file must be readable

**Post Conditions:**
- All HTML code has been tested.
- All browser dependency tests in L have been performed on X.
- R contains the results of performing each test in L on X.

**Side-Effects:**
None

## *Specification 1.3*

**Function:** Check for spelling and grammar mistakes.

**Description:** This function will check HTML Code for the following:

- Misspelled words.
- Duplicate words.
- Irregular capitalisation.
- Style errors.
- Grammatical structure.
- Level of readability.

It will perform this test in accordance with the configuration given as input.

**Input:**
- A HTML Document, X.
- A list, L, of all tests to perform on X.

**Output:**
- A List, R, of all results of all grammar and spelling concerns in X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- X must be readable.
- L must be a valid list of tests.

**Post Conditions:**
- All HTML code in X has been examined.
- All tests in L have been performed on X.
- R contains the results of running each test in L on X.

**Side-Effects:**
- None

## *Specification 1.4*

**Function:** Test for inconsistencies in code.

**Description:** This function will test for inconsistencies in the code throughout the HTML document. This includes inconsistent nested table structures, fonts, colours, alignment and formatting. It will perform these tests in accordance to the configuration information provided as input. It will return all of the inconsistencies discovered in the tests.

**Input:**
- A HTML Document, X.
- A list, L, of tests to perform on X.

**Output:**
- A List, R, of all inconsistencies in X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- X must be readable.
- L must be a valid list of tests.

**Post Conditions:**
- All HTML code in X has been examined.
- All tests in L have been performed on X.
- R contains the results of running each test in L on X.

**Side-Effects:**
- None

## *Specification 1.5*

**Function:** Construct suggestions on how to improve HTML document.

**Description:** This function will construct and return suggestions on how to improve HTML document. This will include suggestions on how to remove unnecessarily nested tables, empty tables, and inconsistencies (Specification 1.4). It will construct these suggestions according to the configuration information provided as input.

**Input:**
- A HTML Document, X.
- A list, L, of suggestions to consider.

**Output:**
- A list, R, of all suggestions that apply to X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- X must be readable.
- L must be a valid list of suggestions to consider.

**Post Conditions:**
- All HTML code in X has been examined.
- All suggestions in L have been considered with respect to X.
- R contains all suggestions in L that apply to X.

**Side-Effects:**
- None

## *Specification 1.6*

**Function:** Check all contained links.

**Description:** This function will check all contained links and return a list of all links in the document that reference invalid addresses. It will perform this test in accordance with the configuration information given as input.

**Input:**
- A HTML Document, X.

**Output:**
- A list R of all invalid links in X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- X must be readable.

**Post Conditions:**
- All HTML code in X has been examined.
- R contains every invalid link contained in X.

**Side-Effects:**
- None

## *Specification 1.7*

**Function:** Verify all pictures, sounds and included files on the page.

**Description:** This function will check all files included in a HTML document. It returns a list of all included files that are corrupt or unreadable. It performs this test in accordance with the configuration information provided as input.

**Input:**
- A HTML Document, X.

**Output:**
- A list, R, of all invalid embedded files in X.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- X must be readable.

**Post Conditions:**
- All HTML code in X has been examined.
- R contains every invalid file that is corrupt or unreadable.

**Side-Effects:**
- None

## *Specification 2.1*

**Function:** Configure test options.

**Description:** This function will allow a user to configure the test options for the HTML Verification System. The configuration will allow a user to determine the subset of tests to perform on a HTML file (See Specification 1.1 for a list of tests).

**Input:**
- None.

**Output:**
- A list, L, of all tests to be performed.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- None.

**Post Conditions:**
- L contains every test option chosen by the user.

**Side-Effects:**
- None

## *Specification 2.2*

**Function:** Configure report options.

**Description:** This function will allow a user to configure all report options for the HTML Verification System. It will allow a user to choose the following:

- The file format used to store the report.
- The test results to include in the Report (See Specification 1.1).
- The location used to store the generated report file.

**Input:**

- None

**Output:**

- The file format, X, used to store the report.
- The test results, Y, to include in the report.
- The location, Z, to store the generated report file.

**Requires:**

- The HTML Verification System must be installed.

**Pre Conditions:**

- None

**Post Conditions:**

- X, Y, and Z are all the options chosen by the user.

**Side-Effects:**

- None

## *Specification 3.1*

**Function:** Produce detailed report file.

**Description:** This function will examine the results of all HTML tests (Specifications 1.1-1.7) and produce a detailed description of all the results of this test.

**Input:**
- File format, X, to use to store report.
- The location, Y, to store the generated report file.
- Results, Z, from all tests (output from Specifications 1.1-1.7)

**Output:**
- A file, F, containing all test results.

**Requires:**
- The HTML Verification System must be installed.
- A Test has been performed, and results from this test available.

**Pre Conditions:**
- There must be sufficient space available in Y to store F.
- X, Y and Z must all be valid.

**Post Conditions:**

- F must be in format X, and located at Y.
- F must include all results in Z.

**Side-Effects:**
- None

## *Specification 4.1*

**Function:** Run Schedule.

**Description:** This function begin a process that will run every scheduled verification at their scheduled times. It will produce reports as defined by the configuration input.

**Input:**
- A list, X, of verifications where each will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Output:**
- A report for each verification performed.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- Each HTML file in X must be readable.
- All values in X must be valid.

**Post Conditions:**
- All scheduled verifications in X with scheduled verification times in the past must have been performed and reports produced for each of these verifications. (Requirements 1 & 3).

**Side-Effects:**
- None

## *Specification 4.2*

**Function:** Add a scheduled verification.

**Description:** This function will add a new verification to the already scheduled tests. The added verification will have an associated configuration.

**Input:**

- An input Verification, X, containing
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.
- A list, L, of verifications where each will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Output:**

- A new list, Lnew, of verifications where each verification will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Requires:**

- The HTML Verification System must be installed.

**Pre Conditions:**

- The HTML file in X must be readable.
- All values in X and L must be valid.

**Post Conditions:**

- Lnew contains all verifications in L as well as X.

**Side-Effects:**

- None

## *Specification 4.3*

**Function:** Delete a scheduled verification.

**Description:** This function will delete a verification to the already scheduled tests.

**Input:**

- A verification X containing
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.
- A list, L, of verifications where each will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Output:**

- A list, Lnew, of verifications where each verification will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Requires:**

- The HTML Verification System must be installed.

**Pre Conditions:**

- All values in X and L are valid.

**Post Conditions:**

- Lnew contains all verifications in L EXCEPT for X.

**Side-Effects:**

- None

## *Specification 4.4*

**Function:** Edit a scheduled verification.

**Description:** This function will add a new verification to the already scheduled tests. The added verification will have an associated configuration.

**Input:**
- An input Verification, X, containing
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.
- Updated information, U, including some combination of
  - An updated HTML Document
  - An updated list of all tests to be performed.
  - The updated file format used to store the report
  - The updated location to store the generated report file.
  - The updated Scheduled time(s) for verification.
- A list of verifications, L, where each will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Output:**
- A new list, Lnew, of verifications where each verification will contain
  - A HTML Document
  - A list of all tests to be performed.
  - The file format used to store the report
  - The location to store the generated report file.
  - A Scheduled time(s) for verification.

**Requires:**
- The HTML Verification System must be installed.

**Pre Conditions:**
- All values in X, U and L are valid.
- X must already be in L.

**Post Conditions:**
- Lnew contains all of the verifications in L.
- Lnew contains X, but the information associated with X adheres to U.

**Side-Effects:**
- None

# Appendix D

Below is attached an example of a HTML file that could be used for white box testing of the HTML Verification System.  It will test several paths of execution since it contains many of the errors that will be detected in the system.

```
<HTML>
<HEAD>
<TITLE>HTML Verfication System</TITLE>
</HEAD><BODY BGCOLOR="FFFFFF" TEXT="400080" LINK="FF0000" VLINK="FF0000"
ALINK="FF0000">

<CENTER><H1>Testing and Verification of HTML Code</H1></CENTER>
<BR>
<BLOCKQUOTE>The Verification Process will search through HTML code and detect any errors,
inconsistencies or suggestions.  The test that it perform on the chosen HTML file will be a subset
of the following test.</BLOCKQUOTE>
<BR>
<UL>

<LI><B>Test for matching opening and closing tags.</I>
<!--Opening and closing tags do not match-->

<LI><G>Test for tags which are not valid.</G>
<!--The tags <G>...</G> are not valid tags-->

<LI><U>Test for missing tags.
<!--The above sentence is missing a closing </I> tag -->

<LI><H5><H5>Test for tags which are unecessary.</H5></H5><I></I>
<!--Heading 5 only needs to be applied once so one set of <H5>...</H5> is not necessary  as well
as <I></I>-->

<LI>Chek for speling and and grammer.
<!--The HTML Verification should find all spelling and gramatical errors in the document -->

<LI>Check for invalid or incorrect links at
<A HREF="HTTP://ww.incorrectlinks.com">ww.incorrectlinks.com</A>
<!--The above link is not a valid link -->

<LI>Check for invalid files.<IMG SRC="FILE://image.bif">
<!--The above file is not the proper format.  The extension .bif is not correct-->

</UL>

</BODY>
</HTML>
```