



On the Application of Metrics  
in Web Systems  
(for Submission)

Darren Stephens  
Centre for Internet Computing  
University of Hull

April 3 2002

**Acknowledgments**

Firstly, I would like to thank both my supervisor, Dr Craig Gaskell and my technical expert Mr Angus Marshall for their support, patience and advice during the gestation of this document and the ones that preceded it.

I would also like to thank Paul Warren, whose work in the area of web evolution and role in the WebSEM project (a collaboration between CIC and the University of Durham) has helped to crystalise my interest in an area I have spent a great deal of time thinking about for a number of years.

Finally, I would also like to thank Nikki and our parents and friends for showing more patience and understanding than could be reasonably expected when I have been close to deadlines.

**Typesetting**

This document has been typeset using several implementations of L<sup>A</sup>T<sub>E</sub>X version 3.14159 in Windows 2000, XP and RedHat Linux 7.x in draft forms between December 2001 and February 2002. Diagrams were produced by a number of tools able to generate PostScript graphics; most particularly Dia and Kontour in Linux.

It has been printed using a Hewlett Packard DeskJet 710C for draft copies and a Hewlett Packard LaserJet 4M for the final versions.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	An Historical Aside . . . . .	1
1.2	What Is Web Engineering? . . . . .	2
1.3	Why Do We Need Web Engineering? . . . . .	3
1.4	Format of this Document . . . . .	4
<b>2</b>	<b>The Nature of Web Systems</b>	<b>7</b>
2.1	Introduction . . . . .	7
2.2	The Origin of Software Engineering . . . . .	8
2.3	The Form and Function of Software . . . . .	8
2.3.1	Validity of Software . . . . .	9
2.3.2	A Conceptual Model for Software . . . . .	10
2.4	The Form and Function of Documents . . . . .	12
2.4.1	Collections of Documents . . . . .	13
2.4.2	Mark-Up Languages . . . . .	14
2.4.3	Validity of Documents . . . . .	16
2.5	The Web - A Hybrid System . . . . .	18
2.5.1	Linking in Webware, Software and Documents . . . . .	20

2.5.2	Summary . . . . .	23
<b>3</b>	<b>Web Engineering</b>	<b>25</b>
3.1	Introduction . . . . .	25
3.2	Recent Work in Web Engineering . . . . .	28
3.2.1	The Web Engineering Problem Domain . . . . .	30
3.2.2	Foundations of Web Engineering . . . . .	33
3.3	The Evolution and Form of Web Sites . . . . .	34
3.3.1	“Web Engineering is Not Software Engineering” . . . . .	36
3.4	Webware, Software, Documents and Hypermedia . . . . .	38
3.4.1	Webware as Software . . . . .	38
3.4.2	Webware as Software-Like Systems . . . . .	39
3.4.3	Webware and Document Engineering . . . . .	40
3.4.4	Webware and Hypermedia Systems . . . . .	42
3.5	Mark-up as a Programming Paradigm . . . . .	43
3.6	Summary . . . . .	44
<b>4</b>	<b>The Use of Metrics</b>	<b>47</b>
4.1	Introduction . . . . .	47
4.2	The Process of Measurement . . . . .	48
4.3	Metrics Within Software Systems . . . . .	49
4.3.1	Types of Software Metric . . . . .	49
4.3.2	The Evolution of Software Metrics . . . . .	50
4.4	Metrics Within Document Systems . . . . .	52
4.5	Metrics Within Web Systems . . . . .	54

4.5.1	An Example of a Potential Web Metric . . . . .	56
4.6	Summary . . . . .	58
<b>5</b>	<b>The Research Proposal</b>	<b>61</b>
5.1	Definition of Thesis . . . . .	61
5.2	The Research Programme . . . . .	62
5.2.1	Overview . . . . .	62
5.2.2	Definition of Metrics . . . . .	63
5.2.3	Construction of Metric Capture Tools . . . . .	64
5.2.4	Evaluation of Metrics . . . . .	66
5.3	Summary . . . . .	67
<b>6</b>	<b>Conclusions</b>	<b>69</b>
6.1	Summary of Progress . . . . .	69
6.2	Further Work . . . . .	70
<b>A</b>	<b>Software Evolution &amp; Taxonomy</b>	<b>71</b>
A.1	A Software Timeline . . . . .	71
A.2	A Taxonomy of Software . . . . .	73
A.2.1	The Software "Family" . . . . .	73
A.2.2	Imperative Programming . . . . .	75
A.2.3	Algorithms . . . . .	76
A.2.4	Object Oriented Systems and Programming . . . . .	77
A.2.5	Alternatives to the Imperative Model . . . . .	79
A.2.6	Self-Modifying Systems . . . . .	81

<b>B Document Taxonomy and Characterisation</b>	<b>83</b>
B.1 A Short Taxonomy Of Documents . . . . .	83
B.1.1 The Structure of Mark-Up Documents . . . . .	85
B.1.2 Document Type Definitions . . . . .	88
B.1.3 Document Validity . . . . .	88
<b>C Provisional Timetable for the PhD Project</b>	<b>91</b>
C.1 Project Timetable . . . . .	91

# List of Figures

2.1	A well formed but invalid Java program . . . . .	10
2.2	A well formed but invalid Perl script . . . . .	10
2.3	A Software Onion . . . . .	11
2.4	An example of SGML style mark-up in HTML . . . . .	16
2.5	Generic Software Transform Model . . . . .	19
2.6	Generic Software Transform Model applied to Software Systems . .	20
2.7	Generic Software Transform Model Applied to Mark-up Systems .	21
3.1	Evolution of Web Projects . . . . .	26
3.2	An Overview of Web Engineering Literature . . . . .	31
3.3	Booch - Diagrammatic Representation of a Web System . . . . .	35
3.4	Hassan - Diagrammatic Representation of a Web System . . . . .	35
4.1	A large but simple HTML document . . . . .	56
4.2	A smaller but more complex HTML document . . . . .	57
A.1	A Software Timeline . . . . .	72
A.2	The Software Family . . . . .	74
A.3	Functional consideration of an algorithmic process . . . . .	76

A.4	Genetic Algorithm Heuristic . . . . .	82
B.1	A Taxonomy of Engineered Artefacts . . . . .	84
C.1	Preliminary GANTT Chart for PhD project - Current at April 3 . . . . .	93

# Chapter 1

## Introduction

### 1.1 An Historical Aside

Towards the end of the Nineteeth Century, what is today described as “Classical Physics”, seemingly explained the world in totality, providing powerful predictive models for describing physical pheomena.

Two phenomena, however, caused a great deal of consternation in the community of physicists. Both the ‘Photoelectric Effect’ and the so-called ‘Black Body Radiation’ problem seemed to not to be explained satisfactorily by existing theories concerning the propogation of electromagnetic radiation (i.e. light). The wave theory of propogation did not appear to explain the results of the two experiments in an adequate way.

The ensuing argument over the nature of light polarised into two camps; those who supported the wave model, the properties of which were most notably formalised by Maxwell, and those who proposed that light was particular in nature. This second camp formed as a result of the work of Planck’s work on black body radiation <sup>1</sup>, Einstein’s work on the Photoelectric Effect [Einstein1905] and de Broglie amongst others.

The supporters of the wave model pointed to the vast body of empirical evidence that not only supported the model but in some cases was actually

---

<sup>1</sup>a black body is both a perfect emitter and absorber of radiation

predicted by it. Even Max Planck himself had great trouble in believing his own quantum theory in the face of the large body of evidence in support of the wave model. In reply, the supporters of the particle model needed only to cite the two cases described above as evidence that that wave model could not satisfactorily explain *all* physical phenomena and was, therefore, somehow deficient.

After much disagreement it came to be realised *both* models could be used at various times and that also there was a theoretical basis for both models to co-exist quite happily, partly as a result of the work of those such Bohr and Schrödinger. In the words of Richard Feynman; [Feynman1964],

“Today, we understand better that what counts are the equations themselves and not the model used to get them. We may only question whether the equations are true or false. ”

At this stage, this particular course of events may not appear especially relevant. As we shall see, however, the idea of two apparently different models being used to describe the same phenomenon is one that holds a great deal of appeal for contemporary work in the field of **Web Engineering**

## 1.2 What Is Web Engineering?

At the present time, Software Engineering models have been successfully<sup>2</sup> used to describe both the process and product of engineering software systems.

Web Engineering is a relatively new computing discipline, identified in an attempt to solve the problems of designing usable and maintainable web systems. Many of the major works in the area are only about five or six years old and many of the most useful contemporary ones two years old or less. The main reason for this has been the prodigious, visible and rapid evolution of the web. The seemingly breakneck speed of evolution of new technologies and tools to enable web-based working and development is constantly adding to the complexity of creating such systems. The methods of creating such

---

<sup>2</sup>the degree of this success is an issue to be considered elsewhere

systems appear to require more than the methods that Software Engineering alone can provide.

Some, such as Boldryreff and Warren, suggested that the issue of whether web systems are software is immaterial. The important issue is that such systems are *observed* to behave in similar ways to software systems, whose behaviour is already relatively well-known. For this reason, the same measurements may be also applicable in these cases [WebSEM2000].

Such an approach is consistent with the philosophy of the Lord Kelvin [Kelvin1891], who insisted that the ability to measure is a necessary condition for understanding a system or phenomenon, although there are some who believe that accurate and objective measurement is not possible within software [Lewis2001].

The ability to be measured, although necessary, is not in itself a sufficient condition to provide full understanding of any process. For this reason, the construction of some model that may in part explain behaviour quantified by measurement is essential to understanding. This model need not be complete, as it may be superceded by others. Although many would like to treat webware as either a software or a document system, it is not unreasonable to believe that it may be, like the quantum models before it, a hybrid that can borrow from whatever models seem appropriate at the time. The reasons for this thinking will be discussed later, when the nature of web systems is dicussed.

The problem of choosing a suitable model has more than a little resonance with the problem faced by the physicists investigating the behaviour of light at the turn of the Twentieth Century. As Feynman so accurately observes, however, in the end, the model is not the end in itself, merely a means to it.

## 1.3 Why Do We Need Web Engineering?

The aim of this document is to bring together and discuss current work in a number of related fields pertaining to the deisgn and maintenance of web-based systems. For convenience, during the course of this paper, the term *webware* will sometimes be used as a generic description for such projects.

When Tim Berners-Lee [Berners-Lee2000] developed the World Wide Web, he devised HTML as a relatively easy way of presenting content. This ease was one of the web's great strengths, being more than partially responsible for the rapid growth in use: almost anyone could create web pages. To do this, Berners-Lee also created one of the web's principal weaknesses; it was easy to create faulty web pages. The early web was made up of predominantly small sites, so the problem of engineering was not a significant one. As usage of the web exploded, though, the size of sites grew extremely rapidly and the problems associated with large sites being updated by multiple users began to be noticed.

The growth in the size of web projects is leading to what some are calling a 'web crisis' [Zelnick1998], in much the same way that the spiralling size and cost of software systems in the late 1960s and early 1970s led to what was called the 'software crisis'. This situation led to the models developed to produce robust, well-engineered systems, such as the rise of Structured Analysis and Structured Design as well as Object-Oriented techniques.

The creation of web projects has been done using a variety of ad hoc methods (such as techniques for site navigation mapping). Part of the reason for this for this lack of rigour is that there is no agreement on a single definitive description of the nature of web site as, let alone how to produce one in an ordered way. This presents web developers with a problem; how to collect meaningful metrics for a system that is difficult to model consistently. The proseed research projects hopes to demonstrate that objective metrics for web systems can be obtained and that they can be a good indicator of the quality<sup>3</sup> of a web site.

## 1.4 Format of this Document

The body of the document begins with chapter 2, which will first consider the nature of the web and its relationship with more conventional software and document systems. To do this, the properties of both documents and software themselves must be examined. By the end of the chapter the relationship between documents, software and webware will have been established.

---

<sup>3</sup>what this 'quality' is will be discussed later

Chapter 3 will survey the field of the new area of web engineering, providing a working taxonomy of current research directions. Much of the work surveyed is relatively recent (i.e. within the last 5 years), although some major works in the area, such as those by Douglas Englebart and Vannevar Bush, are up to 60 years old.

Chapter 4 then goes on to discuss how the work done so far in the web engineering field may allow consideration of what types of metric could be devised later during the PhD project. To help do this, some background about the use of metrics and their application within software engineering will be introduced, before wider discussion of potential applications in the web context are considered.

The research proposal will be presented in Chapter 5, where the initial form of the PhD thesis will be presented. Chapter 6 provides a summary of the work outlined within this document.



# Chapter 2

## The Nature of Web Systems

“Ceci n’est pas un pipe”  
- *René Magritte (1926)*

### 2.1 Introduction

The World Wide Web has now been in existence for a little under eleven years <sup>1</sup>. In that time, the size of the web (and the Internet in general) has increased hugely. As result, the creation of first web documents and then fully connected systems has had to evolve rapidly also. This has involved a great deal of work on the part of developers to ‘scale’ their systems. This has, though, been done a relatively unscientific way. Part of the problem is that no real agreement exists as to what kind of system a web site is.

This chapter aims to describe briefly the nature of both software and document systems and then to describe web systems in relation to them. To do this, the nature of software and documents themselves must be considered, as well as how such systems are engineered.

---

<sup>1</sup>The W3C web site <http://www.w3c.org/History.html>) notes that the official release of the CERN web server tool place on May 17 1991

## 2.2 The Origin of Software Engineering

Almost from the very beginning of the computer age, it has been apparent to practitioners that software was not a manufactured good, but an engineered item. However, as distinct from other complex (usually physical) engineered systems, software is highly mutable and subject to rapid and significant change during its lifecycle. The term “Software Engineering”, describing software and the process used to create it appears to have first been used at the NATO Software Engineering Conference, held at Garmisch, Germany in October 1968. The Garmisch conference provided the first public acknowledgement of an impending “software crisis”. [Dijkstra1972]

In the early days of programming, programs had to be hand-written for specific machines and tasks because of different programming environments and operating systems. Even today, in the era of object orientation and so-called code re-use, large portions of computer programs are still written on an individual, bespoke basis. To help us arrive at a meaningful definition of what we call software, we need to consider two things:

1. the evolution of software as an engineered entity
2. the differing conceptual models that describe such artefacts and their production.

These definitions need to be arrived at if any meaningful discussion of the difference between software and the web is to be considered at all. A fuller discussion of the evolution of software is provided in Appendix A

## 2.3 The Form and Function of Software

Roger Pressman [Pressman2000] describes software as an “information transformer”. As a useful starting point, Pressman’s definition of software can be presented in the following way:

Software -

Any system or agent (whether singular or composite) that acts to transform information from one state into another.

Somerville's definition of software is similar but includes not only program code, but also any documents or configuration information associated with it [Somerville1989].

Such definitions are open to many interpretations. They could, conceivably, define systems that would not normally be considered to be software, such as mechanical or even to physical systems that serve to transform information. Clearly then, a software system can be described as a complex entity containing both code and non-code components.

### 2.3.1 Validity of Software

In a "traditional" software system, the process of compilation occurs in three stages discussed by [Bennett1996]:

1. **Lexical Analysis** determines whether correct the 'words' have been used. In other words, whether the language used is correct.
2. **Syntactical Analysis** determines whether words are arranged in a way consistent within some prescribed grammar. For a piece of software source code this prescribed grammar may be in form of a language specification or of a DTD or schema for a document.
3. **Semantic Analysis** attempts to produce meaningful constructions from syntactically analysed structures produced in previous stage. There is no guarantee that a pass on syntactical analysis will produce code that will satisfy a semantic analysis, in much same way as a grammatically sound piece of prose may make no sense.

The Java and Perl programs shown in Figures 2.3.1 and 2.3.1 perform similar tasks similar but have different outcomes. The former will not compile but the latter *will* execute, although possibly with unpredictable consequences because the variable `$third` is undefined. Although neither of these programs is strictly valid (undefined variables exist in both) they are both syntactically correct. In other words both programs are not valid but they are *well formed*. The only difference is one of implementation, the treatment of errors in semantic analysis.

```
public class Test
{
    int first=1;
    int second=2;

    int theAnswer = first + second + third;

    System.out.print("The answer is: " + theAnswer + "\n");
}
```

Figure 2.1: A well formed but invalid Java program

```
#!/usr/bin/perl

$first=1;
$second=2;

$theAnswer = $first + $second + $third;
print "The answer is: $theAnswer\n";
```

Figure 2.2: A well formed but invalid Perl script

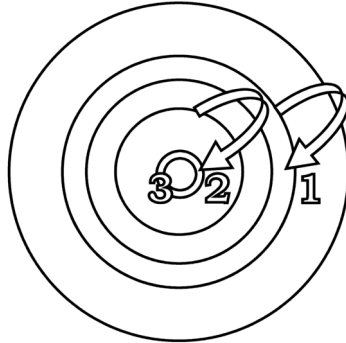
These illustrations are given to show that the property of correctness within a ‘program’ is mutable. This is an issue that provides a connection to document and web systems, as we shall soon see.

We can then describe a web system as a composite software entity and also a document system. The documents therein are also software. We can see this by considering the following model.

### 2.3.2 A Conceptual Model for Software

The bare description of software does not describe such a system in an entirely satisfactory way. Figure 2.3.2 shows a representation of a layered process describing the interactions between each stage of an artefact’s development cycle. Each stage, is reached by creating an item which is a representation, in an alternative form, of the item preceding it. We can visualise this system

rather like the layers of an onion. Each layer wraps and encapsulates the one below it.



Key:

1. Design Conceptual Space. This contains an abstract space corresponding to a design addressing a particular problem domain.
2. Program Conceptual Space. The expression of the design space in a machine understandable form.
3. Artefactual Context. The context chosen for the execution of the programmatic representation to produce the final artefact. This may be, like Java, bytecode; a directly executable item of object code or even transient outputs like webpages or scripts.

Figure 2.3: A Software Onion

Software is generally written to be built and executed within a certain environment, either on a real or a virtual machine. This is simply a specialisation of a more general principle: software is highly contextually dependent. A Bourne Shell script outside of a Bourne shell in which to run it is merely a text file or, more accurately, a document..

Each successive shell (or skin) of the onion is simply a representation (or contextualisation) of the one surrounding it. The move from each space to the next is accomplished by one (or more) transforming processes. It may be helpful to consider two examples to illustrate this process:

1. A UML design document to map a solution to a problem may be represented as a computer program using an appropriate language (C++ or Java, for example). This process may use either some sort of CASE tool to help generate skeleton code or may be entirely hand-edited. The source code might then be compiled or interpretation (for instance by use of the gcc compiler in a Linux environment) to produce final, executable code. This code would only be meaningful as executable code in that particular context, such as Java bytecode within a Java Virtual Machine (JVM).
2. A DTD or schema for an XML document type can be used to model the underlying structure of a set of data. This may describe the structure of some real world entity like a technical manual, for example. This XML entity may then be compiled by a browser interpreting the entity, dynamically linking presentational style information and outputting the transformed entity in some form. This would only be possible in a context where the DTD or schema was understood by the interpreting agent (i.e. the agent could interpret XML/SGML data).

The second scenario suggests that in certain contexts a document can be considered in the same way as a piece of software. In this case, it is only distinguished by the method of transformation from programmatic to artefactual context (i.e. a browser acting as an interpreter, having inputs and output in a visual form).

At this point, the distinction between software and web systems is becoming increasingly blurred. This is not entirely bad. Although a simplistic classification of web systems as software may not be appropriate, it may be possible to apply a more dualistic model, as discussed in the introduction to this paper, to extract useful properties of webware that can be observed and measured.

## 2.4 The Form and Function of Documents

The SGML specification document [ISO8879] provides the following two definitions within its glossary:

- **Document** : A collection of information that is processed as a unit. A document is classified as being of a particular document type.
- **Document Type** : A class of documents having similar characteristics; for example, journal article, technical manual or memo.

The first of these definitions describes the salient characteristics of a document. The definition is a very broad one and could encompass a very wide range of entities. The key words to understand within the definition are, however, *collection* and *unit*. The former implies that a document can be a composite entity, consisting of a number of parts. The latter implies a sense of ordering or structure which can be used to describe the whole.

The second definition states that documents may be classified depending upon features or properties that a particular class of documents may have. The definition further implies a type or commonality classes of documents and that this can be represented in an ordered and reproducible way.

Documents may take many forms, both electronic and not. For the purposes of this document however, discussion will be restricted to documents that are represented and transferred in electronic formats.

### 2.4.1 Collections of Documents

It is more usual for documents in areas other than hypermedia to be self-contained but also to reference other documents, and in addition share common elements by means of inclusion, generally by bibliographies and footnote systems. The problems of consistent addressing and referencing are interesting within themselves but will not be discussed within this document except in their most general sense. This issue is already addressed within web hypermedia by use of URI system.

Hypermedia documents (more often called “nodes” by hypermedia practitioners), however, have associative relationships (links) between them which enable users to move freely between nodes. It is possible to consider such a number of nodes as comprising a single interconnected, composite, distributed document entity. Even then it is difficult to ascertain the extent or boundaries of such composite entities as Warren has discussed [Warren2001b].

This relationship between nodes or groups of nodes has been examined in work done in the area of hypermedia research. Such node systems can also be considered in an object-oriented way.

## 2.4.2 Mark-Up Languages

Documents can be constructed using what is known as a *mark-up language*. Such a language is used to identify and show elements of document structure within the document itself. The term was first used by Charles Goldfarb during the development of GML, the embryonic mark-up language developed by IBM in late 1960s.

Bryan[Bryan1997] states that any device that stores formatted text for later retrieval and use must use some form of mark-up to do so. This category therefore includes program source code. The mark-up may only be machine readable in some cases however. In general terms the following types of document exist:

- **Unformatted Text**

The simplest form of document is the unformatted, raw text document, The data contained inside is not subject to any formal structure although it is actually to provide a (trivial) DTD for such a document. It can also be argued that, without such a DTD, a document would, in fact, conform to a notional type definition, especially in the case of natural language documents. This notional definition would have to be determined by the reader for the document to have any meaning, however.

- **Specific Mark-Up**

Specific mark-up is used in systems such as PostScript or RTF processing in word Processors like Microsoft Word or Corel WordPerfect. The mark-up is usually application or task specific and is not designed to be transferable between environments or applications. This can also include styles such as delimiter separated lists, where each field's meaning may only be known by the parsing application. As such, specific mark-ups are commonly used in typesetting systems, where the major degree of formatting control is exerted by the machine.

- **Generic Mark-Up**

One of the stated goals of generic mark-ups is to allow encoded information to be exchanged within a number of environments and to allow the resultant documents to be viewed and edited by both machines and people. Generalised mark-ups are commonly stored in plain text files, making them highly portable between systems, as tools to convert text file formats are effectively universal.  $\LaTeX$  is an exception to the specific mark-up situation that usually occurs with typesetting languages. The mark-up used in the  $\TeX$  family is actually generic and is easily transformable into HTML or PDF, for example. <sup>2</sup>

The description of generic mark-up documents is further divided in two ways, Presentational and Logical;

- **Presentational Structure** describes how a document will *appear* when used. It is specifically concerned with the description of visual elements and how a document will appear when viewed. As far as version 3.2, HTML had begun to evolve along this route. HTML 4 and its variants attempts to reverse this trend.
- **Logical Structure** is more concerned with the layout of document in terms, not of its visual appearance, but how it may be interpreted by machines or how its *meaning* may be interpreted by a human user.

The move to develop generic mark-ups began in the late 1960s as a result of first Gencode, and later GML (both at IBM), led by Goldfarb [Goldfarb1970]

Marking-up can take a number of forms and could be done both by machines and by human authors. This does not however, strictly speaking, have to be case, although one of stated aims of SGML standard was to create document mark-ups that were both human readable and editable . For instance, HTML is easily written and read by both human and mechanical readers but a Microsoft Word document (to choose an arbitrary example) is not. In that case, the underlying structure is not obviously apparent to a document author although such a structure does exist.

---

<sup>2</sup>Many of these tools are part of the standard  $\TeX$  distributions

A document is marked up by means of “tagging”. and this tagging may (or may not) be human readable. Data within the document are enclosed by tags to describe *elements*. The tag itself is enclosed by delimiters and the choice of such delimiters is arbitrary. An example of HTML mark-up, showing the nesting of elements and the use of tagging can be seen in Figure 2.4.2.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type"
      CONTENT="text/html; CHARSET=iso-8859-1">
<TITLE>An Example of HTML</TITLE>
</HEAD>

<BODY>
<H1>An HTML Document</H1>
<P>This is an example of an HTML document.</P>
</BODY>
</HTML>
```

Figure 2.4: An example of SGML style mark-up in HTML

The particular style shown has become increasingly popular because of the rise in use of mark-up languages like HTML and XML. Both mark-ups are descended from the SGML standard that Goldfarb defined. Elements do not have to be atomic and can contain others, although this is dependent upon the mark-up language used and the DTD (q.v) that defines it.

### 2.4.3 Validity of Documents

Stated simply, a valid document is one that conforms to its associated Document Type Definition (DTD). However, it is apparent on the Web at large that a great number of documents are not strictly valid HTML but are still viewable by a range of user agents. Clearly, validity alone is not a complete description of a document’s ‘correctness’.

Documents may also be *well-formed*. A well-formed document is one that

conforms to a grammar defined by DTD. By definition a valid document is also well formed. However, a well-formed document may also contain elements or attributes not defined within the DTD. Typically, in an HTML context, such items not within a DTD will be ignored but the document will still be processed if it is structured correctly (i.e. they are syntactically correct). This provides a connection to software systems. The Perl program shown above in Figure 2.3.1 is, as stated previously, invalid but it *is* well-formed. This comparison is valid even though an HTML document does not look like a ‘program’ as such.

For mainly historical reasons, HTML browsers tend to be “sloppy” when parsing. to allow user agents to display nodes in at least some form, however ugly, and not to burden end users with error messages. Browsers such as W3C’s Arena and Amaya browsers have broken with this convention and do allow users to view errors in badly-formed HTML, if they so wish.

In web browsers, display of HTML is done in three stages:

1. **Retrieval**, where document is fetched, whether from network or from a local source.
2. **Parsing**, where the document is read and composed into a document tree, containing document elements. Whether this parsing is done by a validating parser or not is an issue of implementation. This stage resembles software compilation.
3. **Rendering**, where the parsed tree is passed to a formatting agent for display. This stage more closely resembles *linking*, where external resources (such as images and external scripts) are retrieved and form part of the ‘execution’ of the resource.

The sloppy parsing of HTML was a key reason for its rapid take-up by amateur users, allowing badly formed mark-up to be written and displayed. However, XML or SGML parsers are usually built to require validity<sup>3</sup> Syntactic and semantic errors are reported, especially because, in many cases, visual rendering is not required, as many mark-ups are not intended directly

---

<sup>3</sup>In the case of some XML parsers, the absence of a DTD is permitted, meaning that well-formed structures are permitted because strict validity cannot be determined

to be display languages. In XML this job should be done by formatting systems such as XSL (eXtensible Style Language).

Authors have long spoken about the process of ‘programming’ in HTML. At this stage it may be instructive to consider mark-up languages such as HTML as a type of declarative programming language. There is more than a little justification for thinking in this way, given the work by Eisenstadt & Brayshaw [Eisenstadt1988] that provides a connection between declarative and object-oriented programming. This is a viewpoint that corresponds with that of Michael Buckland [Buckland1997], who also makes comparisons between documents and ‘objects’ in a general sense. This relationship is even more specific in the use of HTML because of the existence of the Document Object Model (DOM) [W3C2001a], allowing developers to exercise a great deal of control over documents using scripting techniques. It is possible, using Javascript, to write and alter HTML documents ‘on the fly’, in a programmatic way. The fact that HTML has no control structures or data flow does not necessarily prevent it from being a purely declarative programming language, especially when one considers the type of programs written in a language like Prolog where the focus is on the relationships between predicates (or in the case of HTML, elements).

## 2.5 The Web - A Hybrid System

We have already seen that current thinking on the issue of web engineering roughly polarises into two points of view; one, that the web is simply another type of software and should be considered as such, the other that the web is a complex heterogeneous system and new approaches must be found to describe it. The former is evident in the discussion within workshops such as WSE2001 [WSE2001] while the second is more common in the work of those like Deshpande and Hansen [Deshpande2001].

In essence, the software process can be distilled to a transformative process (as seen in Figure 7), changing information from one form into another by the use of some agent. This can be seen to happen during the parsing of a document, if we interpret the definition given in 2.4 correctly and treat the language specification as a largely notional DTD. This is easy to do, given the structure of specification languages like the Backus-Naur Form (BNF)

[ISO14977] and its descendants. In effect, software source code is a human-readable representation of the machine-readable artefact in document form <sup>4</sup>. This is a major reason why HTML documents, being generalised mark-ups, can be more comparable to software than other (more specific) mark-ups. Generally, although mark-ups are human-readable they are commonly machine generated and used for the purposes of data transformation. Although the use of tool-based generation of HTML takes place, much of it is still produced or tuned by hand. This factor alone is a significant one.

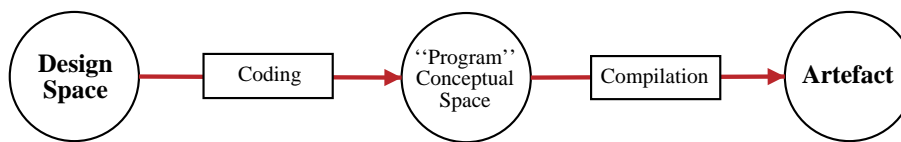


Figure 2.5: Generic Software Transform Model

Source code is indeed processed as a unit (by a compiler) to produce an artefact, whether it is transient (interpreted) or tangible (compiled). Transient items may also include web systems, where pages are generated as a result of a transformative process (parsing and rendering). This being the case, the question of whether web systems are classifiable as software can be raised.

If we consider a browser to be a transformative agent, we can also consider the types of linking within web systems and relate them to common software.

Figure 7 shows the relationship between source documents and the artefacts they represent. There will be at least some mirroring of higher level structure

Jelliffe [Jelliffe1998b] also provides a mapping in this respect, putting forward the opinion that user interfaces are also documents, because especially in the online context, users don't just read documents - they interact with them. Further, the process of UI design should always be done with an

---

<sup>4</sup>The W3C's specification for XML [XML2000] states that the formal grammar can be written in an extended BNF form

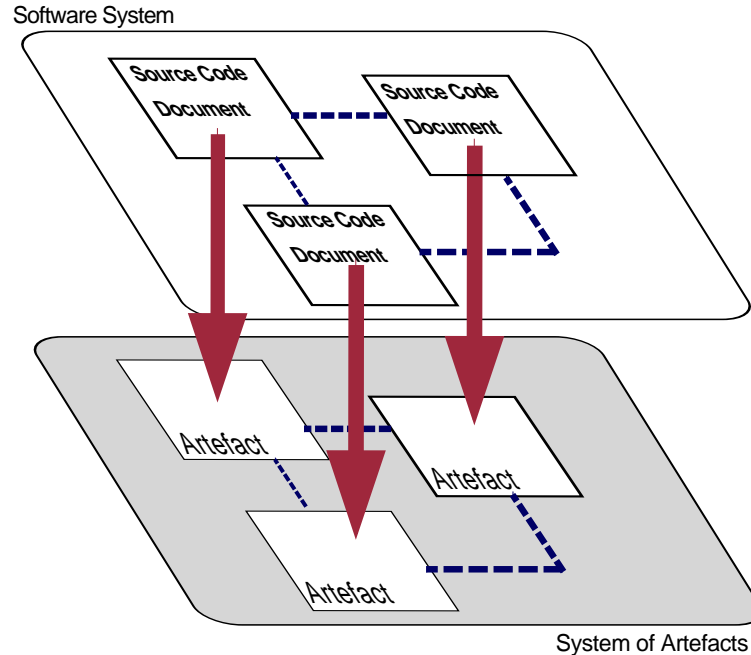


Figure 2.6: Generic Software Transform Model applied to Software Systems

appreciation of the users who will use such a system. Techniques such as *Extreme Programming* (XP) [Maurer2002] attempt to add this approach to the engineering of projects. It is perhaps not surprising that XP has found acceptance in web development communities for this very reason.

### 2.5.1 Linking in Webware, Software and Documents

Linkage, the defined association between nodes within a web system, can be classified broadly in one of two ways, in a similar way to software systems:

#### 1. **Dynamic**

Items such as the IMG element in HTML or the use of LINK to retrieve externally linked stylesheets can be classified as dynamic linkages. The resource is requested and (possibly) retrieved at run-time<sup>5</sup>. They are

<sup>5</sup>here, we define 'run-time' as the time of viewing at the user agent

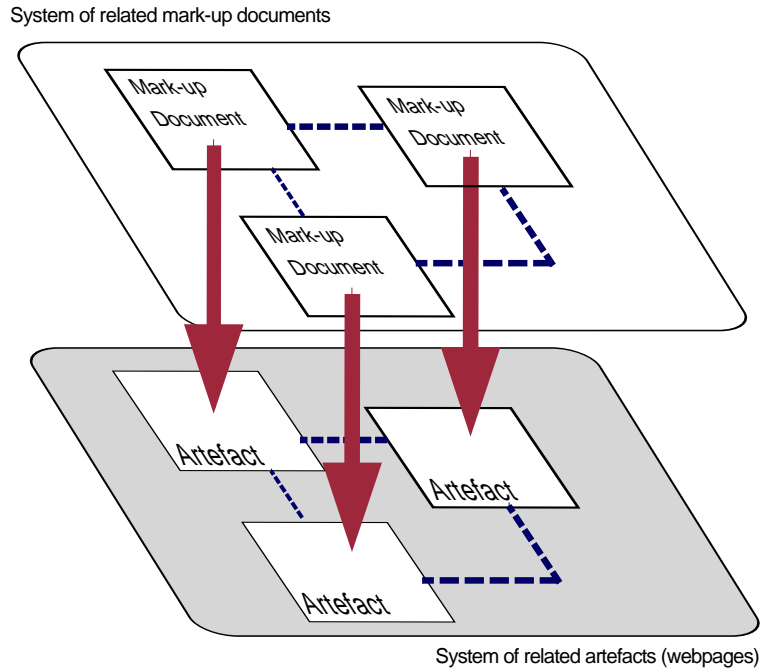


Figure 2.7: Generic Software Transform Model Applied to Mark-up Systems

held separately from the calling node until the time they are requested.

## 2. Static

Static linkages occur when external elements are linked *before* before run-time<sup>6</sup> The principle mechanism for this in the WWW context is the use of of server-parsed documents or server-side scripting. This mechanism effectively acts as a static linker.

We can further characterise two subtypes of dynamic link available to the *end user*:

### 1. Elective

These links are *elective* for the user, such as A HREF (typical links) or AREA HREF (used in image maps) in as much as users can choose

---

<sup>6</sup>This can be thought of as ‘compile-time’ linkage

whether these links are followed at the time of viewing and navigation paths can be followed directly.

## 2. Non-elective

Links of this type (such as LINK, BASE or SCRIPT elements) are *structural* and describe externally linked objects to the node. Such links are usually used to contextualise and orientate links and resources relative to the chosen node (BASE and LINK) or to link to embedded or external content (e.g. SCRIPT, which can utilise externally stored script content). Items such as IMG elements can also be placed within this category as the ability of a user agent about whether to render externally linked items is one of user agent configuration (usually an implementation issue or customisable by the user at ‘run-time’).

It can be argued that even the so-called structural links have an element of electivity. An end-user can choose to disable functions of the client to disable these features. However, such functionality is not within the control or scope of the document, merely an implementation issue connected with the user agent and its configuration.

Warren [Warren2001b] proposes that Lehman’s laws of software evolution miss an important category of maintenance, that of *speculative maintenance*. However, he does not specify how this property manifests itself in a software context. In hypermedia systems, URLs, being pointers can sometimes “break”. The prevalence of the HTTP 404 response code and the use of URL redirection demonstrate this. Warren proposes that speculative maintenance can be performed on web systems by link checking, a form of pre-emptive maintenance to prevent ‘link rot’.

The problem of speculative maintenance *does* also exist for software systems, although not in a such an obvious way as most details of linkage are hidden from the end user.

Consider a software system that makes use of dynamically linked libraries (i.e. the library code is located and executed at run-time). In the C language, for example, the library code could be referenced by using a function prototype declared as **extern**. The calling code would then be directed to look in a known list of locations until it finds a match. This match order is determined by environment variables (such as PATH in DOS or LD\_PATH

in some variants of Unix). If a linked library is missing or damaged then execution will be impaired or perhaps even impossible.

This is a very similar situation that which one finds in web based systems and, in fact, tools now exist within operating systems now which server to perform a relatively limited form of speculative maintenance to make sure such libraries are both present and current. Some of these features can be found in Microsoft Windows XP, released in 2001 [MS2002].

It is, therefore, possible to consider external function prototypes and URLs to be equivalent, as they both identify named external resources. Internal anchors are a special case of this, in essence being recursive calls to the same resource, merely choosing a separate entry point.

### 2.5.2 Summary

In this chapter we have discussed the nature of web systems, documents and software. It can be seen that each of them is a system that requires sound principles of engineering and management to produce an artefact that can be described as 'good' in some way (the ways of determining this quality are discussed in Chapter 4).

Broadly, both web and software systems are documents in nature. Each is a human readable representation of a system at a particular level. Traditional notions of software, especially considering here the works of Wirth [Wirth1976] and Kowalski [Kowalski1979] expect a separation between content (data) and program (logic or control). In the case of mark-up systems, this is not the case: data and 'program' are part of a single entity; the **node**.

Software and mark-ups (especially HTML and some of its XML based relatives) share common notions of re-use and modular design, evident in the parallels drawn in dynamic and static linking.



# Chapter 3

## Web Engineering

“If it looks like a duck, walks like a duck, and quacks like a duck, it’s a duck.”

- *proverb*

### 3.1 Introduction

The discipline of “Web Engineering” is a relatively new one. The term seems not to have been in (wide) use before 1998 and the 9th ACM Conference on Hypertext and Hypermedia [Bieber1998], although Murugesan claims to have used the term as early as 1997[WebE2001]. This is in contrast to the more general area of hypermedia engineering, which had been defined at least as far back as the early 1990s. A similar term “Web Document Engineering” had been used earlier, in 1996 by Bebo White of SLAC (q.v) in a session at the 5th World Wide Web Conference in 1996[White1996].<sup>1</sup>

Given the relative immaturity of the area of web engineering, much of the early literature consists mainly of position papers and overviews of the problem domain of engineering quality in web projects. Many of these papers also describe issues of macroscopic structure, the need for component reuse and process engineering and the problems of mapping the limits of web

---

<sup>1</sup>This conference was held in May of 1996, in Paris, France. The conference programme is available on-line at: <http://www5conf.inria.fr/>

sites. Other early works such as Powell [Powell1998a] acknowledge the rapidly changing nature of web projects and attempt to codify some ground rules for working on such projects.

As an aid to understanding the need for more structured methods for the creation of web projects, it is, perhaps, useful to refer to Powell, who divides the evolution of web applications into a number of generations, illustrated in Figure 10.

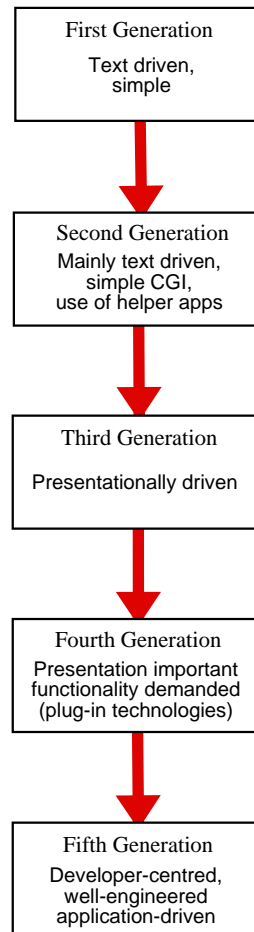


Figure 3.1: Evolution of Web Projects

The generations can be explained in the following way:

1. The first generation predates the development of the Mosaic browser and inline images. Browsers of this nature were text-based systems. Mark-up existed purely to define content, not its presentation. The first example of this type was the initial site at CERN in Switzerland (<http://info.cern.ch/>), the first public web server.
2. The second generation, like the first, was simple, mainly consisting of static webpages and very basic CGI processes. These initial projects were usually ad hoc and lowly budgeted. Extra functions were generally provided by the use of helper applications that needed to be configured at the client.
3. In the third generation, begun by the release of Netscape Navigator, presentational issues assumed greater importance. As Powell observes, many third generation sites were driven more by appearance than usability. This was an opinion shared by Jakob Nielsen in his regular surveys of web usability.
4. In the fourth generation, users became used to “pretty” sites and demanded more functionality. This was provided by additional content, the use of multimedia and technologies such as Java to provide interactivity and increased functionality. Much of this was (and is) provided by ‘plug-in’ technologies or the use of scripting and other client side technologies.
5. The web now appears to be entering a fifth generation. This generation is driven, not by user requirements but by those of developers. The growing size and complexity of web systems and the tasks they are being designed to perform is the cause of this evolution. Part of this is a perceived need to provide component based services (using J2EE or .NET, for example)

Powell refers to this most recent generation, describing it as ‘Software Centred Web Site Development’, making direct comparisons with software development while acknowledging differences, most particularly in terms of timescales and lifecycles. Once again, there is a return to using markup purely to define content, separating it from presentation, especially now through the burgeoning use of XML and its applications. Berners-Lee talks about this at some length in his book ‘Weaving The Web’ [Berners-Lee2000].

This change is as a result of the growing need for interoperability between systems as Internet platforms become more fragmented <sup>2</sup>. The size of sites also forces developers to implement this type of more regular design.

In this chapter we examine the nascent discipline of Web Engineering and attempt to place frame it in terms of its relationship to its close relative, Software Engineering.

## 3.2 Recent Work in Web Engineering

Warren et al. [Warren2001a], in a paper presented at WSE2001, have already proposed a basis for understanding and classifying research being done on engineering and metrics for the World Wide Web. According to them, much of the current work in web engineering is based around three major areas, viz.:

### 1. Software Metrics

The area of software quality measurement is an enormous one. A comprehensive review is well beyond the terms of this document. Readers are advised to refer to the work of both Pressman (q.v.) and Somerville [Somerville1989] for an introduction to the area of software metrics. In his paper, In the WSE2001 paper, Warren describes the use of systems such as GQM (Goal-Question-Metric) [Berghout1999] to define general principles for creation of metrics . The application of this has been principally in software systems and the possibility exists that at least some of these can be extended to the web. There is an underlying assumption that web systems can be described as ‘software-like’ in nature. This is by no means obvious.

### 2. Hypertext

Web systems are generally seen to be specialisms of more general hypertext systems. Current work in this area may have impact on the more specialist one. Of particular interest are applications and metrics derived from the use of methods such as OOHDM [Schwabe1995] and

---

<sup>2</sup>The increasing number of versions of the Windows platform as well as the number of -not always-compatible Unix variants is evidence of this

RMM [Isakowitz1995], the most common in use. This is in addition to the original works of Hatzimanikatis et al [Hatz1995]. and Botafogo et al. [Botafogo1992], which attempted to address the problem of defining metrics for idealised hypermedia systems. This area also describes document engineering, which can, with some justification, be described as a subset of hypermedia engineering in general. Latterly, the work of Mendes et al. [Mendes2001] attempts to define and measure metrics by means of case studies in hypermedia development.

### 3. Human Computer Interaction

A great deal of work has been done in the area of HCI on the web, especially (although not exclusively) by Jakob Nielsen, who has a long history in the investigation of usability engineering. Some may argue that any study of engineering for the web must give great consideration to user interaction because of the highly user-centric nature of any systems developed. This is a valid observation, but not necessarily of direct concern when considering the production of systems in the first instance, for example. Although this field is without doubt of interest it will not be discussed at any further length in this document.

It appears that little thought is being given at present to the issues of considering mark-up based systems in much the same way as one would consider programs. In fact, as recently as 1996, authors such as Stross [Stross1996] could say with some confidence:

“A web is a publication, not a piece of software.”

This view is one that is quite clearly becoming outdated. especially when one considers the opinions of the WebSEM project (q.v) a mere four years later [WebSEM2000]. Web projects are growing rapidly, both in size and complexity. This can be seen in two ways:

1. The number of 'pages' on the web is rising very rapidly, faster than the number of web users indicated by recent surveys. Google estimates this figure at 1,610,476,000 'pages' at November 16 2001. This figure has grown by around 60% in less than a year. By February 2002, this figure had risen to over two billion. Nielsen estimated that there were

around 100 million web sites in January 2000. This number would rise to 25 million by the end of that year and then further to around 100 million by the end of 2002. Conversely, Internet usage in the UK at least had fallen very slightly in mid 2001, according to a recent survey by the UK Office of Telecommunications (OfTel) [OfTel2001]. It appears that internet usage, in the short term at least appears to be reaching a plateau, but the number of pages on the web is growing at great speed.

2. The number and scope of content development and delivery tools for the web is growing extremely rapidly and significant communities are rising up to exploit them (e.g., PHP, ASP, ColdFusion, SourceForge, Zope etc.)

The process of producing such systems, marrying the disparate technologies they use and for measuring their quality is becoming rapidly ever more complex. Already there is an appreciation that this increase in the size and complexity of web projects is a burgeoning problem. A so-called “web crisis” [Zelnick1998], mirroring the “software crisis” that was seen in software in the 1970s and 1980s. The concerns of practitioners of that time, such as Dijkstra, Parnas and Wirth, calling for increased refinement, modularisation and re-use are finding resonance with current web developers. Many of the overview and position documents identify this parallel between current web development and earlier software engineering.

### 3.2.1 The Web Engineering Problem Domain

As has previously been alluded to, managing the production and maintenance of web systems is a non-trivial task, analogous to the management of software systems. This issue is explored in early work by Murugesan et al. [Murugesan1999], defining the problem domain of web engineering. The theme is continued by Deshpande and Hansen [Deshpande2001]. Both works make a distinction between ‘traditional’ software and web-based systems. These differences amount to a combination of three main factors;

1. The importance of the user interface - more so than ‘usual’ software.
2. High Network Overhead - web systems are network intensive in general.

3. Heterogeneous components - web projects are likely to comprise markup, program code and other types of content, typically multimedia.

Each of these can be seen in other types of software development but a combination of these factors, as well as hugely compressed lifecycles for developing such projects, sets them apart.

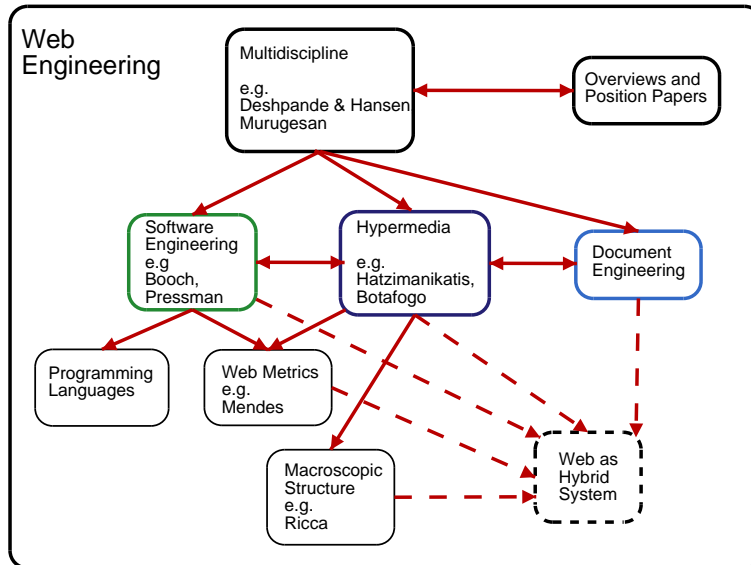


Figure 3.2: An Overview of Web Engineering Literature

These works put the case that web engineering is a discipline of itself and should be treated as such, separately from software engineering. To illustrate how the areas within Web Engineering are connected to other disciplines, a broad illustration of the area is presented in Figure 3.2.1.

Importantly, Roger Pressman, in an article entitled “What a Tangled Web We Weave” [Pressman2000b] uses the phrase “software-related” to describe what he calls “WebApps” (what this paper describes as ‘webware’).

The implication of this article is that he does not see webware systems as *purely* software systems. This is an issue that the Warren WSE2001 paper also addresses, stating that the “software-like” nature of web systems has yet to be established fully. Sensibly, Pressman contends, like others in the

area, that a disciplined engineering approach is essential for the creation and maintenance of useful web systems.

To define ‘disciplined’ he cites the following criteria:

1. Problem solving
2. Good Design
3. Thorough Testing
4. Maintenance

He clearly equates these with the well-established criteria applied in software engineering. Pressman is not the only person to make this connection between software and the web, although differences do appear in the degree of the comparisons that are made. Powell (q.v.) also asserts that such approaches are preferable to the use of the more prevalent RAD [Boehm1997]<sup>3</sup> in web environments. Although RAD is seen as useful because of its use of prototyping (seen as especially useful for interface design), as a method it is wasteful and, judging by the current state of many sites, does not result in well-engineered systems [Powell1998b]. He concludes that this is because many web practitioners are not conversant with more structured engineering methods.

Warren et al. speculate that webware may be only one of a family of more generally engineered artefacts. Engineered items are created using a structured method, such as the waterfall method in software engineering amongst others [Pressman2000c]. Any general method for the engineering of artefacts follows the same basic steps, whatever the artefact may be. These steps are:

1. **Analyse** - Define the problem domain and evaluate current approaches where appropriate.
2. **Design** - Provide one (or more) possible solutions to the problem along with a rationale for choosing the preferred solution.
3. **Build** - Construct the system defined by the preferred solution using the most appropriate methods and tools available.

---

<sup>3</sup>A similar idea is also found in the practice of ‘Extreme Programming’

4. **Test** - Ensure that the construction phase of the project satisfies design criteria and is fit for the purpose of its construction.
5. **Maintain** - Once the construction has reached the required standard and is deemed complete (or a stable configuration agreed), ensure that it continues to meet the required operational standards set and to repair and enhance it if necessary.

Clearly a web system (like software) is subject to all of these and is clearly an engineered system of some type when constructed in an ordered way. This is extended in software engineering by the Spiral [Boehm1988] and Waterfall models (and its variants) . Much of the work by the authors in this field are merely restatements of this seemingly facile assertion. Best practice in the area , defined by the huge number of HTML Authoring texts, largely conforms to these general principles, confirmed by the work of authors such as Powell. The early chapters of Jakob Nielsen's Designing Web Usability [Nielsen2000] concur with much of this thinking. Nielsen firmly places himself in the camp who believe in the use of engineering principles, although he also stresses the importance of creativity and inspiration. Even within this context however, the value of user-centred design is emphasised and the need for sound principles of design is established.

### 3.2.2 Foundations of Web Engineering

As has been established, much of the earliest work in the field only goes as far as calling for a methodical approach in the development of web projects. Work in the OOHDM and RMM systems have tried to establish more regular processes to describe the production process in general hypermedia systems. Such an approach is, however, complicated by the fact that web projects may not, as discussed earlier, be like traditional software. There are special problems to be faced because web systems are heavily loaded toward end-user functionality, use elements that cannot easily (if at all) be described in a way satisfactory to traditional software engineering and are subject to lifecycles generally dissimilar from more conventional software projects.

The area of hypermedia has been extant for a great number of years. It is commonly acknowledged that the defining work is that of Vannevar Bush,

whose article “As We May Think”, published in 1945 describes a prototypical hypermedia system. Nelson, who is believed to have first used the term ‘hypertext’ [Nelson1965] and Englebart, who demonstrated a working hypermedia system (Augment) in the late 1960s, also made huge contributions to the development of hypermedia. Conklin [Conkin1987] provides an introduction to the area of hypermedia a little before the first papers published by Botafogo, Rivlin and Sheiderman.

Much work however has been done on the organisation of hyperdocuments. Much of this later work sprang from two major sources; the work of Botafogo, Rivlin and Shneiderman [Botafogo1992] and work of Hatzimanikatis, Tsalidis and Christodoulakis [Hatz1995]. The thrust of both sets of work was not concentrated on Web documents because, at this time of the writing of the earlier document the web had only been a public entity for about a year. Take-up of web services, although rapid, had not reached sufficient levels to warrant special treatment.

Apart from the consideration of the web as a hypermedia system, others, such as Bray [Bray1996] and Pitkow [Pitkow1995], attempted to consider engineering problems concerned with the web in other ways. Pitkow was, for example, far more concerned about underlying network performance and how to optimise internet systems in this way. Bray’s concerns seemed to be more about mapping the extent and scope of web sites, a handy precursor to the work of those like Warren.

### 3.3 The Evolution and Form of Web Sites

The linear development of web systems, in which there is some considerable overlap between generations, was greatly affected by browser development. This development went hand in hand with the evolution of the HTML mark-up ‘language’ itself [W3C2001c] and related technologies such as XML. The general progression has been from static document-based systems to dynamic systems of much more complexity, including elements of document systems, software and other, less easily classified components.

Much current work on the engineering of web systems concentrates upon navigational issues (e.g Ricca and Tonella [Ricca2000a],[Ricca2000b]). This

is largely in line with the often cited Botafogo and Hatzimanikatis papers that also concentrate on ‘global’ structures and metrics, giving little if any consideration to the node level measures that both papers do identify as an issue.

Booch [Booch2000], Hassan [Hassan2001a] and Hassan & Holt [Hassan2001b] also discuss the macroscopic structure of web systems. In Hassan’s case this is mainly out of a concern to be able to reverse engineer web system architecture. The representations they provide for the modelling of such systems are shown in figures 3.3 and 3.3

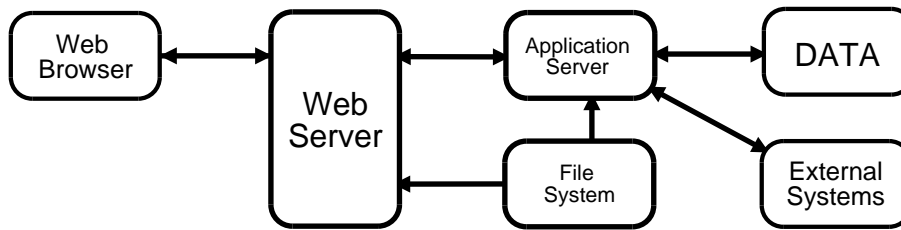


Figure 3.3: Booch - Diagrammatic Representation of a Web System

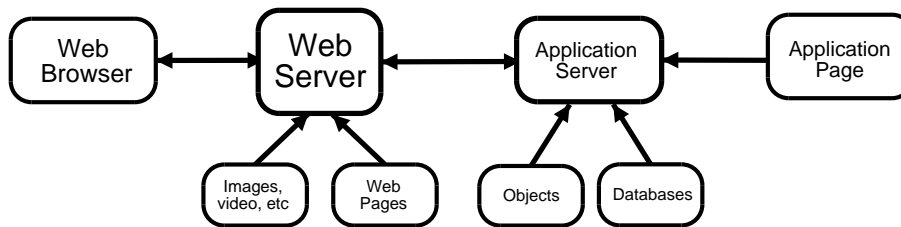


Figure 3.4: Hassan - Diagrammatic Representation of a Web System

In both cases, the architecture is essentially three-tiered, comprising a browser (end user), Web Server (middleware system) and Application Server.

Fine detail within these systems differ but Booch alone *directly* illustrates the filesystem level as an important component. Booch’s analysis goes a little further by classifying the right-hand side of the diagram as more in keeping with more traditional client-server systems, while the left-hand side details the more newer web-based extensions to the process, an evolution from more traditional software. The elements of the user interface and network are

added at this point, concurring with other analyses of the properties of web projects.

### 3.3.1 “Web Engineering is Not Software Engineering”

Warren, in his work (especially [Warren2001b]), suggests that web sites obey Lehman’s Laws of Software Evolution, but also that web systems require one type of maintenance that is not extant in standard software, *speculative maintenance*. Powell too spends a great deal of time considering the web in terms of software engineering [Powell1998c]. Part of this consideration is a list of properties of a well-engineered web site he provides:

1. **Correctness** - The site performs its specified functions and does so as if error free. Correctness is a difficult quality to specify entirely: sites that may *appear* to be correct may contain incorrect (invalid or badly formed) components.
2. **Testability** - The site can have functionality and usability tested as thoroughly as possible, preferably with test data and test scenarios for sites containing interactive components.
3. **Maintainability** - Making changes to the site should be as easy as possible, including the ability to make small changes or even to add or delete entire sections easily.
4. **Scalability** - The site is able to support increases in the number of users. The site should also be able to be ported to other servers easily, for the purpose of mirroring or clustering, for example.
5. **Reusability** - The site should be component-based where possible, allowing developers to re-use similar code in other projects or to incorporate code from other sources into the current one.
6. **Robustness** - The site should be able to be used by users with the confidence that services will be available and properly functioning. This applies not only to interface issues, but also those of bandwidth, server uptime and properly functioning back-end components.]

7. **Readability** - Source files and resources used to build site components should be perspicuous for developers, once again adinig maintainability. Many of the issues involved, such as commenting and formatting of source code are those that apply in ‘traditional’ programming.
  
8. **Well-Documented** - Well-documented sites aid maintainability, in that the project history is there for future maintainers to see and consult when needed. The importance of on-line help for users as sites expand should also not be underestimated.
  
9. **Appropriately Presented** - The function and target audience of the site should be key considerations for developers when developing interface components.

These qualities are more specific restatements of Pressman’s application of sound engineering principles. The first six items certainly describe the quality of the product in engineering terms. Two of the final three items also define the quality of the systems for its developers, alluding the more inclusive concept of *habitability* (q.v.) of a system for **all** of its users. The very last items refers to user interface creation and branding within commercial projects.

Powell does however state, quite categorically, **that web and software engineering are different**. This is a direct contradiction of the WebSEM project [WebSEM2000], which stated that that webs were, effectively, software. This is partly as a result of the more document and content-focused nature of many sites and also because of the culture of the developer base. With a large emphasis upon user interface, the aesthetics of the system take a much more prominent role. However, as we discuss in Chapter 2, there is some common ground to be seen in these principles with areas such as Document Engineering, where the problem of organised structured documents has been considered as part of other work in the field of SGML, by Rezgui and Debras for example [Rezgui1995].

## 3.4 Webware, Software, Documents and Hypermedia

As has been discussed earlier, most of the earlier literature (and much of the later literature too) identifies clear parallels between web and software systems. Even considering the work of people like Booch, Pressman that we have already discussed, there is little at present to connect the two areas in any significant way. Little progress has seemingly been made since the first publications in the late 1990s.

### 3.4.1 Webware as Software

The programmes for the major software evolution conference ISPSE / IW-PSE <sup>4</sup> have contained nothing specifically related to web evolution in the recent past, while the ICSM conference incorporates the WSE workshop on web evolution. Generally, the software evolution community appears to be paying little attention to the specific problem of web evolution.

The work of Boldyreff and at the Research Institute in Software Engineering (RISE) in Durham in the UK, amongst others, does try to place Web engineering within a wider software engineering context. RISE has, in the past, and is continuing today, to do a great deal of research in the area of software metrics, but most particularly in the field of software systems evolution and in the field of web maintenance, evolution and metrics (viz WWWMaint, the WWW Maintenance project and WebSEM-Web Site Evaluation Metrics, in conjunction with CIC Scarborough) [RISE2001]. Much of this work has connections with the work of Lehman and his laws of software evolution ([Lehman1985], [Lehman1999] *inter alia*), which is a starting point for some of Warren's work [Warren2001b], [Warren1999]. There are also connections to more general software evolution, apparent in the work of Glover [Glover2001], also working at Durham, who describes the improvement of software systems to allow easier maintenance. In fact, in the project home page of the WebSEM project [WebSEM2000] the following quote appears to confirm their opinion;

---

<sup>4</sup>the International Symposium / Workshop on Principles of Software Evolution

“We have shown that websites behave according to many of the same rules as software (to all intents and purposes, a website *is* (sic) software).”

Defining the evolution of web systems in this way seems only to manage to describe web systems *purely* in software terms. Given some of the emerging properties of document systems, and the way in which they (both documents and their properties) are being considered by those who create them, this is not likely to provide satisfactory answers to the problem of the nature of the web.

To some degree Warren’s work shares some of these concerns, although his paper presented at WSE2001 [Warren2001a] does admit the possibility that web and software systems may be members of a higher level family of engineered artefacts. This is likely to be a rich area of exploration in the near future.

### 3.4.2 Webware as Software-Like Systems

Many software engineers have found themselves commenting on the production of web-based systems, including Pressman [Pressman2000a], Boldyreff & Lavery. There seems to be a belief in these quarters that web engineering is merely a specialised case of engineering a software system.

Pressman acknowledges that major differences exist between “traditional” software engineering projects and web engineering [Pressman2000b]. Most of these are connected with software lifecycle, where many theories about software evolution can be greatly twisted. Lehman’s laws of software evolution are applicable but the lifecycle of a web project is likely to be highly compressed and discontinuous. This can lead to differences in engineering approach in web systems.

The work of people such as Hassan and Holt [Hassan2001a], [Hassan2001b] as well as commentaries by Booch [Booch2000] amongst others point to an interesting middle ground: WebApps (as Pressman calls them) are like software systems but are also heterogeneous, containing different types of components and need to be largely platform independent. Pressman also identifies some

characteristics of WebApps that make them distinct from so-called “traditional ” software.

- Firstly, development lifecycles for WebApps tend to be shorter <sup>5</sup>.
- Secondly, security is a major consideration when using a web based application.
- Thirdly, the user interface and aesthetics are given a much higher priority largely because of their greater role. Unlike software projects (with the possible exception of hypermedia projects), web projects are largely much more content driven.

### 3.4.3 Webware and Document Engineering

The use of the term “document engineering” as applied to the World Wide Web is also a relatively new one. The need to create ordered and well-made documents seems to have been a key issue when the first mark-ups were defined in the mid-1970s. An important part of the development of mark-up languages was a recognition that many different types of data may need to be represented (in some cases) or referenced, where representation was not possible <sup>6</sup>. Indeed, Rick Jelliffe [Jelliffe1998] identifies this feature as a highly important one in any mark-up language. Earlier browsers used the helper application to access the data referenced by HTML documents in the usual way.

The rise in the use of XML and also in the use of technologies such as XSLT has resulted in improved engineering of systems because the construction of such items has required them to be done in a more regular, modular way. The recognition of the need for engineering principles to be applied to web documents was noted in a session at WWW6 in 1995 by Bebo White of SLAC (Stanford Linear Accelerator) <sup>7</sup> In this session at the conference he defines a ‘Web document’ as a collection of linked ‘Web Pages’. He also goes on to

---

<sup>5</sup>Discontinuity notwithstanding

<sup>6</sup>The SGML standard makes provision for utilising data that the author does not wish to be, or cannot be, marked up

<sup>7</sup>SLAC ran the first non-European web server, going on-line 12 December 1991, according to <http://www.w3c.org/History.html>

define the term ‘Web Document Engineering’ a precursor to the more generic term ‘Web Engineering’ coined around two to three years later.

White’s definition of web documents harks back to some of the earliest work in hypermedia by those such as Bush [Bush1945] and Whorf [Whorf1956], who can characterise systems in terms of associations between entities within them. Clearly, such entities are *linked* - creating a definition of a hypermedia system. In Bush’s *Memex* system, these links were entirely personal in creation, while Englebart, for instance, found this problematic, believing that limited human cognitive abilities required the imposition of hierarchies and abstractions to allow data to be processed more efficiently [Englebart1961]. This creation of hierarchy may itself be a useful measure of site quality.

Although little published work seems to exist in this specific area at present there is a definite increase in interest, resulting in the creation of a new forum, the ACM Symposium on Document Engineering [ACMSDE2001], held in Atlanta, USA in November 2001. Unfortunately, the proceedings of this conference were not available at the time of writing although they are expected in the very near future. The call for papers to this conference acknowledges that, ‘documents’ are no longer static, physical entities’. It continues, to provide this instructive definition of a document from a more contemporary viewpoint:

“A document is a representation of information that is designed to be read or played back by a person. It may be presented on paper, on a screen, or played through a speaker and its underlying representation may be in any form and include data from any medium. A document may be stored in final presentation form or it may be generated on-the-fly, undergoing substantial transformations in the process. A document may include extensive hyperlinks and be part of a large web of information. Furthermore, apparently independent documents may be composed, so that a web of information may itself be considered a document.”

This may seem to illustrate the major connection between webware and software in a traditional sense; In the source state, both webware and software are human readable systems, designed to be read by its inhabitants (to use

Christopher Alexander's terminology again); developers. Thus, there can be no doubting that, in the source state, software projects are also document systems. This, once again, reinforces the idea that web and software systems may be part of a more general family of artefacts.

### 3.4.4 Webware and Hypermedia Systems

Low and Hall, during the course of a survey of web engineering practice, found little, if any, evidence of well-defined metrics extant in the more general area of hypermedia. Given the amount of work done in the hypermedia field, this is somewhat surprising. It is, however an area of burgeoning interest in the software engineering sphere as even more traditional texts such as Pressman's [Pressman2000] now make reference to engineering quality in webware, even though it may only be at a superficial level.

Work by Boldyreff and associates does address this to some small degree but almost purely from the perspective of software engineers. Little attention seems to be being paid to the uncomfortable (for software engineers) issue of how non-software components are engineered and their roles within wider systems. For a web project this is a major issue, both in terms of product and process.

Even some of the most recent work (such as that of Ricca and Tonella) seems to concentrate on the macroscopic structures within web hypermedia. Little seems to be being written about the previously mentioned relationship between mark-up languages and what Wirth describes as, "formal notations" [Wirth1977] rather than programming languages per se. It appears that little progress has been made from previously mentioned papers published in 1992 by Botafogo et al. and in 1995 by Hatzimanikatis et al. where idealised hypermedia systems are modelled, mainly to analyse navigation and movement through them.

This emphasis upon large scale structure influences work on usability issues [Nielsen2001],[Krug2000]. None of them sufficiently describe the problem of the creation and maintenance of web systems from the point of view of the developer. Work in large scale hypermedia appears to be like macroeconomic theory: it may help to explain large scale phenomena like inflation or interest rates but does not, for instance, seem to explain the operation of smaller

systems such as the markets for certain products. In a similar sense, explanations of global navigation do not describe behaviour inside nodes within that system. As Deshpande and Hansen discuss (q.v), this may be an issue that is uppermost in the engineering of web systems.

### 3.5 Mark-up as a Programming Paradigm

The architect Christopher Alexander describes the concept of “inhabitability” in an architectural system. He asserts that those who finally use the structure are not, in fact, its *only* users. Even those who build and maintain such systems also need to use it in some way. Good architecture also takes account of the needs of these users. Thus, the engineering quality of a web system affects not only its end users but also those who ‘inhabit’ the code while writing, testing and maintaining it. In building such systems the needs of the developer must also be given appropriate weight.

This is a subject spoken about by the programmer Richard Gabriel in his book ‘Patterns of Software’, [Gabriel1996]. The importance of building code that is easy for developers to ‘inhabit’ is an established one for normal software systems. Software metrics attempt to quantify the factors that developers feel to be important indicators of code’s complexity compared to its ease of use.

Although not widely accepted and controversial, there are those who classify HTML (and markups generally) as VHLLs (Very High Level Languages) in much the same way that a language such as SQL might be [Frostburg2000]. Even Lowe and Hall, as discussed previously, make at least some connection. Open as this point is to argument, it has not prevented some from attempting to provide more conventional measurements for HTML. Capers Johnes [Johnes2001] provides figures for function points within HTML, classifying it as a Very High Level Language in a similar grouping to languages like Perl and Python.

Such languages are more naturalistic for programmers (and non-programmers) and are generally more declarative in nature. A cursory examination of introductions to HTML and web authoring introduce users to “programming in HTML”, showing that there is a widely held perception that construction

of HTML is a programming task of *some* kind, although it is not immediately clear what kind that may be. It is puzzling as to why this impression of writing HTML as programming is not more evident in literature about hypermedia authoring.

More contentiously, such languages may also be more suited to visual programming methods. They are also generally highly limited in scope and are not useful for general purpose activity, as are many lower level languages. Previous internal discussion papers [Stephens2001a], [Stephens2001b] have proposed similar ideas to explain why some types of software metric may be appropriate for measuring quality in web hypertext systems.

## 3.6 Summary

Most of the work so far in the field of Web Engineering has focused on only a few specific areas. This is perhaps not so surprising, given the highly disparate nature of web systems and the inherent difficulties in trying to tie them together cogently.

In the web's infancy, the major research concerns centred around the topology of the web and delivery of services. The engineering quality of the sites themselves were not considered formally. Early webmasters developed support networks, used ad hoc engineering methods and passed them on as best practice. Thoughts about the engineering quality of web systems only started to become a pressing issue as the size of projects started to balloon. Indeed, the suggestion that Lehman's Laws of Software Evolution [Lehman1985] could also apply to web systems is an indicator that some sort of relationship exists between the two systems and, that being the case, some of the quantitative methods used in software engineering could be applied to the web.

The time when such ad hoc methods could be relied upon as a method of ensuring quality has now passed: the size of web projects necessitates group working and production of such systems on a large scale.

Now that the engineered quality of web systems is being more actively considered the work divides into several relatively distinct areas, each flawed in some small way as part of a larger picture.

One of these areas is ‘Idealised Hypermedia’, as seen in the work of Hatzi-manikatis et al, Shneiderman et al and latterly with Ricca & Tonella. Each of them concentrates mainly on hypermedia systems at the global level, only focussing on those metrics. Node-based measures are hardly considered at all.

Others, like Boldyreff, for example, seem to consider web systems simply as adjuncts to software systems. This attitude persists because, seemingly, only the software components of such systems are being considered and not the mark-ups contained at the node level. The role and nature of markup itself seems to have been largely forgotten or relegated to a level of little importance in the wider scheme of web systems development, possibly because the production mark-up it is not considered as a type of software development.

Document engineers, if the evidence of recent activity is to be believed, are beginning to re-evaluate the function and role of documents. There is a growing appreciation that documents are now no longer static entities. There is seemingly no acknowledgement, however, that the construction of a document system has some visible similarities to the construction of software systems.

Given the discussion in Chapter 2 ( which re-iterate material included in Appendices A and B), that suggest that web systems are, like software systems, instances of a more general type of engineering principle, a case can certainly be made for placing web engineering as a companion to software engineering. This is an approach taken by the work of Powell which was discussed earlier in this chapter. They are not the same, simply because webware and software are not the same, merely similar.

As a result of conventional thinking that software and webware are essentially the same, there appears to be little or no work of a more ‘holistic’ nature with regard to web engineering. Such a hybrid approach, mixing programming paradigms with the creative process of content creation, would seem to be a necessity given the web’s rapid evolution and its need to be a truly heterogeneous cross-platform environment. Once the relationship between software and document system is more clear, the application of appropriate metrics can be considered afresh.



# Chapter 4

## The Use of Metrics

“What is not measurable, make measurable”  
- *attributed to Galileo Galilei*

### 4.1 Introduction

The application of software measurement is now an established and useful discipline within Computer Science. It was the Lord Kelvin [Kelvin1891] who most famously insisted that the ability to measure is a necessary<sup>1</sup> condition for understanding a system or phenomenon,

There are those, however, who do still assert that accurate and objective measurement is not possible within software because software metrics are essentially subjective in nature [Lewis2001]. Given the extensive body of work in the area of software metrics, this is clearly a minority view. Early work such as Boehm’s COCOMO [Boehm1981] model does, however, admit that such models are mere estimates of complex systems and processes and that process metrics are at best what can be described as ‘best guesses’.

---

<sup>1</sup>but not a sufficient one, as we have already discussed in Chapter 1

## 4.2 The Process of Measurement

*Metrics* are abstractions that ascribe values to properties of systems. A metric may be composed of one or more *measures*, which provide a quantification of some property of the system. This property may be measured either directly or indirectly to provide the measure. The IEEE Standard Glossary of Software Engineering [IEEE1993] defines a **metric** as,

*“a quantitative measure of the degree to which a system, component, or process possesses a given attribute.”*

*Measurement* occurs when one or more measures are taken. Metrics are then calculated by interpreting those measurements in some way, usually by summation or some other statistical technique. The implication is that a metric is most often an *indirect* quantifier of some attribute. These metrics will then be used as *indicators* to determine the quality of an engineered product.

Measurement is used to allow us to quantify the properties of attributes that we view around us in the world. The process of measurement assigns some values to those attributes. Clearly, the types of measure that can be made fall into a number of different categories, although they can be roughly classified in the following ways:

1. **nominal**, in which entities within a classification are merely labelled. No further comparison or ordering is possible, like biological taxonomy.
2. **ordinal**, in which an ordering of entities is possible but the ranking represents only the order of sequencing, such as placings in a race.
3. **interval**, where the differences between entities can be assigned but the magnitude of the measures themselves cannot. Consider, for instance, the temperatures 20 and 40 degrees Celsius. It is then meaningful to talk about a temperature difference of 20 degrees. This does not, however, mean that the latter is twice as hot as the former (whatever we choose ‘hot’ to mean).

4. **ratio**, where the comparative size of objects can be quantified, such as length, where it is meaningful to use measures and say that, for example, one object is twice as long as another.
5. **absolute**, where entites are enumerated, such as frequency counts.

A general approach for developing appropriate software metrics <sup>2</sup> is known as the GQM (Goal-Question-Metric) paradigm. In GQM, developed by Victor Basili and associates at the University of Maryland in the USA [Basili1984], metrics are developed using an essentially inductive process in three stages;

1. **Goal**

The major goals and targets of the project are listed.

2. **Question** The major goals re examined and framed in such a way as to determine whether the identified goals are being met.

3. **Metric** The metrics that must be collected to properly answer the questions proposed at the Question stage are identified.

## 4.3 Metrics Within Software Systems

As Pressman [Pressman2000d] remarks, software presents its own measurement problems, with major disagreements as to what to measure and how to do it. Part of this problem is that software systems, unlike structures like bridges or buildings, for example, are not physical structures and direct measures of physical properties are often easier to make and to visualise.

### 4.3.1 Types of Software Metric

There are essentially two types of software metric; *process* and *product* related. Process metrics are, as one might expect, concerned with the efficiency of the process used to produce a system while product metrics are more concerned with the properties of the item produced as a result of that process.

---

<sup>2</sup>or any metrics, in fact

The properties of the product are more likely to be directly measurable, while those of the process are more likely to be indirect (although cost and effort are the major exceptions).

The measures that make up these metrics can be of two broad types; direct and indirect. Direct measures can be made (for want of a better word) directly upon a system. Direct measures include physical properties such as size, execution speed or defects. These are objective measurements and are more likely to be atomic. Indirect measurements are usually composite and measure some less tangible property of a system, such as its reliability or maintainability. At this point, some of these measurements may be said to be more subjective as there may be disagreement over whether the composite nature of the metric models the property fully.

### 4.3.2 The Evolution of Software Metrics

The earliest models (COCOMO [Boehm1981], Function Points [Albrecht1979] and SLIM [Puttnam1978] ) define (production) effort as functions whose independent variables are taken from a pool of possible measures, such as size, level of re-use and developer ability. These functions have a pre-defined structure with coefficients determined by numerical analysis methods and retro-fitting from empirical data (regression). The goal of these models was to measure cost and effort measures for software projects. Hence, the earliest metrics were process oriented. Issues of product quality had been touched on as issues connected with the early days of structured design but quantifying these qualities was not widespread.

In tandem with these measures of process effort, methods of more reliably defining the quality of the products under development were also necessary. Boehm's model tied both product and process measurements together. McCall's FCM model [McCall1977] attempted to achieve this, although Pressman argues that many of the factors suggested by McCall are only subjectively measurable [Pressman2000e].

Many of the measures in these earlier metrics commonly made size a key determinant of complexity. Albrecht's Function Point (FP) model (later extended), however, disputed this and attempted to use measures that more accurately reflected the functionality of the systems and its interfaces to the

external world. According to Pressman [Pressman2000f], function points are counted in five domains <sup>3</sup>;

- *Number of User Inputs*, counting each user input that provides distinct data to the application.
- *Number of User Outputs*, counting each user output that provides information to the user, such as confirmations, errors or reports
- *Number of User Inquiries*, counting immediate responses to any input in the form of output.
- *Number of files*, counting the number of data files used by the application.
- *Number of External Interfaces*, counting the number of interfaces via which data can be transmitted to other systems (eg: network connections, disks etc.).

These items are then assigned complexity values for weighting purposes and a set of complexity adjustment values are applied. Such measure can then be used in a similar way to LOC measures to judge complexity. A weakness of the Function Point method, however, was that it was designed to accommodate business logic and systems. To remedy this, a superset of function points exists, known as *Feature Points* [Jones1991]. The use of Function and Feature Points can be said to provide programming language-independent methods of measure the ‘utility’ of software. It can also be counter-argued that the metrics obtained are less reliable as some subjective evaluation is required to compile them.

### Object-Oriented Metrics

The use of metrics for Object-Oriented systems followed a broadly similar pattern of earlier metrics for structured systems. They were primarily process oriented with a concentration on size as the determinant of complexity. This arose mainly as a result of the work of Pleeger and Palmer [Pfleeger1989],

---

<sup>3</sup>although he does state that this picture is simplified somewhat

[Pfleeger1990]. Later metrics proposed by Chidamber and Kemerer (the CK Metrics suite) [Chidamber1994], however, addressed specific issues of object-orientation more specifically, such as the depth of inheritance tree, as well as other structural attributes.

Berard [Berard1995] defines five properties that can be applied specifically to object-oriented systems;

- *Localization*(sic), describing how closely an object is related to the others around it.
- *Encapsulation*, which describes how a class encompasses the properties and methods of the object on which it is modelled.
- *Information Hiding*, which is related to abstraction. The internal operations of components are hidden from others that do not need to use it.
- *Inheritance*, allowing the properties of an object to be propagated to others.
- *Abstraction*, allowing the developer to concentrate on functionality at the appropriate level.

Whitmire [Whitmire1997] outlines nine properties that exist in object-oriented systems. Some of these are related to the basic properties outlined by Berard but others extend the model. Further metrics exist, such as those proposed by Lorenz and Kidd [Lorenz1994]. These are mainly concerned with items such as class size and issues of inheritance.

For web systems, some of these metrics may prove fruitful at a future point, as it is possible to model web systems in an object-oriented fashion.

## 4.4 Metrics Within Document Systems

Many metrics associated with documents are primarily concerned with their validity and conformance with associated DTDs or Schemas (in some cases

with XML). It can be argued that metrics applicable to hypermedia describe a specialised document class and that many of these metrics are more generally applicable to the base class itself.

Sametinger has approached the issue of measuring and quantifying document systems in work done to better engineer documentation for object-oriented software projects [Sametinger1994]. The structure proposed for such documentation is itself object-oriented, with the same rules governing relationships as that in the software it documents. This allows document metrics to be derived from the software engineering domain to such systems.

Jelliffe [Jelliffe1998a] remarks that most engineering of documents is concerned with re-use of DTD and modularising design. In this sense, many of the concerns are ones shared with software engineering. For this reason, common metrics associated with modularity, coupling and cohesion are also possible to apply to document systems.

The issue of markup quality is now starting to become more widely discussed, as the introduction of journals such as the *MIT Markup Languages Theory and Practice* journal illustrates. As the introduction to the first edition of this journal states, however, many of the practices and metrics are ephemeral or so dissipated as to be impossible to collate [MITML1999].

In more specialised terms however, a larger base of metrics is extant within hypermedia. We must, therefore, also consider whether hypermedia metrics are applicable to webware. As Hall and Lowe comment, however, metrics in the area of hypermedia are not common [Lowe1999a], Mendes, Mosley and Counsell's paper, published in *IEEE Multimedia* in Summer 2001 [Mendes2001] attempts to address this situation by attempting to use some existing metrics to provide a starting point for the production of meaningful metrics. This is partially successful but does not attempt to consider the role of programming in the engineering process.

Work on hypermedia metrics has fallen into two broad categories; global (site) and local (node) metrics. Much of the current work in web engineering takes its starting point as the work of Botafogo et al. and Hatzimanikatis et al. The major issue here however, is that much of the work done is concerned with global metrics, with little or no thought being given to the nodes themselves. This is hardly surprising given the origin of this work is in idealised hypermedia systems. For this reason some, such as Warren and

Boldyreff ([Warren1999] *inter alia*), have attempted to place the collection of metrics for web systems into a more software-centred context.

Much current work in hypermedia engineering is focused on the application of the major design methods, Schwabe and Rossi's OOHDM [Schwabe1995] and Isakowitz, Stohr and Balasubramanian's RMM [Isakowitz1995]. Broadly speaking these two methods are similar in nature although it can be argued that RMM is slightly more user-centric due to its explicit definition of construction and run-time testing stages. As mentioned earlier, the object-oriented nature of these systems suggests that they would be a fertile ground for the application of some types of object-oriented software metrics. Puzzlingly, this seems not to have happened to any great degree.

The use of the term 'programming language' when applied to HTML/XML etc. seems to be applied erratically in literature surrounding the subject. Hall and Lowe do appear to link mark-ups and programming languages together when they discuss the 'Publishing Model' for hypermedia development [Lowe1999b] but, as discussed earlier, there is a view that developing web content is not a programming activity [Stross1996]. This lack of consensus about the nature of web systems obviously presents a problem when considering what kind of metrics are applicable for the web context.

## 4.5 Metrics Within Web Systems

As has been previously mentioned, web projects appear to be undergoing some type of development crisis. There are distinct parallels with software projects of the late 1960s and early 1970s. The discipline of Software Engineering was defined to help address some of these development issues, signposted by such papers as Dijkstra's deprecation of the GOTO statement in languages of the time [Dijkstra1968] and the work of Parnas [Parnas1972] and Wirth [Wirth1971] in calling for more structure, modularisation and re-use. These are issues that are rising up again in the creation of web systems

Traditional software product metrics are broadly comprised of size metrics, complexity metrics and density of comment. Each of these can be measured in a variety of ways. Object-oriented metrics are primarily applied to classes, message passing, coupling, cohesion and inheritance. Many class

based metrics measure the complexity of a class by considering its constituent methods. Traditional metrics can be applied in these cases. Little of importance appears to have been done to produce a mapping basis for metrics between object-oriented systems and node-based hypermedia systems, although object-oriented development methods do exist within hypermedia.

Metrics applicable to the web can be placed into three broad categories, each containing some overlap with the others:

- **Development Metrics** are those concerned with the creation of HTML and other mark-ups, the topology of the hypermedia navigation space and the creation of other components, whether they are programmatic like Java applets or servlets, script based like Javascript or PHP or of other types, such as SVG, Flash, RealMedia etc. These are the already discussed node and global metrics.
- **Deployment Metrics** are concerned with how the engineered items are placed into the server environment and how they interact with other services, such as databases and mail transport, for instance. This is, in part, addressed in the work of those like Pitkow, Bray and similar. This is a major part of engineering a web project, much more so than many in ‘traditional’ software engineering, given the non-deterministic nature of internet performance. Some may class these as ‘operational issues’ and not of relevance to the engineering of the initial system. These may also be classified as global metrics.
- **Delivery Metrics** are those measures which are applicable to the final delivery of the artefact mainly characterised by usability and end-user performance metrics. The principal work in this field has been done by Nielsen and associates amongst others and will not be dealt with further.

It is a misapprehension to believe that any of these fields are totally distinct from one another. Measures taken to address issues in one area may have significant impacts in either or both of the others. This may go some way to refuting the argument that deployment metrics are not relevant. At this stage, however, the areas of delivery metrics and deployment will be glossed over to some degree, leaving development metrics to be discussed at this point.

Unlike conventional software systems, web systems, with their dualistic nature present significant problems for those wishing to use software engineering techniques to measure web site quality. An example of this is the use of LOC metrics to measure complexity within software systems. In these situations, the number of lines of code is used as a direct measure of the complexity of the source. For document systems and particularly for the web, this is not necessarily a valid assumption.

### 4.5.1 An Example of a Potential Web Metric

To illustrate some potential differences between software and web metrics, let us consider the problem of measuring the ‘complexity’ of a web document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>A Simple document</TITLE>
</HEAD>
<BODY>
<PRE>
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed diam nonummy nibh euismod tincidunt ut laoreet dolore
magna aliquam erat volutpat...and so on for several thousand
lines...
</PRE>
</BODY>
</HTML>
```

Figure 4.1: A large but simple HTML document

The file shown in figure 4.5.1 could potentially be several thousand lines long but it is still, in essence, a very simple document with a shallow parse tree. Compare this with the code fragment in 4.5.1. The HTML in this fragment is clearly more complex and would involve more maintenance. It is however, considerably smaller in size than the simpler example. Clearly, a raw size metric will not sufficiently describe the complexity of web documents. Of more interest are two things; first, the relative amount of markup the

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<LINK REL="stylesheet" TYPE="text/css" HREF="style/style.css">
<TITLE>A Simple document</TITLE>
</HEAD><BODY><DIV ALIGN="CENTER">
<OBJECT CODEBASE="/" WIDTH="100%" HEIGHT="100%" ID="logo">
<PARAM NAME="SRC" VALUE="flash/flash_movie.swf">
<PARAM NAME="LOOP" VALUE="false">
<PARAM NAME="SCALE" VALUE="exactfit">
<EMBED SRC="flash/gestalt_movie.swf"
  TYPE="application/x-shockwave-flash"
  WIDTH="100%" HEIGHT="100%"
  LOOP="false" SCALE="exactfit">
  </EMBED>
</OBJECT></DIV></BODY>
</HTML>

```

Figure 4.2: A smaller but more complex HTML document

document contains and second, the complexity of the attributes present that can alter the behaviour of the elements. These are issues that the SGML community have already considered to some degree [NOC1997]

More appropriate measures for static HTML or XML could be proposed as an example; the Specific Element Density (SED) and Attribute Density (AD), of either documents/nodes or entire sites. The former metric represents the density of markup elements per markup node, while the latter represents the number of attributes per tag per page node. The density is normalised to attempt to make the density measure an accurate determinant regardless of absolute size. If we use the symbols  $E$  to represent node element count,  $T$  the node token count,  $A$  the node attribute count and  $n$  the number of nodes. These equations are shown in (4.1) for SED and (4.2) for AD

$$SED = \frac{1}{n} \sum_{i=1}^n \frac{E_i}{T_i} \quad (4.1)$$

$$AD = \frac{1}{n} \sum_{i=1}^n \frac{A_i}{E_i} \quad (4.2)$$

The element density of the node is defined in terms of two variables; the element count and token count. The element count is the number of delimited markup elements (not including closing tags (such as `</a>`, etc), while the token count can be defined as any word within text content or whitespace separated item within the document, i.e. the textual information content.

In the above example (Figure 4.5.1), the amount of text is large and the number of tags small in comparison, giving a more realistic view of a document's complexity. These figures could also be collected from each individual node to give distributions of density across nodes in a site.

These properties may be the major differentiating factor between mark-ups and more conventional types of software. In conventional systems there is a distinction between program and data, following the Wirth model. Markups have more in common with the Kowalski model for Logic Programming [Kowalski1979], although in the case of mark-ups, both logic and data are contained within the node structure and are not separated.

## 4.6 Summary

Software metrics are used to quantify the attributes of systems and are made up of either singular or composite measures. Paradigms such as GQM can be used to derive metrics. Metrics, once obtained, fall into two broad categories:

1. Product Metrics, which measure quantifiable attributes of the product itself, such as size or structure. These metrics tend to be more direct in nature than measures of effort.
2. Process Metrics, which measure development effort and the quality of the methods used to develop the product, including items such as development effort estimates.

The process of metric collection (and the underlying reasons for the choice of some of these metrics) is now, about 30 years on, relatively mature and well-understood. Common measures of software product quality include size metrics (such as LOC), complexity metrics and measures of modularity and

re-use. Some metrics are unavoidably subjective and this leads some to question whether software metrics are inherently unreliable.

Document metrics too are well-established, given the relative maturity of standards such as SGML. Many of the metrics concerned with documents are structural and discuss items such as resource re-use and generalisation of DTDs.

Web systems are less well-understood and this problem makes it difficult to agree upon common measures and metrics to measure quality. However, some of the metrics applied to both software systems and documents could be applicable in the web context, while others, such as size metrics, may have to be remodelled. This problem arises because, unlike usual software, mark-ups are composite objects containing both logic and data.



# Chapter 5

## The Research Proposal

Thus far, this document has concentrated upon more abstract underpinnings for the nature of software, documents and webware in particular. These underpinnings do not necessarily suggest an obvious route for the evolution of the PhD project. This chapter will be used to outline this evolution, to explain in more depth about the nature of the ongoing project and to construct the research proposal for the PhD itself.

### 5.1 Definition of Thesis

At the present time, many heuristics exist for determining the engineered quality of a web system, built up over the (short) lifetime of the web and passed on as examples of 'best practice'. These practices occur commonly in the huge volume of works in the contemporary market place that concern themselves with web design and 'programming' in HTML.

Metrics can be a useful determinant of web site quality and could be used to quantify the degree to which these examples of best practice are effective. Here we define quality as the ability of the site to evolve under the control of developers, allowing maintenance and evolution to take place in an efficient way <sup>1</sup>.

---

<sup>1</sup>this is separate from, but not mutually exclusive to, concerns about usability

It is possible to construct *objective* metrics to measure quality. although presently few, if any, objective metrics exist to fully determine the engineering quality of a web system, although it appears that current measurements either are not focused in this area [Pitkow1995] or are ineffectual for this purpose.

Many measurements are biased towards idealised hypermedia systems, where much research has been done by the likes of Hatzimanikatis et al. [Hatz1995], Botafogo et al. [Botafogo1992] and their successors; usability ([Nielsen2000] etc.) or towards more general topography [Bray1996] and evolution of the macroscopic web or even the topography and evolution of particular sites [Warren2001b]

Metrics may be derived from existing metrics and measurements applied to software and documents, because web sites are, in fact, related to both of these classes. New metrics may have to be formulated in addition, although we should be aware that some elements of ‘quality’, such as aesthetics, are subjective not necessarily measurable, indeed to quote Einstein,

“Not everything that can be counted counts, and not everything that counts can be counted.”

The evolution of such metrics will be a reliable indicator of the engineering quality of the system and can also be used to illustrate the evolution of web systems. This should allow us to see if web sites do, in fact, obey Lehman’s Laws of Software Evolution, as Warren believes in his work [Warren2001b].

## 5.2 The Research Programme

### 5.2.1 Overview

The arc of the research programme is covered in greater detail in Appendix C but is presented in brief summary here for the convenience of the reader.

The end of the PhD project is set for December 2004. In this space of the 34 months from this milestone the following tasks must be completed to allow successful submission of the PhD thesis:

- *Prosecution of full-scale questionnaire study.*
- *Choice of candidate metrics.*
- *Construction of a metric harvester* to collect the chosen metric data.
- *Testing of the harvesting tool.* Testing of the tool is envisaged to be completed by December 2002, allowing deployment to begin in the first quarter of 2003.
- *Selection of candidate sample sites.* This stage will take place in parallel with the development and testing of the harvest tool. The nature of the tool may be a constraint on the types of site that can be used within the study.
- *Deployment of tool,* where the tool is rolled out to the candidate sites and data is collected, possibly over a period of months. It is envisaged that this stage will be ready to begin by January 2003.
- *Collection of results,* to be completed by project month 40 (April 2004), although the primary deadline falls at month 36 (December 2003). The extra four month has been added to create a buffer in the event of problems being encountered.
- *Analysis of results.* including evaluation.
- *Writing up of the thesis submission.* This process is likely to start running (thinly) from the start of the final year of the project, stepping up within the final 4-6 months before completion.

### 5.2.2 Definition of Metrics

Once the relationship between software, documents and webware has been postulated, the use of a wide range of metrics can then be considered. The major task will be to examine the metrics available and to choose relevant measure for webware systems. A group of candidate metrics will be developed to measure characteristics of the web system. Possible metrics could (but may not) include markup density (as previously discussed as an example in Chapter 4), document size (LOC or other appropriate measure), modularity,

reuse of elements etc. Many of these metrics may have connections to some of the heuristics developed to produce better web site design. Many of these heuristics are quite widely used <sup>2</sup>. It should be noticeable at this point that product metrics are being emphasised. It is envisaged, however, that appropriate metrics concerning the development process of web sites will have to be considered as the project demands it.

The work undertaken so far has been to establish a relationship between software, documents and webware (a hybrid entity). Current work tends to treat web based systems as a subset of more general software systems. Many attempts to derive metrics for web systems are predicated upon this assumption. Others, as we have seen, perceive webware as a specialism of more general hypermedia. Their metrics are predicated upon that assumption.

The previous chapters aimed to show that each of these assumptions have some validity due to the common heritage for software and documents. Both of these classes can be related to webware, providing a justification for using all of these types of metrics within web-based systems.

Further to this, the approach of authors creating web systems can be analysed to see how different groups approach the process of writing web documents. A pilot questionnaire to obtain data from a population of web authors <sup>3</sup> has been developed and distributed to a small sample group of 50 authors. Of the fifty distributed, fifteen (15) were returned. When this study is attempted in full it is anticipated to widen the target group to include authors from, non-computing disciplines. This will, it is hoped, provide a more representative view across a wider population of web authors of their perceptions of the development process. The questionnaire has not been reproduced here, mainly for reasons of space but also because the questionnaire itself still requires drafting into a final form.

### 5.2.3 Construction of Metric Capture Tools

Once a set of candidate metrics has been defined, a tool must be produced to allow these metrics to be collected at a particular point in time. By necessity,

---

<sup>2</sup>an example of this is the advice to use server side includes (SSI) to generate common and consistent navigation within web sites

<sup>3</sup>all of whom are students within CIC

this tool must reside on the originating server<sup>4</sup> and must therefore interfere with normal server use as little as possible. There are two approaches that could be successful:

1. Construct the tool to use a batch processing method of harvesting data at given intervals, using a mechanism similar to the Unix `crontab`. This may involve little or no loading during normal use but higher loading while collection takes place.
2. Integrate the tool more closely with the web server by constructing it as a server module for Apache or Microsoft Internet Information Server, for example. Data could then be collected in real-time, with the possibility of continuous but lighter server loading.

Until such time as appropriate metrics are chosen, it is not possible to say which approach is more desirable for the purposes of this project. Once the tool is developed, it can be deployed on a range of target sites, providing ongoing information about their evolution. The metric data thus obtained can then be analysed to determine whether the use of commonly advocated heuristics does produce an increase in the engineered quality of web sites and whether this results in lower amounts of developer effort being expended. This may also require an action research or diary component for developers to record effort for development tasks.

A problem to be considered here is that for the possible cross-section of sites tested, it will not be feasible to use a control group method to evaluate different approaches to a single project. This will be especially true if the projects chosen are commercially based. Care must therefore be taken to select target groups of similar size and type to allow as meaningful a comparison between differing sets of data to be made.

## Limitations

The major limitations of the proposed project are twofold and related:

---

<sup>4</sup>the use of techniques like SSI should be transparent to the end user, so measurement cannot take place at the client side.

### 1. Time Constraints

The time available may not allow an exhaustive survey to be carried out. It is fair to assume that the quality of metric data obtained would improve with higher sample sizes. Even if a very large survey could take place, it is doubtful whether all of the data could be fully analysed within the lifespan of the project.

### 2. Sample Size

The sample size used (and its composition) is not likely to produce a sufficient number of results to allow a definitive evaluation to take place. At this stage, there is a suggestion that further work could take place to make a larger and wider survey of large sites in order to test the choice of metrics further.

These limitations will serve to limit the size of the target group selected for analysis. Great care must therefore be taken to select sites to illustrate the range of possible sites and design options available to provide meaningful measures.

## 5.2.4 Evaluation of Metrics

Once a set of candidate metrics has been proposed and employed within the chosen test environments, their efficacy as performance indicators can be evaluated. Broadly, they can either be effective or ineffective. The reasons for either being the case may reveal more about the properties such systems during analysis and may allow a refinement of the proposed hybrid model of web systems.

At this stage, of course, it is not possible to predict how effective candidate metrics will be, although it is to be hoped that more will be good indicators than not.

The collection of sets of static metrics is, in itself, a relatively uninteresting task. Of much more interest is the evolution of these measures over time to produce indicators of quality. This will be the main tranche of work used to support the thesis itself, outlined at the start of this chapter.

## 5.3 Summary

The timescales envisaged for the PhD project are presented in detail in Appendix C and will not be reiterated here. They are, however, roughly in accordance with the guidelines suggested by Philips and Pugh for the structure of PhD projects [Phillips2000b].

At this point, however, it is apparent that a great deal of thought will be required in the early stages of the report to devise environments to produce high enough amounts of data of the required quality. The method of collection is likely to place constraints on what kind of sites can be used in the sample. It is on this experimental configuration that the success of the project will depend.

The thrust of the PhD thesis is to show that indicators derived from the collected experimental data will demonstrate that web site "quality", however we choose to define it during the choice of candidate metrics, can be measured. Further, these measures can be used to quantify common guidelines for 'good' design laid out in contemporary literature. The process will depend to some degree on how different constituencies of developer approach the problem of webware development. The use of a developer's questionnaire may help to provide insight into the way web authors/developers approach their projects. A questionnaire may also provide some useful quantitative data about the construction of small to medium-sized projects. This data will not, it is likely, come from the wider study.

Such evolutionary measures will help web developers to choose techniques to minimise development effort and produce systems that are well-designed and easy to maintain and service.

The final aims of this PhD project is to give web developers an opportunity to echo the works of the object-oriented software developer Scott Whitmire:

"Many of the design decisions for which I had to rely on folklore and myth can now be made using quantitative data."



# Chapter 6

## Conclusions

### 6.1 Summary of Progress

The first part of the PhD project has involved the investigation of the nature of software and web systems. This has now been done. During the course of the investigation, a relationship between documents, software and web artefacts has been proposed that places all three within a more general class of ephemeral systems capable of being engineered. This is in contrast with tangible systems that can be engineered, such as physical systems like bridges, buildings etc.

Web systems can be considered as both software and document simultaneously, while software itself, in the source form, is most certainly a document system. More contentiously, documents (in the form of mark-ups) can be said to be a form of programming language.

As Christopher Alexander has posited, however, the importance of good construction for all *inhabitants* of an engineered system is extremely important. For ephemeral systems, this manifests itself in the form of engineering methods that attempt to improve usability (for the end user) and maintainability (for the developer). Much of the emphasis in web systems has been loaded toward the end-user in this respect, as demonstrated by the work of Jakob Nielsen. Although work has been done using design methodologies such as RMM and OOHD, the design of heterogeneous web systems is not compa-

rable in maturity to that of software, for the obvious reason that it is much younger.

The view that webware is *either* a software or document is simply not sufficient to describe all aspects of its behaviour. To this end, the dualistic nature of webware (as discussed in Chapter 1) has been posited as an explanation for its properties. The metrics that describe webware will themselves be dualistic, some describing the documentary qualities and others its programmatic ones.

All of this considered, it is possible to derive metrics that will allow developers to make a reasoned evaluation of web site quality. This is, in fact, what the first part of the PhD project has been about; establishing a framework for choosing appropriate metrics for web systems.

## 6.2 Further Work

The next stage of the PhD project will be to examine the area of metrics in more detail. This was discussed in Chapter 5. Such an examination will allow a group of candidate metrics to be developed. The metrics that will be considered for the candidate set are likely to be drawn from both document and software sources, most particularly object-oriented ones.

Once such a candidate list is complete the development, testing and deployment of a metric capture tool can be prosecuted. This will provide data over an interval of time yet to be determined, but long enough to give an indication of the evolution of the web systems selected, especially since web projects, on the whole, tend to have shorter lifecycles than software projects.

This data will then be analysed, which will form the empirical evidence that will support the thesis (or not, as the case may be). At this point, because the metric capture tool and metrics have not been defined, it is not possible to state exactly what data will be collected, or indeed how, although it is envisaged that a method of remote capture can be implemented to allow archiving of the data as it is collected.

The timescales and project plan for the next stage of the project are presented in Appendix C.

# Appendix A

## Software Evolution & Taxonomy

(This appendix is a truncated version of an internal technical paper produced in March 2001)

### A.1 A Software Timeline

Figure A.1 shows the evolution over time of software and software systems. The diagram aims to highlight significant milestones in software evolution, such as the development of the early languages that provided the archetype for certain paradigms, for instance Simuila67 for Object Orientation or Pure Lisp (1958) for functional programming. Sammet [Sammet1972] [Sammet1976] has carried out fuller surveys of extant programming languages at various times in the past. These surveys are interesting for historical reasons.

The diagram illustrates the following important milestones:

1. 1947 - The developemnt of the von Neumann computer model which dominates to this day.
2. 1958 - LISP, the oringinal functional programming language was developed.

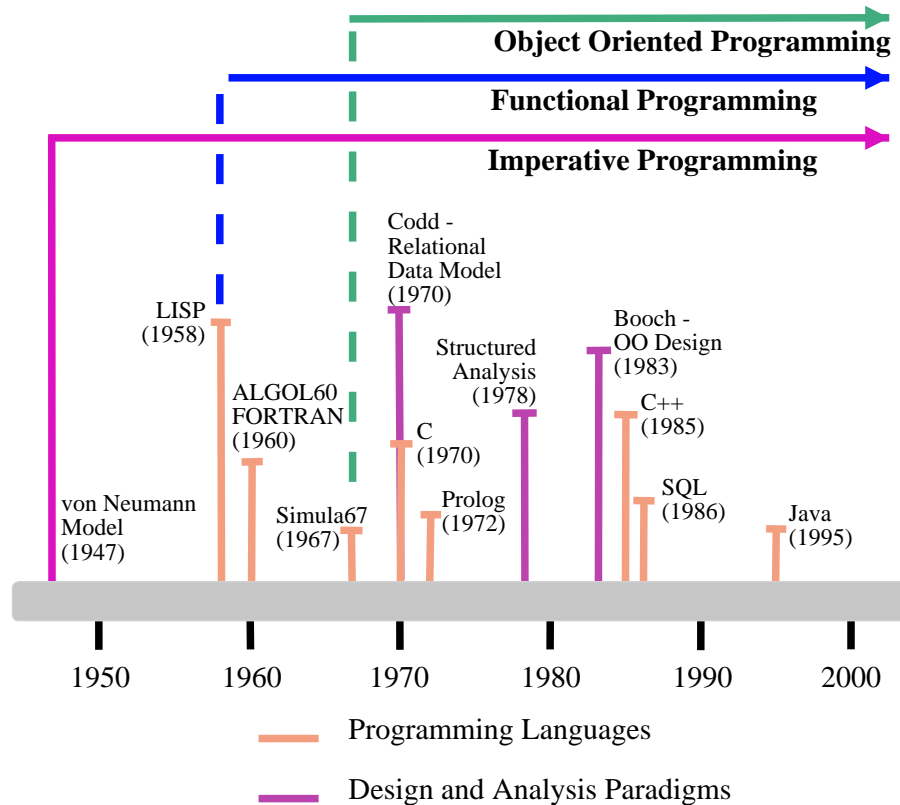


Figure A.1: A Software Timeline

- 1960 - ALGOL60 and FORTRAN, the prototypical imperative languages were released to the world. The language *Pascal*, developed by Wirth in 1971, was one of their successors, designed as a language to encourage the techniques of **structured programming**. Wirth was a major proponent of this technique.
- 1967 - Simula67, the first Object Oriented programming language was released. This was succeeded by the language *Smalltalk* in 1972. Its principal use was in the development of the Dynabook project at Xerox PARC [UCNZ1998].
- 1970 - Codd's Relational Data Model, the precursor to Object oriented design and analysis, is first published.

6. 1972 - Prolog - the first language for logic programming is developed.
7. 1978 - Structured Analysis and Design, the first methodology to present a unified model for design and to address analysis issues. This was the first time the entire software engineering process had a sound theoretical base.
8. 1982 - Foundation of the ANSI SQL committee and development of SQL, following on from the CODASYL project. The first SQL standard is published in 1986.
9. 1983 - OO design. The early 1980s saw the first movements in applying similar rigour to OO systems as to structured ones by people such as Booch, Rumbaugh, Jacobson and others. Coincided with the beginnings of development of C++ by Stroustrup.
10. 1985 - C++ released [CPlus2001]. C++ was built by Bjarne Stroustrup on the foundations of C language (released in 1970) and its ancestors.
11. 1995 - Java is released. Java embodied the principle of *Write Once Run Anywhere* (WORA). Such an approach was not new, having already been used in UCSD p-code compilers during the 1980s. The use of network services and the principle of the applet, does, however, make it a major milestone in software, particularly in relation to the web.

## A.2 A Taxonomy of Software

### A.2.1 The Software "Family"

Bobrow and Stefik [Bobrow1986] define the following kinds of programming style:

- Procedural programs are algorithmic and generally concerned with data flow. This form also includes scripting languages and can also include functional programming.

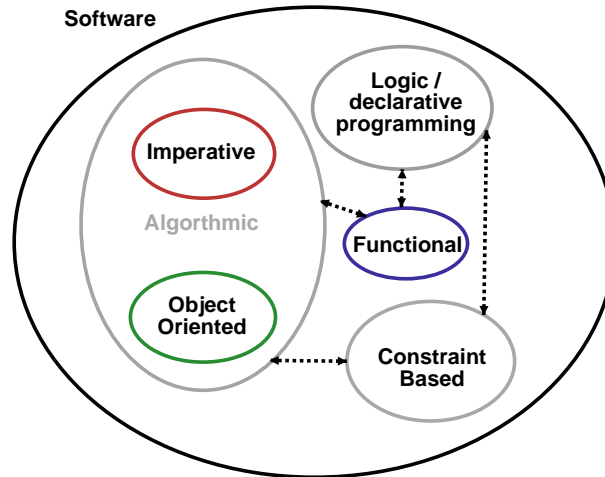


Figure A.2: The Software Family

- Object Oriented programs, where code may have algorithmic elements but there is more concern with mapping software objects to real world ones where possible.
- Declarative programs, encapsulating logic and rule based programming. Declarative systems may also utilise imperative or functional components but have much in common with O-O systems.
- Constraint Oriented programs. including self-organising systems such as neural networks and genetic algorithms . Software of this type can use code generated using any of the other styles for reasons which will be explored later.

Some of the above families are covered in more detail by Bal and Grune [Bal1994a]. To this list, it is proposed that another category could be added, given the conceptual space model described for software earlier.

- “Semantically” Oriented  
here there is no “flow” as such. This style is concerned with the organisation of information in order to be semantically meaningful. This form includes mark-up languages and formatting languages such as

Postscript and T<sub>E</sub>X. It can be argued, however, that such a paradigm is merely a variation on a declarative one, with mark-ups being considered as purely declarative languages. Each element could be said to describe a clause and the tree derived from parsing maps the relationships between such elements for each document.

Bobrow and Stefik state that a style is simply a way of organising programs based on a conceptual programming model and an associated language to make the programming style clear [Bobrow1986].

Each style in this sense is interpreted as equally valid and indeed, each particular style has applications and context in which they may be most successfully used. This goes part of the way to describing the onion-skin model described earlier.

## A.2.2 Imperative Programming

One interpretation of the idea of transformation is that software involves a functional or algorithmic process. This is a key element of views of software such as Structured Analysis [DeMarco1979] that treats the information transformation process as a flow of data through a number of functional components. It corresponds with the view taken by Functional Programming (q.v.), albeit in a slightly different way. This also corresponds with our starting definition as we can define the application of functional components of a software process in well-understood manner.

For instance, consider a transformation of the form, transforming an initial set  $x$  into a final set  $y$ :

$$y = f(x)$$

We can consider a software process to describe a flow through a number of such operations that could be applied sequentially, viz.

This representation, however, could be interpreted in two different ways: imperatively, by being able to create assignments at each intermediate stage

$$\begin{aligned}
 x_a &= f_a(x) \\
 f_{a+1}(x_{a+1}) &= f_{a+1}(f_a(x_a)) \\
 &\vdots \\
 y &= f_n f_{n-1} \dots f_{a+1} f_a(x)
 \end{aligned}$$

Figure A.3: Functional consideration of an algorithmic process

of evaluation and functionally by considering the whole without these intermediate actions. In this context, imperative systems may also encounter problems because of evaluation order: this is a problem that would not occur in the functional context.

### A.2.3 Algorithms

In many views of programming and software production until the introduction of logic programming (q.v.), the role of the algorithm is of vital importance. Knuth [Knuth1969], states five properties that define an algorithm:

1. **Finiteness**  
The algorithm has a finite number of iterations or repetitions. That is, there must be some defined exit condition.
2. **Completeness**  
The algorithm encapsulates the problem domain completely.
3. **Definiteness**  
The algorithm encapsulates ONLY the problem domain and does not focus on irrelevant concerns.
4. **Input**  
The algorithm requires input conditions or states to work from, or must be able to define them.
5. **Output**  
The algorithm should produce discernible results or outputs.

## 6. Effectiveness

The algorithm should, in some way, effectively address the domain of the problem.

It is arguable whether this final item, although extremely desirable is a defining characteristic of every algorithm. It is possible to define a set of algorithms to solve a particular problem which would have markedly different levels of effectiveness. Each would, however, still be a “valid” algorithm in as much as it provides a working solution.

The main implication here again is that software, being algorithmic, is essentially a sequential, deterministic process (specifically in terms of the finiteness condition). This determinism *only* applies to the algorithm itself, *not* to the results of applying it. In other words, the algorithm maps a set of possible conditions and outcomes. The results of applying them are not necessarily known at the time of execution. This, in fact, as discussed later, may be considered the major difference between imperative (algorithmic) and logic programming

### A.2.4 Object Oriented Systems and Programming

The development of Relational Data Modelling by Codd [Codd1970] and others in the early 1970s had successfully introduced the idea of Entity Relation Modelling as an effective tool for data modelling. In fact, in the abstract of his 1970 paper, Codd himself proposed that one of the main reasons for developing his relational model was to hide the underlying structure of data from the user.

Even in the early 1970s this is a common concern shared with Parnas ideas on modularisation. The Parnas idea posited that, instead of simply decomposing programs as subprograms, that each module should have a “responsibility assignment”. In other words, Parnas was proposing information hiding (along with Codd) as well as being an early proponent of the abstraction principles that object orientation would expand upon. Many considered this system to be a powerful tool for data modelling as it closely mapped with human traits of characterisation by categorisation. Codd himself recognised this fact.

Structured Analysis integrated ERA to provide a description of the static

model of the software system. Software practitioners such as Booch, Rumbaugh and Jacobson (separately at first, together later in the UML methodology [UML1999] as well as Yourdon (again) all saw the promise of using such a system more prominently in software design. Previously, software systems had been seen as large entities able to be split into parts and developed separately to model the flow of data around a system. Each of the components or modules could manage a part of the flow. This modularisation process had begun with Wirth, Mills and Parnas, as described earlier.

The O-O system departed from this and (like the model used for Unix system commands) postulated that systems were better modelled as smaller, communicating independent entities. To use the object oriented phraseology; such systems should be componentised, cohesive and lowly coupled.

Proponents of O-O claimed that it offered numerous benefits, including:

- Code reuse
- Component based systems to allow code "manufacture"
- Modular abstraction
- Scalability

These benefits were achieved by use of the fundamental concepts used to build object-oriented systems:

- **Abstraction** Following on from the concept of modularity, objects should connect in well-defined visible ways. Objects need not know about internal implementation issues within other objects with which it communicates. In theory this allows reusable components to be built and allows them to be used to build very large projects more easily. In other words, objects promote scalability.
- **Encapsulation** Objects should not be thought of as data structures, to be fully formed an object should also define what actions it can perform, how it communicates with other objects.
- **Polymorphism and Inheritance** This is possibly the key feature of object orientation; objects can form hierarchies and have the ability to

be classified according to type. New types can be easily derived from existing ones.

Booch also made the distinction between algorithmic decomposition of the kind described by Wirth and others and Object decomposition, providing a level of abstraction in defining the entities participating in a software system. In many senses, this did indeed echo the entity-based approach to relational data analysis begun by Codd.

The communicating entities approach had also been used in the idea of semaphoring between modules in real-time systems, particularly by Ward and Mellor. This model allowed for large systems to be built more easily and, according to its supporters has encouraged code reuse and highly component-based systems.

### A.2.5 Alternatives to the Imperative Model

As can be seen from Bobrow and Stefik's taxonomy, one of the styles of programming they describe is that of declarative programming. Declarative programming re-interprets one of the fundamental assumptions made so far in this discussion of software and software systems: software is essentially an algorithmically based entity. The early proponents of software design and programming had thought this way, giving rise to ideas like the flowchart and later, other analysis tools as has already been mentioned. However, this is not a wholly valid assumption to make.

Imperative programming works by executing a sequence of defined commands (be they singular or grouped). Part of this process involves changing the values of variables within the domain of the program using explicit assignment operations. One result of this is that the order of command execution within a program can affect the values of variables in different ways. In effect, the programmer has to exert some low-level control over the program conceptual space, explicitly controlling the state of the system in the code.

In declarative programming the possibility of confusion regarding evaluation order does not happen. Declarative programming was developed to hide this lower level process away from the programmer. It was seen as both a source of error in thinking and an incorrect level of abstraction when con-

sidering software problems, mainly because the act of assignment made evaluation an important consideration. Functional programming developed to allow a propositional calculus to be implemented in a machine environment. A propositional calculus describes the functional process in more rigorous mathematical terms.

The nature of this style, lent (and still lends) itself to design using formal methods which provide a more mathematically rigorous and complete design framework for such software to operate within. Indeed, the illustration shown in Figure A.2.2 corresponds more to this view of software than to purely imperative programming. The programmer thus has little control over the state of the system once the operational parameters are defined and boundary conditions specified.

Declarative programming is not new, So-called “Pure” LISP, although not strictly a purely functional programming language, certainly contained many such features. It was first introduced in 1958 [MacLennan1990].

“Common” Lisp, introduced later however, contained many imperative programming constructs. By the middle of the 1970s, languages such as Lisp and LOGO [Chakrabary1999] were in widespread use. To abstract the process of explicit assignment from the programmer the programming environment itself must be responsible for it. One of the main methods of doing this is by use of recursion, placing great emphasis on stack operations.

By the early 1980s many were concerned that, to increase computing power, systems would have to become much more adept at processing in parallel. To handle this underlying complexity, authors of software should be divorced from the low-level problems of algorithmic programming as such processes were prone to error. Kowalski [Kowalski1979] (amongst others), proposed a fundamental rethink in the way programming was pursued. These systems relied much more on logical than functional composition, leading to the birth of logic programming.

Where functional programming removed the need for low level assignment, logic programming sought to remove the need for functional considerations entirely. Systems could be devised that would operate under conditions determined only by the relationships between data entities. Systems of this nature involve the use of a predicate calculus, most famously described in the first instance by Alonzo Church [Church1936], to represent these relation-

ships. The engine which evaluates these rules may contain functional (or even imperative) components at lower levels but seeks to make the programmer think in purely abstract terms.

Logic Programming relies on the use of the *Horn Clause* [Bal1994b]. A Horn Clause is one which states that a particular condition is true if zero or more other conditions is also true. A further definition is that a horn clause with no conditions is a fact. Unlike imperative logical evaluations, however, each of these other conditions could be evaluated in parallel. The goal of the logic programmer is to reduce a problem domain into a set of horn clauses which can be evaluated. As a consequence of parallelism, however, logic programming is non-deterministic in the sense that evaluation orders are not necessarily known.

Work by Eisenstadt and Brayshaw [Eisenstadt1988] suggested however, that there was a conceptual link between algorithmic, functional and object-oriented approaches. A more rigorous discussion of logic programming will ont be undertaken in this paper

### A.2.6 Self-Modifying Systems

Self-modifying systems, which include Genetic Algorithms (GAs) and Neural Networks), unlike the algorithmic methods described earlier are essentially non-deterministic. GAs stem form initial work by Holland [Holland1975] but ideas about evolutionary programming in general can be found as far back as Fogel [Fogel1966]

A diagrammatic representation of this process is shown in Figure 24 <sup>1</sup>.

Instead of well-predicted algorithmic systems, the GA system involves the use of a "fitness function" to determine the evolution of the algorithm. Thus, a programmer has no control over the algorithm or the system's state until completion. The fitness function itself does not have to be algorithmic, rule based or functional tests are also allowed (and in some cases may be preferable).

The genetic search is an iterative sequence (a generation) through the fol-

---

<sup>1</sup>This diagram is adapted from The Genetic Algorithm Toolbox [GAT2001]

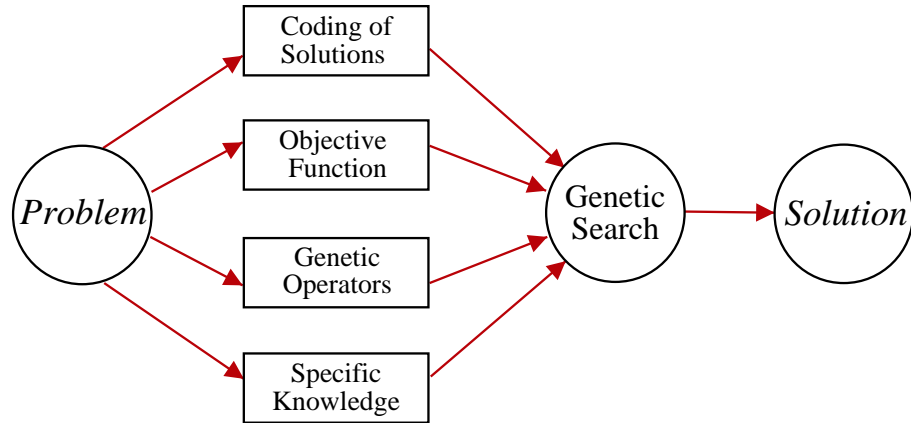


Figure A.4: Genetic Algorithm Heuristic

lowing stages:

- Fitness assignment
- Selection
- Replication
- Recombination Crossover
- Mutation

Replications that do not meet the specified fitness criteria are then discarded. In theory this produces successive generations of solution that satisfy the fitness function more closely. The self-modifying system is therefore non-deterministic. Not only are the outcomes unpredictable but the algorithms used to arrive at that outcome are also not known at the outset as these solutions are generated during the execution cycle.

A more detailed consideration is beyond the scope of this document but readers are advised to refer to the work of Fogel [Fogel1994] for a more rigorous discussion.

# Appendix B

## Document Taxonomy and Characterisation

(This appendix is a truncated version of an internal technical paper produced in June 2001)

### B.1 A Short Taxonomy Of Documents

In order to understand various forms of documents, it is perhaps useful to provide a short review of such items to better facilitate classification. Graphically, this structure is shown in figure B.1. Items at bottom levels are shown as examples only.

Bryan[Bryan1997] states that any device that stores formatted text for later retrieval and use must use some form of mark-up to do so. This category therefore includes program source code. The mark-up may only be machine readable in some cases however. In general terms the following types of document exist:

- **Unformatted Text**

The simplest form of document is the unformatted, raw text document, The data contained inside is not subject to any formal structure although it is actually to provide a (trivial) DTD for such a document.

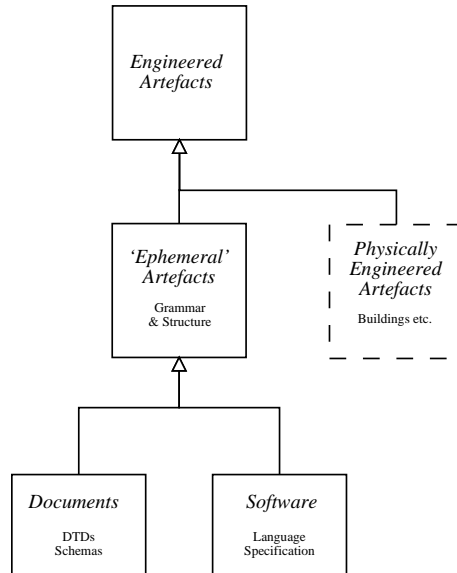


Figure B.1: A Taxonomy of Engineered Artefacts

It can also be argued that, without such a DTD, a document would, in fact, conform to a notional type definition, especially in the case of natural language documents. This notional definition would have to be determined by the reader for the document to have any meaning, however.

- **Specific Mark-Up**

Specific mark-up is used in systems such as PostScript or RTF processing in word Processors like Microsoft Word or Corel WordPerfect. The mark-up is usually application or task specific and is not designed to be transferable between environments or applications. This can also include styles such as delimiter separated lists, where each field's meaning may only be known by the parsing application. As such, specific mark-ups are commonly used in typesetting systems, where the major degree of formatting control is exerted by the machine.

- **Generic Mark-Up**

One of the stated goals of generic mark-ups is to allow encoded information to be exchanged within a number of environments and to allow

the resultant documents to be viewed and edited by both machines and people. Generalised mark-ups are commonly stored in plain text files, making them highly portable systems as tools to convert text file formats are effectively universal.  $\LaTeX$  is an exception to the specific mark-up situation that usually occurs with typesetting languages. The mark-up used in the  $\TeX$  family is actually generic and is easily transformable into HTML or PDF, for example. <sup>1</sup>

The move to develop generic mark-ups began in the late 1960s as a result of first Gencode, and later GML (both at IBM), led by Goldfarb [Goldfarb1970]

Such documents *do not* need to be entirely textual. They are able to contain references to (or have embedded within them) other objects such as multimedia content. Implications of this in hypermedia documents are discussed elsewhere.

### B.1.1 The Structure of Mark-Up Documents

The description of generic mark-up documents is generally presented in one of two ways: Presentational vs. Logical

- **Presentational Structure** describes how a document will *appear* when used. It is specifically concerned with the description of visual elements and how a document will appear when viewed. As far as version 3.2, HTML had begun to evolve along this route. HTML 4 and its variants attempts to reverse this trend.
- **Logical Structure** is more concerned with the layout of document in terms, not of its visual appearance, but how it may be interpreted by machines or how its meaning may be interpreted by a human user.

Jelliffe prefers to extend the presentational versus logical division into six sub categories which uses to describe the structure of a document more fully. These he defines as:

---

<sup>1</sup>Many of these tools are part of the standard  $\TeX$  distributions

1. **Page Layout**, describing the high level visual layout of the document, as most people would use when working with a desktop publishing environment.
2. **Page Objects**, describing individual items within the document such as images, figures, tables, text blocks, etc.
3. **Glyphs**, are another visual indicator: the type faces and sizes used for type within a document.
4. **Characters**, the meaning associated with a particular glyph [ISO9541], such as the choice of language, environment or character set.
5. **Editorial Structure** - word, sentence and language structure within the document.
6. **Topic Structure**, the arrangement of content at a higher, conceptual level.

Jelliffe further states that there is a flow of dependence, progressing in reverse order up this list. Unlike so-called traditional software, the topic structure of a document may affect its page layout. For some documents of course, some of these categories have little or limited meaning. A document made up solely of a sequence of images may require or exhibit little or no glyph information, for instance.

The simple linear flow of dependence may not, then, be sufficient to describe the structure of documents. It is perhaps, more realistic to describe feedback mechanisms (akin to lifecycle models in software engineering) to model the document design process and the artefact it produces. This is especially true because documents are likely to undergo more constant, greater and varied re-engineering throughout their lives.

Document mark-up languages can be further classified into another two broad categories:

- **Procedural Mark-up**, which describes how a document WILL be displayed. This is most commonly used in formatting languages such as PostScript or Word Processing systems (most especially WYSIWYG ones) or where presentational cues *must* be adhered to maintain a correct and consistent visual appearance.

- **Descriptive Mark-up** only provides suggestions to a rendering agent as to how the marked-up document should be displayed (i whatever context that may be). The user agent does not have to abide by these instructions, they are merely hints. “Classic” HTML and the T<sub>E</sub>X languages were designed to work in this way, allowing individual user agents or rendering agents to decide on the exact presentation used, either internally or by the introduction of an external entity defining presentational characteristics. In a web context, this commonly takes the form of a *stylesheet*

To illustrate this point, we can consider the example of the typesetting language T<sub>E</sub>X (and its variants, including L<sup>A</sup>T<sub>E</sub>X). L<sup>A</sup>T<sub>E</sub>X is described at the web site of the L<sup>A</sup>T<sub>E</sub>X project [LaT<sub>E</sub>X2000] in the following way:

“L<sup>A</sup>T<sub>E</sub>X is a document preparation system for high-quality typesetting. It is most often used for medium-to-large technical or scientific documents, but it can be used for almost any form of publishing.

“LaT<sub>E</sub>X is not a word processor! Instead, LaTeX encourages authors not to worry too much about the appearance of their documents, but to concentrate on getting the right content.”

“To produce this in most typesetting or word-processing systems, the author would have to decide what layout to use, so would select (say) 18pt Times Roman for the title, 12pt Times Italic for the name, and so on. This two results: authors wasting their time with designs, and a lot of badly designed documents!”

The key paragraph is the second. It is an almost complete re-iteration of the principles of a generic, descriptive mark-up language.

Generic document mark-up is derived from the concept of the physical mark ups (note the difference in spelling) used by typesetters. Mark up of that type is used to show compositors the physical layout of manuscripts prior to printing. Hence it is a physical procedural mark-up.

L<sup>A</sup>T<sub>E</sub>X is not a hypermedia environment, It does not have the capability to hyperlink. <sup>2</sup> :it can however, contain embedded links to other resources,

<sup>2</sup>Extensions are available to embed hyperlinks within documents, to be opened by external agents, however

acting like inclusion mechanisms.

### B.1.2 Document Type Definitions

The concept of classification is a highly important one. as is apparent from the earlier definitions of what documents are. The mechanism of the Document Type Definition or **DTD** exists to allow such classification and to define document structures consistently, The DTD describes what elements of a particular type a document can contain and further, how the grammar of aggregations of these elements is defined.

In the area of XML this concept been further extended to allow creation of XML Schemas [XML2001]. XML Schemas define a DTD using XML syntax and are themselves XML documents. Proponents of schemas say that they are easier to build and maintain than the more opaque DTD. They are also easier to generate and change dynamically.

### B.1.3 Document Validity

What constitutes a valid document? Stated simply, a valid document is one that conforms to its specified DTD. However, it is apparent on Web at large that a great number of documents are not valid HTML but still manage to be viewable by a range of user agents. This is not entirely due to user agents themselves more generous in their document parsing. Clearly, validity is not a complete description of a document's ability to be used, or to be "complete", in some way.

Documents may also be *well formed* . A well-formed document is one that conforms to a grammar defined by DTD. By definition a valid document is also well formed. However, a well-formed document may also contain elements or attributes not defined within the DTD. Typically, in an HTML context, such items not within the DTD will be ignored but the document will still be processed if structured correctly i.e.: they are syntactically correct.

For mainly historical reasons, HTML browsers tend to be "sloppy" when parsing. This is to allow user agents to display nodes in at least some form, however ugly, and not to burden end users with error messages. Browsers

such as W3C's Arena and Amaya browsers do allow users to view errors in badly-formed HTML if they wish.

In web browsers, display of HTML is done in three stages:

1. **Retrieval**, where document is fetched, whether from the network or from a local source.
2. **Parsing**, where the document is read and arranged into a document tree containing document elements. Whether this parsing is done by a validating parser or not is an issue of implementation.
3. **Rendering**, where the parsed tree is then formatted for display by a display manager.

Sloppy parsing of HTML was a key reason for its rapid take-up by amateur users, allowing badly formed mark-up to be written and displayed. However, an XML or SGML parser is more strict. Syntactical and semantic errors are reported, especially because in many cases, visual rendering is not required. In XML this job should be done by formatting systems such as XSL (eXtensible Style Language).



# Appendix C

## Provisional Timetable for the PhD Project

### C.1 Project Timetable

The end of the PhD project is set for December 2004. In this space of 34 months <sup>1</sup>, the following tasks must be completed to allow successful submission:

- *Prosecution of full-scale questionnaire study*, including the collection and analysis of results. This item is not, strictly speaking essential to the course of the project but is being done to provide some additional insight into the web development process by a number different developer constituencies.
- *Choice of candidate metrics*. This will first entail some investigation of existing metrics for document, software and hypermedia systems before choosing the most useful and appropriate ones to use.
- *Construction of a metric harvester* to collect the chosen metric data.
- *Testing of the harvesting tool*. The testing will need to test functionality and quality of individual components as well as the testing of the

---

<sup>1</sup>these timescales are fitted to the mode of study: part-time

prototype system in operation.

- *Selection of candidate sample sites.* This task is, in itself, a major undertaking because the selected sites must be large enough and evolve at a rate that will provide useful metric data. Possibilities at this early stage include certain university sites, possibly useful as they are maintained by a wide cross-section of developers. Although this stage is not illustrated in the chart it occupies the same timeframe in which development of the harvesting tool takes place.
- *Deployment of tool,* where the tool is rolled out to the candidate sites and data is collected, possibly over a period of months.
- *Collection of results,* which may be ongoing. One possible design goal for the tool to be constructed is to collate this metric data at set time intervals to produce both cumulative and interval indicators.
- *Analysis of results.* Not only will the results themselves require analysis but the choice of analysis methods must be considered in detail, requiring some further reading and research. This stage will also include the evaluation of the analysis, to determine whether the arguments of the thesis are supported.
- *Writing up of the thesis submission.* This process is likely to start running (thinly) from the start of the final year of the project, stepping up within the final 4-6 months before completion.

The preliminary timeplan for these tasks is shown in the GANTT chart in Figure 27. Estimates for this chart were prepared using guidelines suggested by Phillips and Pugh [Phillips2000b], who suggest that the thesis proposal should be formed by approximately 18 months into the PhD. Data collection should start about 24 months, while data collection should be completed at around month 36-40. On the timescales provided this suggests that experimental work should be completed before April 2004, having started around January 2003. The plan proposed, at present, is roughly in accordance with these figures.

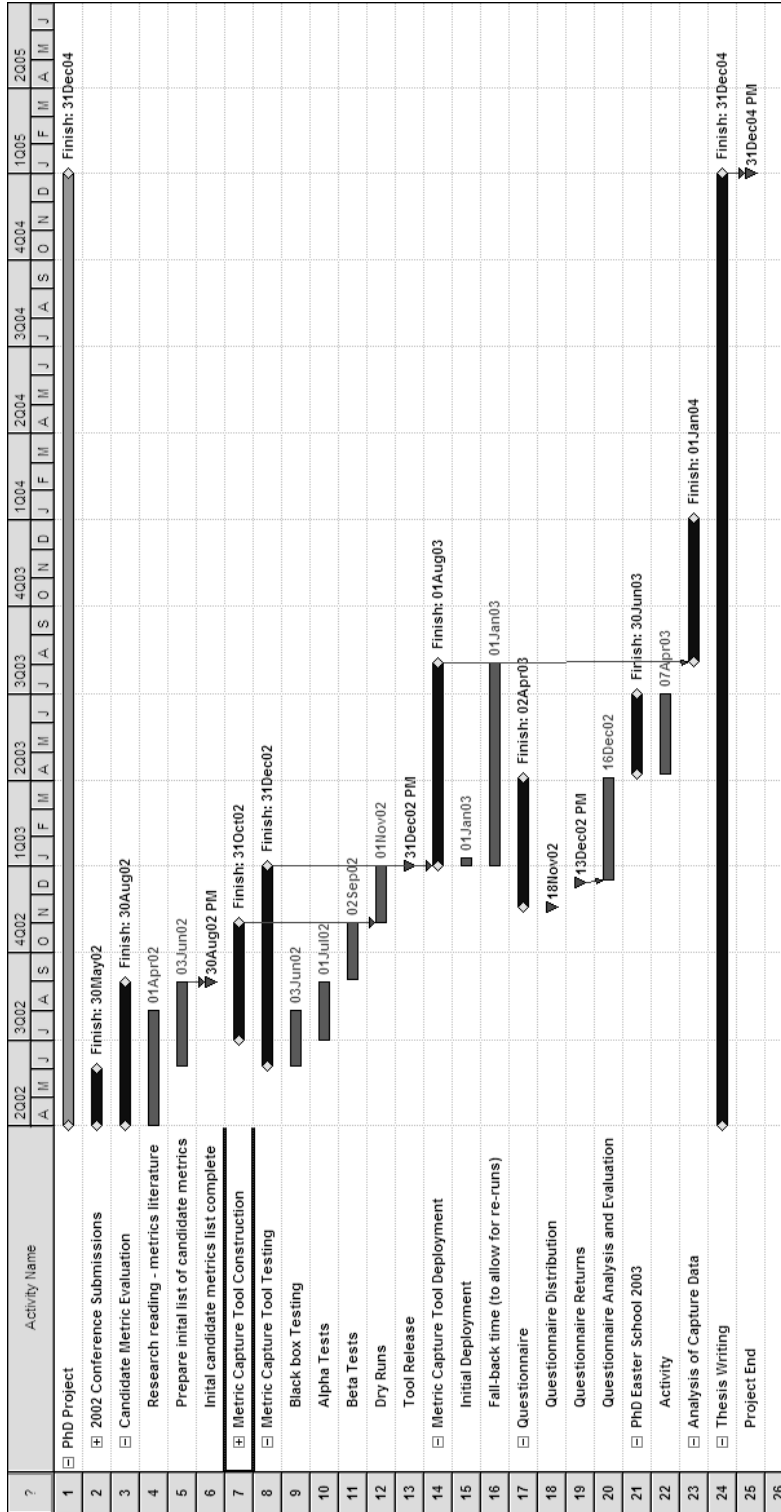


Figure C.1: Preliminary GANTT Chart for PhD project - Current at April 3



# Bibliography

- [ACMSDE2001] Munson EV, *ACM Symposium on Document Engineering*, Available on-line at:  
<http://www.documentengineering.com/> Accessed: October 12 2001
- [Albrecht1979] *Measuring Application Development*, Proceedings of IBM Applications Development Joint SHARE/GUIDE Symposium, Monterey CA USA, 1979, pp83-92
- [Alexander1977] Alexander C, *A Pattern Language*, Oxford University Press, 1977.  
  
Alexander proposes that pattern languages exist within architecture and can be used to describe common or repeating themes or properties within projects. This is a theme continued by the programmer Richard Gabriel (q.v.)
- [Backus1978] Backus J. *Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs*, CACM 21(8) Aug 1978 pp613-641  
  
Along with Kowalski, Backus (who had worked on the development of languages like FORTRAN) was a champion of functional programming, arguing that it offered great benefits in terms of efficiency and power for the programmer.
- [Bal1994a] Bal HE, Grune D, *Programming Language Essentials*, Addison-Wesley 1994, p8

This book includes a brief discussion of the various programming paradigms and gives brief examples of the differences between them.

- [Bal1994b] *ibid.* p166
- [Bardini2000] Bardini T, *Bootstrapping: Douglas Englebart, Coevolution and the Origins of Personal Computing*, Stanford University Press, 2000
- A book that details the earliest developments in hypermedia and human computer interaction. The work of pioneers such as Whorf, Bush and Nelson are also discussed
- [Barron2000a] Barron D, *The World of Scripting Languages*, Wiley, 2000, pp5-6
- [Basili1984] Basili VR, Weiss DM, *A methodology for collecting valid software engineering data*, IEEE Transactions on Software Engineering, vol.SE-10, Nov. 1984. pp728-738
- [Berard1995] Berard E, *Metrics of Object-Oriented Software Engineering*, posted to the USENET newsgroup comp.software-eng, January 28 1995
- [Berghout1999] Berghout E, van Solingen R, *The Goal / Question / Metric Method*, McGraw-Hill, 1999
- A description of the approach used when applying the GQM method.
- [Bennett1996] Bennett JP, *Introduction To Compiling Techniques A First Course Using ANSI C, Lexx and Yacc*, Second Ed, McGraw Hill,1996
- An introduction to the process of software compilation.
- [Berners-Lee2000] Berners-Lee T, *Weaving The Web*, TEXERE, 2000, pp38-46
- A book describing the evolution of the early web, from the point of view of its inventor. Useful particularly for

its insights into the reasoning behind early decisions for the structure of the HTTP protocol and other design issues.

- [Bieber1998] Bieber M, *Hypertext and Web Engineering*, In Proc. 9th ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space-Structure in Hypermedia Systems, 1998, pp277-278
- [Bobrow1986] Bobrow DG, Stefik MJ, *Perspectives on Artificial Intelligence Programming*, Science, vol 231, Feb 1986, p951  
Bobrow and Stefik provided a taxonomy of then currently used programming paradigms, dividing them into the categories discussed in this report
- [Boehm1981] Boehm BW, *Software Engineering Economics*, Prentice Hall, 1981  
This work introduces the principles of the COCOMO model and follows on from earlier work about software quality in 1978 in conjunction with Brown and Kaspar. This work is a hugely influential one in the area of modern software metrics.
- [Boehm1988] Boehm BW, *A Spiral Model of Software Development and Enhancement*, IEEE Computer 21(5), May 1988, pp61-72  
Boehm's work on the spiral model has been a major part of the issue of process engineering within the Software Engineering field.
- [Boehm1997] Boehm BW, Devnani-Chulani S, Egyhed A eds, *Knowledge Summary: Focused Workshop on Rapid Application Development*, USC, Los Angeles CA, USA, 23-27 June 1997.
- [Booch1994] Booch G, *Object Oriented Analysis and Design With Applications (Second Ed.)*, 1994 Benjamin/Cummings, ISBN 0-8053-5340-2, 1994, pp28-29

An extension of Booch's earlier pioneering work in the field of Object Orientation. Together with Rumbaugh and Jacobson, Booch's work was consolidated to provide the basis of UML.

- [Booch2000] Booch G, *The Architecture of Web Applications*, Available on-line at IBM Partnerworld for Developers, [http://www.developer.ibm.com/library/articles/-booch\\_web.html](http://www.developer.ibm.com/library/articles/-booch_web.html), Accessed: October 12 2001
- [Botafogo1992] Botafogo RA, Rivlin E, Shneiderman B, *Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics*, ACM Transactions on Information Systems 10(2), April 1992, pp142-180
- The precursor to the work of Hatzimanikatis et al. The work mainly concentrates on global metrics, mainly concerned with navigation through hypertext systems. Although node based metrics are mentioned in passing, little about them is discussed in any detail.
- [Bray1996] Bray T, *Measuring the Web* In Proc. 5th WWW Conference, Paris, France, 1996
- An early example metrics for the web being considered. In this case Bray was attempting to define limits for the web, both in terms of the boundaries of individual sites and the web as a whole.
- [Brooks1995a] Brooks FP, *The Mythical Man Month (Anniversary Edition)*, Addison Wesley, 1995, p271
- Originally published in the 1970s, at the height of the software crisis, this seminal book talks about the process of building software and the problems associated with that construction. This edition was published with updates to some of the original material, asking what (if anything) had changed in the intervening years.
- [Brooks1995b] Brooks FP, *The Mythical Man Month (Anniversary Edition)*, *ibid.*

- [Bryan1997] Bryan M, *SGML and HTML Explained (Second Edition)*, Addison-Wesley, 1997, p32  
Used primarily as reference and background material for discussion of SGML
- [Buckland1997] Buckland MK, *What is A 'Document?'*, Journal of the American Society of Information Science 48(9), Sept 1997, pp804-809  
In this paper, Buckland discusses the relationship between documents and objects. He cites a number of works to create a cognitive link between an artefact and a representation of it for study.
- [Bush1945] Bush V, *As We May Think*, The Atlantic Monthly, July 1945  
One of the very earliest writings about associative linking. Bush introduces a device he calls the 'Memex', which, although mechanical, does exhibit many qualities of current hypermedia systems.
- [Chidamber1994] Chidamber SR, Kemerer CF, *A Metrics Suite for Object-Oriented Design*, IEEE Transactions on Software Engineering 20(6), 1994, pp476-498
- [Chakrabarty1999] Chakrabarty A, Graebner R, Stocky T, *The Conception of LOGO*, 6.933J, Dec 1999, MIT  
<http://mit.edu/6.933.www/LogoFinalPaper.pdf>, pp13-16  
An historical document documenting the development of the LOGO language and its place in the range of functional programming languages.
- [Church1936] Church A, *A Note on the Entscheidungsproblem*, Journal of Symbolic Logic, Vol 1, 1936, pp40-41  
One of the earlier papers outlining the use of the lambda calculus by one of its principal creators.

- [Codd1970] Codd EF, *A Relational Model for Large Shared Data Banks*, CACM 13(6), June 1970, pp377-387
- The defining paper about the relational data model, the abstract dicusses the idea of data hiding and abstraction, a concept later borrowed in object-oriented analysis and design.
- [Conkin1987] Conklin, J *Hypertext: An Introduction and Survey*, IEEE Computer, 20(9), September 1987, pp17-41
- An early article providing a survey of the area of hypermedia design. This is, of course pre-web.
- [CPlus2001] CPsusPlus.com, *History Of C++*, Available online at: <http://www.cplusplus.com/info/history.html>, Accessed June 22, 2001
- [Crocker1997] Crocker, D ed., *RFC2234 - Augmented BNF for Syntax Specifications: ABNF*, Avaialble online at: <http://www.ietf.org/rfc/rfc2234.txt?number=2234>, Accessed June 22, 2001
- A document describing a variant of the BNF notation for describibng programming languages. The comparision between BNF notation and SGML DTDs is, in some ways, quite striking.
- [Dalton1996] Dalton S, *A Workbench to Support Development and Maintenance of World-Wide Web Documents*, Available Online at :<http://www.dur.ac.uk/dcs0cb/workbench/>, Accessed November 10 2001, Department of Computer Science, University of Durham, UK, 1996
- [DeMarco1979] DeMarco, T, *Structured analysis and System Specification*, Prentice-Hall, 1979
- This work introduced DeMarco's development of Structured Analysis, to formaise the creation of systems that could then be engineered using Structured Design

- [Deshpande2001] Deshpande Y, Hansen S, *Web Engineering: Creating a Discipline among Disciplines*, IEEE Multimedia 8(2) April-June 2001, pp82-87
- An article that posits that web engineering is not simply an adjunct to the related area of software engineering. The article describes man of the disciplines required to deliver successful web projects.
- [Dijkstra1968] Dijkstra EW, *Go To Statement Considered Harmful*, CACM, 11(3), March 1968, pp147-148
- This article presaged the need for more order in the design of software and paved the way for the work of Wirth and Parnas.
- [Dijkstra1972] Dijkstra EW, *The Humble Programmer (1972 ACM Turing Award Lecture)*, CACM 15(7) July 1972, p859-866
- While re-iterating the points made in his deprecation of the GOTO statement, this article also notes the parallels between software development in the early 1970s and the early 1960s.
- [Einstein1905] Einstein, A, *Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt ("On a Heuristic Viewpoint Concerning the Production and Transformation of Light")*, Annalen der Physik, v.17, 1905, pp132-148.
- The paper that introduced the quantum theory and provided verification of Planck's conjecture that light was quantised.
- [Eisenstadt1988] Eisenstadt M and Brayshaw M, *The Transparent Prolog Machine (TPM): An Execution Model and Graphical Debugger for Logic Programming*. Journal of Logic Programming, 5(4), 1988, pp277-342
- This paper proposes an implicit link between declarative and object-oriented systems. This link is useful when

considering the relationship between mark-ups and the Document Object Model within web systems

- [Englebart1961] Englebart DC, *Special Considerations of The Individual as a User, Generator and Retriever of Information*, American Documentation 12(2), 1961, pp121-125.

This project (Augment) was the first working hypertext system, demonstrating that Nelson's and Bush's ideas of an associative navigation system could work in practice

- [Englebart1963] Englebart DC, *A Conceptual Framework for the Augmentation of Man's Intellect*, In *Vistas in Information Handling*, Spartan Books, 1963, pp1-29.

This project (Augment) was the first working hypertext system, demonstrating that Nelson's and Bush's ideas of an associative navigation system could work in practice

- [Englebart1995] Englebart, DC, *Boosting Our Collective IQ - Selected Readings*, Bootstrap Institute/BLT Press, 1995

- [Fenton1997] Fenton NE, Pfleeger SL, *Software Metrics - A Rigorous & Practical Approach* (Second Ed.). International Thomson Publishing, 1997.

- [Fogel1994] Fogel, DB, *An Introduction to Simulated Evolutionary Optimization*. IEEE Trans. on Neural Networks, vol. 5, No. 1, 1994, pp3-14.

- [Fogel1966] Fogel LJ, Owens AJ. and Walsh MJ, *Artificial Intelligence through Simulated Evolution*, John Wiley, 1966

Fogel's work provides an introduction to the major ideas present within the area of Genetic Algorithms and self-evolving code.

- [Feynman1964] Feynman RP, Leighton RB, Sands M, *The Feynman Lectures on Physics Volume II, The Electromagnetic Field*, Addison-Wesley, 1964, p18

- [Frostburg2000] *Survey of Programming Languages*, Available on-line at:  
[http://www.frostburg.edu/dept/cosc/htracy/cosc120/MODULES120/NetPL/PL\\_Net.htm](http://www.frostburg.edu/dept/cosc/htracy/cosc120/MODULES120/NetPL/PL_Net.htm),  
Accessed November 2, 2001,  
Last alteration date: December 13 2000, Department of Computer Science, Frostburg State University MD USA, 2000  
A web page provideing course materials that puts HTML firmly in the category of high level programming
- [GAT2001] *Genetic and Evolutionary Algorithm Toolbox for use with MATLAB*, Online version at <http://www.geatbx.com/docu/index.html>, (Version 1.92), last modified November 1998, accessed March 30th 2001  
A comprehensive online archive relating to self-modifying and evolutionary systems, Useful for general descriptions and discussion of genetic algorithms.
- [Gauthier1970] Gauthier, R,Pont,S. *Designing Systems Programs*, Prentice-Hall, 1970
- [Gabriel1996] Gabriel RP, *Patterns of Software*, Oxford University Press, 1996  
Gabriel discusses Alexander's ideas of pattern languages and habitability and attempts to extend these into the area of writing software.
- [Glover2001] Glover JJ, *Inherently Flexible Software*, PhD Thesis, University of Durham UK, 2001  
A thesis that discusses the software evolution by attempting to develop self-maintaining and evolving systems that obey Lehman's Laws.
- [Goldfarb1970] Goldfarb CF, Mosher EJ, Peterson TI, *An Online System for Integrated Text Processing*, Proc, American Society for Information Science,July 1970, pp147-150

An introduction to a prototypical mark-up language (GML), the precursor to the modern mark-ups in existence today. The authors were also involved in the creation of SGML, from which HTML and XML has evolved.

- [Goldfarb1997] Goldfarb CF, *SGML: The Reason Why and the First Published Hint*, Journal of the American Society for Information Science. 48(7), July 1997, Online version at: <http://www.sgmlsource.com/history/jasis.htm>

- [Hassan2001a] Hassan AE, Architecture Recovery of Web Applications, MSc Thesis, University of Waterloo, Ontario, Canada, 2001

A discussion of the reverse engineering of web applications. As a part of this, the general structure and form of web applications is discussed.

- [Hassan2001b] Hassan AE & Holt RC, Towards a Better Understanding of Web Applications, In Proc. Third International Workshop on Web Site Evolution WSE 2001, Florence, Italy, November 2001,

This paper is an adjunct of the Masters thesis above, presenting it in a summary form

- [Hateley1988] Hateley DJ and Pirbhai IA, *Strategies for Real-Time System Specification*, Dorset House Publishing, New York USA, 1988.

Hateley and Pirbhai extended the real time analysis and design models first developed by McMenamin & Palmer several years before.

- [Hatz1995] Hatzimanikatis AE, Christodoulakis D, Tsalidis CT, *Measuring the Readability and Maintainability of Hyperdocuments*, Software Maintenance: Research and Practice. Vol 7, 1995, pp77-90

This paper is a major work in the development of measurements in hypermedia, although, like earlier work, it

tends to concentrate on global issues of navigation and path traversal.

- [Hoare1972] Hoare CAR, Dahl O, Dijkstra E, *Structured Programming*, Academic Press 1972  
Together with Parnas and Wirth, this work collected together thinking regarding the ordered development of software.
- [Holland1975] Holland, JH, *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.
- [IEEE1993] *IEEE Software Engineering Standards*, 610.12-1990, pp47-48
- [Intel2001] *Moore's Law*, Available online at:  
<http://www.intel.com/intel/museum/25anniv/hof/moore.htm>  
, Intel Corp - Accessed 2.3.2001 last date of modification not specified
- [Isakowitz1995] Isakowitz T, Stohr EA, Balasubramanian P *RMM: A Methodology for Structured Hypermedia Design*, CACM 38(8) August 1995, pp34-44  
An introduction to the general hypermedia design methodology RMM. This methodology has been used consequently to design web systems in a number of cases.
- [ISO8879] *SGML (ISO 8879:1986)*, International Standards Organisation, 1986  
The definition document for the SGML standard, from which HTML and XML derive.
- [ISO9541] *Font Information Exchange (ISO/IEC9541-1:1991)*, International Standards Organisation, 1991
- [ISO9621] International Standard ISO/IEC 9126. *Information technology -Software product evaluation - Quality characteristics and guidelines for their use*, International

- Organization for Standardization, International Electrotechnical Commission, Geneva  
A prototypical standard for software quality.
- [ISO14977] ISO/IEC 14977:1996, *Extended BNF*, ISO, 1996  
ISO Document defining the standard for EBNF.
- [Jelliffe1998] Jelliffe R, *The XML and SGML Cookbook: Recipes For Structured Information*, Prentice Hall 1998  
A comprehensive reference work from a major figure in the development of the XML standards within W3C, covering general principles of mark-up languages.
- [Jelliffe1998a] Jelliffe R, *The XML and SGML Cookbook: Recipes For Structured Information*, Prentice Hall, 1998, pp47-72
- [Jelliffe1998b] Jelliffe R, *The XML and SGML Cookbook: Recipes For Structured Information*, Prentice Hall, 1998, p68
- [Johnes2001] Johnes C, *Sizing Up software*, Scientific American, xx(x), Dec 1998.  
Available online at:  
<http://www.sciam.com/1998/1298issue/1298jones.html>  
Accessed October 15 2001  
A document providing function point measures for a number of languages of various levels, including HTML, which is classified as a high/very high level language.
- [Jones1991] Jones C, *Applied Software Measurement*. McGraw-Hill, 1991
- [Kelvin1891] William Thompson, Lord Kelvin, *Popular Lectures and Addresses*, 1891-1894
- [Kleene1956] Kleene, SC, *Representation of events in nerve nets and finite automata*, in C. E. Shannon and J. McCarthy, eds, Automata Studies, Annals of Mathematics Studies No. 34, Princeton University Press, 1956, pp3-42.

Together with Fogel, Kleene's work was used to gain insight into self-modifying and evolutionary systems.

- [Knuth1984] Knuth DE, *The T<sub>E</sub>XBook*, Addison-Wesley, 1984
- [Knuth1969] Knuth, D, *The Art of Computer Programming*, Vol1: Fundamental Algorithms, 1969, pp4-6
- A key work in computing, discussing and collecting many different algorithms and problem solving techniques. The first chapters deal with the question of what an algorithm is and what properties one must have.
- [Kowalski1979] Kowalski, RA, *Algorithm = Logic + Control*. CACM 22(7), 1979, pp424-436
- Kowalski challenges Wirth's pivotal Algorithm=Program+Data paper, by proposing an alternative, non-imperative or procedural framework for programming.
- [Krug2000] Krug S, *Don't Make Me Think*, Que, 2000
- [Lamport1986] Lamport L, *L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System*, Addison-Wesley, 1986
- [LaT<sub>E</sub>X2000] *The LaT<sub>E</sub>XProject Home Page*, Available on-line at <http://www.latex-project.org/>, Last modification date August 10, 2000, Accessed May 15, 2001
- Both this and the previous item have assisted in the preparation of this report and have also served as background material about the structure, evolution and philosophy of the L<sup>A</sup>T<sub>E</sub>Xsystem.
- [Lee1998] Lee HS, Suh WJ, *A Workflow Based Hypermedia Developing Methodology*, Graduate School of Management, Korean Advanced Institute of Science and Technology, Available online at: [http://kms.kaist.ac.kr/re\\_center/paper/1998-200.pdf](http://kms.kaist.ac.kr/re_center/paper/1998-200.pdf), 1998, Accessed June 25, 2001

- [Lehman1985] Lehman MM, *Program Evolution*, Academic Press, 1985
- [Lehman1999] Lehman MM, *Software System Maintenance and Evolution in an Era of Reuse, COTS and Component Based Systems*, IEEE International Conference on Software Maintenance, Oxford, England, August 30 - September 3, 1999
- This item is a later presentation following from the previous work, laying down laws of evolution for software systems in terms of their development and useful life
- [Lewis2001] Lewis JP, *Large Limits to Software Estimation*, ACM Software Engineering Notes, 26(4), July 2001, pp54-59
- A recent paper declaring that, by definition, precise measures of software cost and effort cannot be made.
- [Lorenz1994] Lorenz M, Kidd J, *Object-Oriented Software Metrics*, Prentice-Hall, 1994
- [Lowe1999a] Lowe D, Hall W, *Hypermedia And The Web: An Engineering Approach*, Wiley, 1999, p200
- A comprehensive work detailing the major areas of hypermedia design and research, encompassing discussion of the major methods and many of the major projects from the recent past.
- [Lowe1999b] *ibid*, pp509-510
- [Lowe1999c] Lowe D, Hall W, *ibid*, p337
- [Lowe1999d] *Web Development Methodologies: Help Or Hindrance?* WebNet Journal 1(3), July-September 1999, pp9-10
- This article surveys current design methodologies and asks what benefits they provide for developers and, more particularly, if there is any agreement about what qualities need to be measured.
- [Maurer2002] Maurer F, Martel S, *Extreme Programming: Rapid Development for Web-Based Applications*, IEEE Internet Computing 6(1), January 2002, pp86-90

- [McCall1977] McCall JA, Richards PK, Walters PF, *Factors in Software Quality*, RADC-TR-77-369, 1977, vols I, II, II, US Rome Air Development Center Reports NTIS AD/A-049 014,015,055,1977
- An early software quality model, used as the template for the first ISO standard, ISO 9126 (q.v.)
- [McMenamin1984] McMenamin, S Palmer, J, *Essential Systems Analysis*, Prentice Hall, 1984
- An early work disussing the use of event partitioning and the importance of state in strucutred analysis and design. Event partitioning introduced the idea of a temporal component in such systems.
- [MacLennan1990] MacLennan, *Functional Programming: Practice and Theory*, Addison Wesley, ISBN 0-201-13744-5, 1990, pp23-26
- A work that provides a useful overview of functional programming languages and a short treatment of the issues underlying functional program design.
- [Mendes2001] Mendes E, Mosley N, Counsell S, *Web Metrics- Estimating Design and Authoring Effort*, IEEE Multimedia 8(1) Jan-Mar 2001, pp50-57
- One of the first papers to consider the problem of metric definition and collection for web systems.
- The paper has an emphasis upon the hypermedia aspects of design but is useful nonetheless in this context.
- [Mills1971] Mills HD, *Top Down Programming in Large Systems*, R Rustin ed. Prentice-Hall, 1971
- Another early work advocating the use of structured and modular design in software systems.
- [MITML1999] *Welcome to Markup Languages: Theory and Practice*, Journal of Markup Languages: Theory and Practice, 1(1), MIT Press, Winter 1999, pp1-6

- [MS2002] Windows XP Professional Features, Available online at: <http://www.microsoft.com/windowsxp/pro/evaluation/features.asp> Accessed January 25 2002
- [Murugesan1999] Murugesan S et al., *Web Engineering: A New Discipline for Development of Web-based Systems*, Online Proc. 1st International Conference on Software Engineering Workshop on Web Engineering, <http://fistserv.macarthur.uws.edu.au/san/icse99-WebE/ICSE99-WebE-Proc/San.doc>
- One of the early documents advocating a process and a systematic approach to the development of high quality Web-based systems while admitting that web systems are different to more conventional software.
- [Nakayama2000] Nakayama T, Kato H, Yamane Y, *Discovering the Gap Between Web Site Designers' Expectations and Users Behavior*, In Proc. 9th International World Wide Web Conference (WWW9), Amsterdam May 15-19 2000, Available online at: <http://www9.org/w9cdrom/start.html>
- This paper presents an analysis of linkage between nodes derived from elements of page layout, improving navigation for users by determining the relevance of connections between nodes in a hypermedia system. The paper is mainly concerned with issues of user interface design and does not have directly address engineering quality.
- [Nelson1965] Nelson T, *A File Structure for the Complex, the Changing, and the Indeterminate*, 20th National Conference, ACM, 1965
- The document that provided one the first appearances of the term 'hypertext' and provided the basis for the creation of Nelson's 'Xanadu' system.
- [Nelson1990] Nelson T, *Literary Machines*, Mindful Press, 1990
- [Nielsen2000] Nielsen J, *Designing Web Usability: The Practice of Simplicity*, New Riders Publishing, 2000, pp10-15

- Survey of web usability by the premier name in the field.
- [Nielsen2001] Nielsen J, Available online: <http://www.useit.com/> Accessed October 1 2001  
Jakob Nielsen's own web site, providing updated content and extensions to his publications.
- [NOC1997] Cover R, *SGML: A Textual Representation for Information Structure*, Notes On Computing 16(5), SIL International, Dallas Texas, 1997, p37
- [OED1993] *The Shorter Oxford English Dictionary*, Oxford University Press, 1993, 4th Edition
- [Oftel2001] The Office of Telecommunications (Oftel), *Consumers? use of Internet Oftel residential survey Q6 August 2001*, Available Online at:  
<http://www.oftel.gov.uk/publications/research/2001/q6intr1101.htm>, Published november 4 2001, Accessed November 18 2001
- [Orenstein1996] Orenstein R, *An HTML 2.0 Pattern Language*, <http://www.anamorph.com/docs/patterns/default.html>, Last modified May 16, 1996, Accessed MAY 21 2001
- [OWLA2002] OWLA Project, University of Oulu, Finland Available Online at:  
<http://owla.oulu.fi/index.html>  
A reasearch prject concerned principlally with the developemnt of mobile systems. The project views the development of web applications as information systems and hyermedia design.
- [Parnas1972] Parnas, DL, *On the Criteria To Be Used in Decomposing Systems into Modules*, CACM 15(12), December 1972, pp1053-1058  
Together with Wirth's 1971 paper and the work of Hoare et al. this work provided a platform for Structured Design. Unlike Wirth, Parnas was interested not just in

breaking down problems into smaller units but giving each unit some kind of functional responsibility within the whole system in a more modular fashion.

[Pfleeger1989] Pleeger SL, *An Investigation of Cost and Productivity for Object-Oriented Development*, PhD Thesis, George Mason University, Fairfax VA USA, 1989

[Pfleeger1990] Pleeger SL Palmer JD, *Software Estimation for Object-Oriented Systems*, Proc, International Function Point Users Group Fall Conference, San Antonio TX USA, 1990, pp181-196

[Phillips2000a] Phillips EM, Pugh DS, *How to get a PhD*, Third Edition, Open University Press, 2000

An indispensable guide for the new PHD student. The book deals with practical issues as well as discussing the academic process of completing the PhD, complete with rough guides to timings within the project.

[Phillips2000b] *ibid.*, pp74-99

[Pitkow1995] Pitkow JE, *Summary of WWW Characterizations*, In Proc. 7th WWW Conference, 1998

An early work whose focus is on network performance in the delivery of web services, concerning itself with optimisation of server performance and underlying network configurations.

[Powell1998a] Powell TA, et al, *Web Site Engineering: Beyond Web Page Design*, Prentice Hall, 1998

Powell's book sets down many of the issues talked about by White, Yale's Style Guide and others into a single volume, arguing that treatment of the web as a simple document systems was no longer viable and that alternative methods needed to be found.

[Powell1998b] *ibid*, pp16,17

- [Powell1998c] *ibid*, pp25-49
- [Pressman2000] Pressman, RS, *Software Engineering: A Practitioner's Approach (European Adaptation)*, (Fifth Edition), McGraw-Hill, 2000
- A leading work in the field of software engineering, describing many of the major processes in current use. The fifth edition also introduces newer concepts such as a web engineering and e-commerce system engineering.
- [Pressman2000a] *ibid*, pp813-842
- [Pressman2000b] Pressman, RS, *What a Tangled Web We Weave*, IEEE Software, 17(1), Jan 2000, pp18-21
- An article in which Pressman examines the current confusion in the engineering of web systems, reiterating that the use of engineering methods is just as valid within a web context
- [Pressman2000c] Pressman, RS, *Software Engineering: A Practitioner's Approach (European Adaptation)*, (Fifth Edition), McGraw-Hill, 2000, pp813-842
- [Pressman2000d] Pressman, RS, *ibid.*, p79
- [Pressman2000e] Pressman, RS, *ibid.*, p496
- [Pressman2000f] Pressman, RS, *ibid.*, p887-88
- [Puttnam1978] Puttnam LH, *A General Empirical Solution to The Macrosoftware Sizing and Estimating Problem*, IEEE Transactions on Software Engineering SE 4(4), pp345-361, 1978
- [Rezgui1995] Rezgui Y, Debras P, *An Integrated Approach For a Model Based Document Production and Management*, Electronic Journal of Information Technology in Construction, Vol 1, 1996, pp1-21

- [Ricca2000a] Ricca F, Tonella P, *Web Site Analysis: Structure and Evolution*, In Proc. 2nd International Workshop on Web Site Evolution, WSE 2000, Zürich, Switzerland, 2000
- [Ricca2000b] Ricca F, Tonella P, *Visualisation of Web Site History*, In Proc. 2nd International Workshop on Web Site Evolution, WSE 2000, Zürich, Switzerland, 2000
- Ricca and Tonella's work concentrates, like earlier work by Botafogo and Hatzimanikatis, on global properties of hypermedia systems, exploring navigation and paths through such systems. Unlike the earlier work, which concentrates upon more general hypermedia, the authors apply this thought to web systems in particular.
- [RISE2001] University of Durham, Research Institute in Software Evolution, Available online at: <http://www.dur.ac.uk/CSM/>, Accessed November 23 2001
- [Rumbaugh1991] Rumbaugh, J et al. *Object-Oriented Modelling and Design*, Prentice Hall International, 1991
- [Sametinger1994] Sametinger J, —em Object-Oriented Documentation, The Journal of Computer Documentation, 18(1), January 1994, pp3-14
- [Sammet1972] Sammet J, *Programming Languages History & Future*, CACM 15(7) July 1972, pp601-610
- [Sammet1976] Sammet J, *Roster of Programming Languages*, CACM 19(12), Dec 1976
- These works by Sammet are used mainly for historical purposes, tracing the development and spread of different programming languages and techniques.
- [Schwabe1995] Schwabe D, Rossi G, *The Object Oriented Hypermedia Design Model*, CACM 38(8) August 1995, pp45-46
- [Somerville1989] Somerville I, *Software Engineering - Third Edition*, Addison-Wesley 1989, ISBN 0-201-17568-1, p3

- [Stephens2001a] Stephens D, *What Is Software?*, Unpublished Paper University of Hull, 2001
- [Stephens2001b] Stephens D, *What Is A Document?*, Unpublished Paper, University of Hull, 2001
- [Stross1996] Stross C, *The Web Architect's Handbook*, Addison-Wesley, Harlow UK, 1996, p180
- [Udell2001] Udell J, Document Engineering, Byte Magazine, available online at: <http://www.byte.com/documents/s=620/byt20010510s0001/index.htm>, May 2001
- [UCNZ1998] University of Canterbury, Christchurch, New Zealand *Basic Aspects of Squeak and the Smalltalk-80 Programming Language* Available online at <http://kaka.cosc.canterbury.ac.nz/wolfgang/cosc205/smalltalk1.html#history>, Last modified June 22, 1998, accessed Feb 10 2001
- An introduction to object-based environments.
- [UML1999] Rumbaugh J, Jacobson I, Booch G, *The Unified Modelling Language Reference Manual*, Addison-Wesley, 1999
- [vonNeumann1947] von Neumann J & Goldstine HH, *Planning and Coding Problems for an Electronic Computing Environment*, Part II vol I, report for US Army Ordinance Department, 1947 - Reprinted in von Neumann J, *Collected Works*, AH Taub ed. Vol V, Macmillan, pp80-151
- In this work, the authors introduce some of the basic tools to be used in the developemt of imperative programs, the most notable of which is the flowchart.
- [W3C2001a] Le Hégarret P, Wood L eds., *Document Object Model (DOM)* Avaiable on-line at: <http://www.w3c.org/DOM/>, Viewed June 25, 2001. Last modified: June 21, 2001

- The DOM provides the link between declarative and objects systems within HTML documents that is proposed within the work of Eisenstadt and Brayshaw
- [W3C2001b] Miller E et al. eds. *Semantic Web* Available on-line at: <http://www.w3c.org/2001/sw/>, Viewed June 21, 2001. Last modified: April 17, 2001
- [W3C2001c] World Wide Web Consortium, Online at <http://www.w3c.org>, Accessed October 30 2001
- [Walker1997] Walker R, *Information Engineering on the World-Wide Web: Drawing Analogies with Software Engineering*, In Proceedings of the Eighth Western Computer Graphics Symposium (SKIGRAPH '97; Whistler, BC, Canada; 6-9 April 1997), 1997, pp. 97-107  
Also available on-line at:  
<http://www.cs.ubc.ca/walker/papers/walker.1997a.pdf>
- [Ward1985] Ward PT & Mellor SJ, *Structured Development for Real-Time Systems*, Volumes 1-3. Vol. 1: Introduction and Tools, Yourdon Press (Prentice-Hall) 1985.
- [Warren1999] Warren PJ, Boldyreff C, Munro M, *The Evolution of Websites*, in Proc. International Workshop on Program Comprehension IWPC99, Pittsburgh PA, USA, IEEE Computer Press 1999 Submitted November 2001
- [Warren2001a] Warren PJ, Gaskell C, Boldyreff C *Preparing the Ground for Website Metrics Research*,  
In Proc. Third International Workshop on Web Site Evolution WSE 2001,  
Florence, Italy, November, 2001
- [Warren2001b] Warren PJ, *The Evolution of Websites*, PhD Thesis, University of Durham, Submitted November 2001
- [WebE2001] Murugesan S ed., *WebE Home Page*, Available Online at:  
<http://fistserv.macarthur.uws.edu.au/san/WebEhome/>  
Accessed November 16 2001

- [WebSEM2000] Boldyreff C, Warren P et al, *WebSEM: Web Site Evaluation Metrics Home Page*, Available Online at: <http://www.dur.ac.uk/dcs3pjw/web-sem/> University of Durham RISE, Accessed November 20 2001, Last updated February 10 2000
- [Weiss1994] *Isomorphism Between OOP and Documentation: Reflections on Sametinger's Object Oriented Documentation*" The Journal of Computer Documentation, 18(2), May 1994, pp8-12
- Sametinger considers document systems in an object oriented way, allowing for object oriented techniques for maintaining such systems. Weiss' work explores some of the issues introduced in Sametinger's work in more detail
- [White1996] White B, *Web Document Engineering*, Available Online at: [http://www5conf.inria.fr/fich\\_html/slides/tutorials/T14/all.htm](http://www5conf.inria.fr/fich_html/slides/tutorials/T14/all.htm), Accessed November 10 2001
- This presentation by Prof. Bebo White of SLAC at WWW5 in 1996 introduces some of key issues in the creation of larger scale web systems. The term 'web document engineering' is also introduced to describe the creation of web systems.
- [Whitmire1997] Whitmire S, *Object-Oriented Design Measurement*, Wiley, 1997
- [Whorf1956] Whorf BL, *Language, Thought and Reality, Selected Writings of Benjamin Lee Whorf*, JB Carol ed. (2nd Edition), MIT Press, 1956
- [Wirth1971] Wirth, N, *Program Development by Stepwise Refinement*, CACM, 14(4), April 1971, pp221-227
- Along with Parnas and Hoare et al. this work was instrumental in the development of Structured Design and programming in the early 1970s. Unlike Parnas, Wirth

concentrates of progressively breaking down problems into smaller units.

[Wirth1976] Wirth N. *Algorithms + Data Structures = Programs*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

Wirth presents the key idea of programs being composed of two separate components: data and algorithm, essentially a cornerstone of the procedural paradigm. This is a separation that has persisted, even with Object Orientation.

[Wirth1977] *What Can We Do About the Unnecessary Diversity of Notation for Syntactic Definitions*, CACM 20(11) Nov 1977, pp822-823

Here Wirth considers the meaning of the term ‘programming language’ arguing that software should not be classified in the same way and that they more closely resemble formal notations, as in mathematics, because they are not spoken.

[WSE2001] Tilley S ed., Proceedings of the Third International Workshop on Web Site Evolution WSE 2001, Florence, Italy, November, 2001

[XML2000] *Extensible Markup Language (XML) 1.0 (Second Edition)*, David C. Fallside ed.  
Available on-line at: <http://www.w3.org/TR/REC-xml>, Created October 6 2000, Accessed March 6, 2002  
W3C recommendation for XML language specification

[XML2001] *XML Schema Part 0: Primer*, David C. Fallside ed.  
Available on-line at: <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>, Accessed May 17, 2001  
W3C recommendation for use of XML schemas, related to the use of DTDs

[Yale1999] Lynch PJ, Horton S *Web Style Guide : Basic Design Principles for Creating Web Sites*,

Yale University Press, Also available online at:  
<http://info.med.yale.edu/caim/manual/contents.html>,  
2001

This resource provides a great deal of the heuristic design techniques alluded to within this document.

[Zelnick1998]

Zelnick N, *Nifty Technology and Nonconformance. The Web in Crisis*, Computer, 31(10) October 1998, pp115-116,119

In which the authors draws parallels between the software crisis of the 1970s and the current direction of web development.