

Master's Thesis

Maintainability Aspects of the Website Development Life Cycle

carried out at the

Information Systems Institute
Distributed Systems Group
Technical University of Vienna

under the guidance of
Univ.Prof. Dr. Shahram Dustdar
and
Univ.Ass. Dipl.Ing. Dr. Clemens Kerer
and
Univ.Ass. Dipl.Ing. Martin Vasko
as the contributing advisors responsible

by

Markus Krumpöck
Kogelweg 11, 2631 Sieding
Matr.Nr. 9571186

Vienna, August 2008

0.1 Abstract

The development of websites often happens in an “ad hoc” manner. Therefore changes in requirements (for example a graphic redesign) often lead to a complete rebuild from scratch. To make things even worse, websites and web applications are, due to a highly dynamic web, more often object of changes, compared to classical software for the desktop, since they not only consist of software but also content and the rollout of changes is much more easy. The variety of programming languages and technologies for the development of dynamic web applications and the availability of many different frameworks and template engines make things more complicated.

Therefore the goal of this thesis is to develop some “best practices” and guidelines for how to design and implement websites for maintainability in order to create websites with a long lifetime without doing complete rebuilds.

The thesis will try to clarify which main points should be considered when designing a website for maintainability and scalability in informational architecture, layout, web applications and so on and to keep the effort to maintain a website within a reasonable extend. Additionally, it should be possible to add new functionality easily without the need for a complete change in the architecture.

In this thesis we will try to demonstrated how maintainability has been achieved in a real world case study done between 2003 and 2008, which consisted of about 6.700 documents after the redesign for maintainability based on these guidelines in 2003 and has increased to more than 55.000 documents nowadays in 2008.

0.2 Zusammenfassung

Die Entwicklung von Websites geschieht in vielen Fällen noch immer “ad-hoc”. Daraus resultiert, dass Änderungen in den Anforderungen (etwa ein grafisches Redesign) oftmals zu einer völligen Neuimplementierung führen. Erschwerend kommt hinzu, dass Websites und Webapplikationen aufgrund der hohen Dynamik des Webs wesentlich häufiger Änderungen unterworfen sind als klassische Software für den Desktop, da diese nicht nur auf Software bestehen sondern auch aus Inhalten und Änderungen wesentlich einfacher ausgerollt werden können. Die Vielfalt an verschiedenen Programmiersprachen und Technologien für die Implementierung dynamischer Webapplikationen sowie die Verfügbarkeit verschiedenster Frameworks und Template Engines verkomplizieren die Situation zusätzlich.

Daher ist es Ziel dieser Diplomarbeit die Erstellung von “Best Practices” und eines Leitfadens, wie man besser wartbare Websites konzipieren und implementieren kann.

Diese Arbeit wird versuchen welche Hauptpunkte bei der Konzipierung einer Website hinsichtlich Wartbarkeit und Skalierbarkeit hinsichtlich der Informationsarchitektur, des Layouts, der Webapplikationen usw. zu berücksichtigen sind und wie die Wartung mit vertretbarem Aufwand durchgeführt werden kann. Neue oder zusätzliche Funktionalität sollte hinzugefügt werden können ohne dass eine komplette Änderung der Architektur notwendig ist.

In dieser Arbeit möchte ich zeigen wie Wartbarkeit in einer von 2003 bis 2008 durchgeführten Fallstudie erreicht wurde, wobei die Website nach dem Redesign basierend auf diesen Leitlinien aus etwa 6.700 Dokumente bestand und mittlerweile (Stand Juni 2008) auf über 55.000 Dokumente angewachsen ist.

Contents

0.1	Abstract	i
0.2	Zusammenfassung	ii
1	Problem Description	1
1.1	Problem Definition	1
1.2	Motivation	2
1.3	Organization of this Thesis	3
2	Review of the state of the art / Related Work	4
2.1	Programming Languages and Technologies	4
2.2	Template Engines	4
2.3	Frameworks	7
2.3.1	MVC Frameworks for the Web	7
2.3.2	Presentation Frameworks	10
2.4	Web Content Management Systems / Wikis	11
2.5	Rich Internet Applications	13
2.6	Model Driven Architecture (MDA)	13
3	Definitions	15
3.1	Maintainability	15
3.2	Portability	17
3.3	Scalability	17
3.3.1	Horizontal Scalability	18
3.3.2	Vertical Scalability	18
3.4	Reusability	18
3.5	Usability	18
3.6	Accessability	19
4	Development Life Cycle	21
4.1	Concepts and Patterns	21
4.2	Informational Architecture	24
4.3	Realization of the Layout	30
4.4	Implementation	32
4.4.1	Static vs. Dynamic Web Pages	33
4.4.2	Architecture for Static Generation	36
4.4.3	Repository for Media - Filesystem vs. Database	38
4.4.4	Business Logic	40
4.5	Testing	41
4.6	Deployment	42

4.7	Operating, Monitoring	43
5	Case study	45
5.1	Problems before the Redesign	45
5.2	Goals and Non-Goals	47
5.3	Data Migration	50
5.4	Informational Structure and Layout	53
5.5	Implementation	54
5.6	Content Management and Deployment	56
5.7	Operating	58
5.8	Statistical Data	63
5.9	Additional new Functionality	64
5.10	Possible Improvements	71
6	Summary and conclusion / Future Work	78
6.1	Summary	78
6.2	Future work	79
A	List of Figures, Tables and Listings	81

1 Problem Description

1.1 Problem Definition

In the past it has been a common practice to implement websites in an “ad hoc” manner, due to a document centered view instead of using an overall concept. Since websites are growing fast and becoming more and more complex the results of the development process are often thrown away and the website is completely rebuilt from scratch. Instead, it would be more cost efficient to design and implement websites from the beginning with focus on maintainability, especially since website are a matter of frequent changes.

Unfortunately, maintainability is a non-functional requirement, and therefore often is not seen by the customer as a necessary characteristic, because it does not change the behaviour or look of the website or web application, but only causes additional costs - at least in the short term. So in general it takes some work to convince the customer to design for maintainability which will produce higher initial costs but will be more cost efficient in the long term.

Another problem is that websites are often optimized for a “nice look” - in the worst case even for a certain browser version or screen resolution, so they are optimized for the customer’s perception. For example if the customer wants to use a certain font which is not widely used the text elements have to be implemented as graphics which are hard to maintain - each time you want to change the labeling you have to generate a new graphic instead of simply changing the text.

To make things worse, there are a lot of different and maybe incompatible technologies, like programming languages, template engines and frameworks, content management systems and so on which complicates it to build a maintainable system based on a common concept.

There are a lot of other examples what can wrong in order to make a website hard to maintain, e.g. copying the same element (graphic, style, code, content, navigation, ...) again and again instead of reusing it, or mixing up style and content, so they can not be exchanged without bothering the other, and so on. Therefore one of the goals of this thesis to focus on the things that can go wrong, how to avoid them and to present some best practices in order to have a guideline when designing and implementing websites.

1.2 Motivation

Various studies [1] have shown that that the costs for the maintenance phase in software engineering typically has been around two third of the overall costs in the past, but even can be more than 90 percent.

Since websites are changed more frequent than classical software for the desktop because they not only consist of software, but also documents, images, audio, video and other media content, informational infrastructure, layout and so on the percentage may be even higher for websites.

Therefore the goal would be to decrease the cost for the maintenance phase. This thesis will show some techniques and concepts in order to reduce the effort for maintenance and therefore costs.

To implement maintainable websites may only be useful if the site a certain size (e.g. measured in number of documents) and if it is used over a longer period in time. A website with only a few documents (maybe smaller than 50) or a short lifetime (e.g. only for one event like a concert or election) may not be designed for maintainability because it adds overhead in effort for design and implementation and therefore additional costs.

To decide when to design a website for maintainability one has to compare the cost estimation without design for maintainability against cost estimation with design for maintainability over the total lifetime of the website (that is, the TCO, Total Cost of Ownership over the whole lifetime). Of course this is not only a decision for the hole site, but also for every single additional feature where you can decide to prepare it for modifications or assume that it will not be changed in the future.

Bommer et al. ([2], pp. 39-40) say in their book about software maintenance that it is always a bet whether it pays off or not on the long term to invest effort for quality characteristics of software, but you have to take this bet into account if your are a project manager, whether you would like to or not. In the long term it will always pay off to invest in quality issues like preparing the software for the maintenance phase initially, because such issues can only hardly be added after the design, but rather every design decision has to take into account the effects regarding maintenance.

Another advantage of designing for maintainability is the possibility for reuse of some parts (mainly business or presentation logic, but also layout, infor-

mational structure, ...) in other projects.

Designing for maintainability also offers better support for a distributed development with distributed role. In bigger web projects usually different people with different knowledge and abilities work together, for example graphic designers, programmers, content managers, administrators, customer, project manager so on. When separating the different layers (layout, business logic, content, ...) you get the possibility to change one layer without bothering the other, so each role can change its artefacts independent of the other role - the only thing which has to be defined are the interfaces between the different layers.

1.3 Organization of this Thesis

This master's thesis is organized as follows:

Chapter 2 discusses the state of the art in web engineering and technologies for website and web application development as well as related work.

Chapter 3 provides definitions for maintainability, reusability, scalability, usability and accessibility.

Chapter 4 discusses the single steps and tasks in the website development life cycle and what should be taken care of regarding maintainability.

Chapter 5 presents a real world case study done from 2003 to 2008, which has been designed and implemented with focus on maintainability, scalability, usability and performance.

Chapter 6 presents a summary and suggestions, what further could be done to improve maintainable website development.

2 Review of the state of the art / Related Work

2.1 Programming Languages and Technologies

Websites and web applications nowadays are based on client side technologies like (X)HTML, WML, DHTML, CSS, Java Script, Flash, Java Applets and server side technologies. The latter can be distinguished in scripting languages like PHP, PERL, Python and Ruby or precompiled languages like Java or the ones based on ASP.NET (C#, VB.NET). XML and XLST could be in use either on the client side or on the server side, but mainly is used for transformation of content in XML format in other output formats like XHTML, WML or PDF with help of a XLST transformation on the server side, due to poor support for XLST by the browser suppliers.

The Common Gateway Interface (CGI) [3] defines an interface for server sided applications between the web server and the external application, usually written in one of the scripted languages above. The main disadvantage of this technology is the low performance, because every CGI call from the browser is executed in a new process, the script is interpreted every time once again, databases connections have to be reestablished for every call and so on.

To avoid this disadvantages there have been implemented several other technologies, for example modules which can be integrated in the web server process (e.g. `mod_perl` and `mod_python` for the Apache web server) which are loaded only once when the web server starts up instead of loading them for every client request, or precompiled technologies based on the Java technology (JSP, Servlets, Java Server Faces, ...), which are compiled automatically only once after a modification in the source code and can reuse database connections over several calls (“persistent connections”).

Most template engines, content management systems and frameworks for web applications use one or more of the technologies and languages described above.

2.2 Template Engines

In order to separate the content from the business logic, template engines have been implemented, which are able to process templates with placeholders for variables or files to include.

The templates may be written in one of the common languages for web applications or they may have their own syntax - depending on the engine. The advantage of having the same programming language for the templates and the business logic is, that you don't have to learn a new language, but the disadvantage is that it is more easy to mix up content and business logic. In addition, the syntax of the template engine may be easier to learn for webdesigners which only have knowledge of HTML, CSS or Javascript, but are no programmers.

[4] gives an overview of different template engines, the programming languages they use and a comparison of the functionality they offer.

Examples for popular template engines are:

- Smarty (PHP) [5]
- Template Toolkit ([6], [7]); Mason (PERL)
- Webmacro (Java)

Some template engines also offer the functionality to generate the output (which may be in several formats, like HTML and PDF) as static documents for all documents of a website, while for other you have to create a wrapper if you want to get a static version of all the final documents. The advantage of having static copies of the generated output is an improvement in performance, because the documents have to be processed only once after a modification and not for every request for the document. Template engines may also offer dynamic caching functionality.

A template for the Template Toolkit may look like the following example:

Listing 1: Sample template for the Template Toolkit

```
1
2 [% INCLUDE header title="Select a journal" %]
3 ...
4 <tr>
5   <td>
6     <select name=journal_id >
7       [% FOREACH journal = journals -%]
8         <option value="[%_journal.id_%]"
9           [% IF formdata.journal_id == journal.id %]
10          selected="selected"
```

```

11         [% END -%]>[% journal.description %]</option>
12     [% END %]
13 </select>
14 </td>
15 ...
16 [% INCLUDE footer %]

```

The calling PERL code may look like this:

Listing 2: Calling PERL code for the Template Toolkit

```

1 #!/usr/bin/perl
2
3 print "Content-type: \text/html\n\n";
4
5 use CGI::Carp qw(fatalsToBrowser);
6 use CGI 'standard';
7 use Template;
8
9 my $template = Template->new();
10
11 my %formdata;
12 $formdata{'journal_id'} = param("journal_id");
13
14 # get list of hashes, where each entry has a id an a description
15 my @journals = getJournalsfromDatabase();
16
17 my %data = (
18     journals => \@journals,
19     formdata => \%formdata
20 );
21
22 $template->process('template', \%data )
23 || die "Template_process_failed:\n", $template->error(), "\n";
24 %%\end{verbatim}

```

The INCLUDE keyword includes another HTML file, where title is a variable in the included HTML file, so the same header file may be reused for different purposes. The PERL code is the program which is called from the browser via CGI, gets the data from the database and passes the values to the template where they are displayed in a loop via the FOREACH statement.

2.3 Frameworks

The Frameworks described below may be based either directly on a common programming language for web applications or on a template engine.

[8] gives an overview of the most common frameworks for web applications and a comparison of features, e.g. if they support AJAX, the Model-View-Controller design pattern (MVC), internationalization (i18n) and localization (l10n), object relational mapping (ORM), template engines, caching, form validation and several other features.

2.3.1 MVC Frameworks for the Web

In order to separate the three different layers - content (Model), layout (View) and business logic (Controller) a lot of frameworks have been implemented which support the MVC design pattern, so it is easier to modify one layer without affecting the two others.

The MVC pattern was first described in 1979 [9] by Trygve Reenskaug, when working on Smalltalk at Xerox PARC. The original implementation is described in [10].

[11] gives an overview of common frameworks for the web which uses separation of concerns via the MVC pattern.

The most important MVC frameworks grouped by concept for the web namely are:

Java - Model 2

When using Java for implementing the view Java Server Pages (JSP) have been used in the beginning. The so called “Model 1” approach uses JSP for the view, but also includes the controller in the same page(s). For the model JavaBeans are used.

The “Model 2” architecture (see figure 1 on page 8) tries to separate the controller and view by using Java Servlets for the controller and only JSP for the view, the model is still implemented as JavaBean.

One framework that uses the Model 2 architecture is Apache Struts [13], but

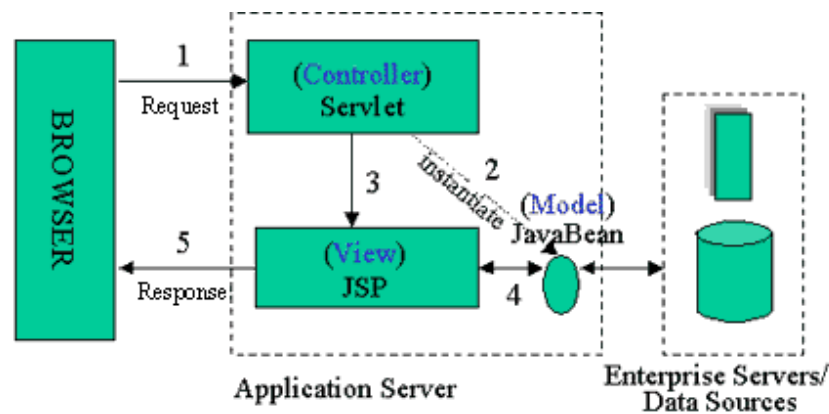


Figure 1: Java Model 2 approach

also frameworks implemented in other programming languages like Struts4PHP [14].

An in detail description of the Model 2 approach can be found at [12]

Frameworks based on XLST and XML / Model 2X

These frameworks highly make usage of XML - XML files are transformed into a final output format by using an XLST stylesheet. When using several different XLST sheets the output can be generated in various different formats and layouts (e.g. for printing, for display on screen, as PDF document, as HTML document,), always using the same XML content as input.

Apache Cocoon ([15], [16], [17]) uses a pipelining mechanism, where a XML file and its content can be processed and transformed into another XML structure which is pipelined into another transformer, so in every step a transformation of the content may be done by another processor, for example one processor for reading files, one for accessing a database, one for adding data from a webservice and so on. Figure 2 on page 9 shows an example for processing XML data by using a Cocoon pipeline [18].

For a short overview of Apache Cocoon also see my paper published in the Linux Magazine [19]. Frameworks with similar concepts are:

- StrutsCX [20], MyXML [21] (Java)
- Apache AxKit [22] (PERL)

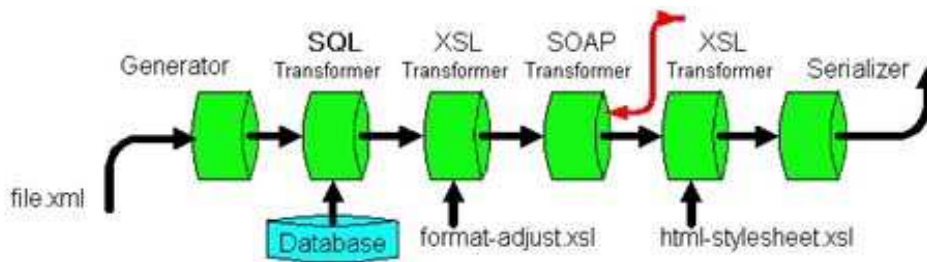


Figure 2: Cocoon XML pipelining mechanism

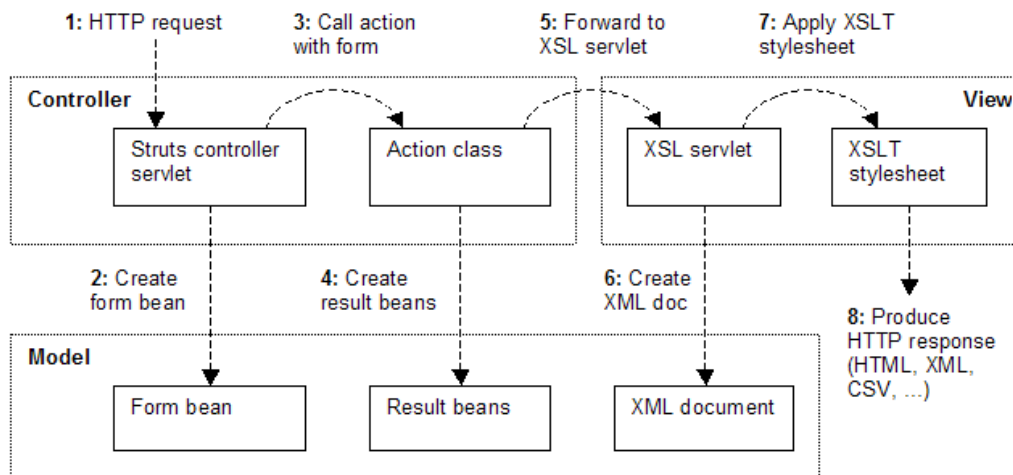


Figure 3: Model 2 X approach using XLST instead of JSP

The “Model 2X” approach [23] tries to replace the JSP from the “Model 2” approach described above by XLST, as implemented in StrutsCX [20] for example (see figure 3 on page 9). The main advantage is to have also business and presentation logic separated from each other.

Java Server Faces and related

JavaServer Faces (JSF) [24] is a framework for the development of user interfaces for web applications based on Servlets and Java Server Pages (JSP). JSF uses a component-based approach where the state of UI components is saved when the client requests a new page and then is restored when the request is returned. By default JSF uses JavaServer Pages for the view but can also make use of other technologies like XUL. Pages are not written in

HTML as with JSP but as higher level components instead.

Based on JSF there are libraries which offer a set of reusable components like drop down menus, calendars, registers, trees, pageable sortable tables, form validation, WYSIWYG-Editor, like Apache MyFaces (Trinidad, Tabago, Tomahawk, ...) [25]

Ruby on Rails and related

Rails [26] is an open source framework implemented in Ruby for rapid development of database-backed web applications according to the Model-View-Control pattern, which is reached by using concepts and patterns like “Don’t Repeat Yourself” (DRY), “convention over configuration” and “scaffolding”.

Convention over configuration means an end to verbose XML configuration files - there aren’t any in Rails. Instead of configuration files, a Rails application uses a few simple programming conventions that allow it to figure out everything through reflection and discovery. This is used for the object-relational (OR) mapping for example - the name of a table should be the same as the name of the object, so it only has to be defined once (don’t repeat yourself), Rails can retrieve this information from the database.

Rails also provides a mechanism called “scaffolding” which means that it can automatically construct some of the models and views needed for a basic website for creating, reading, updating, and deletion (CRUD) of database entries.

Similar frameworks to Ruby on Rails are:

- Grails [27] (implemented in Groovy, which is a scripting language based on Java)
- Symfony [28], CakePHP [29] (PHP)
- Catalyst [30] (PERL)
- Grok [31] (Python / Zope)

2.3.2 Presentation Frameworks

(X)HTML/CSS frameworks

(X)HTML/CSS frameworks provide templates for the implementation of the layout of a website or web application, based on W3C web standards and well tested with several common browsers. “Yet Another Multicolumn Layout” (YAML) [34] and Mollio [35] are examples for such frameworks.

Frameworks for DHTML, AJAX and JavaScript

There exist several frameworks for easier development of client side scripting. The functionality they offer may be support for drag and drop, form validation, auto completion search forms, animation, and so on.

Prototype [36] is a JavaScript Framework that aims to ease development of dynamic web applications, featuring a unique, easy-to-use toolkit for class-driven development and a AJAX library.

script.aculo.us [37] provides an easy-to-use, cross-browser user interface JavaScript library for animation, drag and drop, AJAX controls, DOM utilities and unit testing, as an an add-on to the Prototype framework.

jsVal [39] is a JavaScript program used for *validating HTML Forms on the client side*. This allows HTML forms to be validated in the browser without having to send a request to the server. For security reasons, input validation always has be done on the server side in order to prevent any kind of injection through unverified and unsanitized user input, validation at client side should only be used as a kind of “pre-validation” to give the user immediate feedback. On problem of this approach is that the form validation has be maintained on client side and on server side, thus doubling the effort for maintenance.

2.4 Web Content Management Systems / Wikis

Content Management Systems (CMS) are used to create, edit, manage, and publish content on the web. Features they offer may be access control, versioning, support for workflow in order to allow controlling, revision and release for the public, ability for full text searches, a repository for several different media types like images, audio and video files, PDF and Office documents, ability to enhance the functionality via plugins or extension, and so on.

They may be directly implemented in one of the common programming lan-

guages, or be based on a template engine or one or more of the frameworks described above.

There are hundreds or even thousands of content management systems around, either commercial or free open source, implemented in any of the common languages for the web, with different sets of features.

The possibility for comparison of CMS gives [32], a list of common CMS gives [33].

The reason why there are so many of them may be that every CMS is optimized for a special purpose or customer, so it may be hard to find a system which also fits the needs for the current project (e.g. the informational structure may only support two levels in the hierarchy, the layout is fixed, the business logic can not be extended, ...). On the other hand even if there is a CMS which fits all your needs then it may have a lot of functionality you don't need for your project, so the software may be very complex and hard to learn how to operate and use it. In addition because there are so many content management systems it takes a long time to evaluate which one fits your needs and once you have selected one to learn the administration and usage of it - therefore CMS are often implemented from scratch for a new project.

Wikis (from the Hawaiian word for "fast") are similar to Content Management Systems, but usually don't have that rich set of features. Therefore they are often used for intranets and as Knowledge Management systems, but usually not for websites and web applications.

Some well known open source content management systems are:

- Typo3, Mambo / Joomla, Drupal, ez Publish, Midgard (PHP)
- Alfresco (Java)
- Apache Lenya, Daisy (Apache Cocoon / Java)
- Plone (Zope / Python)
- Bricolage (PERL)

2.5 Rich Internet Applications

One of the latest trends are frameworks and technologies to develop “rich internet applications” (RIA), which should offer the features and functionality of traditional desktop applications. The advantage of this kind of applications is that there should be no or only little need for local installation but having the same rich feature set as traditional desktop applications. In addition, some of this frameworks allow the applications to be run offline. Some of this frameworks are around for some years, but most are quite new, so their APIs are subject to change.

Frameworks for RIA are:

- Google Gears
- Adobe AIR , Adobe Flex
- Microsoft Silverlight (formerly WPF/E)
- Sun JavaFX
- Mozilla Prism, Mozilla WebRunner
- Java Web Start
- OpenLaszlo [38]

2.6 Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) approach supports the automatic generation of code and other artefacts from a descriptive, platform independent model (PIM) using an appropriate domain specific language, like the Unified Modeling Language (UML) or the Web Modelling Language (WebML), in order to improve the quality of software and to reduce the development effort and time.

On one hand the model is much more readable and therefore analyzable then the source code itself and therefore may be also used as part of the documentation or to communicate with the customer since it may be also understood by non technical persons by using graphics. Therefore the functional expertise is separated from the technical expertise.

On the other hand it may allow to generate code for different environments (platforms, frameworks and programming languages), therefore supporting

portability and reusability by using the model as an additional abstraction layer ([42], pp. 16).

One of the disadvantages is that this approach requires an high degree of formalization of the specification in order to be able to generate code automatically, which is not always easy to obtain, since the specification usually is written in a natural language which is not precise and may be ambiguous. Therefore usually not the hole code can be generated automatically - at least not with reasonable effort, since an additional generator, sometimes called cartridge, has to be implemented, which may not make much sense from an economical point of view.

Depending on the used MDA tool the portion of automatically generated code may be between 40 and 80 percent (see [41], page 173). Usually only repetitive code (like definitions of classes, methods and interfaces, get- and set-methods, methods for database access, ...) is generated automatically, where the business logic or at least parts of it (depending on the MDA tool) may have to be implemented manually. Therefore it is important to distinct between automated generated artefacts and manual artefacts ([40], pp. 10). There are different approaches to handle this - either the manual and generated code may be completley separate or they may be mixed. In the second case either the manual code is integrated in the generated code by using added protected regions and user sections in the generated code. Or the generated code may be integrated into the manual code by using include and import mechanisms or by inheritance, depending on the artefacts (for example HTML code or Java code) and the possibilities the related frameworks and technologies offer.

MDA was launched by the Object Management Group (OMG) which provides specifications, standards and guidelines, for example the MDA Guide [43].

AndroMDA [45] is an Open Source MDA Generator but only supports Java, openArchitectureWare [46] is a powerful open source generator framework that can read any kind of model (XMI, XML, any textual representation) and transform it in any kind of textual output. It has an explicit representation of the source metamodel, and uses templates to generate the output. The target languages include C, C++, C#, Java, Perl, Python, Ruby, Javascript, XSLT, JSP, PHP, ASP.NET, VB.NET and much more.

3 Definitions

According to ISO 9126 [47] there are six quality characteristics in software engineering, where each of them includes several subcharacteristics:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

3.1 Maintainability

The ISO 9126 standard defines “Maintainability” as follows:

A set of attributes that bear on the effort needed to make specified modifications. (ISO 9126: 1991, 4.5)

Modifications may include corrections, improvements or adaptation of software to changes in environment, and in requirements and functional specifications. (ISO 9126: 1991, 4.5)

Subcharacteristics for Maintainability are:

- **Analysability:**
Attributes of software that bear on the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified. (ISO 9126: 1991, A.2.5.1)

Examples of techniques which improve the analysability are the use of well known design patterns, coding standards, use of comments, providing up to date documentation and Refactoring.

- **Changeability:**
Attributes of software that bear on the effort needed for modification, fault removal or for environmental change. (ISO 9126: 1991, A.2.5.2)

The environmental change may concern the software environment (for example the operating system, the programming language, as well as the hardware environment. Thus this may also imply design for Portability and Scalability.

- **Stability:**
Attributes of software that bear on the risk of unexpected effect of modifications. (ISO 9126: 1991, A.2.5.3)

For example to support this characteristic every unit and module should have one single and well documented purpose and no unexpected side effects. In addition, changes should be necessary in only one single location. If changes have to be done on different locations it may be forgotten to do the modification in every place.

- **Testability:**
Attributes of software that bear on the effort needed for validating the modified software. (ISO 9126: 1991, A.2.5.4)

For example, when splitting the code in small units automatic unit testing can be used, which reduces the manual test effort after modifications.

IEEE 90 [48] defines “Maintainability” as “...the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment...”.

Maintainability in software engineering is one of the non functional requirements (also called qualities, quality attributes, quality goals or quality of service requirements), which specify criteria that can be used to judge the operation of a system, rather than specific behaviors as functional requirements do. For this reason it is often hard to communicate the need for this requirements to the customer and to explain and to justify the additional initial costs.

Since a website not only consists of documents and web applications, but also a hardware and software environment and a lot of other components, I would like to mainly focus on maintainability according to the above definitions, but also keep portability and scalability in mind when discussing

about maintainability for websites. In my opinion the three qualities portability, scalability and maintainability are strongly related, since scalability could mean to be able to handle a higher number of levels in the navigation, a higher number of users or a higher number of documents or entries in the databases. If a system or architecture scales out well than it is also maintainable, since the architecture does not have to be changed when being confronted with an increase in any of these items.

Regarding to websites following components may be subject to change (among other things):

- content
- design / layout
- informational structure
- changes in software requirements, new or additional functionality
- traffic and load generated by users
- hardware environment
- software environment (operating system, frameworks, template engine, programming language, database engine, web server, ...)

3.2 Portability

According to to ISO 9126 [47] “Portability” is defined as follows:

A set of attributes that bear on the ability of software to be transferred from one environment to another. (ISO 9126: 1991, 4.6) The environment may include organizational, hardware or software environment. (ISO 9126: 1991, 4.6)

3.3 Scalability

Scalability is the ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. Regarding to websites the changes concerning scalability may not only be in the load and traffic generated by users and therefore performance of the web server(s), but also in the informational structure, namespaces, and so on.

From point of view of the hardware, it can be differentiated between horizontal and vertical scalability:

3.3.1 Horizontal Scalability

Horizontal Scalability means to have only one system which is getting bigger and bigger as more load is produced by users. The advantage is that one system is easier to maintain than several systems, because no shared storage is necessary, no data or programs have to be synchronized or distributed. The disadvantage is, that it usually scales only to a certain measure, at some point the costs for additional performance gain are disproportional - at this point it may be better to use vertical scaling instead if possible. In addition, having only one single system is a single point of failure.

3.3.2 Vertical Scalability

If a website and the web applications are prepared for vertical scalability it is easy to add additional computers or nodes if the load generated by users grows. Another advantage is to gain fault tolerance by redundant servers (as long as there is no single point of failure in the concept, e.g. one shared storage). The disadvantage of the schema is, that this concept usually needs some kind of shared storage, data distribution or data synchronization in is therefore harder to maintain. Also some load balancing mechanism (round robin DNS, explicit load balancing hardware) is necessary, which ideally should not be designed as a single point of failure.

3.4 Reusability

Reusability is the likelihood a segment of source code can be used again to add new functionalities with slight or no modification. Reusable program components reduce implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required. [51]

Examples of design features and patterns for software reuse are parameterization, implementing generic code, modularity, support for internationalization and localization and use of Service Oriented Architecture (SOA).

3.5 Usability

Usability is a quality attribute that assesses how easy user interfaces are to use. The word “usability” also refers to methods for improving ease-of-use

during the design process.

Usability is defined by five quality components:

- **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design?
- **Efficiency:** Once users have learned the design, how quickly can they perform tasks?
- **Memorability:** When users return to the design after a period of not using it, how easily can they reestablish proficiency?
- **Errors:** How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- **Satisfaction:** How pleasant is it to use the design?

On the Web, usability is a necessary condition for survival. If a website is difficult to use, people leave, if users cannot find the product on the site, they cannot buy it either.

Usability can be improved by the use of usability guidelines from earlier studies in other projects or published research, testing users or by analyzing the log files of the website, doing “web usage mining”. [49], [50]

3.6 Accessibility

Accessibility refers to the practice of making websites usable by people of all abilities and disabilities. When sites are correctly designed, developed and edited, all users can have equal access to information and functionality.

For example, when a site is coded with semantically meaningful HTML, with textual equivalents provided for images and with links named meaningfully, this helps blind users using text-to-speech software and/or text-to-Braille hardware. When text and images are large and/or enlargeable, it is easier for users with poor sight to read and understand the content. When links are underlined (or otherwise differentiated) as well as colored, this ensures that color blind users will be able to notice them. When clickable links and areas are large, this helps users who cannot control a mouse with precision. When pages are coded so that users can navigate by means of the keyboard alone, or a single switch access device alone, this helps users who cannot use a mouse or even a standard keyboard. When videos are closed captioned

or a sign language version is available, deaf and hard of hearing users can understand the video. When flashing effects are avoided or made optional, users prone to seizures caused by these effects are not put at risk. And when content is written in plain language and illustrated with instructional diagrams and animations, users with dyslexia and learning difficulties are better able to understand the content.

When sites are correctly built and maintained, all of these users can be accommodated while not impacting on the usability of the site for non-disabled users. [52]

The Web Accessibility Initiative [53], a project by the World Wide Web Consortium (W3C) [54], published the Web Content Accessibility Guidelines (WCAG), so websites can be validated against this guidelines.

4 Development Life Cycle

In this chapter I would like to present common tasks in the website development life cycle, what should be paid attention on if a website should be built for maintainability and how this could be achieved.

First of all the Informational Architecture has to be defined, that mainly is which content should be presented in which way and how it should be organized. Afterwards the layout and templates will be created for each document type. After that, technologies and frameworks have to be chosen which support the required functionality so developers can start with the implementation. After testing the website and applications the website will be deployed and launched to a live system, where it has to be monitored, in order to improve the service and correct errors. Finally, after the go live there will be changes in the requirements, so the whole process starts again.

In the thesis I would like to focus on the maintainability aspects of each of these phases and which things to keep in mind in every of these phases in order to improve the maintainability of the website.

4.1 Concepts and Patterns

Whenever possible and applicable following concepts should be taken into account in order to improve the maintainability of a website. Some of them (for example DRY, Automation) are relevant for the whole development life cycle while others only may concern the coding itself.

- Usage of a well documented and scalable architecture, based on well known **Design Patterns**. Design patterns are general reusable solutions to a commonly occurring problem in software design and can speed up the development process by providing tested, proven development paradigms. Reusing design patterns helps to prevent subtle issues that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns. Design patterns gained popularity after the publication of the book “Design Patterns. Elements of Reusable Object-Oriented Software” [55] by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, which are often referred to as the GoF or Gang of Four. An example for a well known design pattern which improves the maintainability of websites is the Model-View-Controller pattern (MVC).

- **Separation of Concerns (SoC)**: in case of web site development this may be achieved by the usage of the **Model-View-Controller (MVC)** design pattern, and separation of content, presentation logic and business logic from each other, so if one of the elements has to be changed, the others should not be affected. Most current frameworks for web application development support this design pattern. In addition, SoC in web site development may also be achieved by separating the style from the content by using **stylesheets**.
- **Modularity**, that is, splitting code into small, easy to test modules with well defined interfaces, or the use of pipelining for processing data in different independent steps.
- **DRY - Don't repeat yourself** (e.g. found in Rails, also known as Single Point of Truth) is a concept trying to reduce duplication. The philosophy emphasizes that information should not be duplicated, because duplication increases the difficulty of change, may decrease clarity, and leads to opportunities for inconsistency. DRY is a core principle of Andy Hunt and Dave Thomas book "The Pragmatic Programmer" [56]. They apply it quite broadly to include database schemas, test plans, the build system, even documentation. When the DRY principle is applied successfully, a modification of any single element of a system does not change other logically-unrelated elements. Additionally, elements that are logically related all change predictably and uniformly, and are thus kept in sync. [59]. If you are about to duplicate some information you should change your architecture or coding, in order to be able to reuse this information on both places you need it. If this is not possible (e.g. because the information is required in another file format) at least only one single place should exist where the information is maintained, the information used on a second place should be at least generated automatically, so you can't forget about the different places where the information is used.
- The **Model Driven Architecture (MDA)** approach supports the automatic generation of code from a descriptive model, usually defined in the Unified Modeling Language (UML). On the one hand the model is much more readable and therefore analyzable and may be also used as part of the documentation to communicate with the customer since it may be also understood by non technical persons by using graphics, on the other hand it may allow to generate code for different platforms and language, therefore supporting portability.

- **Reuse of Code**, that is the use of existing and well tested software, e.g. modules, libraries or frameworks. Some characteristics that make software more easily reusable are modularity, loose coupling, high cohesion, information hiding and separation of concerns. For newly written code to use a piece of existing code, some kind of interface, or means of communication, must have been defined.
- **Convention over Configuration** (e.g. found in Rails). Instead of the extensive usage of XML configuration files, Rails application uses a few simple programming conventions that allow it to figure out everything through reflection and discovery. This is used for the object-relational (OR) mapping for example - the name of a table should be the same as the name of the object, so it only has to be defined once (don't repeat yourself), Rails can retrieve this information from the database.
- **Scaffolding** is a technique supported by some model-view-controller frameworks (e.g. Rails and related), in which the programmer may write a specification that describes how the application database may be used. The compiler uses this specification to generate code that the application can use to create, read, update and delete (CRUD) database entries, effectively treating the template as a scaffold on which to build a more powerful application.
- **Refactoring** of source code means modifying it without changing its behavior (sometimes also referred to as "cleaning it up"). Refactoring neither fixes bugs nor adds new functionality. Rather it improves the understandability (and therefore maintainability) of the code and changes its internal structure and design, and removes dead code, to make it easier to comprehend, more maintainable and amenable to change. [58]. To support refactoring automatic testing should be established in order to be able to check if the behavior really has not been changed by refactoring. (also see Martin Fowler et al. [57])
- **Pipelining** (e.g. found in Apache Cocoon, the Unix-Shell, ...), that is, processing the content of an file with one processor, convert it to another format (e.g. using XSLT when the data is in XML style), and piping the output of this conversion into another processor, having the possibility to combine several different processors.
- **Automation**. In order to improve maintainability, as many tasks as possible should be automated - automation is one of the key factors to achieve a maintainable website. Tasks that can be automated can

be (among others): testing (e.g. module/unit testing, checking links or mail addresses, ...) generation of code (e.g. Scaffolding) and other artefacts, e.g. images, documents in various file formats (PDF, HTML, XML, ..), static HTML files for performance reasons, and so on. According to the DRY pattern everything should be generated from one single source.

- **Documentation.** Use of documentation standards and maintaining an up to date documentation, which is especially important for interfaces. According to DRY and Automation as much of the documentation should be generated automatically as possible, e.g. from special tags and comments within the source code. After changes to the source code or other components the documentation has to be updated. Documentation also should styleguides for the layout and a description of the architecture in order to prevent the architecture from deterioration when people who built the site leave the project.
- **Coding Standards,** that is, use of coding standards and development guidelines, defining namespaces, usage of useful comments, in order to create readable and consistent code. Namespaces are also important for the addresses of the documents (uniform resource locators - URLs) since they should not be changed for usability reasons after launching the website.
- **Monitoring,** that is. usage of utilities and tools for diagnosis of the operative live system, e.g. monitoring the logfiles and system performance and creating an automatic notification in case of unusual events, high system load and possible security incidents for example. Also these tools can be used for web usage mining to improve stability and usability and to find problems and program errors.

4.2 Informational Architecture

According to Rosenfeld and Morville ([60], page 4) the information architecture is defined as follows:

- The combination of organization, labeling, and navigation schemes within an information system.
- The structural design of an information space to facilitate task completion and intuitive access to content.

- The art and science of structuring and classifying web sites and intranets to help people find and manage information.
- An emerging discipline and community of practice focused on bringing principles of design and architecture to the digital landscape.

The following issues try to focus on the maintainability aspects of the informational architecture, which are independent of the programming language, template engine or any other framework used for the website.

Reuse of informational components

A typical web page consists of several different content areas ([61], Web page anatomy, page 4), like

- Header, usually consisting of the logo of the company or organization with a link to the home page and a search form
- Navigation areas (local or global)
- Content area
- Footer
- Areas for advertising

Some of them are repeated on every web page of a website or web application (like the header or the footer), some of them are unique for every page (like the content), some are repeated on all pages of a single section. For example the global navigation will be repeated on every page, containing the main navigational elements where the local navigation menu is only repeated on every page of a subsection of the website. When designing the layout for a web page these different areas have to be identified in order to split them up in different files, so they can be reused on different pages.

Frames vs. Non-Frames.

In earlier days of web development there was a highly usage of HTML frames to implement such repetitive areas like navigation. The advantage of using frames is the ease in development by just adding a few HTML tags, but there are several disadvantages regarding usability using frames. e.g. the content of the frames can hardly be bookmarked and printed, when entering a website

from a search engine the user may only see the content of one frame, but not the surrounding frameset and the contents in the other frames, the buttons for navigation in the browser (back / forward) don't work as expected by the user and so on (also see Jakob Nielsen, Why Frames Suck (Most of the Time) [62]).

For this reasons frames will not available anymore in the newer HTML standards (e.g. HTML 5) from the W3C.

Due to the usability problems with frames other methods and technologies have to be established to support the reuse of web page areas, like the functionality of most template engines to include content from other separate files.

Scalable informational architecture

In some projects I have seen layouts which only support an information structure of two levels, there has not been the possibility to add a third or even more levels. In order to be scalable (and therefore extensible and maintainable) the information architecture and the layout should support some kind of **hierarchical navigation**. For example the global navigation menu provides the first level of the hierarchy, where the local navigation menu provides all further levels. Since the content for the local navigation menu changes for every level, there should be added some **breadcrumb navigation**, so the users sees the full path to the current local navigation menu and can easily figure out where he currently is located in the hierarchy and does not get lost. From point of usability the width of the page should be constant, so the user does not have to scroll to the right, therefore the global navigation may be aligned horizontal, because it should not change (much) over time and contain only a few elements. Since the number of elements in the local navigation may highly vary it would be better to align it vertically, so the user only has to scroll down, but not to the right.

In the navigation the link to the current page should be disabled (because it does not make sense to link to the same page you currently are on) but rather should be highlighted instead, in order to show the user where he currently is located. Ideally the framework should support a way where this can be done automatically based on the address of the current document.

Since the addresses of the documents (URLs) should not be changed for

usability reasons once they have been established it is important to create a guideline for the **namespace for documents** and the different document types. If a URL has to be changed after going live for good reasons at least an redirect to the new URL should be configured, so users get to the new address automatically without interruption. Also other websites, search engines may link to your website, so they would have to update their links when changing addresses on your site. In addition, content should not disappear from a website, it only should be “archived” but still available on the web with the same address. Therefore it is important to have some timestamps on every page, like a **created date and a last modified date**, so the users can estimate how appropriate and up-to-date the content is or if it may be obsolete [63].

Users often don't want to learn the navigational structure of the website since this may be time consuming (at least if the site has a bad structure or bad labelling of the navigational items, or it is only a one time visit). For this reason many sites offer a **search form** to allow users a full text search in the hole website or only in some parts of the site or in special databases (e.g. a knowledge database, user forum, ...) or only even in the meta information like title, author, summary, and so on. For performance reasons these search engines use an index, which has been built automatically in advance and contains a list of all words and the documents in which they appear, every time the website is updated or at least on a regular base. In general it is time consuming to build such a search tool including the program to generate the index for full text search for different document types like HTML, MS Word or PDF. Therefore many search engines like Swish-E [65], ht:Dig [66] or Perfect [67] offer a template system which can be used to integrate these search engines into the website. Good search engines also offer some kind of error tolerant approaches in order to support the user when misspelling the search term.

Some sites also offer a **Sitemap or Table of Contents (TOC)** to get an quick overview of the structure of the website. Some also offer an **Index** with the most important keywords, so the user gets used to the wording the organization uses and to search by this keywords. This pages should be generated automatically if possible in order to be up-to-date and to reduce the effort for maintenance, although my experience in my projects has shown that this approach is also quite hard to maintain if not all starting pages (e.g. one for each subdirectory) should show up on the sitemap, so you would have to maintain a blacklist for pages that should not be included.

To improve the visibility of the website to the most important search engines like Google, Yahoo and MSN in order to increase the traffic to the website (**Search Engine Optimization, SEO**, in order to acquire new customers or to increase the volume of the ad impressions) they offer the possibility to upload a sitemap to these search engines including all document addresses of the website. These sitemaps should be generated as well automatically after an update and submitted to the search engine [64]. For SEO also the type of document addresses are important - in general addresses should look like static addresses instead of having a lot of dynamic parameters added to the URL, even if the page is generated on the fly based on these parameters a rewrite should be used to show static addresses to the browser respectively robot.

When designing the information structure it should be also **prepared for web usage mining** in order to improve the usability; the namespace of the documents has an affect on the **possibility to analyze the logfiles**. In addition, the address of the documents is analyzed by search engines, URLs that match the keyword the user has searched for will get an better ranking and therefore show up earlier in the search results.

For indexing by the own search engine or other foreign search engines some **meta information** has to be added on every web page, like title, keywords, language, author. This meta information will be analyzed by the search robots and documents which match the search term in the meta information will be ranked higher.

The content should be **split up into smaller chunks of information** instead of having all information on one single big page. This increases the download speed and decreases the time the user has to wait to get the information displayed on the screen, especially if the document embeds other documents, like images. In addition the user does not have to scroll down (at least not that much). For example an page to get an overview of the contents of an archive may consist of one page per year (or month, week,... depending on the amount of articles per time unit) instead of one page for all articles or issues. Using smaller chunks of information scales well, while using one single big file does not scale well. When splitting up the content usually some additional intermediate navigational pages or pages to get an overview of the content have to be added. Also, smaller chunks improve the possibilities for web usage mining, because you know more exact and in greater detail in which pice of the information the user has been interested in. Smaller chunks are also more easy to read than one big page.

Multi-lingual websites

Some websites have to be presented in different languages (e.g. English and German) to the user or to support regional differences, e.g. in the currency or due to different legal requirements to support and reach more users from different markets. Therefore the website has to support **internationalization** (often referred to as “i18n”) so it can be adapted to various languages and regions without engineering changes and **localization** (“L10n” or “l10n”) which is the process adapting software for a specific region or language by adding locale-specific components and translating text. While translating existing text to other languages may seem easy, it is more difficult to maintain the parallel versions of texts throughout the life of the product. For instance, if a message displayed to the user is modified, all of the translated versions must be changed. This in turn results in somewhat longer development cycle [68].

One way to handle different languages for a website is by the use of Content Negotiation [69] defined in RFC2616 section 12 [71]. Using this mechanism the user can configure his browser to tell the web server which are his preferred languages (including their priorities) and the server tries to reply according to that requests. Using the Apache web server one has to put a HTML file for each language for each document in the web tree, for example “index.de.html” and “index.en.html” in order to support the German and English language. [70]

Support for different end user devices or document formats

One and the same content sometimes should be presented to different end user devices, e.g. the browser, the mobile phone, PDA, television screen,... or in different document layouts or formats for different purposes, e.g. as HTML document for the browser, as **printable document** without navigation and advertising, as PDF document, as WML document for the mobile and so on. Ideally, all this different kinds of final document formats should be generated from one single source of the content, in order to maintain only this one single source to reduce the effort for maintenance and to avoid inconsistencies. Therefore the framework should support the generation of different kinds of final document formats from one single source.

4.3 Realization of the Layout

When the information structure has been defined usually a graphic designer has to create a layout, one layout each for the most important document types of the site. Therefore he or she creates “comp”s (an abbreviation of the phrase **comprehensive dummy**, which is a term that comes from the print design world and is a complete simulation of the printed layout before it goes to press), which are images of a layout that are created before beginning to prototype the design in HTML. Comps usually are created by using graphic programs ([61], page 2).

Graphics vs. Text

One common pitfall which may decrease the maintainability of the layout is the usage of graphics instead of textual elements for navigation elements and headings. Since only a restricted set of fonts is available on different operating systems and browsers graphic designers often use graphics to keep their preferred font throughout all operation systems. This graphics are generally harder to maintain than simple text elements, because you have to know the way the graphic has been produced, you have to install the fonts on your computer, you need an graphic program (ideally the same as initially has been used), and so on. If you don't know the way how the graphics have been produced you may have to recreate all graphics, otherwise the look of the new graph icon may differ from the old ones. Instead, when adding an element or changing the content of the element it is much more easy when simply using a textual description.

Another problem with graphics for navigational elements or headings is that the text could not be read by a program, like a spider of a search engine or a screen reader, therefore also decreases the accessability. For this reason the HTML standard defines ALT and TITLE tags for images which contain a textual description if it is not possible to show the image to the client, but these tags again increases the maintenance effort, since you have to change HTML tags in addition the the graphic.

One possible workaround if the font is not available but the use of this font is a mandatory requirement (e.g. in order to maintain the corporate identity) can be the automatic generation of the graphic elements, including the HTML tags from one single source (e.g. a repository with all navigational elements).

Stylesheets

In order to separate the content from the style of the presentation the **Cascading Style Sheet (CSS)** standard has been established by the W3C. CSS is a stylesheet language used to describe the presentation of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can be applied to any kind of XML document, including SVG and XUL.

Prior to CSS, nearly all of the presentational attributes of HTML documents were contained within the HTML markup; all font colors, background styles, element alignments, borders and sizes had to be explicitly described, often repeatedly, within the HTML. CSS allows authors to move much of that information to a separate stylesheet resulting in considerably simpler HTML markup.

In the earlier days, when the support for the CSS standard was not well implemented by the different browsers, layout designers used **pixel graphics as placeholder** to get the exact layout they wanted and made highly use of **nested tables** to get the same kind of grid in order to align the different elements on one page. Both lead to HTML code which is hard to read and therefore to maintain. Some design programs and layout designers still make use of nested tables and pixel graphics, which should be avoided.

Another advantage of stylesheets is that it can be used to present the same content to different devices in a different way, e.g. using one stylesheet to display the document to the browser and one stylesheet to present a **printable version** of the same document, but without navigational or advertising elements or another version to present to speech readers to provide an accessible version of the document, for example:

```
<link rel="stylesheet" type="text/css"
      media="all" href="print.css" />
<link rel="stylesheet" type="text/css"
      media="print" href="print.css" />
```

Additionally, when using CSS the style for a whole website can be maintained in one single stylesheet, so when changing the font style or size or even the alignment of content boxes or navigational areas in the stylesheet the change

is applied to all documents which refer to this stylesheet.

The term “cascading” says that it is possible to **inherit stylesheet definitions**, e.g. having one global stylesheet for the hole site, which may be overruled by a local style sheet for a single section, which may be overruled by different style sheets in every single document (although, from point of view of maintainability as much as possible should be defined in a global stylesheet to decrease the number of different styles and to keep the design easy to maintain). [72]

Another technology to present the same content in different ways to different clients or in different document formats is the use of **Extensible Stylesheet Language Transformations (XSLT)**, which is an XML-based language used for the transformation of XML documents into other XML or “human-readable” documents. In theory, this conversion could also be done on the client side by the browser, but in practice this conversion is only done on the server side, due to poor support for XSLT by the browser suppliers.

For example, Apache Cocoon makes highly use of XSLT. By using the Formatting Object Processor (FOP) Cocoon is even able to convert XML documents into binary output documents, like PDF, PS, PCL, and RTF, driven by XSL formatting objects (XSL-FO). In addition, Cocoon supports the mechanism of pipelining, where XML documents can be transformed by different specialized processors in several steps, feeding the output of one processor into the input of another processor.

4.4 Implementation

This thesis mainly references to open source frameworks, tools and programming languages, since they are available for free, usually are well documented, the source code is available, so you can analyze the functionality in detail if you need to or you can modify the source code in order to adopt it to your own needs, although this is not recommended because it would decrease the maintainability, since upgrades to newer releases are harder to handle. If you need some extra functionality for a framework you should instead use the extension mechanism the framework offers or try to incorporate the modification into the standard version by working together with the developers.

The programming languages used are PERL (Practical Extraction and Report Language) and PHP (PHP Hypertext Processor, [73] [74], [75]) because they provided by most web hosters. Also for both languages a lot of mod-

ules are available through the Comprehensive Perl Archive Network (CPAN) respectively the PHP Extension and Application Repository (PEAR). PHP has been created in 1995, while PERL is available since 1987, therefore both languages are quite stable and well tested. In addition both languages are quite well suited for text processing (like the generation of HTML code) due to the integrated support of regular expressions. Development environments for both languages usually come with most Linux distributions, but they are also available for other operating systems.

For the support of templates with PERL the Template Toolkit ([6], [7]) has been chosen which has been implemented in 1996, for PHP no template engine is mandatory, since PHP itself is an template engine, but Smarty [5] would be a good choice, since it offers additional functionality like a fine-grained caching mechanism which allows caching of all or parts of a rendered web page, or leaving parts uncached. The disadvantage of template engines like the Template Toolkit or Smarty are that they use their own syntax, so you have to learn another “programming language” besides from PERL respectively PHP.

4.4.1 Static vs. Dynamic Web Pages

One of the fundamental design decisions in web site development is if a web page has to be static or dynamic.

When a web page is requested by a user clicking a hyperlink or entering a URL, the web server returns the document to the users computer and the browser displays it. On a static web page, this is all that happens. The user only may interact with the document through clicking available links but the document has no capacity to return information that has not already been generated before.

On a dynamic web page the user can make requests (often through a form) for data contained in a database on the server that will be assembled on the fly according to what is requested. For example the user might want to find out information about a theatrical performance, such as theater locations and ticket availability for particular dates. When the user selects these options, the request is relayed to the server using an intermediary script embedded in the page’s HTML. The intermediary tells the server what information to return. Such a web page is said to be dynamic. [77]

There are two methods how dynamic web pages are created. In the first method called the client side scripting the dynamic behaviour is initiated by the keyboard or the mouse. The second method uses the server side scripting where the source of actual page being supplied changes or there is change in the sequence of the pages. The response is decided by conditions as in a posted HTML form, URL parameters, session cookies, types of browser used, passage of time, or state of the database or server. Simultaneous use of both techniques can also be made to create dynamic pages, e.g. for forms it is a good style to validate them both on client and server side - the validation on the client side gives immediate feedback to the user while the validation on the server side is necessary for security reasons in order to ensure that the parameters have not been modified for a break in attempt by parameter injection.

Web pages use two basic types of presentation technologies. For the client side scripting, the web page presentation has to be using the “Rich Interface Pages” technology. Here scripting languages like Javascript and Flash are used. In case of the server side scripting, languages such as PHP, PERL and ASP are used. [78]

The main advantages of static web pages over dynamic web pages are:

- **Optimal performance.** Static pages are very quick to process by the web server, the usage of the central processing unit is very low since only a file from the filesystem (which may be even cached in the memory) has to be read and delivered to the client.
- **Security aspects.** Dynamic web pages allow the input of parameters, which can be modified by the user. If these parameters are not validated properly it is possible to use them for break in attempts to our system by SQL injection, Shell injection, FORM POST hijacking, and so on. With static web pages you don't have to handle parameters, therefore these security risks do not appear, making development and maintenance much more easy.
- **Search engine optimization.** Static web pages are easier to index by search engine robots, since they only have a static address, while dynamic pages may have a lot of parameters where each of them can have a lot of values. Therefore search engines prefer static addresses to addresses with parameters, because with dynamic addresses there is

also the risk to run into an endless loop when indexing, because of the endless possibilities to combine parameters and values.

- **Configuration.** Web servers that provide only static pages are much easier to configure and thus to maintain, since they only serve files from the filesystem, where dynamic web pages usually need an additional programming language and other frameworks, where the web server has to be configured to handle the communication with this additional environments.
- **Caching.** In order to reduce load and traffic and to increase download speed caching mechanisms have been established everywhere - in the browsers, intermediate proxies of the internet providers or companies or prior to the actual web servers as reverse proxy in order to reduce the load to the actual web servers. With static web pages caching is much more easy, while with dynamic web pages usually only some parts of the web page can be cached on the server side, so it hasn't be processed again and again after the first request.
- **Web usage mining** is much more easy with static web pages, since the content the user got is defined by the address of the document which shows up in the log files of the web server. With dynamic pages there may be additional parameters sent from a HTML formular via a HTTP POST request which do not show up in the log files of the server, therefore you may have to implement additional mechanisms to record the requested information. Also, the content sent back to the client may change over time, since the content of the database where the information comes out from may change over time.
- **Platform independence.** Static pages can be copied to any web-space, where dynamic web pages depend on the platform, like programming language, template engine or frameworks. Even an offline version to copy the content on DVD or CD-ROM is no problem with static pages.
- **Load distribution** to different servers of a cluster is easier with static web pages, since no state has to be maintained - each request can be answered by any host of the cluster with static pages while with dynamic pages - especially ones which have to maintain a user session and not only request information but also do updates to a database - it may be important that every time the same host answers the requests for one user session, depending on the architecture of the cluster.

For the above reasons wherever possible static web pages should be preferred to dynamic web pages in order to be scalable by reducing the system load and to be easier to handle and therefore to maintain. Unfortunately it is not always possible to use static web pages, since they don't offer that much possibilities to interact compared to dynamic web pages. Although it is possible to use client side scripting with static web pages it is not possible to use server side scripting. For example it is not possible to access to a database with static web pages to retrieve information or to buy items in an online shop. Also the handling of user sessions for personalization or user tracking is not possible. The additional functionality of course comes with additional complexity, which makes web sites harder to maintain.

In order to take advantage of both concepts dynamic content can be integrated into static web pages using AJAX (“Asynchronous JavaScript and XML”) or inline frames, which may be used to integrate rotating advertising banners for example.

Some bigger websites also divide their web servers or clusters into servers respectively clusters for static and for dynamic delivery, so every cluster could be optimized for each purpose.

If the content of a website or at least some areas like an online archive changes only very rarely it may be a good idea to *generate static web pages* or documents, where the original documents may be based on template engines or dynamic web pages. This may happen offline in order to decrease the system load on the productive system, after the static pages have been generated they can be transferred to the live system. Some frameworks (like Apache Cocoon) also offer the possibility to generate an offline copy of the pages, another solution may be to create a static mirror of the site with tools like “wget” (a free software package for retrieving files using HTTP, HTTPS and FTP, which has many features to make retrieving large files or mirroring entire web or FTP sites easy) [79].

4.4.2 Architecture for Static Generation

An architecture for static generation of a website could look like the following. This approach also has been used in the case study described in the next chapter.

For each single template or document type following steps are executed:

- Since every document of one type or all documents for one section have to be generated, a wrapper is necessary which selects all document identifiers and feeds them into the program to generate all documents of one type based on the template.
- For each document the business logic is used to get all the data needed for that document from the database. The business logic returns one or more data objects, which are passed over to the presentation logic in the next step.
- The presentation logic (mainly consisting of loops and handling of conditions) takes the data objects and creates the output document, which can either be based on a template (which usually has its own syntax but may be more ease to learn for web designers) or be directly implemented in the presentation logic (which usually is more easy, since the same programming language can be used, but may be harder to maintain for web designers which are not used to that language). In this step mainly the content is generated, while the navigation, headers, footers, advertising, and so on are not yet included - only the definition, which elements to use and the parameters they need are defined in this step, the inclusion itself will be done in the next global step, in order to be able to provide context sensitive navigation menus and to check for the existence of other files.

The implementation of this approach could be also used to present the documents as dynamic web pages with small modifications: the content type has to be returned to the browser, which is not necessary when creating static documents and the way the input parameters are handled have to be changed from parsing input parameters from the shell to parsing CGI parameters. In this case also the first step is omitted.

The common general part, which is independent of the document class or template processes all the documents generated in the steps above (the hole website or the sections of the website which a generated):

- In this step the documents created in the previous step are processed by the template engine, that is to include the templates which previously only have been referenced. Since in the local navigation the link for the current page should be disabled and highlighted in this step it is checked

if the current page is the same as the one referenced by one of the navigational elements. Also, if one navigational “master” template is used for several similar subsections where some items in the navigation may not show on every of the subsections, a check for existing can be integrated in this step - therefore all pages of one subsection have to be created at once in order to be able to check the existence of the other navigational elements.

- Since now the page is complete, all pages can be processed with a program to clean up the markup (e.g. HTML Tidy [82]). This guarantees that all tags are well formed, and the generated HTML source code is readable (and therefore easier to analyze and maintain) due to consequent indentation. This step could be also used to reduce whitespaces in the output in order to improve the download speed for the client (while this also could be done by compressing the document sent to the client by using the Apache module `mod_gzip`, for example).
- In the next step a validation of the generated HTML code can be done, that is to ensure that is in conformance with the standards defined by the World Wide Web Consortium (W3C). This step has not to be done for every generation, it should be sufficient if its done after the implementation of a new document type. Valid HTML code according to the W3C improves the accessibility since screen readers are supported better for example, but also make it easier for any other program (e.g. spiders) to parse the web pages.
- In this step the generated web pages should be checked for broken links, errors in this step as well as in the steps before should be reported into a log file for the generation process.
- If the steps before have been successful then the generated pages can be moved from the temporary location to the final location, that is deployed to the productive system.

4.4.3 Repository for Media - Filesystem vs. Database

Actually there is an ongoing debate about whether to store media in the database or whether to store media on the filesystem and just store the reference in the database, see for example [81].

In general I would use the filesystem as repository for documents (PDF, images, audio and video files, ...) and only the database for metadata since

both are optimized for this two purposes: a database is designed to store data to be searched and retrieved - it would be a rare case when you send a query to a database that consists of an image blob, that is search for an image that matches certain binary data.

This approach offers also the advantage to be able to deliver the files statically from the filesystem by the web server, while with storing media in the database every page has to be created dynamically on request. Therefore this approach benefits from all advantages of static web pages. In addition, it is more easy to handle documents in a filesystem, for example in shell scripts, since your are able to use standard tools for automatic generation and processing of PDF documents and images, while when storing them in a database you would have to implement wrappers which handle the management of the files in the database.

Main arguments for using a database to store binary data are to have one single point and mechanism for backups and data replication, as well as the support for consistency - if you store the binary data in the filesystem and only the reference in the database you have to check if the file really exists - otherwise you will run into broken links.

To go even further in a project I realized for the Austrian Mountain Rescue Service I even used the filesystem as repository for metadata, a database has not been used at all. Every article of the content management system has been stored in one directory, where each article could have any number of additional documents attached to the article, as well as any number of subdirectories. The hierarchy of the articles and the navigation structure is built automatically from the hierarchy of the articles. Each article respectively directory has one file for metadata as well as one for comments and one to store the order in which the subdirectories should show up on the website (which can be changed by the content management system by a single mouse click - articles can be moved up and down in the hierarchy as well as the sort order within one hierarchy level can be changed by clicking on different arrows). Every single article can be easily locked by creating a lockfile in that directory if someone currently is changing the article, in order to ensure that another person is not able to modify that article at the same time and therefore overwrites the changes of the other person. The advantage of this approach is consistency, since you have only one single location for each article and the easy handling, since you can easily move around every article in the filesystem and therefore hierarchy. If an article has been expired or deleted you can easily move the directory into an archive outside

from the web tree, you can restore a single article easily. This approach also allows to have different versions of one article, where one version may be already released, while another version has to be reviewed and replaces the other version after release. All articles are stored outside the webtree and are delivered to the client via a wrapper script, which allows to check the permissions for each document and user. To provide static URLs the Apache module `mod_rewrite` is used.

4.4.4 Business Logic

The business logic consists of the functional algorithms which handle information exchange between a database and a user interface, that is, the business logic delivers to and gets its data from the presentation logic, which validates the input and presents the data to the user. The presentation logic may be implemented on the client side (e.g. with JavaScript, DHTML, AJAX, RIA, ...) or on the server side, included in (HTML) templates.

Coding and documentation standards

Coding standards or coding conventions are a set of rules or guidelines used when writing the source code for a computer program. Especially when working in a team following a particular programming style will help programmers quickly to read and understand source code conforming to the style as well as helping to avoid introducing faults.

The standard include rules for indentation, that is the whitespaces to structure control blocks and blocks of code, vertical alignment, spaces and tabs, usage of appropriate names for variables, functions or methods, classes and files, using names in upper or lower case or mixed (CamelCase), usage of underscore for names, rules for comments and documentation and so on. One example are the GNU coding standards [80].

In order to format the source code (generated automatically or manually) properly some “code beautifiers” exist, like Tidy for HTML [82] and PERL [83] or the PEAR class `PHP_Beautifier` [84].

Documentation should be automatically generated from the source code as far as possible in order to be consistent and up to date. Tools which support this are `phpDocumentor` [85] for PHP, `Javadoc` for Java or `Doxygen` [87] for C++, C, Java, Python, PHP, C# and some more languages. They usually

are able to create the documentation in different output formats (for example HTML and PDF) from comments within the source code, where special tags have to be used.

Database Abstraction Layer

In order to enable independence of the underlying database management system an abstraction layer should be used, usually there is for every programming language at least one library available which implements generic access methods for several database management systems. For example for PERL “DBI” is the standard database interface module, where Java uses JDBC (Java Database Connectivity), and with PHP the Pear::DB class can be used (among some others). In addition, only standard SQL statements defined by the ANSI standard should be used, proprietary features of a database management system should be avoided as far as possible in order to be portable to another database management system.

Business Logic via SOAP

In order to provide an platform independent implementation the Simple Object Access Protocol (SOAP) or XML-RPC (XML Remote Procedure Call) could be used to include content from webservices, whether they are provided from a third party or by yourself on the same or another platform.

For example, Apache Cocoon provides SOAP logicsheets (xsp.SOAPHelper) based on eXtensible Server Pages (XSP) to include content from webservices. [88] shows an example how to use the functionality of the Cocoon SOAP logicsheet to query the Googles webservice. In addition there has been an integration of Apache Axis [90] (which is an implementation of the SOA protocol) into Cocoon - the AxisRPCReader allows to serve SOAP requests from your Cocoon application.

4.5 Testing

Testing should be automated as far as possible. Things that could be easily automated are the validation of HTML, CSS and WAI according to the W3C standards, and the check for broken links.

Unit Testing can be done with frameworks like JUnit [91] and PHPUnit [92], which is focused on the source code and modules, while HttpUnit [93] allows to test the functionality of the web application since HttpUnit also emulates the relevant portions of browser behavior, including form submission, JavaScript, basic HTTP authentication, cookies and automatic page redirection.

Regarding to website also spellchecking can be relevant before releasing a document.

Automated testing also supports refactoring in order to produce readable and therefore maintainable source code.

For websites also a load and performance tests may be relevant. This can be accomplished by using Apache JMeter [94] or using the native Apache HTTP server benchmarking tool (ap, [95] that comes with the Apache web server.

In order to ensure that all mail addresses on a website are valid they should be check as well. Unfortunately, there is no generic method to prove the validity of an email address. In the earlier days of the internet some mail servers also provided a method to verify the validity of an email address, but this feature has disappeared from most servers to prevent spammers from discovering mail addresses. So, what one can do to verify the validity of email addresses is to check if the format is compliant to RFC822 [96]. This is especially important if you allow someone to enter an email address in an form field on your website or content management system, maybe for subscribing to a newsletter. The second thing that can be done is to check if there is an MX record in the Domain Name System for the domain of a given email address. If there is no MX record, the mail address is invalid. If you are owner of a domain, you usually also have access to the mail system, so one could scan the website for mail addresses and look them up in the directory (e.g. LDAP database) where they are stored.

4.6 Deployment

In an ideal world there should be three different independent systems: a development system, a test system and a productive system, where the test system should have the same resources as the productive system in order to be able to perform load tests. The development environment is used to implement modifications and new functionality, so users on the productive

system don't run into errors. The test system should be a copy of the productive system as far as possible (except new development and modifications) in order to be able to test modifications. In some cases development and testing may be done on the same system, while development or testing never should be done on the same system as the productive system in order to provide continuous operation of the productive system ([116], Chapter 3).

In smaller projects there may not be a dedicated test and development system, so each developer may have its own virtual host on his computer for every project.

Once the development has been finished it will be deployed to the test system, where the customer may test if it matches his requirements, and after that is deployed to the life system.

In order to analyze errors which appeared in the life system but also for web usage mining in order to improve the usability in addition to the (error) logfiles of the web and database server application logging should be implemented. All logfiles should be automatically monitored and unusual events should be forwarded automatically to the webmaster.

For the source code some kind of versioning should be establish (for example by using CSV or Subversion) in order to be able to roll back to an previous version, if the new version seriously fails.

For efficient data synchronization between the different systems tools like Unison [112] (which allows bidirectional synchronization) or rsync (synchronization only in one direction) can be used, which can make use of a secure tunnel over SSH.

4.7 Operating, Monitoring

To be able to do a restore of the data in case of disaster, if the provider quits his service, or if some files are deleted by accident an automatic backup mechanism has to be established.

As to provide a stable platform and continuous business operation some kind of monitoring the system load, memory usages, number of busy and idle server, traffic, hits per second, availability and other system parameters have to be checked on a regular base, an alert should be created and sent to the webmaster automatically if a threshold value is reached.

After a security hole has been detected in a program available security updates should be applied as soon as possible in order to close this security hole and therefore reduce the risk of being misused - this process should be automated as far as possible.

In addition, a web application firewall (for example `mod_security` [97] for the Apache web server) could be established in order to reduce the security risks.

All logfiles should be automatically monitored and unusual events should be forwarded automatically to the webmaster. In case of a frequent invalid URL a redirect could be established in order to support the users to find what they actually looked for.

5 Case study

The goal of this case study done from 2003 till 2008 was to redesign the company website of an Austrian publishing company, which mainly publishes scientific medical journals on a regular base but also medical books, where some of them have been transformed into web applications. The website is mainly in German, where some parts exists in English as well.

5.1 Problems before the Redesign

The old website was relatively unstructured because it has been grown through the years (starting in the second half of the 90s) without a clear concept - neither regarding informational structure, nor the design, nor data maintenance or operating. It consisted of several subsites like a portal for cardiovascular diseases, a database for cardiology in Austria, a database for medicinal plants, the subsite for books, one subsite for each journal and so on, which where not integrated, each subsite had its own design and navigational concept.

The content has been maintained redundant from different persons and for different purposes. For example, the titles, authors, links to summaries, ... for articles have been manually maintained in a “Table of Contents”, a big non-scalable single HTML document, for each journal. Additionally, a list of authors has been manually maintained for each journal. Some articles, which have been rated “scientific interesting” by the company have been entered in a database to be searchable by metadata (authors, title, keywords, summary), while others where only stored as PDF documents, where the links to them has been maintained manually in the table of content. Some articles, especially older ones, only appeared in the table of content, without linking it to a PDF or entry in the database.

Articles of a special kind (e.g. reports from congresses) were cluttered around the header as linked-image to gain special attention from the users.

Some journals also (still) have different issues for different countries with nearly the same content (the content of the articles is the same throughout the different issues, the issues only differ in the articles they choosed to publish) except pharmaceutical news and advertising (due to legal reasons, depending on the approval of a drug for one country), so a lot of articles had to be maintained more than once for different issues.

The big single table of contents for each journal also forced the user to wait longer than necessary for the requested page. Therefore, depending on the size of the files, they were maintained partially redundant, once as a whole table, and once as small table, where only newer issues of a journal were listed.

The old website was based on frames, which has some usability problems, and did not make use of stylesheets - the design was maintained in each single HTML document, so content and design (and for the few existing applications, like the search engine, also the business logic) were mixed together, which made a change in layout difficult. Additionally, for each subsite and each new project (e.g. new databases and/or different structuring of existing content), a new design has been developed, which is not cost efficient.

There has been no overall navigational structure, so you could not gain an overview of the whole site or jump between different subsites. Also, a common search functionality was missing. There only has been an input form to search the meta data (where only some of the articles have been entered). If a new journal had to be added, the input form for the search had to be modified for each single journal, because of the drop down for selecting the journal to search in.

The data was also stored on different servers to reduce hosting costs, which made it difficult to check if all manually maintained pages, links and content really were in the right place. In fact, such a check never happened and no one had ever a look into the error logfiles of the web servers to check for bad links and to correct them. Additionally, many links started with the IP address rather than the domain name (due to temporary problems with the domain name system), which led to invalid URLs after changing the web-hoster and caused problems for a long time after the relaunch because many other websites also linked to this IP address URLs.

The files were stored in a single directory and had no defined namespace, which made it more difficult to find the right one.

Several different portals, projects and web applications have been implemented during the years, every of them with a different layout, different navigational structure, different concepts, and so on, which made it more and more difficult to maintain the different subsites and increased the maintenance costs. The different subsites did not reference each other, there was no overall concept for navigation, no possibility to search in all subsites and so on.

The screenshot shows a Mozilla browser window displaying the homepage of 'JOURNAL FÜR MENOPAUSE online'. The page is cluttered with various elements:

- Header:** Includes the journal title 'JOURNAL FÜR MENOPAUSE online' and 'ZEITSCHRIFT FÜR DIAGNOSTISCHE, THERAPEUTISCHE UND PROPHYLAKTISCHE ASPEKTE IM KLIMAKTERIUM'. There are several search boxes and promotional banners for abstracts and full-text databases.
- Left Sidebar:** Contains a navigation menu with links for 'Datenbank', 'SCHWEIZ', 'DEUTSCHLAND', and various issues (Heft 1/2003, Heft 4/2002, Heft 3/2002, Heft 2/2002, Heft 1/2002, Archiv, Sonderhefte). It also features a 'Bücher&CD' section and a 'Remans Online AWB' button.
- Main Content Area:** Titled 'JOURNAL FÜR MENOPAUSE online', it lists the publisher (AUSGABE ÖSTERREICH, SCHWEIZ, DEUTSCHLAND) and provides a search box. Below this, it lists the contents for 'Nummer 1/2002' and 'Nummer 2/2002'. Each entry includes a PDF icon, a summary, and page numbers.

Issue	Article Type	Title	Author(s)	Page(s)	
Nummer 1/2002	PDF	Editorial	F. Fischl	p. 5	
	Summary	Prämatüre Menopause - eine Übersicht	U. Ulrich, C. Dorn, J. Schmolling, H. van der Ven	pp. 7-15	
	PDF	Kardiovaskuläres Risiko und personale Defizienzen bei Frauen im mittleren Lebensalter	S. Bergmann, C. Mix, K. Kocis, P. Richter, W. Jarof	pp. 16-21	
	PDF	Niedrigpenetranz-Gene und Brustkrebs	N. M. Probst-Hensch	pp. 22-26	
	PDF	"Evidence based medicine" am Beispiel der präventiven Langzeitsubstitution mit Estrogenen-Gestagenen in der Postmenopause - Teil 1: Grundlagen	C. Lauritzen	pp. 27-30	
	PDF	Maturitas aktuell			
	PDF	Für Sie gelesen			
	PDF	Pharma-News			
	Nummer 2/2002	PDF	Editorial	W. Braendle	pp. 4, 5
		Summary	Hormonsubstitution zur Prävention des Herzinfarktes?	A. O. Mueck	pp. 7-18
PDF		Sind Estrogene Karzinogene?	H. Kuhl	pp. 19-27	
Summary		Vorstellung der ersten deutschen epidemiologischen Studie zu HRT und Mammakarzinom	W. Braendle, J. Berger, D. Flesch-Janys, St. Hentschel, J. Chang-Claude, G. Bastert	pp. 28-36	
PDF		Über das therapeutische Potential von Estrogenen beim Mann	M. Oettel, D. Hübner, B. R. Winkelmann	pp. 37-45	
PDF					

Figure 4: www.kup.at - homepage of a journal before the redesign

Figure 4 on page 47 shows the homepage of one journal before the redesign, which is based on frames, cluttered with abstracts in the header to gain special attention from the user and includes the content tables of all issues in one single HTML file.

5.2 Goals and Non-Goals

The main goals were to get rid of above problems by designing and implementing an overall concept for layout and navigation (and therefore reduce

implementation cost by not reinventing the wheel for each subproject and subsite) and to reduce manual maintenance effort and thus reducing maintenance costs while improving the quality and consistency of data. Therefore as much as possible should be generated automatically (e.g. tables of contents for the issues, list of authors and keywords per journal, PDF documents, images,...). Also new functionality should be added, like a database with images from the articles, full text search for the complete site, newsletters for user notification on new issues, interfaces to be able to query the databases from other sites and so on. Based on the analysis of the web server logfiles the site should also be continuously improved regarding usability. The data and business logic should be reusable to present the same data in different context (e.g. the search engine can be used for on the fly queries entered by users or to generate a portal consisting of articles for predefined search terms).

Internationalization and localization were none of the mandatory requirements, although there has been one English journal (by the time drafting the concept in 2003 the English journal was discussed not being published any longer, nevertheless in 2007 the journal still existed) and some English and French articles in the mostly German journals. Nevertheless, there are country specific issues for some journals (special issues for Austria, Germany and Switzerland) due to legal reasons.

Portability also has been none of the requirements, the website had to work only in one dedicated environment, that is operating system, web server software, programming language and so on. Although, some months after launching the redesign there was the requirement to run the website and all web applications on a notebook for presentation purposes on another operating system - which worked without problems and only slight modifications.

In order to improve the usability the use of HTML frames were not allowed in the new website in order to get printable, bookmarkable pages. Independent of the page the user enters the website he always should have access to the navigation menus.

The web pages should be split up in smaller information units instead of one big file, for example before the redesign all books published by the company were listed on one single page, including the detailed description, which caused long download times, forced the user to wait and to scroll down a lot instead of getting a quick overview of all offered books and getting into more detail on extra pages per book.

Another requirement by the customer was to generate parts of the PDF documents automatically, in order to decrease manual maintenance effort and therefore reduce costs.

The website should be prepared for advertising and sponsoring at different locations (e.g. in the global header and footer, in the local navigation menus, in PDF documents, in newsletter mails, ...) in order to refinance the implementation and internal and external maintenance costs. In order to increase the user retention a newsletter should be implemented, where users could be informed about new issues from their subscribed journals but also about new offerings on the website which may be of interest for them. Additionally an online shop to be able books and to subscribe to the printed issues of the journals should be implemented.

Another requirement was to deliver better analysis and reports of the web site usage by implementing additional logfiles and doing web usage mining, which is necessary to provide profound numbers to the sponsors and advertisers and to improve the usability of the website in order to attract more and more users and therefore increase the ad impressions. In order to attract more users the decision has been taken that there must not be a user registration, every content has to be available for free, the website will be only financed by sponsors and advertising and is seen as additional advertising platform for the company itself. By having all documents available for free the content is fully searchable by search engines like Google, Yahoo and MSN. Also, scientific organizations and universities can reference to the articles online.

All subsites had to be available on one single domain (while before the redesign the website was distributed across different providers, e.g. one part was hosted by the Medical University of Vienna, one by EUnet).

Since the information of the scientific articles was distributed across three different sources (manually maintained tables of content, databases, PDF documents) on different hosts a data migration and unification had to be done.

Before the redesign the only dynamic web pages were the ones for the search interface to the database for scientific articles, where the layout was mixed up with the business logic in the Java code. In the new system separation of concerns should be implemented in order to separate the layout from the business logic to be able to exchange one without affecting the other.

Another goal was that the maintenance of the content could be done by employees of the company, while in the past it has been done by an external webmaster (since there was no separation of concerns, at least content and style was mixed up).

An unified navigation for all subsites and subprojects (like a portal for cardiovascular diseases, a database for cardiology in Austria, a database for medicinal plants, the subsite for books, and so on) should be implemented, including the possibility to do a full text search.

Figure 5 on page 51 shows the portal for cardiovascular disease before the redesign, where the layout differs from the layout for the journals (while including content from the journals), an overall navigation between the two subsites is missing.

Figure 6 on page 52 shows the database for cardiology in Austria before the redesign, where the layout differs from other subsites, an overall navigation between the subsites is missing.

5.3 Data Migration

One of the main problems with implementing the redesign was the migration and merging of legacy data, that is, data which has not been entered into a database, but rather maintained manually in HTML files. Since there have been three different sources for the articles (tables of contents for each journal, a database with metadata of some articles and some documents available as PDF) there were also a lot of inconsistencies which had to be uncovered and solved.

Since the manually maintained table of contents had different structures even within one table the migration could be made only semiautomatically, that is, manually prepare the table of contents, so that it could be processed by a program, handling all entries the same way. One difficulty was to find a reasonable tradeoff between manually preparing the input files and implementing a script for automatic processing of the manually prepared input files. The more effort you put into manually preparing the input files, the less effort is it to implement the parsing and processing by a program and vice versa. For example, even for a human reader it was difficult to detect, if some information belonged to the previous or the next entry, even when

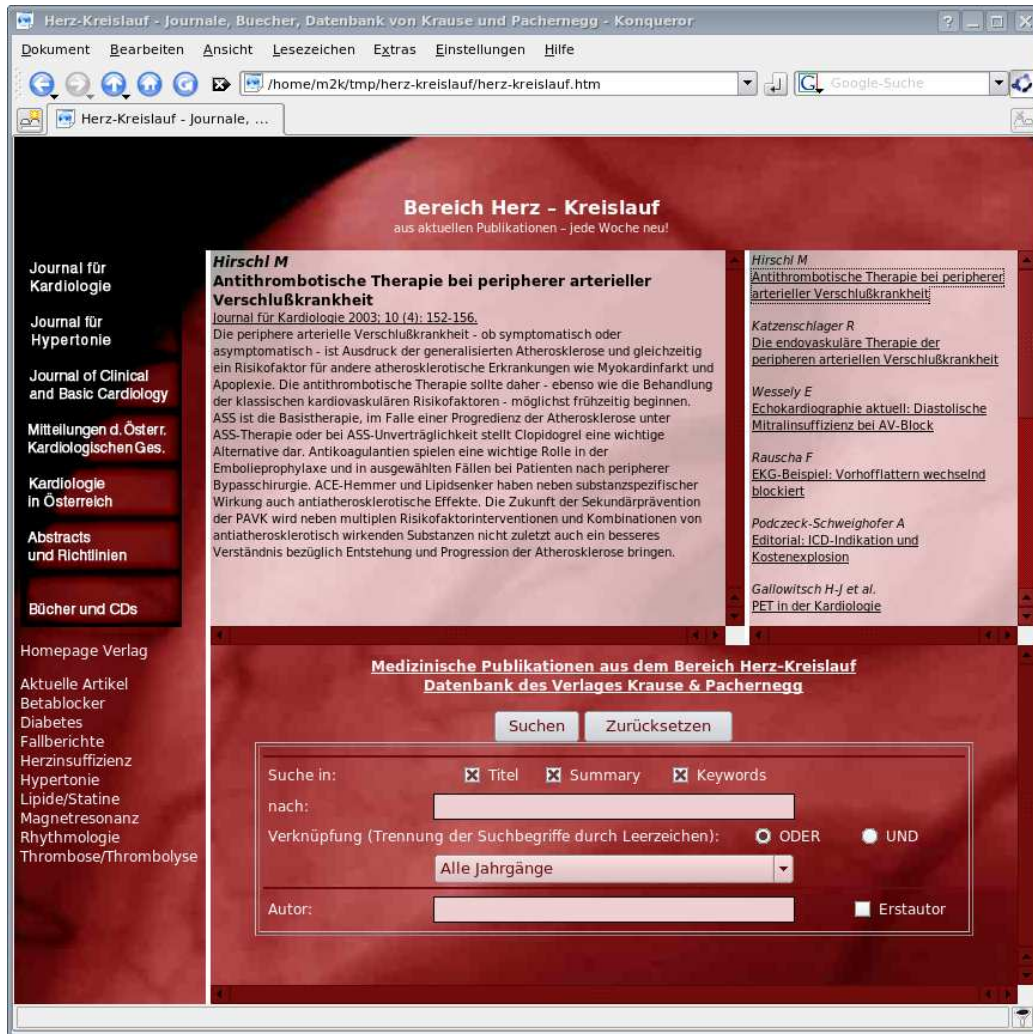


Figure 5: www.kup.at - portal for cardiovascular diseases before the redesign

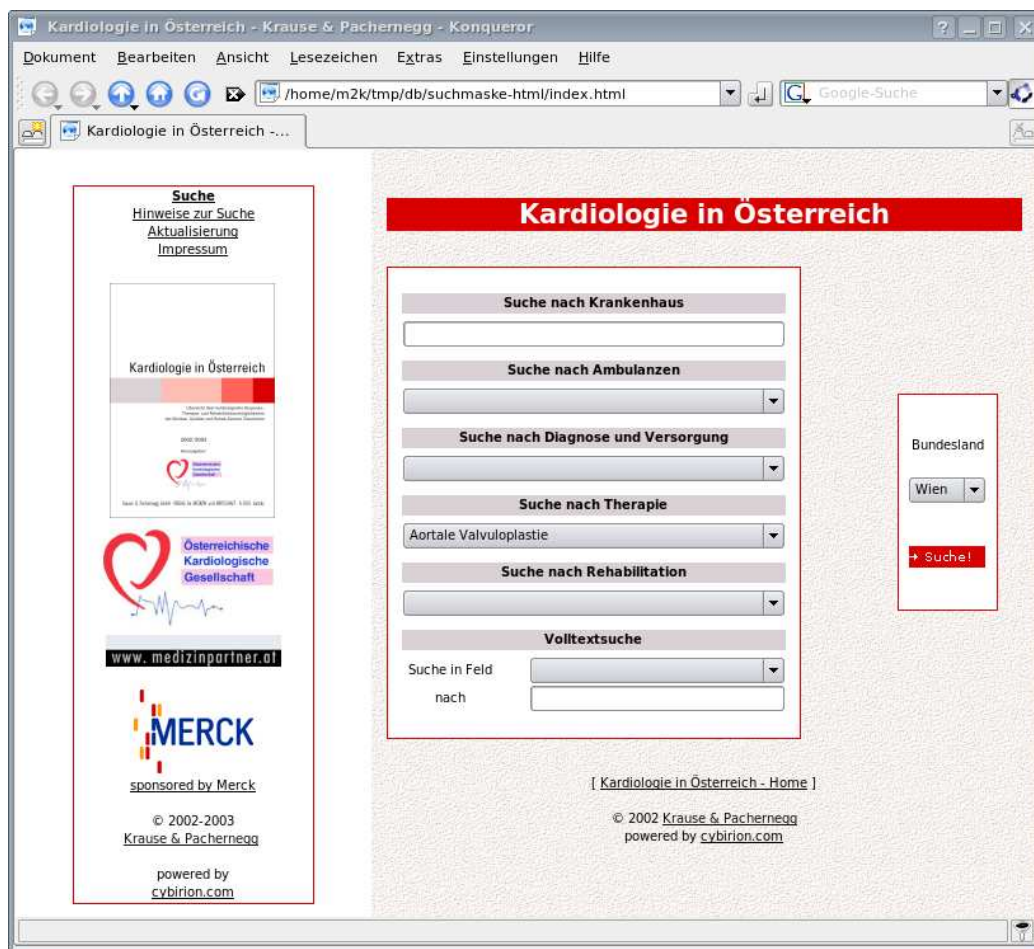


Figure 6: www.kup.at - database for cardiology in Austria before the redesign

looking at the pages in the browser.

Fortunately, I found some structures in the text, which were unique for some attributes. For example, page numbers always have the format pp. 23-25 or p. 35 and authors always were of the format “F1. Surname1, F2. Surname2”. URLs could be detected by a starting “http:”, the ID of an article and the name of the PDF file could be determined by parsing the URL, which always had the same structure. Everything else was taken as title (there were also some “articles” without authors and/or page numbers and so on). If - after removing all HTML tags - there was more than one newline without content, an entry has been taken as new article. The program displayed a warning message, if one entry had an attribute twice or an required attribute was missing, so I could iteratively improve the input files and the processing script until there were no warnings. If for an article also an entry in the database was found, the information has been taken from the databases, which I assumed was more accurate.

Since in the past the number(s) of the issue(s) in which an article appeared has not been entered into the database, this information was taken from the tables of content as well. If an article appeared in more than one issue, the article has been created only once in the database (if any entry did not yet exist) with an reference to the different issues.

Finally I made some crosschecks using the three data sources which have been merged before, for example, if for each link to an PDF file in the database really existed a PDF an vice versa. The detected conflicts were solved manually.

5.4 Informational Structure and Layout

The informational structure and layout has been designed mostly according to the guidelines from the previous chapter. The only exception is because at time of implementing the layout CSS was not yet well supported by all browsers in order to provide a complete layout, arranging the alignment of the different content elements. Therefore CSS only has been used for formatting the textual elements but not for the layout - instead nested HTML tables have been used which are hard to maintain.

The different areas of a web page include a global header, consisting of a

company logo, a search form for three different type of searches (scientific articles, images database and full text search), global navigation and global advertising, where the ad banners initially have been included statically, which turned out to be not efficient to maintain, therefore later have been included via an inline frame and therefore could be exchanged and rotated easily, where the position of the banner and the start and end time is defined by an employee of the company by an database entry. The content area was divided into two columns - one for local menus including local navigation and local advertising (which initially also was included static, but later was implemented using AJAX since it was not possible to use an inline frame because the height would have to be fixed, while the height of the local ad banners is not predefined and may vary) and one for the actual content. Finally a global footer is added on every page, containing again the global navigation and space for global advertising.

For the local navigation for all journals the same template has been used but some links (e.g. congress reports, guidelines, abstracts, country specific issues for Austria, Germany and Switzerland, , ...) do not exist for all journals, therefore a check has been necessary for the existence of these files, a link will be only generated if the related file exists. For this reason all files of a journal have to be generated at once, in order to be able to check the existence of the other links in the local navigation.

Figure 7 on page 55 shows the homepage of one journal after the redesign, which is not based on frames anymore and includes the content table for only one issue per page.

5.5 Implementation

Because of the advantages of static web pages listed in chapter 4 the static generation of the documents was preferred to dynamic web pages wherever possible. Since most of the content is uploaded once and not changed anymore afterwards and there are only a few updates - updates are done about once a month - all documents can be pregenerated. The only things which have to be dynamic are pages where user interaction is necessary, like the search engine where the user enters search terms or the online shop for ordering printed issues of the journals or books, but most of the content is static. The implementation has been done according to the rules and patterns defined in the previous chapter.

File Edit View Bookmarks Widgets Tools Help

Krause & Pachernegg
Verlag für Medizin und Wirtschaft

Artikel Bilder Volltext

Werbung
AT-101
3. Alpenländisches Angiologiesymposium
Vaskulärmedizin von Kopf bis Fasse
06. Oktober 2008 - Kitzbühel

Abn :: News :: Datenbanken :: Journale :: Vorträge :: Fortbildung :: CDs & Bücher :: Sitemap :: Impressum

Journal für Kardiologie

Aktuelle Ausgabe

Neuere Ausgabe Ältere Ausgabe

Aktuelle Ausgabe
Archiv



Sonderhefte
[Forum Rhythmologie](#)
[Abstracts](#)
[Leitlinien](#)
[Kongressberichte](#)

Editorial Board
[Autoren](#)
[Richtlinien für Autoren](#)
[Keywords](#)
[Impressum](#)
[Abonnement](#)
[Newsletter](#)

Suche nach
[Publikationen](#)
[Abbildungen/Graphiken](#)

[Diaspräsentationen](#)
[Fortbildung](#)
[Produktmonographien](#)

Ankündigung


INTERDISZIPLINÄRE HERZDIAGNOSTIK V

10./11. Oktober 2008
Hörsaalzentrum
Universitätsklinik Innsbruck

Nummer 7-8/2008

Huber K Brief des Herausgebers Volltext (PDF)	p. III
Alfonso F European National Society Cardiovascular Journals: Background, Rationale and Mission Statement of the "Editors' Club" (Task Force of the European Society of Cardiology) Volltext (PDF)	pp. 205-209
Bartel T Intrakardiale Echokardiographie und "Intraluminal phased-array imaging" zur Führung interventioneller Prozeduren Volltext (PDF) Summary	pp. 211-217
Glaser F, Rohla M EKG-Differentialdiagnostik der Breit-QRS-Komplex-Tachykardien Volltext (PDF) Summary Abbildungen	pp. 218-235
Desole S, Kähler CM Alveoläre Hypoventilation und pulmonale Hypertonie Volltext (PDF) Summary	pp. 236-242
Toplak H Update: Das Metabolische Syndrom Volltext (PDF) Summary	pp. 243-246
Kunert M, Gremmler B, Konzen G, Ulbricht LJ Fallbericht: Angina pectoris bei Tachykardie-induzierter Obstruktion im linksventrikulären Ausflusstrakt Volltext (PDF) Abbildungen	pp. 247-249
Beran E, Reiter G, Mächler H, Reiter U, Rlenmüller R Das MR-Flussbild am Beispiel eines poststenotischen Aortenaneurysmas - ein Fallbericht Volltext (PDF) Abbildungen	pp. 251-252
Pürerfellner H Kongressbericht: ATHENA-Studie: Vorläufige Ergebnisse und deren mögliche Auswirkungen auf die antiarrhythmische Pharmakotherapie bei Vorhofflimmern. Volltext (PDF)	pp. 253-254
Säly Ch, Drexel H Aktuelles: Die PERISCOPE-Studie - Hintergrund, Kernaussagen und	pp.

Figure 7: www.kup.at - homepage of a journal after the redesign

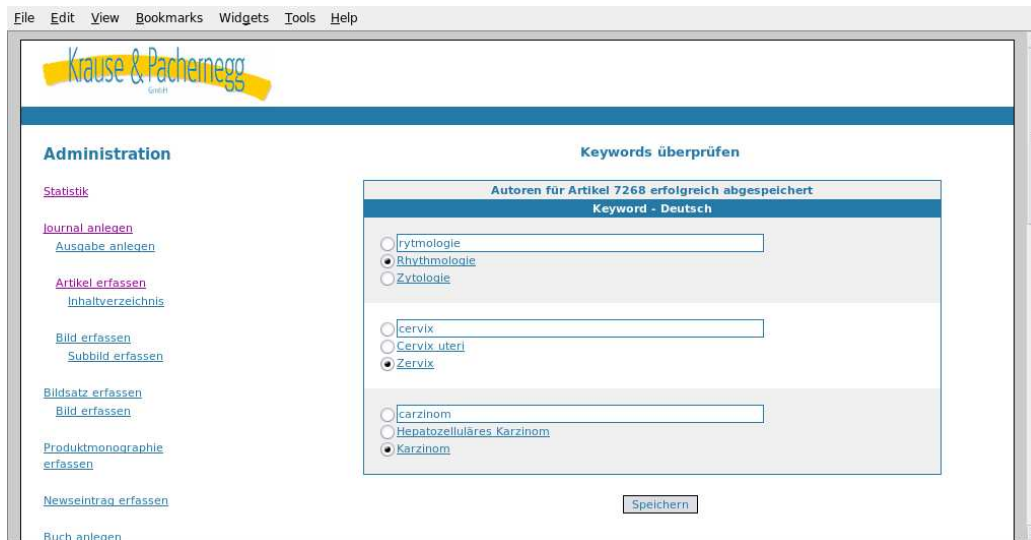


Figure 8: www.kup.at - fault tolerant database administration frontend to enter keywords

5.6 Content Management and Deployment

In order to allow the employees of the company to enter new journal issues, scientific articles, images, books and so on by themselves user interfaces have been implemented.

To reduce typos, misspellings and different spellings for the same word a user interfaces has been implemented which offers similar terms to the user for entering the names of the authors and keywords, where the user can keep the one entered or select one of the presented already existing terms.

Figure 8 on page 56 shows a part of the database administration frontend to enter new articles. When entering authors or keywords for an article they are checked against existing authors respectively keywords, and similar terms are presented to the users in order to avoid misspellings.

Since content is usually only entered once and not changed afterwards there are no user interfaces to change or delete entries from the database for cost reasons. In cases of typos or changes a generic web based database frontend (phpMyAdmin [104]) can be used. Since changes can be made directly to the database and therefore the consistence may be violated by deleting entries which are still referenced (which is possible since the database management system MySQL did not support foreign references in earlier implementations)

consistency checks are done automatically when regenerating the indices for the search engines and interfaces.

If a PDF document or image has to be replaced because of a mistake or the metadata in the database has been changed or corrected it can be done by uploading the new document to the original directory for the particular document type while the automatically generated document can be deleted from the webtree. With the next regeneration step started by the user all missing PDF documents and images are automatically created. The replacement of the documents is done by using the Secure Copy Protocol (SCP) with a native Windows Client for SCP.

In order to decrease the time for generation of images, HTML files and PDF documents the user can select which parts and sections to regenerate and if only new documents which do not yet exist or if all documents should be recreated, independent if they existed before.

In the first phase after the relaunch there was an extra internal host for the content management system, located at the customer's site, generation of static documents and indices was always done on this host offline, which reduced the system load to the productive system. After generation and revision by the content manager the changes were transferred to the productive system by using Unison [112], a file synchronization tool using a secure tunnel via SSH, which allows to transfer only the files that have been modified since the last synchronization.

The concept with the additional content management system for offline generation at the customer's site turned out to be hard to handle for the employees, especially the synchronization. Also the deployment of templates and changes in the business logic was difficult, since the host was not reachable from the outside, so only the employees would have been able to deploy such changes which they got by mail.

Therefore after one year the additional host for offline generation and content management was shut down, and the functionality was migrated to the productive system. Since all documents and indices were static changes to the database had no immediate impact on the online content, only after the generation of the static content the changes are online, where the generation is done in a temporary directory (since it may take hours for the whole site if regenerating everything) and all regenerated parts are put together online, therefore the online website always is in a consistent state. The only

disadvantage of this concept is the additional system load generated by the regeneration of database indices and static documents.

In addition to the productive system (and in earlier days the host for content management and offline generation) there always has been a development and test system to develop modifications and new functionality. Since this test system is located at the developers site and not acting as a static server to the internet in order to present the additional functionality or modifications to the customer usually a copy of the existing programs has been made, which finally replaced the old versions.

Figure 9 on page 59 shows the user interface to start the automatic generating of the website, which includes options to check the consistency of the database, checking links, regenerating the database indexes, generating a thesaurus, generating all or only new PDF documents and images, and web pages for different subsites.

5.7 Operating

- **Backups.** To be able to do a restore of the data (currently 18 GB of data have to be backedup) in case of disaster, if the provider quits his service, or if some files are deleted by accident several backup mechanisms have been implemented.

One is based on rsnapshot [105], which uses rsync and hard links, therefore it is possible to keep multiple, full backups instantly available, where the disk space required is just a little more than the space of one full backup, plus incrementals. rsnapshot is used to create disc to disc backups, it allows to get an older version of a file or to quickly restore files which have been deleted by accident. Of course, a disc to disc backup does not help in case of a crash of the harddisc (although, currently a RAID-1 system is in use where two identical harddiscs are used for mirroring the data - if one of the discs fails the system can still operate using the other disc. The failed disc can be exchanged in the meantime by a new one.)

Another backup system to backup files to a non-trusted FTP server (which is offered by the provider in the same size as the RAID-1 hard discs) uses ftplicity [106] respectively duplicity [107]. Duplicity backs up directories by producing encrypted tar-format volumes and uploading them to a remote or local file server. Because duplicity uses librsync,

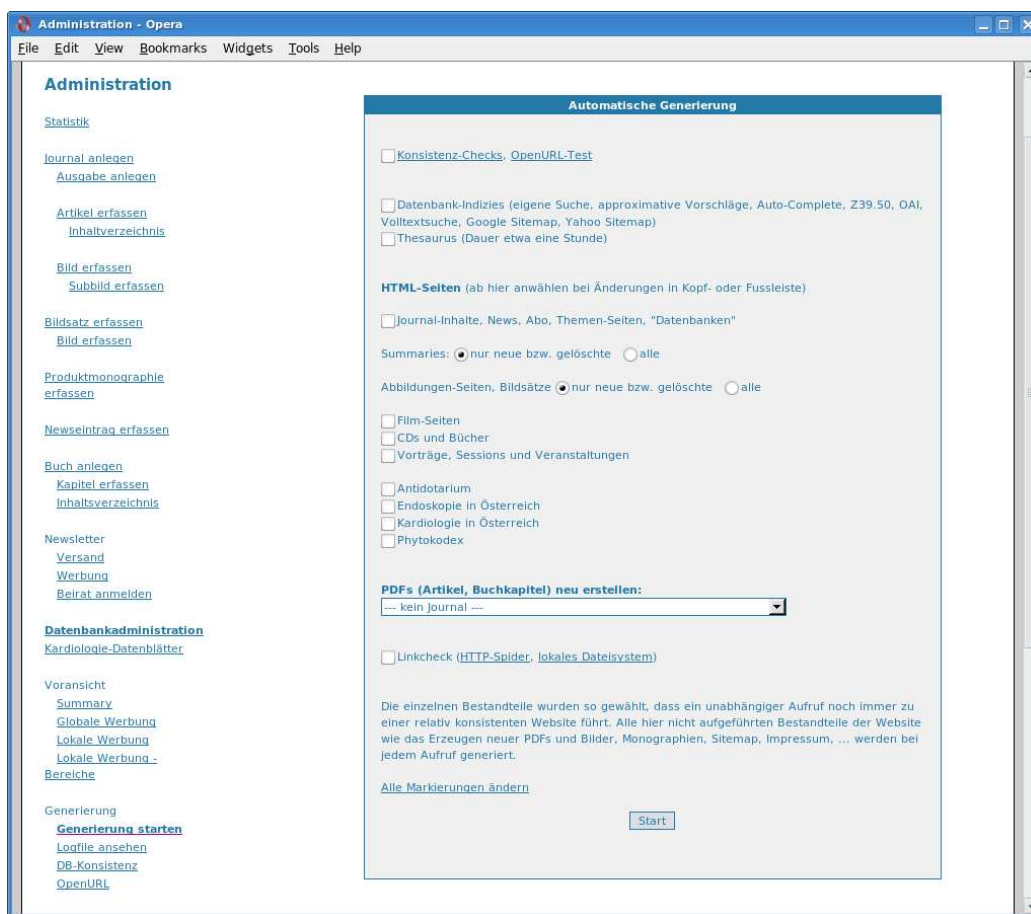


Figure 9: www.kup.at - user interface to start the automatic generation of the website

the incremental archives are space efficient and only record the parts of files that have changed since the last backup. Because duplicity uses GnuPG to encrypt and/or sign these archives, they will be safe from spying and/or modification by the server.

In addition to the two backup systems described above all files are synchronized to a local development and test system using the Unison file synchronization tool [112] via SSH where it can be copied to some DVDs and an external USB harddisc.

- **Monitoring**

As to provide a stable platform and continuous business operation the system load, memory usages, number of busy and idle server, traffic, hits per second, availability and other system parameters have to be checked on a regular base. This has been solved by using a round robin database provided by the RRDTools [108] which allows to store and rotate time related data with different time intervals while using a constant disc space and based on a script developed by Ivan Ristic presented in his book “Apache Security” [109], page 209 to 217. The script has been extended to also include the CPU load average and the current memory usage. Figure 10 on page 61 shows this monitor for the CPU load, memory usage, traffic and number of servers for the Apache web server.

This figures can be used for sizing the server platform, that is estimate the hardware needs for the future, but also for finding out some unusual system activity like Denial of Service (DoS) attacks, too many concurrent requests and therefore a backlog of the requests, search engine robots which create high system load by indexing dynamic web pages (although this is disallowed by the file robots.txt in the root directory of the webtree) in an endless loop and so on.

Basic availability is checked from the outside from another host by using “mon” [110], which tests for the availability by sending ICMP echo requests and doing basic HTTP requests on a regular base. In case of a serious problem (that is, if one of the tests failed several times) an alert is created, which is sent to the webmaster.

In addition, all logfiles are checked by “logcheck” [111], which filters out normal system events and sends a summary of real errors and un-

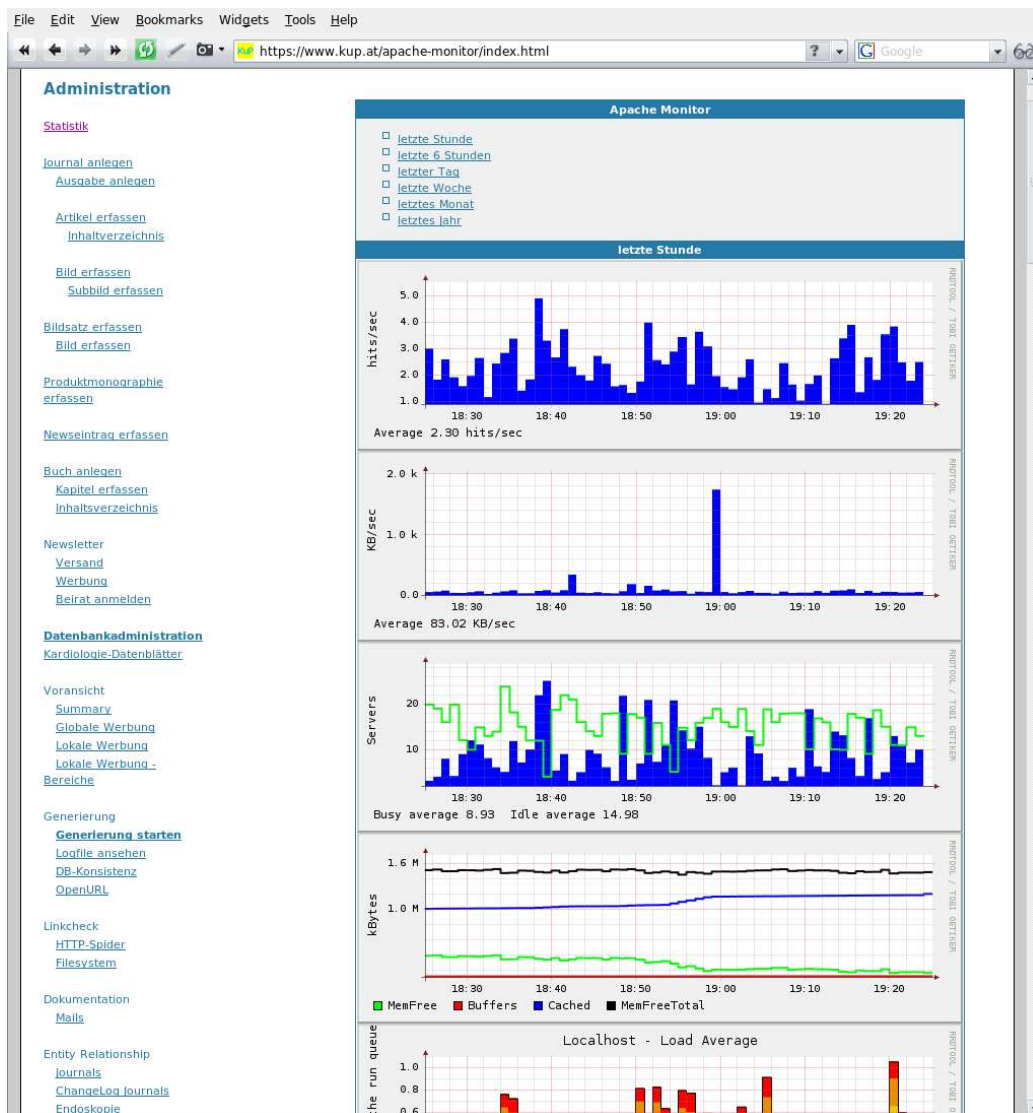


Figure 10: www.kup.at - monitor for the hits per second, traffic, number of busy and idle Apache servers, memory usage and CPU load average

usual system events to the webmaster. If for example an invalid URL appears for several times (e.g. by mistyping by the user) a redirect may be created for this URL in order to support the users to find the content. To provide further error tolerance for typos in URLs the Apache module `mod_speling` has been activated, which allows one wrong letter in the URL and also supports case-insensitivity. Wrong URL and unusual events that should not be reported in the future can be included in list of regular expression which should be ignored.

In order to reduce the traffic the Apache module `mod_gzip` has been activated, which tries to send compressed content to the customer, if the client accepts this kind of reply. This may slightly decrease the performance on server side since documents have to be compressed before returning them to the client, but increases the download speed on client side.

- **Security**

After a security hole has been detected in a program available security updates should be applied as soon as possible in order to close this security hole and therefore reduce the risk of being misused. For the operating system (Linux Debian) and all installed packages this can be done automatically by using the Debian package “`cron-apt`” [113], which gets updates every night and deploys them automatically.

As the web page for ordering journals and books has been misused by entering a random mail address the attacker wanted to send spam mails to (so called “Form Post Hijacking”) no automatic order confirmation is sent to the inserted mail address anymore, the confirmation is only sent to the company itself - they have to decide if it is a spam mail or a real order. Another approach to get rid of this problem could be the use of CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart), asking a user to complete a simple test which the computer is able to generate and grade, but unable to solve, for example the user has to type the letters or digits of a distorted image that appears on the screen. This approach would reduce the comfort for the users and therefore has not been chosen.

In addition, if there are too much requests from one single IP address which turns out to be some kind of misbehaviour (for example using order confirmations as spam mails or doing many concurrent request) an IP address may be locked out temporarily or permanent by config-

Table 1: Case Study - Statistical Data

	June 2003	August 2003	June 2008
Metadata in the database for			
- scientific articles:	1.510 ¹	3.440	7.176
- images:	n/a	554	10.859
Number of journals:	15	15	20
journal issues:	n/a	598	1054
Database tables	31	37	89
PDF documents:	2.679	2.679	6.266
Images (including thumbnails):	153	1.158	31.627
HTML documents:	180	2.885	17.468
Videos:	n/a	n/a	292
Total number of documents:	3.012	6.722	55.361
Number of Templates:	n/a	24	144
Lines of Code:	7.793 ²	7.423 ³	35.449 ³
Overall data size:			18 GB
Traffic per month:	11,5 GB	14 GB	141 GB
Visits per month:	41.554	44.342	226.526
Page views per month:	198.446	299.817	3.749.140

uring Apache to send “HTTP 403 forbidden” replies to this IP address, so no further request will be answered to that address.

5.8 Statistical Data

Table 1 on page 63 compares the website in numbers before the going live in June 2003 to the numbers in August 2003 after the relaunch and to June 2008.

The numbers of HTML documents have increased from 180 before the relaunch to 2.885 after the relaunch, since they have been maintained manually

¹1.510 scientific interesting articles have been entered into a database in addition to manual maintained HTML files while 1.169 articles were only added manually to the HTML files. These articles have been added into the database during the semiautomatic data migration

²The code has been written in Java. Only the search engine and database administration frontend for scientific articles have been available, templates have not been used, so business and presentation logic are mixed up in the source code

³The code has been written in PERL, the search engine is not included, since an external component (Swish-E) has been used for this purpose

before and automatically generated after so it is much easier to add another document class. In addition the content has been split up in smaller units per page, e.g. the table of content has been maintained in one big HTML file before the relaunch, where after the relaunch every issue got its own HTML page, and the list of authors and keywords which are referenced by one journal have been split up into one page per initial letter and journal instead of only one page per journal for all initial letters. All this pages are generated automatically.

From August 2003 to June 2008 the number of HTML documents has increased from 2.885 to 17.468. The reasons for the increase in number of documents are on the one hand additional entries in the database, that is, new articles, issues and journals, but on the other hand a lot of new classes of document types have been introduced - since now everything is created automatically from one single source it is much more easy to maintain additional document classes.

For example new theme portals have been created which are generated completely automatically. The only thing which is predefined are the queries to use for each subject in the theme portal to search for articles in the database.

The articles in the database only doubled within the five years, where the number of templates and different document types increased from 24 to 144 and the number of database tables increased from 37 to 89, so the main source of growth were additional document types which were easy to add with the new overall concept, since only the content of the new document types had to be discussed, but not the informational structure or the layout, also in general functionality from the business logic could be reused for new templates.

5.9 Additional new Functionality

- **Image database** Figure 11 on page 65 shows the search engine for the image database, which has been implemented after the relaunch and only took four hours to implement, since most concepts could be taken from the search engine for scientific articles and business logic could be reused. Before the relaunch every project had its own layout and concepts, so it took longer to discuss, implement and test layout, navigation, functionality and so on.

File Edit View Bookmarks Widgets Tools Help

Krause & Pachernegg
Verlag für Medizin und Wirtschaft

Werbung
SCHILLER Austria
3. Alpenländisches Angiologiesymposium
Vaskulärmedizin von Kopf bis Fuß
06. Oktober 2008 - Kitzbühel

Abd :: News :: Datenbanken :: Journale :: Vorträge :: Fortbildung :: CDs & Bücher :: Sitemap :: Impressum

Suchergebnis - Abbildungen und Graphiken

Alle Journale wurden nach **Echokardiographie** durchsucht Treffer **1 - 20** von **402**

Medizinische Publikationen (111) :: Abbildungen und Graphiken :: Volltext (694)

Neue Suche:

In allen Journalen nach Bildtitel Bildkeywords Bildsummary Autoren Mindestens ein Wort Alle Wörter

Oberbegriffe: Graph
Unterbegriffe: CW-Doppler-Echokardiographie, Dipyridamol-Streßechokardiographie, Dobutamin-Stress-Echokardiographie, Doppler-Echokardiographie, Doppler-Stress-Echokardiographie, Echokardiographiekontrolle, Kontrast-Echokardiographie, Kontrastechokardiographie, Spektraldoppler-Echokardiographie, Streßechokardiographie, transösophageale Echokardiographie
Verwandte Begriffe: Abschlusangiographie, Abschlusarteriographie, Aktiographie, Ammoniak-Positronenemissionstomographie, Ammoniak-Positronenemissionstomographie, Angiocardiography, Angiographic, Angiographie, Angiographieaufnahme, Angiographiebefund, Angiographieeinheit, Angiographielabor, Angiographisch, Angiographischer Schweregrad, Angiography, Arteriographie, CT angiography, CT-Angiographie, CT-Kolonographie, Cholangiopankreatographie
Verwandte Begriffe (Quelle: Morphosaurus): Kardiologie, disease, Herzkrankheiten, Herzinsuffizienz, heart, kardiovaskulären, Myocardial, erkranken, cardiac, Myokardinfarktes, Herzerkrankung, kardiale, cardiovascular, Cardiol, heart disease, kardiologischen, Cardiology, Verschlusskrankheit, Herztransplantation

[weitere](#)

1. **Michalski T;** Chmelizek F; Pichler M
EKG-Beispiel: Präklinischer Ultraschall beim Akuten Koronarsyndrom, Non-STEMI (ACS)
Ultraschallgerät
Journal für Kardiologie 2008; 15 (7-8): 261
Präklinische Anwendung der **Echokardiographie** im Salzburger Notarztbuschrauber Christophorus 6.
Keywords: Fotografie, Kardiologie, Ultraschallgerät

Artikel:
[Volltext \(PDF\)](#)
[Abbildungen](#)



[Detailansicht](#)

2. **Kunert M;** Gremmler B; Konzen G; Ulbricht U
Fallbericht: Angina pectoris bei Tachykardie-induzierter Obstruktion im linksventrikulären Ausflusstrakt
LVOT
Journal für Kardiologie 2008; 15 (7-8): 247-249
Doppler-Stress-**Echokardiographie** mit dynamischer Flussbeschleunigung im LVOT (Gradient 53 mmHg) unter High-dose-Dobutamin-Stimulation
Keywords: Doppler-Echokardiographie, Kardiologie, LVOT

Artikel:
[Volltext \(PDF\)](#)
[Abbildungen](#)



[Detailansicht](#)

3. **Kunert M;** Gremmler B; Konzen G; Ulbricht U
Fallbericht: Angina pectoris bei Tachykardie-induzierter Obstruktion im linksventrikulären Ausflusstrakt
LVOT

Artikel:
[Abbildungen](#)



Figure 11: www.kup.at - search engine for the image database

- **Automatic generation of PDF documents.** For scientific articles it had been an requirement to add a title page and a final page. The title page should look like the first page of the printed issue of the journal, but containing the author, title and issue of the article. The final page contains a formular for subscribing the print version of the journals and advertising for the publishing company, but also the possibility to offer advertising space to sponsors. These two pages had been added manually in the past, after the redesign they have been created automatically, where the content is taken from the database and templates have been implemented for each journal. In addition, metadata from the database is copied automatically to the metadata of the PDF document, which is used by search engines for indexing. Per default only PDF documents are created which do not yet exist in order to reduce the time for generation. Sometimes it is necessary to recreate all PDF documents of one journal, for example if the template for the first page has been changed or there is a new sponsor for the advertising page, therefore there is an additional option to regenerate all PDF documents of one journal. Figure 9 on page 59 shows the user interface to start the automatic generation.
- **Automatic generation of images.** After uploading the images to the directory outside the webtree in original size they are resized into three different sizes for different purposes, that are, detailed view, thumbnail and small thumbnails for navigation. To the images for detailed view an additional bar with annotation text is added automatically, where the text is taken from the database and depends on the type of the image (images of scientific articles, books, presentation slides, ...). For images of scientific articles also some keywords are added automatically into the database, depending on the journal and the keywords of the article, so the person who uploads the image does not have to enter this keywords manually. In addition for images IPTC [98] meta information is added automatically for each image from the database. As with PDF documents only images are created which do not yet exist in order to reduce the time for generation.
- **Approximate error tolerant search engine.** By doing web usage mining I found out, that about 20 percent of all queries to the search engine done by users returned no match at all. Reasons may be that there is no relevant content for the search term entered by the user, but also different spellings or misspellings. Therefore I implemented a error tolerant mechanism to present users similar terms out of the key-

words and authors if the search returns no matches. The implementation is based on `agrep` (approximate `grep`) [102] which is a fuzzy string searching program, developed by Udi Manber and Sun Wu. It selects the best-suited algorithm for the current query from a variety of the known fastest (built-in) string searching algorithms, including Manber and Wu's `bitap` algorithm based on Levenshtein distances [103]. The algorithm tells whether a given text contains a substring which is "approximately equal" to a given pattern, where approximate equality is defined in terms of Levenshtein distance - if the substring and pattern are within a given distance k of each other, then the algorithm considers them equal. In case of the search engine k is the minimum number, where at least one author respectively keyword is found. By offering a list of "did you mean one of this terms?" the queries which returned no match could be reduced by 6 percent.

Figure 12 on page 68 shows the approximate error tolerant search - if no database entry matches the search term entered by the users similar search terms are presented to the user in order to correct maybe misspelled search terms.

- **Auto completion.** In order to further help the users to find what they want and to reduce the misspellings and problems with different spellings when entering search terms an auto completion functionality has been implemented for all search forms, that is if a user types a letter a list of possible search terms pops up, if he types a second letter the list will show the words which match this two letters. The more letters he types the less search terms will appear in the pop up, finally the user should be able to select the search term from this small list. If the term does not show up in the pop up then there is no content for it in the database. Therefore for different search forms querying different databases and tables different lists of search terms have to be prepared, which are generated automatically after updates to the databases.
- **Stemming.** In order to further improve the quality of the results of the search engines stemming has been implemented, that is, every word is reduced to its stem, where the stem does not have to be a valid word, but words with the same stem could be considered as identical. This mechanism also supports the user by finding entries which match the same meaning, but with different endings for example (e.g. plural vs. singular). The word index of the databases is already reduced to its stem (which is generated automatically after updates to the databases),

The screenshot shows a web browser window displaying a search results page. The browser's address bar is not visible, but the page content is as follows:

- Header:** Logo for Krause & Pachernegg Verlag für Medizin und Wirtschaft. A navigation menu includes: Abo :: News :: Datenbanken :: Journale :: Vorträge :: Fortbildung :: CDs & Bücher :: Sitemap :: Impressum.
- Search Results:**
 - Suchergebnis - Medizinische Publikationen
 - Alle Journale wurden nach **carzinom rythmolgie mayyer** durchsucht
 - Die Suche nach 'carzinom rythmolgie mayyer' in den wissenschaftlichen Publikationen ergab leider keine Treffer.
 - Meinten Sie vielleicht einen der folgenden Begriffe?**
 - [Bayer](#), [Bayer-Schering-Pharma](#), [Endometrium-Karzinom](#), [Hepatozelluläres Karzinom](#), [Karzinom](#), [Karzinom-Prävention](#), [Karzinom-positiven](#), [Maier-Kur](#), [Mamma-Karzinom](#), [Marker](#), [Marker-Symptome](#), [May](#), [Prostata-Karzinom-Risiko](#), [Rhythmologie](#), [Inflammatorische Marker](#), [tumour marker](#)
 - Meinten Sie vielleicht einen der folgenden Autoren?**
 - [Mayer A](#); [Mayer Ch](#); [Mayer E](#); [Mayer G](#); [Mayer H](#); [Mayer B](#); [Mayer S](#); [Mayer P](#)
 - Tipps, um das Ergebnis der Suche zu verbessern:** Gestalten Sie die Suchanfrage **allgemeiner** indem Sie nur nach **einem Wort** anstatt mehrerer Wörter suchen oder den **Platzhalter *** für beliebige Zeichenfolgen verwenden
- Filters:**
 - Medizinische Publikationen :: [Abbildungen und Graphiken \(0\)](#) :: [Volltext \(0\)](#)
 - Neue Suche:
 - In allen Journalen nach:
 - Titel
 - Keywords
 - Summary
 - Autoren
 - Mindestens ein Wort
 - Alle Wörter
- Footer:**
 - Navigation: Abo :: News :: Datenbanken :: Journale :: Vorträge :: Fortbildung :: CDs & Bücher :: Sitemap :: Impressum
 - Copyright © 2003-2008 Krause & Pachernegg | Technik: Markus Krumpöck
 - Advertisements:
 - AGV ARBEITSGEMEINSCHAFT FÜR NOTFALLMEDIZIN
 - agnsh p gratis download agnsh p AGNFIBEL für PDA
 - BOOKSHOP agnsh p gratis download AGNFIBEL für PC + DESKTOP
 - NOTA MEDI
 - Werbung:
 - Stents u. Ballonkatheter**: OptiMed Ihr Partner bei Gefäßinterventionen www.opti-med.de
 - Schöne Mariengeschichten**: Erlebnisse, Tatsachenberichte mit der Wundertätigen Medaille www.die-wundertaetige-medaille.at
 - EKG**: Hersteller EKG Flüssigkeit EKG messgeräte/ Aufzeichner www.eximholland.nl

Figure 12: www.kup.at - approximate error tolerant search

so if a user enters a search term the single terms are also reduced to their stems and compared to the stems in the index - if they match the entry is returned to the user.

- **Thesaurus, Morphosaurus.** As to support users to find what they want an Thesaurus has been built which is generated automatically. Although it is only based on substrings - if one term contains another term as substring, then the substring is considered as generic term while the first term is a subtopic, where the terms are taken from keywords and summaries of the scientific articles - the results are quite good. If two terms have the same generic term they are considered as related and are presented to the user as well. Very promising results have been delivered by the “Morphosaurus”, where we cooperated with the Department of Medical Informatics in Freiburg [118]. The results of Morphosaurus have been integrated in our automatically built thesaurus as well. Figure 11 on page 65 shows the search engine for the image database including the terms of the thesaurus and morphosaurus in the head of the search result.
- **Z39.50, OAI, OpenURL and XML - Interfaces.** In order to enable other websites to search our content we implemented several interfaces to our database.

We have installed and configured a server which supports the Z39.50 standard: Zebra [99] is a high-performance, general-purpose structured text indexing and retrieval engine. It reads structured records in a variety of input formats (e.g. email, XML, MARC) and allows access to them through exact boolean search expressions and relevance-ranked free-text queries.

In order to support the Open Archive Initiative (OAI) [100] standard we have implemented an OAI repository for our scientific articles using an implementation from the Virginia Tech university [101].

In order to improve the visibility to search engines (Search Engine Optimization, SEO) the list of available documents is generated and submitted to Google Sitemaps automatically.

After an update to the database the indices for Z39.50, OAI, full text search and search in meta data of articles and images can be automatically regenerated, the files for the Google Sitemap are regenerated after

every update and uploaded automatically to the servers of Google.

- **Performance optimization of the search engine.** Due to the implementation of a lot of new features like auto completion, thesaurus, stemming and so on it became necessary to improve the performance of the search engine - therefore it has been reimplemented (without changing the user interface), based on static generated flat files where all relations were denormalized instead of accessing directly to the database (which is in third normal form and therefore includes a lot of joins which decrease performance and does not scale well, e.g. for every keyword an additional join would be necessary). For performance reasons the search engine only returns the identifiers of the entries which match the search terms, when presenting the results to the user only the detailed information for the entries presented to the user (usually 20 entries per page) are read from the database, so the load to the database is constant for every query, independent of the number of entries. Some entries should not show up in the search result, therefore a list of to ignore expressions for articles has been implemented as well as a list of stopwords which are ignored.
- **Full text search.** In order to offer an performant full text search to the user an ready to use full text search engine has been configured, which also indices the content of the PDF documents (currently 1.5 GB of data). To improve the performance of indexing these documents an additional caching mechanism has been implemented, which stores the result of the conversion from PDF to text format. Since search engines use the meta information of the HTML and PDF documents every document had to have this information, which is added automatically for both document types.
- **Portability.** Although portability has not been one of the initial requirements some months after launching the redesign there was the requirement to run the website and all web applications on a notebook for presentation purposes (just in case if no network connection is available or other possible troubles which can appear when giving a presentation) on another operating system - which worked without problems with some small modifications (e.g. in the configuration different paths have been selected automatically depending on the current operating system).
- **Newsletter.** In order to increase the user retention and traffic (and therefore incoming by advertising) a newsletter has been implemented,

where users could be informed about new issues from their subscribed journals but also about new offerings on the website which may be of interest for them. At the time of subscribing to the newsletter the user has to enter a valid mail address, which is validated against RFC822 and checked if MX records exists for that domain. Afterwards an mail is sent to the address entered by the user with a unique link the user has to click in order to confirm this mail address is valid and the owner of the mail address really wants to subscribe to the newsletter and the address has not been misused by someone else - this mechanism is called "double opt in". The content of the newsletter itself is created automatically for new journal issues which have not yet been sent, while the content manager can add manually some free text, for example announcements of new offerings on the website. The newsletter can be sent to all subscribers by a few mouse clicks, the content manager does not have to type anything.

- **Video- and audio streaming.** As to present short video clips for e-learning purposes, presentations from conferences and in addition to the scientific articles in the printed journal issues first a streaming server based on Quicktime / Darwin has been implemented. Since it turned out that this may lead to problems with firewalls at the client side, we decided to use the Macromedia Flash technology for this purpose, where the video files are stored as FLV files, the audio files are stored as MP3, for conference presentations one single generic Flash application has been created to present the video respectively audio file to the user, which allows to pass the identifier of the video/audio to the Flash application. For this reason conference presentations are generated completely automatically, the content manager has only to upload the media files. Figure 13 on page 72 shows the video stream of a conference presentation. Right to the video the slides can be found, a click on the thumbnails of the slides leads to a larger view of the slides.

5.10 Possible Improvements

Issues which are not yet solved in an optimal manner and therefore future projects may be one of the following:

- **Automatic conversion of full text.** In order to increase the number of ad impressions and therefore revenue one goal was to convert the content which is mainly contained in PDF documents automatically

File Edit View Bookmarks Widgets Tools Help

Krause & Pacherneegg
Verlag für Medizin und Wirtschaft

Artikel Bilder Volltext

Tagungsort:
St. Wolfgang
- Salzkammergut

Kongreß-
organisation:

Abo :: News :: Datenbanken :: Journale :: Vorträge :: Fortbildung :: CDs & Bücher :: Sitemap :: Impressum

Veranstaltungen > Gemeinsame Jahrestagung der Österreichischen Gesellschaft für Endokrinologie und Stoffwechsel und der Österreichischen Gesellschaft zur Erforschung des Knochens und Mineralstoffwechsels > Session Nr. 1 - 15.5.2008 - Nebensitzung Vorsitz: V. Stepan (Graz), J. Patsch (Innsbruck)

Vortrag von Univ.-Prof. Dr. med. V. Stepan (Graz). Gemeinsame Jahrestagung der Österreichischen Gesellschaft für Endokrinologie und Stoffwechsel und der Österreichischen Gesellschaft zur Erforschung des Knochens und Mineralstoffwechsels, 15.5.2008 - 17.5.2008
Clinical Practice Guideline - The Diagnosis of Cushing's Syndrome: An Endocrine Society Clinical Practice Guideline

Vortrag von Univ.-Prof. Dr. med. V. Stepan (Graz)

Kardiovaskuläres Risiko bei subklin. CS

Parameter	normale (n=100)	subklin. CS (n=10)
LDL	~1.5	~1.8
HDL	~1.0	~0.8
triglyceride	~1.0	~1.2
total cholesterol	~2.5	~2.8
LDL/HDL	~1.5	~2.2
triglyceride/HDL	~1.0	~1.5

Chromidol Hyperkortisolismus ist ein wichtiger CV-Risikofaktor!

Stathopoulos et al. JCEM 2008

"Occult Cushing's syndrome" in patients with uncontrolled diabetes mellitus type 2

```

graph TD
    A[200 patients with DM2 (HbA1c > 8.5%)] --> B[26% Cushing's disease]
    B --> C[8.5% Cushing's syndrome]
    C --> D[5.5% Cushing's syndrome and Cushing's disease]
  
```

Copyright © 2003-2008 Krause & Pacherneegg | Technik: Markus Krumpöck

Figure 13: www.kup.at - video stream of a conference presentation including the slides

into HTML documents. While this worked in principal, there were several problems, which brought us to the decision not to take this approach. One problem was that the documents where generated using Adobe Pagemaker, but the stylesheets in Pagemaker for creating the full text files had not been used according to their meaning, but rather according to what they look like. In addition a lot of text has been formatted without using any stylesheet. Therefore the conversion could be done only semi automatically, which would have been too much effort for the employees and therefore is not maintainable. To support the automatic conversion from Adobe Pagemaker respectively InDesign to HTML documents in the future stylesheets would always have to be used according to their meaning. The usage of “Tagged PDF” [114] where the logical structure is expressed via “tags” would also increase the accessibility.

- **High availability** Currently the productive system only consists of one hosts which is a single point of failure. One thing which improves the fault tolerance is a RAID-1 system, where two identical harddiscs are used for mirroring the data - if one of the discs fails the system can still operate using the other disc. The failed disc can be exchanged in the meantime by a new one. The provider offers several internet connections to the internet backbone, therefore a failure of one of the internet connection should not be a problem. Other problems with the hardware, e.g. the memory will cause a system halt, also problems with the software may bring the system down or cause it not to work properly anymore. Therefore some redundant systems would be nice to have in order to ensure business continuity by providing failover if one of the systems fails. The redundant hardware also could be used to reduce the load to each single system by implementing a load balancing mechanism.
- **Load balancing** If the load increases additional servers may be necessary to build a cluster and distribute the load, therefore a load balancing mechanism and synchronization of the data may be necessary, although load performance tests and the round robin database monitor of the web sever show that currently there is no bottleneck with the current memory or central processing unit, the single server can scale for some years based on the growth in the past since nearly all web pages are static and only a few dynamic web pages are in used. The main problem in the past where two downtimes because of some overload situations and therefore system downtimes because of too many

concurrent requests at the same time which filled up the memory, where load balancing would not help that much. This problem has been solved by decreasing the number of maximum concurrent apache servers according to the available memory and by changing to an new hardware with four times as much memory.

For load balancing usually a load balancer (that is additional hardware) is necessary which decides which traffic to route to which host - this load balancer(s) usually are quite expensive, in addition the may be a single point of failure unless you have at least two of them. Therefore the simplest and cheapest approach for load balancing would be to use round robin DNS, which does not need any additional hardware and no single point of failure is introduced. This approach adds for every host in the cluster the IP addresses if the hosts for a single hostname. If there are multiple entries in the DNS for the same hostname the DNS resolution on the client side decides which hosts of the list of hosts to use. According to [117] page 64 this leads to an uniform distribution over a large time period.

Data synchronization between the different hosts could be realized by using Unison file synchronization (in both directions) or rsync (in one single direction) using an encrypted connection via SSH or data replication can be done using DRBD (Distributed Replicated Block Device) [115]. Database replication can be done by using Master/Slave-Replication, where updates to the databases always must be done to the Master and are replicated to all slaves afterwards. If the Master fails another Slave has to be configured as Master, where the switchover could be done manually or automatically. In order to have one single source for logfile analysis a syslog(-ng) server could be configured on one of the hosts, all other hosts log to the syslog server. An alternative technology would be using mod_log_spread (see [116] page 263 and following).

Further concepts to build high performance website can be found in Cal Henderson "Building Scalable Web Sites" [116] (pp. 202-256) and Theo Schlossnagle "Scalable Internet Architectures" [117].

- **Improve generation mechanism.** Since the generation of the complete site takes some hours and since between two times of generating the complete site some documents may not be requested by users at all, in this cases the generation of the document was unnecessary. There-

fore the use of a caching mechanism instead of complete generation of all documents would make sense, that is, to generate the document only when it is requested for the first time and to serve it afterwards from the filesystem cache. In addition it would be nice to have a user interface to be able to regenerate single documents (for example only the summary page for a single scientific article, e.g. by deleting the cached document so it has to be regenerated when requesting it the next time) instead of regenerating the whole section (that is, all summaries for all scientific articles) which would be more efficient and faster.

- **Scalability.** Currently all documents of one document type are stored in one single directory which may cause performance problems with some filesystems (see [116] page 246) since there may be thousands of files of one document type. Newer Linux file systems like ReiserFS and Ext3 on a 2.6 kernel don't have this problem anymore, since they use a hash of filenames to lookup the inode entry, therefore the time for looking up an entry remains relatively constant. In order to avoid such problems anyway a more scalable directory structure may be used (for example such as the one used by the popular mail transfer agent (MTA) postfix for storing mails in the mail queue, where the number of files per directory is constant, for each digit of the ID one subdirectory is created). To achieve better read and write performance, it nearly always a good idea to add the "noatime" flag when mounting a filesystem, where atime is short for access time. By default Unix stores the time a file was last accessed (for read or write), which reduces performance. Adding the noatime flag will give a boost in performance [116], page 247).
- **Automatic bounce management for the newsletter.** At the time of subscribing to the newsletter the user has to enter a valid mail address, which is guaranteed by using double opt in. As time goes by mail addresses may become invalid, since a person may leave a company, the domain name changes, the user changes to another provider, the account has been canceled, the account may be permanently over quota since it is not used anymore by the user and so on. Therefore the newsletter is also sent to some invalid mail addresses which are sent back to the newsletter owner as undeliverable. Currently this bounced mails are handled manually, in order to improve maintenance this should be handled automatically - invalid mail addresses should be removed automatically when mails bounce for several times.
- **Using CSS instead of layout tables.** At time of implementing the

layout CSS was not yet well supported by all browsers in order to do a complete layout. Therefore CSS only has been used for formatting the textual elements but not for the layout - instead nested HTML tables have been used which are hard to maintain. In addition, the layout could not be changed by just modifying one single stylesheet, but rather the layout is fixed in all (currently 144) templates, which makes it hard to change the layout, for example to add a third vertical column for advertising.

- **Text mining.** In order to automatically build an thesaurus a program has been implemented which tries to find common co-occurrences of (key)words, that is words that often appear together and therefore are considered as related in some kind. This implementation has been relatively time consuming and took long to return with the results, in addition it found some very interesting and really related terms, but also some terms which are not related at all and therefore would have to be excluded manually. Although, text mining (find co-occurrences of terms, clustering of related documents, ...) would be a very interesting issue. Some promising results have been delivered by “Morphosaurus”, where we cooperated with the Department of Medical Informatics in Freiburg [118]. The results of Morphosaurus have been integrated in our automatically built thesaurus as well.
- **Automatic generation of summaries.** The Open Text Summarizer [119] is an open source tool for summarizing texts. The program reads a text and decides which sentences are important and which are not. Several academic publications have benchmarked it and praised it. This program could be used to generate summaries of the PDF documents automatically.
- **Improved logfile analysis.** Currently no generic framework or overall concept for web usage mining is in use, for every single point of interest another program to analyze the logfile is implemented. In the future it would be nice to have a general framework to be able to analyze the data like in an Business Warehouse or OLAP system. An interesting question to analyze may be for example the percentage of which PDF documents are loaded by the users (a PDF document may be only loaded partially due to the HTTP Byte-Range retrieval). In order to further improve usability and to meet users needs for example one could offer only the most used options on the regular page (found by web usage mining) and all other options on a page with further options. It may also be of interest if the newsletter mails are actually read - this

could be implemented by using HTML newsletters (currently the mails are sent as plain text, since every client accepts this kind of information) and using pixel graphics or by using links with an unique identifier for each recipient instead of uniform links.

- **Dictionary English-German.** Currently all keywords are translated manually, that is German keywords and English keywords have to be entered for each article. To avoid this additional effort a dictionary for translation of German keywords into English keywords could be implemented, so the user has to enter the translation only once and not for every single article every time a word is (re)used - the user should be only prompted for an translation if the translation does not yet exist.
- **User interfaces to delete and update databases entries.** In order to further improve the database consistency interfaces for updating and deleting databases entries should be implemented since the entity relationship model may be hard to understand because of the growing number of database tables and the user may miss some relationship. Currently only a generic database frontend is used to maintain updates and deletions. In addition an WYSIWYG editor could be implemented, so the user does not have to understand HTML, which is currently necessary for some kind of content (e.g. the editorial board, imprint, guidelines for authors, ...).
- **Offline versions for delivery on CD-ROM or DVD.** For some people it might be interesting to get an offline version of some databases or journals (as for most of them exists a printed copy). Since most content is static this would not be a great deal, the main challenge would be to offer the search functionality.

6 Summary and conclusion / Future Work

6.1 Summary

Since there are a lot of different technologies, frameworks and content management systems for building websites it is hard to choose the one which fits your needs for the current project. Because there are so many of them (which may be optimized for a special purpose or company) a lot of time would be needed to evaluate which is the right one for your project, in addition it may be a lot of effort to learn how to operate, extend or modify this framework or content management system to fit your needs. Therefore content management systems for websites are often implemented from scratch - in this case you need some guidelines how to build a website in order to keep it maintainable with reasonable effort.

In this thesis I developed an approach to build a maintainable website. First, a general overview of the state of the art and the different technologies and frameworks was given, while in the next chapter the different steps and tasks in the website development life cycle were presented with focus on maintainability.

The real world case study done from 2003 to 2008 (which is still ongoing and online) showed how to build such a website and to keep it maintainable by generating as much as possible automatically from one single source. Also the use of static documents instead of dynamic pages has a lot of advantages, which has been made use of in this project. At first it took some effort to convince the customer to build such an overall concept, but in the end - after being online for five years now - it turned out that this concept still works, is scalable and still maintainable without changes to the architecture or concepts. In addition, a lot of new features have been implemented in this five years, some of them also increased the income and were able to improve other commercial figures by adding additional possibilities for sponsors and advertisers, like paid content, advertising in PDF documents, additional space for advertising, sponsoring of video presentations, but also by increasing the traffic and therefore reach more users.

Nevertheless, there is still room for improvement, as we saw at the end of the last chapter.

Spoken in general maintainability is not an “add on” which can be added afterwards if there is enough time at the end of a project phase (which actually

never is) but rather is a permanent process and every design decision has been taken with taking the consequences and effects regarding to maintainability into account.

6.2 Future work

For faster development of maintainable websites it would be desirable to be able to use an integrated framework, which unites the advantages of native programming languages for implementing web sites and web applications (that is, flexibility and the availability of modules for the most important functionality needed for websites) and the functionality of generic frameworks, template engines and content management systems. So, the goal for such a system could be an framework, which offers modules for frequently needed functionality, like newsletters, full text search, generic content management, automatic building of sitemaps and tables of contents, support for internationalization and localization and so on. The behaviour of this modules should be configurable, the layout should be configurable from one source for all modules, so a single informational or navigational element could be reused for different modules. Besides it should be also possible to use your favorite programming language to implement your own business logic respectively web application.

Another goal could be to develop a library with user interface components like the ones offered by Apache MyFaces [25] for different programming languages like PERL and PHP, but independent of the framework or template engine used. This library could ease the development of user interfaces for web applications, since it offers ready to use components for calendars, WYSIWYG editors, pageable and sortable tables or lists, tabs, navigation trees, panels, progress bars, and much more. The layout of this components should be customizable by using cascading stylesheets.

Another approach could be some kind of meta model or abstraction layer for different programming languages, where the meta model could be automatically mapped into any supported programming language, so platform and language independent websites could be implemented. This could be solved by using a framework to support model driven architecture (MDA) web application development by defining the UML model and automatic generation of the source code in different languages (PERL, PHP, Java, ...), so the definition would be language independent and the platform could be changed more easily, so the application would be more portable [44]. For example AndroMDA [45] is an Open Source MDA Generator but only sup-

ports Java, openArchitectureWare [46] is a powerful open source generator framework that can read any kind of model (XMI, XML, any textual representation) and transform it in any kind of textual output. It has an explicit representation of the source metamodel, and uses templates to generate the output. The target languages include C, C++, C#, Java, Perl, Python, Ruby, Javascript, XSLT, JSP, PHP, ASP.NET, VB.NET and much more.

A List of Figures, Tables and Listings

List of Figures

1	Java Model 2 approach	8
2	Cocoon XML pipelining mechanism	9
3	Model 2 X approach using XLST instead of JSP	9
4	www.kup.at - homepage of a journal before the redesign . . .	47
5	www.kup.at - portal for cardiovascular diseases before the re- design	51
6	www.kup.at - database for cardiology in Austria before the redesign	52
7	www.kup.at - homepage of a journal after the redesign	55
8	www.kup.at - fault tolerant database administration frontend to enter keywords	56
9	www.kup.at - user interface to start the automatic generation of the website	59
10	www.kup.at - monitor for the hits per second, traffic, number of busy and idle Apache servers, memory usage and CPU load average	61
11	www.kup.at - search engine for the image database	65
12	www.kup.at - approximate error tolerant search	68
13	www.kup.at - video stream of a conference presentation in- cluding the slides	72

List of Tables

1	Case Study - Statistical Data	63
---	---	----

Listings

1	Sample template for the Template Toolkit	5
2	Calling PERL code for the Template Toolkit	6

References

- [1] Jussi Koskinen. *Software Maintenance Costs*. Information Technology Research Institute, Finland <http://users.jyu.fi/~koskinen/smcosts.htm>, 2003.
- [2] Christoph Bommer, Markus Spindler, Volkert Barr. *Software-Wartung: Grundlagen, Management und Wartungstechniken*. dpunkt, 2008.
- [3] National Center for Supercomputing Applications. *Common Gateway Interface*. <http://hoohoo.ncsa.uiuc.edu/cgi/>
- [4] Comparison of Template Engines. http://en.wikipedia.org/wiki/Template_engine_%28web%29#Comparison_of_template_engines
- [5] Smarty PHP Template Engine. <http://www.smarty.net/>
- [6] Template Toolkit. <http://www.template-toolkit.org/>
- [7] Darren Camberlain et al. *Perl Template Toolkit*. O'Reilly & Associates, first edition, 2004.
- [8] *Comparison of Web application frameworks*. http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks
- [9] Trygve Reenskaug. *Model-View-Controller Pattern*. Xerox PARC, 1978. <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>
- [10] Steve Burbeck. *Applications Programming in Smalltalk-80: How to use Model-View-Controller*. 1987. <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html>
- [11] Model View Controller Frameworks for the Web. http://en.wikipedia.org/wiki/Model-view-controller#Implementations_of_MVC_as_web-based_frameworks
- [12] Java Model 2. <http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html>
- [13] Apache Struts. <http://struts.apache.org/>
- [14] Struts4PHP. <http://www.struts4php.org/>
- [15] Apache Cocoon. <http://cocoon.apache.org/>

-
- [16] Bill Brodgen, Conrad D’Cruz and Mark Gaither. *Cocoon 2 Programming*. Sybex, Inc., 2003.
- [17] Matthew Langham and Carsten Ziegeler. *Cocoon: Building XML Applications*. New Riders, first edition, 2002.
- [18] *Getting Started With Cocoon 2*. <http://www.xml.com/lpt/a/2002/07/10/cocoon2.html>
- [19] Markus Krumpöck. *Serving XML with Apache Cocoon*. Linux Magazine, Issue 06. <http://www.linux-magazine.com/issue/06/ApacheCocoon.pdf>. 2001.
- [20] StrutsCX. <http://it.cappuccinonet.com/strutscx/>
- [21] MyXML. <http://www.infosys.tuwien.ac.at/myxml/homepage.html>
- [22] Apache AxKit. <http://axkit.org/>
- [23] Model 2 X. <http://www.javaworld.com/javaworld/jw-02-2002/jw-0201-strutsxslt.html>
- [24] Java Server Faces (JSF). <http://java.sun.com/j2ee/javaserverfaces/>
- [25] Apache MyFaces. Trinidad, Tobago, Tomahawk. <http://myfaces.apache.org/trinidad/> <http://myfaces.apache.org/tobago/>, <http://myfaces.apache.org/tomahawk/>
- [26] Ruby on Rails. <http://www.rubyonrails.org/>
- [27] Grails. <http://grails.org/>
- [28] Symfony. <http://www.symfony-project.org/>
- [29] CakePHP. <http://cakephp.org/>
- [30] Catalyst. <http://www.catalystframework.org/>
- [31] Grok. <http://grok.zope.org/>
- [32] Comparison of Content Management Systems. <http://www.cmsmatrix.org/>
- [33] http://en.wikipedia.org/wiki/List_of_Content_Management_Systems

-
- [34] Yet Another Multicolumn Layout (YAML). <http://www.yaml.de/en/home.html>
- [35] Mollio CSS/HTML Templates. <http://www.mollio.org/>
- [36] Prototype - a JavaScript Framework. <http://www.prototypejs.org/>
- [37] script.aculo.us - a JavaScript Framework based on Prototype. <http://script.aculo.us/>
- [38] OpenLazlo. <http://www.openlaszlo.org/>
- [39] jsVal. <http://jsval.fantastic-bits.de/>
- [40] Roland Petrasch, Oliver Meimberg. *Model-Driven Architecture: Eine praxisorientierte Einföhrung in die MDA*. Dpunkt Verlag, 2006.
- [41] Klaus Zeppenfeld, Regine Wolters. *Generative Software-Entwicklung mit der Model Driven Architecture*. Spektrum Akademischer Verlag, 2005.
- [42] Georg Pietrek et al. *Modellgetriebene Softwareentwicklung. MDA und MDSB in der Praxis*. Entwickler.Press, 2007.
- [43] MDA Guide Version 1.0.1. <http://www.omg.org/docs/omg/03-06-01.pdf>
- [44] Code Generation Network. <http://www.codegeneration.net/>
- [45] AndromDA Open Source MDA Generators. <http://www.andromda.org/>
- [46] openArchitectureWare. <http://www.openarchitectureware.org/>
- [47] ISO (1991). *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.
- [48] IEEE 90. *Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York, NY: 1990.
- [49] Jakob Nielsen. *Usability 101: Introduction to Usability*. August 25, 2003 <http://www.useit.com/alertbox/20030825.html>
- [50] Jakob Nielsen. *Designing Web Usability*. New Riders, third edition, 2000.

-
- [51] Reusability. <http://en.wikipedia.org/wiki/Reusability>
- [52] Web Accessability. http://en.wikipedia.org/wiki/Web_accessibility
- [53] Web Accessibility Initiative (WAI). <http://www.w3.org/WAI/>
- [54] World Wide Web Consortium. <http://www.w3.org/>
- [55] Erich Gamma et al. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1999.
- [56] Andrew Hunt and David Thomas. *Der Pragmatische Programmierer*. Carl Hanser Verlag, 2003.
- [57] Martin Fowler et al. *Refactoring. Improving the Design of Existing Code*. Addison-Wesley, 1999.
- [58] Refactoring. http://en.wikipedia.org/wiki/Code_refactoring
- [59] http://en.wikipedia.org/wiki/Don't_repeat_yourself
- [60] Louis Rosenfeld and Peter Morville. *Information Architecture for the WWW*. O'Reilly & Associates, second edition, 2002
- [61] Jason Beaird. *The Principle of Beautiful Web Design*, Sitepoint, 2007.
- [62] Jakob Nielsen. *Why Frames Suck (Most of the Time)*. <http://www.useit.com/alertbox/9612.html>. December 1996
- [63] Jakob Nielsen. *Top 10 Web Design Mistakes of 2003*. <http://www.useit.com/alertbox/20031222.html>
- [64] Google Sitemaps. <https://www.google.com/webmasters/sitemaps/>
- [65] Swish-E. <http://swish-e.org/>
- [66] htdig. <http://www.htdig.org/>
- [67] Perlfect Search. <http://www.perlfect.com/freescripts/search/>
- [68] http://en.wikipedia.org/wiki/Internationalization_and_localization
- [69] Content Negotiation. http://en.wikipedia.org/wiki/Content_negotiation

-
- [70] Apache Content Negotiation. <http://httpd.apache.org/docs/2.0/content-negotiation.html>
- [71] Fielding, et al. *Hypertext Transfer Protocol - HTTP/1.1 - 12 Content Negotiation*. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec12.html>
- [72] Cascading Style sheet (CSS). http://en.wikipedia.org/wiki/Cascading_Style_Sheets
- [73] PHP Hypertext Processor. <http://www.php.net/>
- [74] Rasmus Lerdorf and Kevin Tatroe. *Programming PHP*. O'Reilly & Associates, Inc., first edition, 2002.
- [75] George Schlossnagle. *Advanced PHP Programming*. sams, 2004.
- [76] PERL - Practical Extraction and Report Language <http://www.perl.org/>
- [77] What is dynamic and static? - a definition from Whatis.com. http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci348104,00.html
- [78] Static and dynamic webpage designing. <http://ezinearticles.com/?Static-And-Dynamic-Webpage-Designing&id=1199113>
- [79] GNU Wget. <http://www.gnu.org/software/wget/>
- [80] GNU coding standards. <http://www.gnu.org/prep/standards/standards.html>
- [81] Database vs. Filesystem for storing files. http://en.wikibooks.org/wiki/Programming:WebObjects/Web_Applications/Development/Database_vs_Filesystem http://asktom.oracle.com/pls/asktom/f?p=100:11:0:::::P11_QUESTION_ID:1011065100346196442
- [82] HTML Tidy. <http://tidy.sourceforge.net/>
- [83] PERL Tidy. <http://perltidy.sourceforge.net/>
- [84] PHP Beautifier. http://pear.php.net/package/PHP_Beautifier
- [85] phpDocumentor. <http://www.phpdoc.org/>
- [86] Javadoc. <http://java.sun.com/j2se/javadoc/>

-
- [87] Doxygen. <http://www.stack.nl/~dimitri/doxygen/>
- [88] Pankaj Kumar. *MyGoogle: A Simple Cocoon Application that Uses Google's SOAP API*. <http://www.pankaj-k.net/sdwest2002/readme.html>
- [89] AxisRPCReader. <http://cocoon.apache.org/2.1/userdocs/optional/axisrpc-reader.html>
- [90] Apache Axis. <http://ws.apache.org/axis/>
- [91] JUnit. <http://www.junit.org/>
- [92] PHPUnit. <http://www.phpunit.de/>
- [93] HttpUnit. <http://httpunit.sourceforge.net/>
- [94] Apache JMeter. <http://jakarta.apache.org/jmeter/>
- [95] ab - Apache HTTP server benchmarking tool. <http://httpd.apache.org/docs/2.3/programs/ab.html>
- [96] RFC822 - Standard for the format of ARPA Internet text messages. <http://www.freesoft.org/CIE/RFC/1123/100.htm>
- [97] ModSecurity: Open Source Web Application Firewall. <http://www.modsecurity.org/>
- [98] IPTC Information Interchange Model. <http://en.wikipedia.org/wiki/IPTC-NAA-Standard>
- [99] Zebra - Z39.50 server. <http://www.indexdata.dk/zebra/>
- [100] Open Archive Initiative. <http://www.openarchives.org/>
- [101] OAI-PMH2 XMLFile File-based Data Provider. <http://www.dlib.vt.edu/projects/OAI/software/xmlfile/xmlfile.html>
- [102] agrep - Implementation for Unix. <ftp://ftp.cs.arizona.edu/agrep/>
- [103] S. Wu and U. Manber, *Agrep - A Fast Approximate Pattern-Matching Tool*. Usenix Winter 1992 Technical Conference, San Francisco (January 1992), pp. 153-162.
- [104] phpMyAdmin. <http://www.phpmyadmin.net/>
- [105] rsnAPSHOT. <http://www.rsnAPSHOT.org/>

-
- [106] ftplicity. <http://www.heise-online.co.uk/security/Backups-on-non-trusted-FTP-servers--/features/79882>
- [107] duplicity. <http://duplicity.nongnu.org/>
- [108] Round Robin Database. <http://oss.oetiker.ch/rrdtool/>
- [109] Ivan Ristic. *Apache Security*. <http://www.apachesecurity.net/>, O'Reilly Media, 2005.
- [110] mon - Service Monitoring Daemon. http://mon.wiki.kernel.org/index.php/Main_Page
- [111] logcheck - a logfile scanner. <http://logcheck.org/>
- [112] Unison, a file synchronization tool. <http://www.cis.upenn.edu/~bcpierce/unison/>
- [113] cron-apt. automatic update of packages. <http://packages.debian.org/etch/cron-apt>
- [114] What is Tagged PDF? <http://www.planetpdf.com/enterprise/article.asp?ContentID=6067>
- [115] Distributed Replicated Block Device. <http://www.drbd.org/>
- [116] Cal Henderson. *Building Scalable Web Sites*. O'Reilly Media, first edition, 2006.
- [117] Theo Schlossnagle. *Scalable Internet Architectures*. Sams, 2006.
- [118] Morphosaurus. <http://morphwww.medinf.uni-freiburg.de/>
- [119] OpenTextSummarizer. <http://libots.sourceforge.net/>