An E-commerce Platform Evaluation based on the DoSAM Framework

OSKAR LIND

KTH Teknik och hälsa

Degree thesis in Computer Technology
Stockholm, Sweden 2012

An E-commerce Platform Evaluation based on the DoSAM Framework

Utvärdering av e-handelsplattformar med ramverket DoSAM

OSKAR LIND

Degree thesis in Computer Technology (15 credits)
at Softronic AB
Royal Institute of Technology year 2012
Supervisor at Softronic AB was Jimmy Thulin
Examiner was Reine Bergström

TRITA-STH 2012:44

KTH School of Technology and Health
KTH Haninge
Marinens väg 30
SE-136 40 Handen
http://www.kth.se/sth

Abstract

The goal of the thesis work was to give Softronic AB, a management and IT consulting company, an overview of strengths and weaknesses of the four content management systems Magento CE 1.6.2, Episerver Commerce 1 R2 SP1, Wipcore eNOVA 5.3 and Umbraco CMS, as platforms for e-commerce. The evaluation was to be made from a marketing perspective as well as from a user and in-house developer's perspective. Additionally, the goal was to provide an overview of how certain functionality from a current project could be implemented on the Magento CE platform.

To be able to evaluate the platforms, a study of current evaluation and comparison models was performed. This resulted in the choice of the framework Domain-specific Architecture Comparison Model. A comparison framework was derived and defined from this model, and applied on the platforms. The choice of what was to be measured, and how, was determined from literature studies of the e-commerce domain, as well as studies of current e-commerce platforms. The demonstration of the capabilities of the Magento platform was solved by creating a new Magento store instance, implementing two components from a reference project, as well as parts of the graphical user interface, and deploying the solution on a cloud server.

The evaluation resulted in a description of the platforms' strengths and weaknesses, and a table showing the properties of the platforms. The evaluation indicated that the platforms Magento CE and Episerver Commerce were the strongest when it came to functionality and market readiness. Wipcore eNOVA and Umbraco CMS's strongest side was performance efficiency, while none of the platforms were assessed as particularly strong in terms of maintainability.

Sammanfattning

Målet med examensarbetet var att ge Softronic AB, ett konsultbolag inom IT och management, en översikt över vilka styrkor och svagheter de fyra CMS-plattformarna Magento CE 1.6.2, Episerver Commerce 1 R2 SP1, Wipcore eNOVA 5.3 samt Umbraco CMS hade som e-handelssystem. Utvärderingen skulle ske dels utifrån ett marknads- och kundperspektiv men också från ett utvecklarperspektiv inifrån företaget. Målet var också att ge en översikt över hur vissa funktioner för ett befintligt projekt kunde implementeras på plattformen Magento CE.

För att utvärdera plattformarna gjordes en litteraturstudie över befintliga jämförelsemodeller, vilket resulterade i valet av ramverket Domain-specific Software Architecture Comparison Model. Ett jämförelseramverk definierades utifrån modellen och applicerades på plattformarna. Vad som skulle utvärderas och hur det skulle mätas bestämdes utifrån litteraturstudier av affärsområdet samt studier av befintliga e-handelsplattformar. Demonstrationen av Magentoplattformen löstes genom att sätta upp en Magento-installation på en molnserver och implementera två komponenter samt delar av det grafiska användargränssnittet för ett referensprojekt.

Evalueringen resulterade i en beskrivning över plattformarnas styrkor och svagheter samt en överskådlig tabell som kan vara till nytta för Softronic i inledningsskedet av framtida e-handelsprojekt. Evalueringen visar att plattformarna Magento CE och Episerver Commerce är starkast när det gäller funktionalitet och att snabbt få ut en lösning på marknaden. Wipcore eNOVA och Umbraco CMS har styrkor när det gäller prestanda, medan ingen av plattformarna bedömdes vara starka när det gäller egenskaper som underlättar underhåll och utveckling.

Contents

1 About this document

The goal has been to evaluate four e-commerce platforms for the company Softronic, and to demonstrate the abilities of the Magento platform. In an attempt to achieve objectivity, a comparison framework has been developed first, to establish what should be measured, and how. The Magento platform has been demonstrated by implementing certain features found in an existing project built on another platform.

The reader should keep in mind that the evaluation was intended to be as objective as possible, however the framework and the metrics used are nevertheless focused on the needs of Softronic. It could therefore be considered less useful for other businesses than Softronic, although the description of the process and the solution method might be of interest to someone who is in the process of selecting a suitable evaluation method for software platforms.

The documents starts with a background (ch. 2) and an introduction of the goals and limitations (ch. 3). An introduction to the platforms is provided in ch. 4.

Chapter 5 provides an overview of different comparison frameworks and a background on why the DoSAM framework was selected for this comparison. This is followed by the actual establishment of the comparison framework (ch. 6), describing what attributes will be measured and in what way, and also how they will be weighed against each other. Chapter 7 describes the appliance of the metrics on the selected platforms, and presents the individual results for the platforms.

Chapter 8 presents possible solutions to the Magento demonstration goals, by introducing the platform from a developer's point of view. It also contains a description of how the problem of implementing selected components of the Mataffären.se system on the Magento platform was solved.

The last part of the document (chapters 9 and 10) summarizes the results of the evaluation and provides a brief overview of the strengths and weaknesses found in the platforms. The solution method is also discussed and evaluated.

Some knowledge on programming and system architecture is required from the reader of the document.

As a last note, it must be emphasized that e-commerce is a business area that it changing very fast. During the thesis work, two out of the four platforms in the comparison were released in new versions that changed the platforms' technical standard considerably (Magento CE 1.7 and Umbraco CMS 5.1). The reference solution, Mataffären.se, was also upgraded to use a different version of Wipcore eNOVA, and the solution was thereby subject to a number of architectural changes. Wipcore even announced in May 2012 that they were planning to integrate Umbraco CMS into the eNOVA platform[1]. The consequence of this is that the comparison and implementation could be considered obsolete already at the time of publishing, and the document should be read with these facts in mind.

All figures were published with permission from their respective owners.

1 Wipcore/Enova leaks. *Kommande release for WebFoundation.*

2 Background

E-commerce is an emerging business area, growing by large numbers every year. According to a recent report, the turnover for e-commerce retailing in Sweden increased by 10 percent from 2010 to 2011, to a total of almost 28 billion SEK, and is estimated to reach 31 billion SEK by the end of 2012[2]. It can be assumed that the market for e-commerce platforms is growing with the sector as a whole, and that the number of platforms is continuing to increase. For companies in the business of designing e-commerce systems for external clients, it is also safe to assume that the market situation requires them to stay "on their toes" and to update their offerings to include new platforms and technologies. One of the companies that currently operates in the area of e-commerce is *Softronic*.

Founded in 1984, Softronic is a consulting firm that offers services within management and IT. Their products and services reaches from consulting guidance and tailored software development to administration and operation of systems. Customers are midsize and large businesses and organizations in northern Europe[3]. Softronic currently offers e-commerce solutions mainly based on two platforms, but would like to extend their offering and explore other platforms. The company needs a review of existing alternatives to the platforms they are currently using, and also to compare them to each other with regards to functionality and other properties.

One of the platforms that have been growing in popularity since its 1.0 release in 2008 is Magento. One of the interesting factors about Magento, from Softronic's point of view, is that it is open source software. This is not the case with the platforms that the company currently works with, which makes it an interesting candidate for participating in a comparison with other platforms.

Comparing and evaluating software has, however, proven to be a complicated subject and it can be difficult to achieve an objective perspective. It it argued that many quality attributes, such as maintainability and usability, are affected by subjective factors and therefore cannot be measured objectively[4]. A suitable comparison framework is therefore needed, to ensure that Softronic's specific platform needs are taken into account while still keeping the perspective generic enough to be able to apply non-subjective metrics. Another way of saying it would be that the comparison has to be as application-independent as possible and instead performing the comparison more on an architectural level, while still allowing for attributes specific to e-commerce implementations to be measured, compared and brought to the surface.

If a suitable comparison framework can be established, good opportunities should exist for contributing to domain knowledge by establishing a comparison framework specifically for e-commerce platforms, based on evaluation methods whose components and metrics are fetched from the demands on e-commerce today. This new framework could potentially be helpful not only for Softronic but also for other developers and businesses.

2 HUI Research AB. *E-barometern* (2011).
3 Softronic AB. *Softronic.se* (2012).
4 Sommerville, I. (2011), p. 669.

3 Introduction

Softronic has a need for an increased knowledge and improved strategies around e-commerce projects. Especially, there is a need for increased knowledge on the non-technical aspects such as customer perception and market-oriented features. One way of contributing to increasing this knowledge would be to analyze different e-commerce platforms to find out how, and if, they can be compared, from perspectives that are relevant to both customers and developers. A recently completed project, Mataffären.se, could be used as a reference for assessing how the conditions for implementation differ between platforms.

3.1 Goals

The purpose of this thesis is to analyze and compare four different e-commerce platforms: Umbraco, eNOVA, Episerver Commerce and Magento. The comparison shall present an overview of the platforms from a number of perspectives which are defined from literature studies, interviews with employees and field studies of existing e-commerce platforms.

The thesis is also expected to deliver a brief conclusion and comparison chart which will serve as an overview of the capabilities of the four platforms and which can be used in the early stages of e-commerce project.

Also, because Softronic is especially interested in analyzing Magento, a selection of components from the project Mataffären.se, currently based on Wipcore eNOVA, shall be re-implemented on the Magento platform. The implementation shall be limited to two parts: a frontend and backend. The frontend part should demonstrate how to design and implement a store frontend view. The backend should include the development of two components: a module for fetching product attributes through web services, and a module for sending and receiving shipping information to/from an external logistics partner.

3.2 Limitations

Only the platforms Episerver Commerce, Magento, Umbraco and Wipcore eNOVA should be analyzed. Work should be completed by June 1:st.

The Magento implementation should be limited to a selected set of features determined as necessary to provide an overview of the capabilities of the platform.

3.3 Solution methods

To make the comparison as objective and generic as possible, a comparison framework will be used. Applicable scenarios and relevant measurable attributes should be determined and developed primarily by conducting a field study in applicable databases and by looking at current demands for systems in the application domain. Developers at Softronic can also be interviewed to add relevant perspectives. The data shall be collected using interviews and applicable measurement methods. The evaluation data shall be presented separately along with results from the interviews.

The comparison should result in a short overview of the capabilities of the e-commerce platforms. This overview should be usable for both clients and developers in early stages of e-commerce projects, when the advantages and disadvantages of different platforms are considered.

Development of the Magento components should be performed in a suitable IDE such as Aptana

studio or Netbeans. The latest version of Magento Community Edition should be installed on a LAMP (Linux, Apache, MySQL, PHP) stack, locally or on a remote server.

3.3.1 Implementing Mataffären.se in Magento CE 1.6.2

Because Softronic is interested in analyzing the platform Magento, an implementation of a recent project, “Mataffären.se”, will be implemented on this platform. Only a selection of the functionality of the project will be implemented. The implementation should be split up into a frontend-, a backend and a module-section.

3.3.1.1 Frontend

The store should feature a start page to demonstrate how layouts are used in Magento and how to work with styles and design, and how the content management system works. The store should also have multiple categories with products in them, to demonstrate product listings and category design. Product pages should also be visible with the purpose of displaying how to work with product attributes in the view.

3.3.1.2 Backend

The backend demonstration should focus on the plugin system called Magento connect, how to administrate products, customers and orders, which CRM related features that are available, and how an administrator can make changes to the configuration.

3.3.1.3 Modules

In this section one or several modules should be developed to display the abilitiy of extending the Magento platform with custom modules. The module(s) should attempt to replicate existing functionality in the Mataffären.se platform. Two components has been selected for implementation in Magento:

- A module should be developed that connects to the OPV product database, so that product attributes such as images, descriptions and other attributes can be updated automatically.
- A second modules should be developed that simulates a connection to the delivery agent Widrikssons. The module should export a list of orders and retreive a new list that contains batch numbers and sort orders. The list should be stored locally and be made available to other components in the system.

The module development should be documented and aimed at developers, providing a good overview over how module development is done in Magento.

Apart from the three parts “frontend”, “backend” and “modules”, no other components should be implemented.

4 Software platforms and application frameworks

This section introduces the software platforms and application frameworks utilized by the e-commerce frameworks in this study. Unless otherwise stated, the information on the frameworks are based on the web sites of the solutions. Addresses to the sites can be found in the References section.

4.1 Episerver Commerce

In this document, Episerver Commerce 1 R2 SP1 is analyzed. The platform runs on Windows Server, uses Microsoft SQL Server 2008 as database storage solution, and is implemented using the C# .NET framework 3.5 SP1.

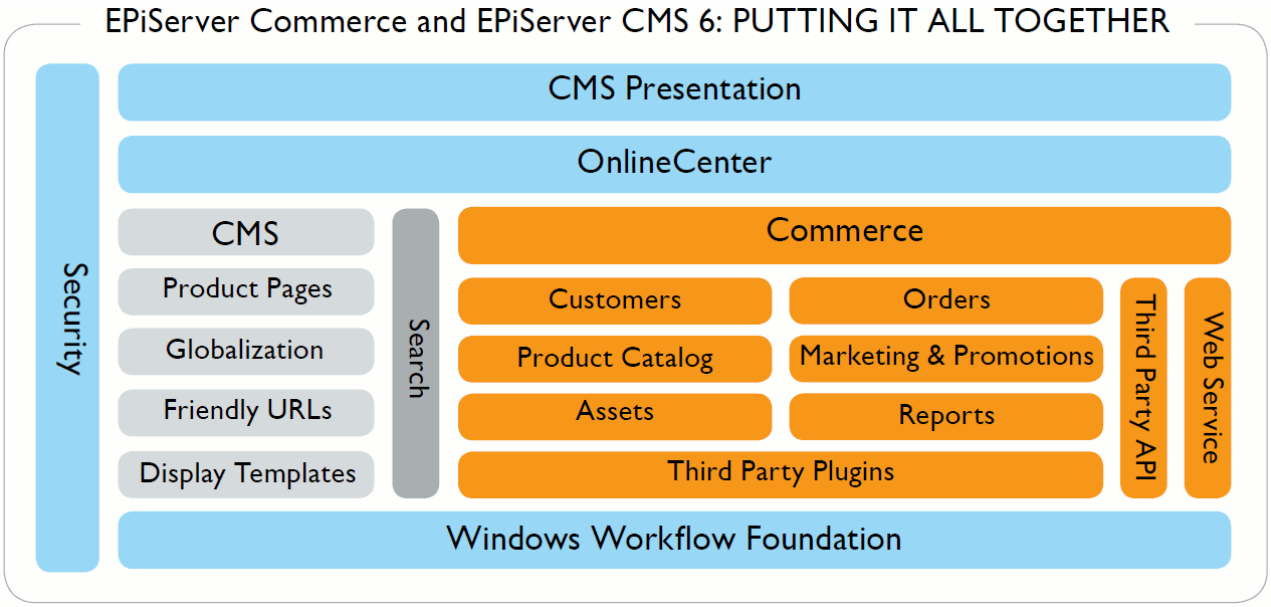

Fig. 4.1: The components of Episerver Commerce and its relationship to Episerver CMS (Episerver, 2012).

Episerver Commerce is based on the Episerver CMS, a content management system, which is required for the installation. Fig. 4.1 shows the components of Episerver Commerce and its relationship to the CMS. Episerver Commerce uses the CMS for presentation, and an additional module, “Commerce Manager”, for store maintenance and order processing. The platform uses two different databases for content storage and e-commerce storage.

Episerver AB, the company behind the product, is based in Sweden. Episerver CMS is used in many intranets and extranets for other purposes than e-commerce, and is used primarily by businesses on the Swedish market. Episerver has a network of partners which adapts the systems for its clients and develops modules.

4.2 Wipcore eNOVA

Wipcore eNOVA is built on C# .NET, with the latest version being 5.3, launched in November 2011. This is also the version that will be used in the comparison. Wipcore eNOVA uses a

Web/Business server running IIS on Windows Server OS, and a database server running on Windows Server OS with Microsoft SQL Server.

Wipcore offers the eNOVA platform as a fully commercial product, meaning there is no open source alternative of the platform. Developer documentation is available online. Onsite training and certification is offered for developers. Wipcore is a Swedish-based company which has led to most of its clients and partners being located in Sweden.

The eNOVA backend differs from many other e-commerce platforms in that it comes with a native Windows client for administration. The platform also features version control and scheduled publishing, which traditionally is offered only on pure CMS systems and not in e-commerce systems. The reason for this is that Wipcore originally was a pure CMS platform, and has since evolved into offering additional e-commerce functionality.

4.3 Umbraco

Umbraco is an open source CMS written in C#. Umbraco commonly sits on top of a software stack consisting of the .NET framework, the Microsoft SQL Server and XSLT, although the platform supports a number of different relational databases for storage. Introduced in 2000, the latest stable release is now 5.0.1, with version 5.1 to be released as a stable version soon. Umbraco is typically deployed on an IIS server. The latest version of Umbraco runs on the ASP.NET MVC3 framework. However, it is assessed that the 5.x version of Umbraco has not been out long enough to be able to use in this evaluation. For this reason, it is determined that the version 4.7.2, the latest version of the 4.x branch, will be used.

Umbraco has a fairly active community and is one of the most popular .NET based CMS systems. Although Umbraco is mainly a content management system and not an e-commerce platform, the open source nature of the framework makes is suitable for extending and merging into different specializations. There are a number of modifications available for Umbraco which adds e-commerce functionality as a plugin.

There are three different implementation packages;

- “Umbraco CMS”, which is the free version
- “Confidence” - a commercial package offering more support and bug fixing warranty
- “Complete” - a lower priced commercial alternative offering a number of add-ons such as a forms builder and a deployment service.

Umbraco is distributed under the MIT license.

The key difference between Umbraco and the other platforms in the evaluation is that Umbraco has no official e-commerce package. Instead, plugins and extensions are available that adds e-commerce functionality to the platform. A brief survey found six different e-commerce packages:

- uCommerce (www.ucommerce.dk)
- TeaCommerce (www.teacommerce.dk)
- commerce4umbraco (http://commerce4umbraco.codeplex.com)
- uWebshop
- Umbraco E-commerce Extension

- Procure eCommerce Engine

From investigating the above alternatives in more depth, it is found that only two of the (uCommerce and TeaCommerce) have been released in stable versions (non-beta). While TeaCommerce requires Umbraco CMS version 4.5 or newer, uCommerce supports all 4.x versions. The .NET and SQL Server requirements are virtually identical, and they both have, at the time of writing, released new versions within the last month (April 2012).

They only major difference that can be found between uCommerce and TeaCommerce is the licensing alternatives. TeaCommerce only offer one type of license, while uCommerce offer three differently prices alternatives; “free”, “pro” and “enterprise”.

With the two above alternatives showing no obvious differences in functionality and system requirements, the question of which e-commerce extension to choose as representative for the Umbraco platform does not have a clear answer. They both offer developer's licenses to enable trial versions to be implemented and tested before buying a license. The licensing options can, however, be seen as a distinct difference, with TeaCommerce only offering one licensing options. This enables for less variables in the comparison since only, and not three, variants of the platform can be evaluated. For this reason, TeaCommerce has been chosen as the extension on which to base the Umbraco evaluation.

Nevertheless, it must be noted that uCommerce should also be evaluated in the future, to provide a broader overview and add perspectives on the Umbraco platform evaluation.

4.4 Magento

Magento is an open source e-commerce platform built on PHP and the ZEND framework. The company behind Magento, Varien, was founded in 2004 and the 1.0 version of Magento was launched in 2008. In 2011, Magento Inc. was acquired by eBay.

Magento is currently offered as four different packages;

- Magento Community Edition, which is the open-source, free version
- Magento Enterprise, offering support, encryption and several market-centered features not found in Magento Community Edition
- Magento Enterprise Premium, with all of the standard Enterprise features plus consulting services, training and multiple licenses
- Magento Go, a hosted solution for small enterprises.

The platform is typically implemented on a LAMP (Linux, Apache, MySQL and PHP) stack and uses the Entity-attribute-value database model to store data.

In this document, the latest stable release of Magento CE (community edition) will be evaluated which is 1.6.2.0.

Magento offers a market for plugins called Magento Connect. This allows for external developers to offer extensions and modules that improves the functionality of the platform, such as checkout customizations, additional administration features and frontend templates/skins. Magento CE is distributed under the OSL v3 license.

Magento uses components from the Zend framework, an open source web application framework written in PHP5. It is object-oriented, uses a loosely coupled architecture and offers an MVC

structure for implementations. Zend comes with a number of ready-to-use components such as forms validation, a high performance MVC implementation, HTML rendering and filtering, and a database abstraction layer. Zend requires a web server with PHP 5.2.x[5]. Magento developers are encouraged to use the Zend Studio IDE for development, however this is not free software. Another option is to use Eclipse with PDT (PHP Development Tools). Zend is distributed under the New BSD License[6].

Zend Technologies, the company behind Zend framework, offers a customized PHP stack called Zend Server, which is optimized for Zend implementations. The package features improved caching, diagnostics and an improved PHP installation. Zend Server is available as a free community edition or as a commercial product which includes support and other extra features[7].

5 Zend Technologies Ltd. *Zend Programmer's Reference Guide.*
6 Zend Technologies Ltd. *Zend New BSD Licence.*
7 Zend Technologies Ltd. *Zend.com*.

5 Comparison framework and evaluation method

Before determining how to compare the platforms, it can be necessary to define what software evaluation actually is. Banani, R. and Graham, N. say the following:

> *Software architectural evaluation is a technique which determines properties of software architectures, or software architectural styles, or design pattern by analyzing them.*[8]

The authors also state that architectural evaluation is about verification of the properties of the system so that it fulfills the specifications required of it, and also about determining to which degree an architecture satisfies the quality requirements.

For this study however, a comparison method is needed that does not evaluate the implementations but rather the architecture itself, and which only evaluates features that can be considered common enough among e-commerce platforms that the framework will be applicable to all of them. This way of analyzing software platforms differ from most studies that has been found, especially regarding e-commerce platforms, where the comparison tends to fall towards concrete "can/cannot"-style evaluations, emphasizing on functionality. However, considering that in the case of Softronic most platforms are heavily modified and customized to a client's needs, it's not a question of what a platform can or cannot do in its basic form but rather how easily certain functionality can be implemented and what preconditions the developer faces. It is assumed that this "soft" way of evaluating a platform requires a comparison framework that is abstract, modifiable but still concrete enough to be able to draw conclusions around the results.

In "Methods for Evaluating Software Architecture" (Roy and Graham, 2008), a number of software evaluation methods are evaluated, categorized based on at which stage of a project they're applied, and if they are scenario-based or based on a mathematical model. Roy/Graham provides an overview of when a certain evaluation method is suitable, and also gives a brief description on how the respective methods are applied.

"A Survey On Software Architecture Analysis Methods" by (Dobrica and Niemelä, 2002), provides valuable insight into evaluation techniques and analysis methods, as well as adding perspectives on some evaluation methods already mentioned by Roy/Graham, such as *SAAM, SAAMCS, ESAAMI, ATAM and SBAR.*

These two publications will be used as the primary sources for the next section, where the evaluation methods are described in more detail.

5.1 Overview of evaluation methods

The common denominator for all evaluation methods is that they require stakeholder-participation, requirements list, and a description of the architecture and its artifacts[9]. The evaluation models can be identified as *Early methods*, where the planned system design is evaluated, and *Late methods*, where the progress is evaluated to determine whether the present design will be able to meet the specifications. Late evaluation methods can use data measured on the actual implementation. The methods can also be categorized as

- *scenario-based*, an *early* evaluation method where a scenario is a brief description of a

8 Banani R., and Graham, N. (2008), p. 16.
9 Banani R., and Graham, N. (2008), p. 8.

single interaction of a stakeholder with the system,

- *mathematical model-based*, another *early* evaluation method where a quantitative assessment is emphasized over the more qualitative assessment-style of the scenario-based based methods,
- *metrics-based*, a *late* evaluation method which uses metrics to prevent degeneration of a system by measuring and correcting deviations, and
- *tool-based*, another *late* evaluation method where algorithms and tools are used to automatically check compliance of design and source code.

For this study, the early scenario-based methods seems the most interesting since they are meant to be used in the beginning of a project, allows for a qualitative analysis approach, and features a lot of well-known and previously verified methods such as SAAM[10].

5.1.1 Early Scenario-based methods

According to Roy/Graham[11] and Dobrica/Niemelä[12], the methods can be summarized as follow:

- *SAAM (Scenario-based Software Architecture Analysis Method)* is often seen as the "mother" of all scenario-based methods. Simply put, the method compares the properties of a system with requirement documents based on the desired properties of the application. The main characteristic of the method is that quality attributes are concretisized in the form of scenarios. However, it is considered that modifiability is the quality attribute analyzed by SAAM.
- *ATAM (Architecture-based Tradeoff Analysis Method)* has evolved as a way for understanding the tradeoffs in software architecture, as often improving one quality attribute comes at a price of reducing another. Dobrica/Niemilä also adds that ATAM can be considered a framework for different evaluation techniques depending on the quality attributes. It integrates the best individual theoretical model of each attribute in an efficient and practical way.
- *SBAR (Scenario-Based Architecture Reengineering)* provides tools for transforming an architecture and measure the cost and gain of this transformation, based on quality attributes. According to Dobrica/Niemilä, the specific goal of the method is to estimate the potential of the designed architecture to reach the software quality requirements.
- *SALUTA (Scenario-based Architecture Level Usability Analysis)* specializes on assessing usability quality attributes, being the only evaluation method that considers usability before implementation of an architecture. SALUTA extracts two types of information from an architecture: Usability patterns (design patterns that are used to solve a particular scenario) and Usability properties (architectural decisions affecting the usability).
- *SAAMCS (SAAM for complex scenarios)* is, as it sounds, an extension of SAAM for handling complex implementation scenarios. Roy/Graham states that its main intention is to expose the boundaries of an architecture and to thereby determine its flexibility. This can be used to assess which scenarios can be hard to realize on the architecture. Dobrica/Niemilä expresses that the only goal of SAAMCS is risk assessment, noting that the purpose of the

10 Banani R., and Graham, N. (2008), 20.
11 Banani R., and Graham, N. (2008), 19-36.
12 Dobrica, L. and Niemela, E. (2002), p. 641-648.

method is to extend the SAAM method and to add specific perspectives rather than offering a broad perspective.

- *ESAAMI (Extending SAAM by Integration in the Domain)* aims to facilitate reuse of scenarios to reduce cost of knowledge-intensive activity.
- *ASAAM (Aspectual Software Architecture Analysis Method)* is more aspect-oriented than SAAM. It purpose is to identify problems that can occur when scenarios span across multiple methods.
- *SACAM (Software Architecture Comparison Analysis Method) and DoSAM (Domain Specific Software Architecture Comparison Model)* were specifically developed for the comparison itself, and to provide a standardized way of performing the comparison so that it would be independent of the person performing it. SACAM determines a standard architectural view so that the candidates can be compared at a common level of abstraction. This, however, can be problematic when comparing architectures from different domains as it can be difficult finding the right abstraction level. For this purpose, DoSAM was developed to facilitate comparison of architectures within a specific domain.

From the above descriptions, no comparison framework mentioned by the two publications (Banani R./Graham, N., 2008) and Dobrica/Niemellä, 2002) appears to be specifically designed for comparison of e-commerce platforms. A survey of publications around evaluation of e-commerce platforms also did not suggest that any comparison framework was more used than others for this software area.

However, as one the goals of the thesis are to analyze platforms from the e-commerce domain, the DoSAM framework can be regarded as suitable because of its emphasis on facilitating the comparison of architectures from a specific domain. The following sections describes the method in more detail.

5.2 DoSAM – Domain-specific software architecture comparison model

Roy/Graham concludes in their study of comparison frameworks that many methods don't allow standard frameworks for comparing several architectures. Most are focused on evaluating a single architecture at a given point in time, and the comparison results are often highly dependent on the person performing the evaluation. The authors state that the aim of the DoSAM method is to provide the rationale for an architecture selection process by comparing the fitness of candidate architectures.

DoSAM was introduced by Bergner, K., Rausch, A., Sihling, M. and Ternité, T. in 2005 as part of a conference on the quality of software architectures. The authors state that:

> *The method provides an evaluation framework for comparing different software architectures in a certain domain. After adapting this framework to the application domain at hand, it can then be used repeatedly for all future evaluations in a methodical and reproducible way*[13].

The DoSAM method defines two parts of an evaluation: the establishment of a Domain Architecture Comparison Framework (DACF) and a Concrete Architecture Evaluation (CAE). Sections 5.2.1 and 5.2.2 aims to describe these two steps of the method and are, unless otherwise stated, based on the original DoSAM article by Bergner et al.

13 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 1.

5.2.1 DACF (Domain Architecture Comparison Framework)

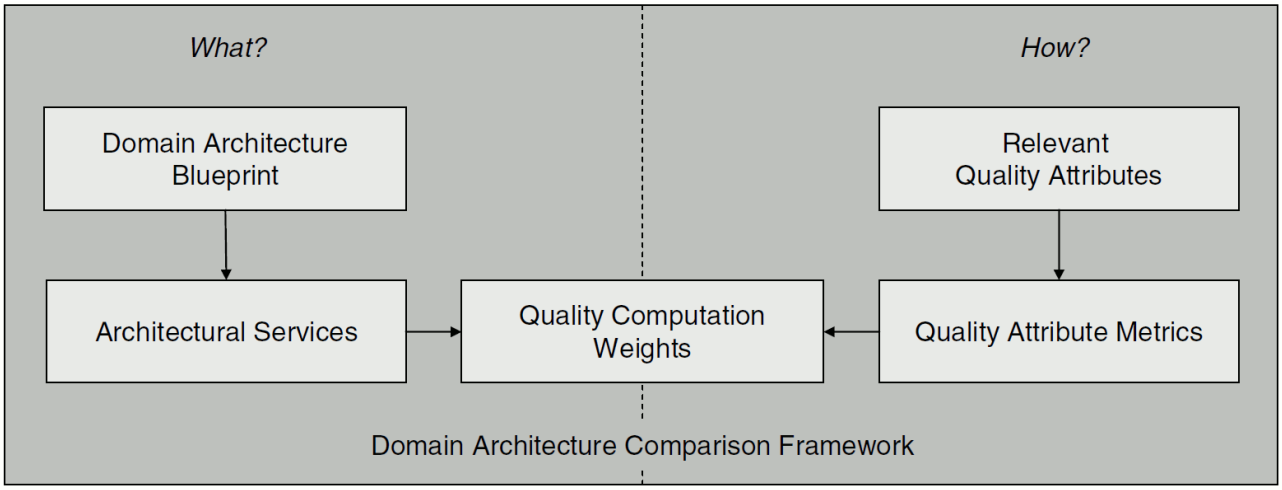

Fig. 5.1: Domain Architecture Comparison Framework (Bergner et al., 2005).

The DACF is performed in three parts:

1. Establishing the *domain architecture blueprint* and the *architectural services*, which combined serves as an abstract description schema for all architectures in the application domain. In this stage, it is important to find the right level of abstraction so that it's representative of the applications in the domain and still able to identify distinct differences between the applications. The proper architecture blueprint is determined by a domain specialist.
2. Establishing Quality Attributes (QA) and Quality Attribute Metrics (QAM), which serves as a representation of which qualities are assessed and how the assessment is performed.
3. Establishing a Quality Computation Weight matrix, which is provided to state the relative importance of each application of a QAM on an architectural service.

5.2.2 CAE (Concrete Architecture Evaluation)

The concrete evaluation of the application is performed using the established DACF. This is also a three-part activity:

1. The architecture of the application is related to the blueprint, yielding the set of all hardware and software components and their connections that are relevant for the subsequent steps.
2. The service implementation of the evaluated architecture is assessed with respect to the QA:s (Quality Attributes). The rating is performed using the QAM:s (Quality Attribute Metrics) previously defined in the comparison framework.
3. The ratings are weighed using the previously established Quality Computation Weight Matrix.

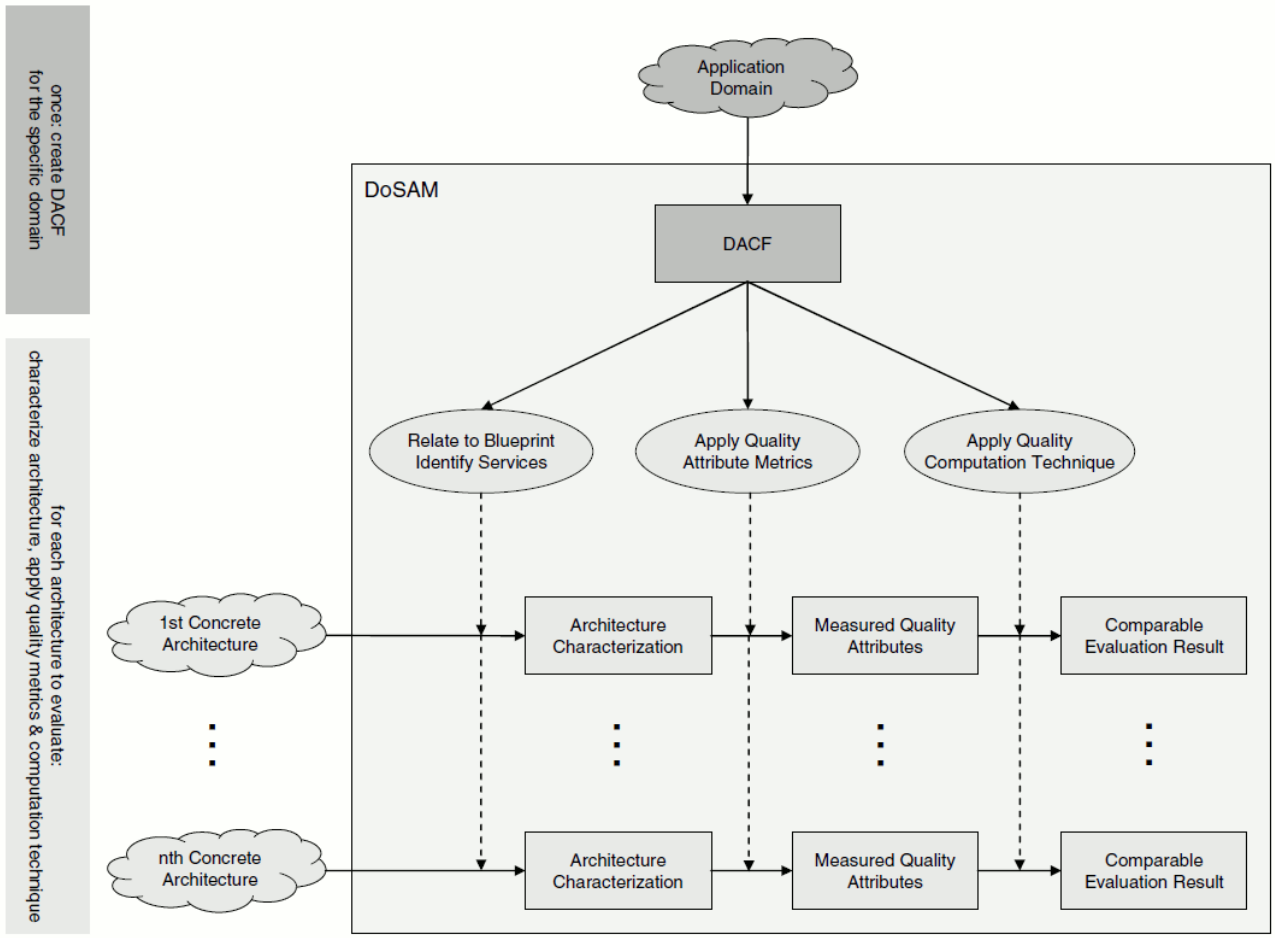

Fig. 5.2: DoSAM application (Bergner et al., 2005).

It is important to note that only the components that are relevant to the domain comparison framework (that is, the components that performs the services specified in the blueprint) are to be involved in the CAE. This is to ensure that only the attributes and services that are relevant to the application domain are evaluated, and that no properties outside of the defined comparison framework are evaluated.

Once a CAE has been performed for each platform, the overall ratings per architectural service for each platform can be compared. Each service is given a score for each quality attribute, for example the data store service is evaluated with regards to availability, modifiability and other attributes. The CAE process is described in Fig. 5.2.

It is clear the DoSAM framework, as well as other frameworks, defines *how* to measure and evaluate the properties of a platform. The question of *what* to measure is left to the author of the framework to decide. Suitable quality attributes need to be defined, to establish from which perspectives the platforms should be evaluated.

One of the established standards for software quality evaluation is the ISO 25010 standard. This standard defines a set of attributes that can be evaluated separately, and also provides a description of what the attribute stands for. It is assumed that this standard could be able to contribute to selecting and adding suitable attributes to the comparison framework. To be more precise, the standard appears to be a good source for *quality attributes*. In the following section, the ISO/IEC

25010:2011 standard will be described.

5.3 ISO and IEEE standards for software evaluation

ISO/IEC 25010:2011, an evolution of ISO 9126, is a standard for the evaluation of software quality. By clarifying and then agreeing on the project priorities and then converting abstract priorities to measurable values, ISO/IEC 9126 aims to develop a common understanding of the project's objectives and goals[14].

The standard defines a quality in use model, applicable for products that are already deployed and in use, and a product quality model, relating to static properties of software and dynamic properties of computer systems. The model excludes purely functional properties, but includes functional suitability. It can be used, for example, to identify system design and objectives, or to establish quality control characteristics[15].

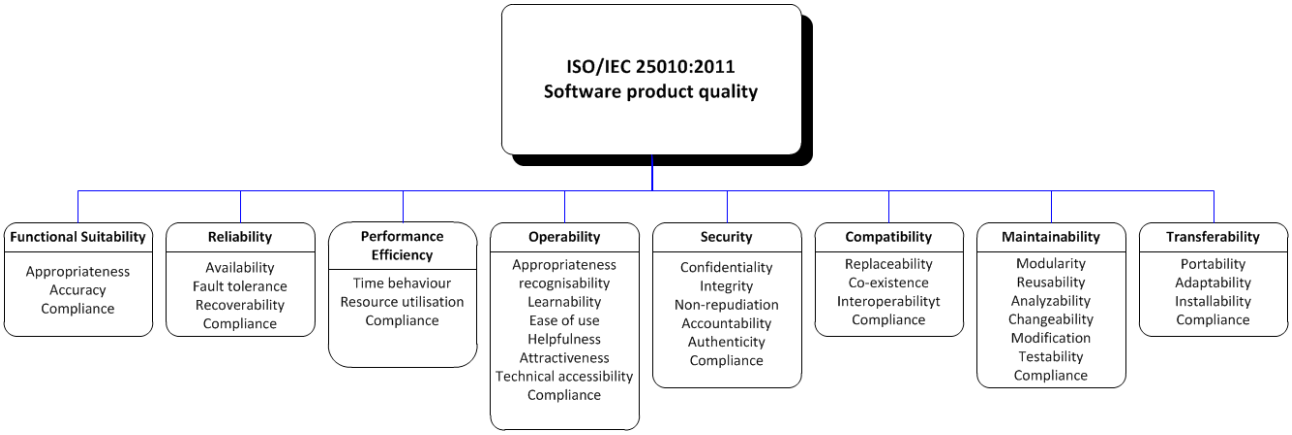

Fig. 5.3: ISO/IEC 25010:2011 Product quality model with subsections (source: the author).

The quality model is divided into eight sections with multiple subsections, as described in Fig. 5.3. Details on the subclasses and what they are intended to measure can be found in Appendix A: ISO/IEC 25010:2011 Product Quality Model.

5.4 Selection of evaluation method

Among the different evaluation methods presented in the previous section, both SACAM and DoSAM was specifically developed for the comparison itself, and to provide a standardized way of performing the comparison so that it would be independent of the person performing it. This is desirable since it can be assumed that this adds to the objectivity of the comparison. Therefore, the choice of comparison platforms can be further limited to these two.

The DoSAM method also enables the platforms to be evaluated on an abstract, architectural level and defines a set of standard specifications that delimits the framework. This makes it possible to create a comparison framework specifically for e-commerce platforms. SACAM does not allow for delimiting the field of study to the same degree, which is desirable since the software compared will all be e-commerce platforms. For this reason, the DoSAM stands out as the stronger candidate compared to SACAM.

14 Wikipedia. *ISO/IEC 9126.*
15 *ISO/IEC 25010:2011* (ISO.org, 2011)

Additionally, the ISO/IEC 25010:2011 standard appears to fit well into the DoSAM framework, where it could be used as a source of *quality attributes*. Using the ISO/IEC standard is assumed to provide established and well-known attributes on which to measure, and it is also assumed that the use of well-known standards in general is always desirable.

For the above reasons DoSAM appears to be well suited for the goals of this theses and has therefore been selected as comparison method.

6 Domain Architecture Comparison Framework (DACF)

This section describes how the Domain Architecture Comparison Framework is established. The sections starts off by defining the *architectural blueprint* and the *architectural services*. This is followed by defining *quality attributes* (*what* to measure) and quality *attribute metrics (how* to measure it*).* The DACF establishment process is finished by defining *quality attribute weights*, which should "tweak" the framework so that it emphasizes the features that are important for the selected application domain.

6.1 Domain architecture blueprint

A component-based style has been identified as a suitable representation method of the architecture blueprint. The components will form the basis of the platform for the services running on the system. As most e-commerce systems of today are web-based, standard web components such as email and database access have to be considered in the blueprint. Also, different service layers for payment and verification services has to considered.

A complete system is made up of both software and underlying hardware, where for example two different presentation services can be run on the same hardware platform. Therefore, the base software and the base hardware has to be evaluated separately.

As previously stated, only components that can be considered as universal for the application domain are to be included in the blueprint.

Hardware components	**Software components**
Web/business server hardware, responsible for presentation and content rendering to the user.	Web/business server software (for example Apache)
Database hardware, for example a server responsible for persistent data storage.	Database software (for example a MySQL installation)

Table 6.1: Domain architecture blueprint

Table 6.1 shows the architectural components have been identified as necessary for the architectural blueprint.

6.2 Architectural Services

According to Bergner et al., the intention of this step is to find constituents of the architectures that are inherent to the application domain at hand, and therefore inevitably represented in all imaginable solutions.[16]

Table 6.2 shows the services that have been identified as inherent to the e-commerce application domain. A store needs, for obvious reasons, a service that achieves the presentation of the graphical user interface. It is assumed that all e-commerce systems uses a web page as GUI. Therefore, the presentation service's main purpose will be to render a html page using the system as a whole. This includes model representations and views, which are constituents of the core system. The presentation service can therefore be said to be the largest of the four services identified.

To manage products, customers and orders and to edit store settings and attributes, a management

16 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 7.

service must be provided. In most systems, this will probably be called “administration panel”, “back office” or something similar.

It is also assumed that an e-commerce system offers some sort of Object-relational mapping system which relies on a database. These components are grouped together to form the Data storage and transaction service.

The last service to be part of the list is the Data access and system interaction service. It is assumed here that an e-commerce system offers a core API that other components can make use of to interact with the system. Also, a system for plug-in components such as modules for payment or shipping solutions is also assumed to be available. Additionally, an interface for system interaction from external sources can also be evaluated, such as a SOAP or XML RPC interface.

The last column states from who's perspective the service should be evaluated. This is provided as a guideline for when identifying and applying metrics that are not quantifiable but instead relies on qualitative judgment from the person performing the evaluation. For example, when evaluating the Presentation service, metrics from the customer's point of view might include such components as product comparison services or navigational aids, and the developer's point of view might include design patterns and other code-oriented features.

Label	**Definition and examples**	**Perspective from which the service is evaluated**
Presentation service	Achieves the presentation of the GUI, the store frontend. The service can be a combination of HTML templates, CSS, jquery, javascript etc.	Customer and Developer
Store management service	Achieves the customer relations management (CRM) of the store. Examples of tasks for the service includes order-, customer- and catalog management.	Customer and Developer
Data storage and transaction service	Achieves the provision of database services, secure transactions etc. Examples of components are the database and database software, and Object-relational mapping (ORM) components.	Developer
Data access and system interaction service	Achieves the provision of data for external and internal components. Examples include built-in API:s for data collection, components that provide integration with other services/web services and extension systems enable the usage of third-party extensions.	Developer

Table 6.2: Architectural services for the application domain. The table shows the chosen label of the service, definitions and examples of usage, and from who's perspective the service is evaluated.

6.3 Quality Attributes

Establishing relevant *quality attributes* and the corresponding *quality attribute metrics* is the second

step in the establishment of the DACF[17]. Bergner et al., as well as Stefani/Xenos, utilizes the ISO 9126 standard to specify relevant quality attributes for the systems analyzed. The authors thus uses the standard as a base for a tailored, more system-specific, standard specification. The ISO 25010 standard also allows for this, as is expressed in the standard specification:

> *"[...] the model should be tailored before use as part of the decomposition of requirements to identify those characteristics and subcharacteristics that are most important, and resources allocated between the different types of measure depending on the stakeholder goals and objectives for the product.[18]"*

The DoSAM framework in combination with the ISO 25010 standard thus allows for several degrees of freedom in the specification of quality attributes, as both models are meant to be adapted and tailored for a specific domain of applications. It is up to the designer of the comparison framework to elaborate and argue which quality attributes are relevant to the application domain, define who the stakeholders are and to act as a domain expert[19].

The ISO25010 standard defines a set of factors that can be used to evaluate software quality. The standard itself has been discussed previously in this document, however the factors need to be discussed in the context of e-commerce systems, to define which of the factors are to be included as quality attributes in the DACF.

In the following subchapter, different views on quality attributes for e-commerce systems are discussed. Following that subchapter, the quality attributes are established as part of the DACF.

6.3.1 External sources

Stefani/Xenos (2001) argues that since users interacts through a web interface, e-commerce quality is related to the quality of the web pages and the services that are provided to the end user. Stefani/Xenos relates e-commerce quality to four quality factors: functionality, reliability, usability and efficiency[20].

V.d.Merwe/Bekker proposes a web site evaluation framework for e-commerce sites which focuses on five main groups: Interface, Navigation, Content, Reliability and Technical. V.d.Merwe/Bekker focuses on the user experience and presents a framework which essentially does not cover the underlying architectural solutions. However, V.d.Merwe/Bekker contributes to the selection of quality attributes by emphasizing the importance of user experience, and how the technical aspects of a solution forms the center around which all user oriented features derive from[21].

Stefani/Xenos (2001) use the ISO 9126 model, and the author's focus on the *usability* attribute can be directly translated into the ISO25010 framework, as can *reliability*. *Functionality* can be expressed as *functional suitability*, and *efficiency* as *performance efficiency*.

In their publication from 2008, Stefani/Xenos also uses the ISO 9126 model as their basis for e-commerce system evaluation. The authors have in this case chosen to evaluate only external metrics from the ISO 9126 standard, in combination with Belief Networks. Again, the authors focus on the user-centered qualities of the systems, such as functionality and usability, but also touches on efficiency and reliability[22].

17 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 8.
18 *ISO/IEC 25010:2011* (ISO.org, 2011).
19 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 7.
20 Stefani, A. and Xenos, M. (2008), p. 2.
21 van der Merwe., R. and Bekker, J. (2003), p. 333.
22 Stefani, A. and Xenos, M. (2008).

6.3.2 Selection of quality attributes

Based on the sources previously mentioned, a strong focus on user-centered quality factors can be seen as the prevailing standard. However, the comparison framework for this study also needs to have a strong developer focus, targeting abstract factors such as extendability, portability and the degree of standardized code in the platform.

Described in the ISO/IEC 25010 standard, two quality factors can be identified as suitable for contributing for obtaining a higher degree of developer focus: maintainability and portability. Maintainability is described as

> *"a set of attributes that bear on the effort needed to make specified modifications",*[23]

with the subcategories Modularity, Reusability, Analyzability, Modifiability and Testability all being focused on the developer side such as if the system can be easily modified, if assets can be easily reused in other components, and the degree of efficiency to which test criteria can be established[24]. Portability, in turn, is described as

> *"a set of attributes that bear on the ability of software to be transferred from one environment to another"*[25].

The subcategories to portability are Adaptability, Installability and Replaceability, which can be used to evaluate the degree to which a system can be adapted to different or evolving environments, the degree of effectiveness with which a system can be installed/uninstalled, and the degree to which a product can be replaced by another product for the same purpose[26].

The remaining categories of quality factors of the ISO/IEC 25010:2011 standard are Functional suitability, Reliability, Usability, Performance Efficiency, Compatibility and Security. However, to limit the scope of the comparison and to narrow the framework down, Reliability is assumed to be fulfilled to a satisfactory level and is therefore left out. For the same reason, Security and Compatibility are also cut from the list of quality factors, and are assumed to be fulfilled to a satisfactory level.

Measuring Usability is usually done using a panel of participants which performs actions on the platform. Their actions are supervised and measurements such as the time it takes for the users to perform certain tasks, how easily they are able to use the services and their general perception of the system form the basis of the usability analysis[27]. However, the use of a users panel is out of the scope of this thesis work. For this reason, the usability quality attribute is judged unsuitable for the comparison framework since a reasonably objective assessment of the attribute can not be performed.

The rest of the categories have been linked to e-commerce framework comparisons in previously mentioned sources and can therefore be seen as necessary to obtain a useful perspective on the application domain.

6.3.3 Market readiness attribute

The quality factors in the ISO/IEC 25010 standard forms the basis for the comparison framework used in this analysis. However, as one of the goals are to address more market-oriented factors such

23 *ISO/IEC 25010:2011* (ISO.org, 2011).
24 Appendix A: ISO/IEC 25010:2011 Product Quality Model
25 *ISO/IEC 25010:2011* (ISO.org, 2011).
26 Appendix A: ISO/IEC 25010:2011 Product Quality Model
27 Wikipedia. *Usability* (http://en.wikipedia.org/wiki/Usability).

as payment gateway compatibility, time-to-market and the degree of search engine optimization of the platforms, an additional quality factor, “market readiness” is suggested. The purpose of this quality factor is to facilitate the assessment of a platform's inherent conditions for a short development time, fast deployment and the degree to which the product is ready for production use without additional tweaking needed for it to hold up to current market standards.

Suggested metrics for the market readiness quality attribute is described later in this document.

6.3.4 Summary of quality attributes

The previous sections of the chapter has attempted to filter out attributes that are relevant for e-commerce platforms. The categories of quality factors that have been assessed as suitable for the comparison framework are:

- Functional suitability
- Performance efficiency
- Maintainability
- Portability
- Market readiness

In the DACF, these quality factors will appear as *quality attributes.*

6.4 Quality Attribute Metrics

As described in the DoSAM specification, corresponding *quality attribute metrics* are to be associated to each of the defined *quality attributes*, to be able to evaluate each *architectural service* with respect to each quality attribute. Quality attributes can be evaluated using a single metric, or a combination of several metrics. For example, a data transfer service can be evaluated with respect to the *performance* attribute by assessing the combination of average throughput and average response time[28].

Jarl, M. (2008) uses the Business Readiness Rating (BRR) method to perform the evaluation. Jarl states that the functionality category needs special treatment, because a certain type of application is often considered to have a standard set of features and another set that is considered bonus features. The author states that a “standard list” must be established, from an external source, if available[29]. Using Jarl's technique, a combination of the feature lists of the four platforms involved in this comparison can be seen as sufficient to represent the quality attribute “functional suitability” for e-commerce platforms.

However, the aim of this thesis is an abstract perspective and to compare the base platforms rather than the actual implementations. Only using metrics that measures weather or not a feature is “existent/nonexistent” could therefore be seen as too stiff, not allowing for enough adaptivity of the evaluation platform, making it less useful. One solution to the problem would be to add factors that measure the expected time and cost needed to implement the feature. This is exemplified in Bergner et al.'s DoSAM publication, where the author defines a metric for the Modifiability attribute using *average change effort*, *number of components to be changed*, *number of persons affected* and other

28 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 9.
29 Jarl, M. (2008), p. 51.

estimation variables[30]. This technique could be further adapted and simplified, for example by using only one factor, the *total change effort.* The feature *cross-sells* would given a value of 0-100 for a certain platform depending on how much effort is needed to implement the feature. The *effort* would represent the *cost* of implementation. As a result, all platforms would be evaluated using the same *scenarios* and therefore be comparable at any given point in time[31].

However it is questionable whether the usage of this technique lies within the scope of this thesis work. The main reason for this is that an insufficient number of metrics have been found in literature that can be proven to have an effect on the selected quality attributes. It is estimated that a substantial amount of work would be needed to define which metrics and factors affect the particular quality attributes selected, and which factors to add, as well as developing formulas to balance between them. Additionally, it is estimated that the usefulness of the results of the evaluation, after applying it to the four platforms, would be lower for Softronic since significantly more effort would be put in establishing the evaluation framework than actually evaluating the platforms.

It becomes clear that the evaluation framework will be required to offer an acceptable balance between measuring quantifiable data which can be proven to have an effect on certain quality attributes, and qualitative data based on interviews and user perception. Additionally, the evaluation framework will have to be able to be defined within the given time frame, and the application of it must render data that is useful to Softronic while still being generic enough not to favor any specific platforms.

The model chosen for this thesis combines Bergner et al.'s standard way of balancing between factors, and the usage of feature lists. The functionality attribute is measured using a list of features that the platforms can be expected to have. The rest of the attributes are measured using a combination of quantitative and qualitative metrics, some of which are extracted through interviews and some through in-depth studies of the implementations. In addition, some quality attributes can be given a lower factor in the weighing table in the DACF so that the effect on the overall score is smaller, thus compensating for the risk of subjectivity when measured.

Interviews will be used to facilitate the assessment of the quality attributes. The interviewees will be asked to elaborate on the metrics for each of the quality attributes except for Functional suitability. The interviews, together with additional sources such as product feature lists, documentation and wiki pages, will be used as sources to determine the metric values. All data, and which sources that were used, will be provided in the evaluation data section of this document.

In the following sections each attribute, and how it is measured, will be described in detail.

6.4.1 Functional suitability

The ISO/IEC 25010:2011 standard describes Functional suitability as

> *"A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs".*

The subcategories of Functional suitability are completeness, appropriateness and correctness. Together, the subcategories state that Functional suitability measures the degree to which the set of functions covers all the specified tasks and user objectives, the degree to which the functions facilitate the accomplishment of specified tasks and objectives and the degree to which a product or

30 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 16.
31 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p.14.

system provides the correct results with the needed degree of precision[32].

Appendix B: Evaluation data shows the metrics that has been selected as suitable for measuring the functional suitability for each service in the architecture. From Stefani/Xenos's model from 2001, the characteristics related to functionality has been adopted and added as quality metrics[33], and also the characteristics described as part of the Functionality subnetwork in their second publication (Stefani/Xenos, 2008). The DoSAM framework also allows for the domain expert to define metrics based on current standards for the platforms in the domain. By analyzing the platforms feature lists, a number of metrics has been added to the list. It can be argued that this method adds a lot of subjectivity to the comparison framework. However, as has been stated previously by Sommerville, assessment of software quality is, in itself, a subjective process. Nonetheless, the functionality of a solution cannot be ignored or it would yield a comparison which would be free of subjectivity but less usable[34].

The services will be evaluated by assessing its features and assigning a binary score (yes/no) on the metrics associated to it. Each service will then achieve a percentage score (0-100) on the functionality attribute. The percentage will represent the score for the service in the evaluation. For example, the Store management service of the Episerver Commerce solution is found to fulfill 40 out of 50 metrics from the metrics section of the service. The score for functional suitability for the service is then 80%, and the score becomes 80/100.

6.4.2 Performance efficiency

Performance efficiency is described in the ISO/IEC standard as a set of attributes that bear of the relationship between the performance of the system and the resources used. In other words, the metrics related to performance efficiency has to to with resource usage, response time, capacity and other metrics which can be seen as quite common metrics in the software world and which can also be quantifiable with relative ease.

The metrics suggested are related to common tasks in the e-commerce software domain, such as handling orders, customers and products. Caching of resources should have alarge impact on user experience, as it reduces response time.

The suggested metrics for the performance efficiency quality attribute are

1. memory usage on the business server running an identified reference project
   $$100 - \frac{memory[GB] * 100}{16}$$
2. the degree to which the service utilizes caching of relevant resources (0-100)
3. the degree to which orders, customers and products can be easily manipulated in code (0-100)

Comments on these metrics can be made regarding the ability to measure them. For example, metric no. 1. is though of as a way of detecting indications of memory usage, not to identify absolute numbers. For example, it is estimated that is is out of the scope of this thesis to determine reference projects that can be applied to all of the solutions and to set up test environments. Instead, the evaluation will attempt to identify recent projects and their maximum memory usage, on which the score calculation will be based. 16 GB is assumed to be a suitable figure for a maximum memory

32 *ISO/IEC 25010:2011* (ISO.org, 2011).
33 Stefani, A. and Xenos, M. (2008), p. 4.
34 Sommerville, I. (2011), p. 655.

limit. Scores evaluating to negative values will be interpreted as 0.

Metric 2 will be measured by looking at caching abilities for the platforms, such as weather or not the service can be cached in full (i.e. full page caching for the presentation service). Metric 3 will be largely based on interview answers.

6.4.3 Maintainability

The ability to test, analyze, maintain and modify a solution can be seen as a very important attribute for any software. The developer must be able to make modifications according to customer preferences, perform regular code maintenance and extend the existing solution.

As Sommerville (2011) describes, software maintenance can be divided into three categories: Fault repairs, Environmental adaption and Functionality addition, with the last category being the most effort-demanding activity. Sommerville also relates some internal attributes to maintainability, such as Length of user manual and Program size, which suggests that the number of source files and documentation completeness can be used as metrics. Also, intuitively, the use of design patterns such as MVC should facilitate the analyzability, modifyability and reusability of a solution.

Based on this information, these attributes are suggested as metrics for Maintainability:

1. the number of source files in the base package: $100 - \frac{no.\ of\ \ files}{500}$ *
2. the degree to which MVC, or equivalent techniques for separation of responsibility areas in an application, is used in the architectural design of the service (0-100)
3. the degree of completeness of the documentation for the service (0-100)
4. the degree to which the service implementation can be debugged easily (0-100).

* It is assumed that the number of files in the source package does not exceed 50.000 source files. A value above 100 will be interpreted as 100, and a value less that 0 will be interpreted as 0.

6.4.4 Portability

The metrics suggested for Portability are;

1. the degree to which the service in question is able to run on different platform configurations, or the degree to which it is independent of non-generic configurations (0-100)
2. the degree to which the service is backwards-compatible with earlier versions of the platforms services on which it depends (0-100)
3. the degree to which the service is easily deployed as a cloud service (0-100)
4. the degree to which installing and setting up the service, as well as uninstalling it, is independent of external resources such as third-party consultants (0-100).

6.4.5 Market readiness

As described previously, the purpose of the Market readiness quality factor is to facilitate the assessment of a platform's inherent conditions for a short development time, fast deployment and the degree to which the product is ready for production use without additional tweaking needed.

Based on this definition, these metrics are suggested;

1. assessed effort required for adaption of the service until it can be used in a live shop: $100-man\,days$ *
2. total cost for licenses needed: $100-\frac{cost}{10000}$ **
3. assessed effort for implementing an analytics tool: $100-man\,days$ ***
4. assessed effort for implementing SEO friendly url:s for product pages: $100-man\,days$
5. the degree to which the service supports plug-in modules for at least five different payment services: $\frac{number\ of\ alternatives}{1/20}$ ****

* It is assumed that an implementation of the service takes between 0 and 100 days to complete. A negative value will be interpreted as 0, and a value above 100 will be interpreted as 100.

** It is assumed that the license costs for a system is between 0-1 000 000 SEK. A negative value will be interpreted as 0, and a value above 100 will be interpreted as 100.

*** It is assumed that the effort required is between 0 and 100 days. A negative value will be interpreted as 0, and a value above 100 will be interpreted as 100.

**** It is assumed that the service must provide support for at least five payment alternatives, which will translate to a score of 100. A negative value will be interpreted as 0, and a value above 100 will be interpreted as 100.

6.5 Quality Computation Weights

Architecture Services	Functional suitability [%]	Performance efficiency [%]	Maintainability [%]	Portability [%]	Market readiness [%]
Presentation service	40	20	25	50	40
Store management service	40	0	25	0	30
Data storage and transaction service	10	60	25	50	0
Data access and system interaction service	10	20	25	0	30

Table 6.3: Quality attribute weighting matrix. Each quality attribute's importance for each architectural service is assessed and assigned a weight percentage. The score of the evaluation is, in the last stage of the CAE, multiplied by the attribute weight.

In Table 6.3 the selected weights of the services are shown. The Presentation service is assumed to be contributing a lot to the functionality factor of the solution, as well as the Store management service. Therefore, the weight of these on the overall functional suitability factor has been set relatively high.

As a contrast, performance efficiency is assumed to be largely dependent on the Data storage and transaction service, for example reading and saving products, customers and orders. This is reflected in the Performance efficiency column, where the Store management service has been excluded from the comparison. The reason for this is that it is assumed that the efficiency of the Presentation service is representable for the efficiency of the Store management service as well.

The weight of the Maintainability quality attribute is distributed evenly across the different services, for the reason that it is assumed that the maintainability of each service is equally important.

With the Market readiness factor, the Data storage and transaction service is assumed to be too difficult to measure. The Data storage service is assumed to be comprised of several low-level components that are not directly related to the market-oriented factors. It is therefore left out of the measurement of the quality factor.

7 Concrete Architecture Evaluation performance (CAE)

According to the DoSAM specification, the Concrete Architecture Evaluation is a three-step-process, iterating over the first two steps for each architecture in the comparison[35]:

1. Relate the architecture to the architectural blueprint and identify the services that are to be assessed. In other words, the task is to identify which of the system's components make up the services of the application domain.
2. Apply the Quality Attribute Metrics to the identified services, to assess and examine the architecture with respect to the identified quality attributes of the application domain.
3. Apply Quality Computation Weights to the result, yielding a comparable evaluation result.

The following sections will perform, if possible, step 1 and 2 for each architecture. To make the CAE comprehensible for the reader, the chapter has been divided into subchapters for each architecture, which in turn is divided into a Blueprint/Services identification part and a Quality attribute application part.

The following section presents only the scores that were the result of the evaluation. Complete evaluation data, showing how the score was calculated, is presented in Appendix B: Evaluation data. Data sources for the determination of the metric values are primarily official documentation and the public websites of the platforms. These sources can be found in the References chapter. Also, where it has been possible to set up a local solution and try out the features, this has been stated in the beginning of the evaluation chapters. To obtain scores for metrics that require personal assessments by developers, interviews have been held with developers on Softronic who were asked to describe how they would rate the platform with regards to the quality metrics defined for the attribute.

In the evaluation tables, quality attributes who was assigned a weight of 0 for an architectural service have a dash (“-”) in the evaluation matrix instead of an actual score. The quality computation weights were determined in chapter 6.5.

7.1 Magento Community Edition

This section relates the architecture to the blueprint and applies the quality attribute metrics for the Magento CE system. A local solution was set up and was used as a source for metric values, where applicable.

7.1.1 Blueprint relations and identification of services

Architectural blueprint component	**Identified architectural component(s)**
Web/business server hardware	Linux x86 server with 2GB memory and two processor cores
Database server hardware	[same as business server]
Web/business server software	Apache 2.2.1, PHP 5.3
Database server software	MySQL 5.2

Table 7.1: Architecture overview of Magento CE 1.6.2.

Table 7.1 shows the architectural components that have been identified for the Magento CE system.

35 Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005), p. 9.

The components have been extracted from the official requirements[36]. The following assumptions have been made:

- The latest version of Apache, PHP and MySQL is used.
- The database server is running on the same hardware instance alongside the business server.

Architectural service component	**Identified participating components**
Presentation service	The Magento frontend rendering system, including the template system, layouts, the MVC implementation and other components responsible for rendering the GUI.
Store management service	The administration panel and its CRM features.
Data storage and transaction service	The ORM system, the MySQL database itself, the SSL services implementation.
Data access and system interaction service	The core API (over SOAP or XML RPC), the Magento Connect system, the administration panel.

Table 7.2: Architectural service components and their implementation in the Magento CE system.

The components providing the architectural services are presented in the right column in Table 7.2. These are the components on which the quality attribute metrics will be applied.

7.1.2 Application of quality attribute metrics and quality computation

Architectural Service	**Functional suitability**			**Performance efficiency**			**Maintainability**			**Portability**			**Market readiness**		
	Weight %	Value	Points	Weight %	Value	Points	Weight %	Value	Points	Weight %	Value	Points	Weight %	Value	Points
Presentation	40	100	40	20	56	11	25	78	19	50	88	44	40	99	40
Store management	40	94	38	0	-	0	25	83	21	0	-	0	30	97	29
Data storage and transaction	10	100	10	60	56	34	25	73	18	50	90	45	0	-	0
Data access and system interaction	10	100	10	20	56	11	25	66	17	0	-	0	30	100	60
Total			**98**			**56**			**75**			**89**			**99**

Table 7.3: Evaluation summary for Magento CE.

The services in Table 7.2 are evaluated by applying the metrics defined for each quality attribute. An overview of the resulting scores is shown in Table 7.3.

36 Magento Inc. *Magento system requirements.*

7.2 Wipcore eNOVA

This section relates the architecture to the blueprint and applies the quality attribute metrics for the Wipcore eNOVA system.

7.2.1 Blueprint relations and identification of services

Architectural blueprint component	Identified architectural component(s)
Web/business server hardware	4 GB RAM, 64-bit processor.
Database server hardware	4 GB RAM, 64-bit processor.
Web/business server software	Microsoft Windows Server 2008 Standard Edition x64, IIS 7, .NET Framework 3.5
Database server software	Microsoft Windows Server 2008, Microsoft SQL Server 2000

Table 7.4: Architecture overview of Wipcore eNOVA.

Architectural service component	Identified participating components
Presentation service	The Wipcore eNOVA Content Management systen renders content, for example products, categories and pages.
Store management service	Wipcore eNOVA has a separate Windows client for store management, “Wipcore eNOVA BackOffice”.
Data storage and transaction service	The SQL server database offers persistent storage. Content and objects are accessed and modified through the built-in API.
Data access and system interaction service	The built-in API provides internal access to components. However no standard component for interaction with external systems could be found.

Table 7.5: Architectural service components and their implementation in Wipcore eNOVA.

7.2.2 Application of quality attribute metrics and quality computation

Architectural Service	**Functional suitability**			**Performance efficiency**			**Maintainability**			**Portability**			**Market readiness**		
	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points
Presentation	40	75	30	20	63	13	25	63	16	50	73	36	40	80	32
Store management	40	81	33	0	-	0	25	59	15	0	-	0	30	90	27
Data storage and transaction	10	100	10	60	90	54	25	59	15	50	73	36	0	-	0
Data access and system interaction	10	50	5	20	63	13	25	59	15	0	-	0	30	100	30
Total			**78**			**79**			**60**			**73**			**89**

7.3 Episerver Commerce

This section relates the architecture to the blueprint and applies the quality attribute metrics for the Episerver Commerce system. A local solution was set up and was used as a source for metric values, where applicable.

7.3.1 Blueprint relations and identification of services

Architectural blueprint component	Identified architectural component(s)
Web/business server hardware	4 GB RAM, 2 64-bit processors
Database server hardware	4 GB RAM, 2 64-bit processors
Web/business server software	Microsoft Windows Server 2008 R2 SP1, IIS 7, .NET Framework 3.5
Database server software	Microsoft Windows Server 2008 R12 SP1, Microsoft SQL Server 2008 SP2 32/64 bit

Table 7.6: Architecture overview of Episerver Commerce

Architectural service component	Identified participating components
Presentation service	The Episerver CMS is responsible for rendering the content, for example products, categories and pages.
Store management service	The Commerce Manager handles orders, products etc.
Data storage and transaction service	The SQL server database offers persistent storage. Content nodes and objects are accessed through the Business API.
Data access and system interaction service	The Episerver CMS offers a plugin system that enables extensions of the platform. Episerver Commerce also uses the Windows Communication Foundation API to enable system interaction through Web services.

Table 7.7: Architectural service components and their implementation in Episerver Commerce.

7.3.2 Application of quality attribute metrics and quality computation

Architectural Service	**Functional suitability**			**Performance efficiency**			**Maintainability**			**Portability**			**Market readiness**		
	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points
Presentation	40	100	40	20	67	15	25	56	12	50	80	40	40	91	36
Store management	40	100	40	0	-	0	25	56	12	0	-	0	30	90	27
Data storage and transaction	10	100	10	60	67	45	25	56	12	50	88	44	0	-	0
Data access and system interaction	10	100	10	20	67	15	25	56	12	0	-	0	30	100	30
Total			**100**			**67**			**56**			**84**			**93**

7.4 Umbraco CMS with TeaCommerce

This section relates the architecture to the blueprint and applies the quality attribute metrics for the Umbraco CMS/TeaCommerce system. A local solution was set up and was used as a source for metric values, where applicable.

7.4.1 Blueprint relations and identification of services

Architectural blueprint component	Identified architectural component(s)
Web/business server hardware	
Database server hardware	
Web/business server software	Microsoft Windows Server 2008, IIS 7, ASP.NET 4, MVC 3
Database server software	Microsoft Windows Server 2008, Microsoft SQL Server 2008

Table 7.8: Architecture overview of Umbraco CMS. No sources could be found that suggested specific hardware specifications for the architectural blueprint. However, it assessed that the software specifications of the platform is sufficient to be able to perform an evaluation of the platform.

Architectural service component	Identified participating components
Presentation service	The Umbraco CMS is responsible for rendering the content, for example products, categories and pages.
Store management service	Umbraco Content management system, handling orders, products etc. as content nodes.
Data storage and transaction service	The SQL server database offers persistent storage. Content nodes and objects are accessed through the built-in API of Umbraco CMS.
Data access and system interaction service	The Umbraco CMS offers a plugin system that enables extensions of the platform. Umbraco also features a Web services API that enables system interaction.

Table 7.9: Architectural service components and their implementation in Umbraco.

7.4.2 Application of quality attribute metrics and quality computation

Architectural Service	**Functional suitability**			**Performance efficiency**			**Maintainability**			**Portability**			**Market readiness**		
	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points	Weight	Value	Points
Presentation	40	54	22	20	83	17	25	75	19	50	85	43	40	84	34
Store management	40	69	28	0	-	0	25	75	19	0	-	0	30	97	29
Data storage and transaction	10	50	5	60	83	50	25	75	19	50	83	41	0	-	0
Data access and system interaction	10	67	7	20	83	17	25	75	19	0	-	0	30	20	6
Total			**61**			**83**			**75**			**84**			**69**

8 Magento implementation of mataffären.se

This chapter provides an overview of the Magento CE platform, as well as a description of the process for implementing a basic design for a Magento website and how to develop the two modules selected from Mataffären.se.

8.1 Developer background

This section describes, from a developers perspective, the Magento system including its components, the code organization and the system design. At the end of the chapter, the implementation process of the selected components of Mataffären.se is described in detail, to provide an example of how to set up, configure and design a Magento installation. The section also provides an overview of how to extend the functionality by creating a *module*, in this case the two selected components of the Mataffären.se system.

There are two distinct code repositories where a developer has to look in order to develop for Magento: the */app/code* and the */app/design* directory. The */app* directory holds the application-specific code for the installation, which in turn is divided into */code*, the directory holding the application logic, and */design*, holding the layouts and templates for the application. The connection between template files and module code, and how a page's html code is generated, will be explained later in this chapter.

8.1.1 Code organization

In Magento, files are grouped into *modules* based on what functionality they offer. This is a bit different from standard MVC applications, where controllers are in one folder, models in another and so forts. There are two main catalogs for modules: the *core* modules are in /app/code/core and the local, “custom-made” modules are in /app/code/local[37].

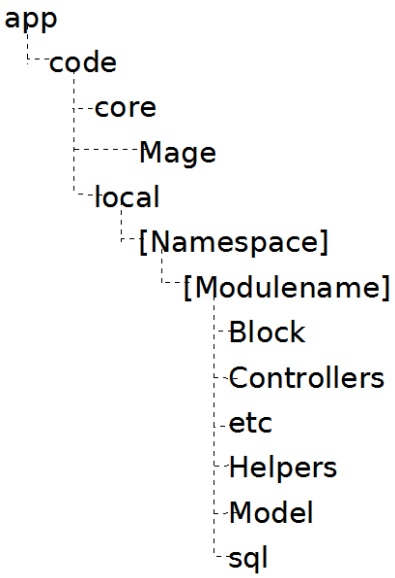

Fig. 8.1: Folder structure for a Magento module.

37 Alan Storm. *Magento for developers.*

Magento is a configuration-based MVC system, as an alternative to a convention-based system. In convention-based MVC, new modules would be recognized automatically when files are created and put in a certain directory. In configuration-based MVC, new modules and components have to be registered in the configuration files in order to function[38]. While a lot of things in Magento applies the convention-over-configuration approach, modules are still required to have a config.xml file with its configuration parameters specified[39].

To customize or extend Magento, rather than editing core files directly, or even placing new controllers, models or helpers next to Magento code, new modules are created in *app/code/local/Package/Modulename*. *Package*, similar to *Namespaces* in Java and C#, is a unique name that identifies your company or organization. As is the case with namespaces in C#[40], the package name is used to declare a scope for the code. It acts as a unique identifier for the company or person behind the module, to avoid code collisions between developers. Fig. 8.1 shows the folder structure for a typical Magento module, where /app is located in the root directory of the Magento installation.

8.1.2 MVC implementation

Sommerville states that:

> *The MVC pattern separates elements of a system, allowing them to change independently. For example, adding a new view or changing an existing view can be done without any changes to the underlying data in the model*[41].

In a typical MVC architecture for a web application, a controller would trigger state changes to the model based on user events in the view. The model then notifies the view of the change. Magento, though, has its own implementation of the MVC design pattern. As in many other MVC-based systems, Magento uses a single point of entry for all page requests: index.php. The file contains code that handles routing to the proper method in a controller. It instantiates and calls the action method, which is called *dispatching*.

38 Wikipedia. *Convention over configuration.*
39 Alan Storm. *Magento for developers.*
40 Microsoft. *C# Programming Guide.*
41 Sommerville, I. (2011), p. 157.

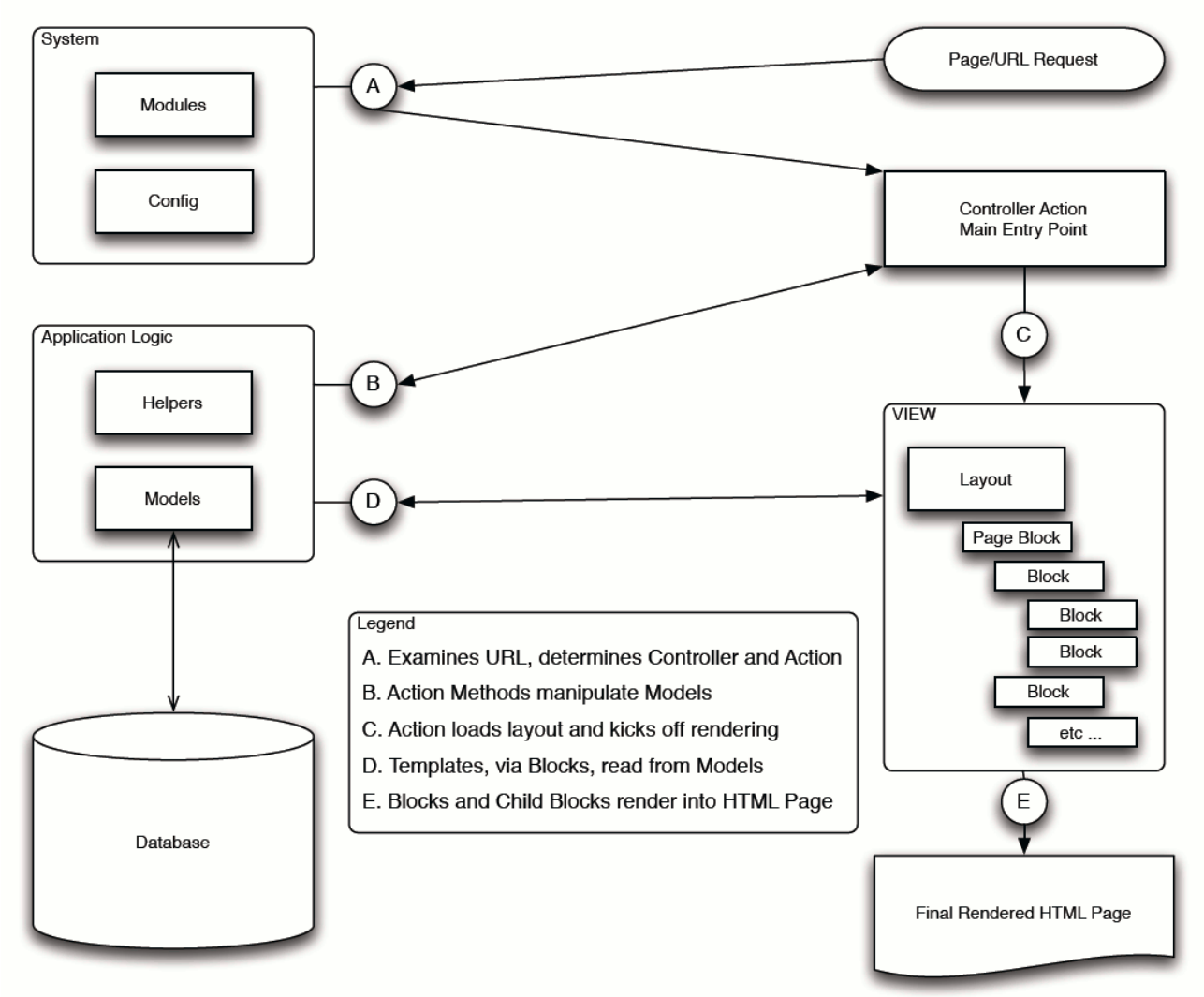

Fig. 8.2: Magento MVC implementation (Alan Storm, 2012).

Described in Fig. 8.2, the request process starts of with url examination and routing. The action then loads the layouts and starts the rendering process. Pre-defined templates and blocks loads data from the models, and the blocks then render into an HTML page. The terms *blocks* and *templates* will be described in more detail later in this chapter.

For example, the url request

```
http://example.com/foo/bar/view/id/25
```

contains information about in which module (“foo”) the controller resides, which controller should be used (“bar”), which action should be called (“view”) and also the arguments to the controller action (“id=25”).

Controllers are placed in the Controllers folder of a module (see Fig. 8.1). For example, the controller BarController.php would look like this[42]:

```
class Kth_Foo_BarController extends Mage_Core_Controller_Front_Action
{
    public function viewAction()
```

42 Alan Storm. *Magento for developers.*

```
    {
        //do something
    }
}
```

The controller extends the core, and implements the “view” action. “Kth” is the namespace for the module, see Fig. 8.1.

8.1.3 Models and Helpers

A Magento module can contain several *models.* In Magento, the term *model* refers to a structural model, representing one of the components that make up a system. An example of a module would be the “category” model of the “catalog” module. Dependencies and logical structure are factors that has to be considered when determining in which module a model should be placed.

To aid interaction with the datastore and avoid having to write SQL code manually, Magento offers an Object Relational Mapping system (ORM). The ORM is accessed through models. Helpers contain utility methods for the models. Helpers are available per module or per model. For example, the helper class for the category class can be loaded through

```
$helper = Mage::helper('catalog/data');
```

providing helper methods such as

```
$helper->getBreadcrumbPath();
```

[43].

An example of a Magento model would be the catalog/product model. Once a model has been loaded, it can be used to interact with the datastore and perform basic CRUD (create/read/update/delete) operations:

```
$_productId=25;
$_product = Mage::getModel('catalog/product')->load($_productId);
$_product->setPrice(199);
$_product->save();
```

A product can be loaded up by the model, and the price, manufacturer or other attributes can be set or retrieved from the object. The *setPrice* methods is an example of Magento's use of *magic get/set methods*, where the method name is mapped to a model attribute. This way, once a product attribute has been added it can be easily retrieved in the frontend.

8.1.4 Observers

Magento implements an Event/Observer pattern that can be used to pick up and react to certain events. As certain actions happen during a Page request (a Model is saved, a user logs in, etc.), Magento will issue an event signal which a module can “listen” to.

Observers are registered in the module's config.xml file. The configuration entry specify which event that should be listened to and which action in which module that should be triggered. An example of an observer entry in the config.xml file would be

```
<events>
    <customer_login>
       <observers>
            <observernamespace>
                <type>singleton</type>
                <class>customerlistener/observer</class>
                <method>exampleAction</method>
```

43 Magento Inc. *Magento Documentation.*

```
            </observernamespace>
        </observers>
    </customer_login>
</events>
```

which represents an observer that listens to the *customer_login* event and triggers the *example* action method in the *customerlistener* module.

8.1.5 Class overrides

Magento offers a way of extending the functionality of the core models by overriding methods in the original classes. For example, the developer might want to change the behaviour of the *login* method in the *customer* model. By extending the original model in the way of

```
class Kth_Foo_Model_Barcustomer extends Mage_Core_Model_Customer
{
    public function login()
    {
        // custom login action
    }
}
```

where the Bar model in the Foo module extends the core Customer model, replacing the original *login* method without touching the original methods[44].

Following the configuration-over-convention-pattern, the rewrite has to be registered in the config.xml file of the module:

```
<models>
    <foo>
        <class>Kth_Foo_Bar</class>
    </foo>
    <catalog>
        <rewrite>
            <customer>Kth_Foo_Model_Bar</customer>
        </rewrite>
    </catalog>
</models>
```

8.1.6 Design and customization

The Magento design and templating system is arguably one of the more complicated parts of the framework. Magento separates the model and the view by adding an additional layer using *blocks* and *templates*. This way, the view component directly references system models to get the information it needs for display.

8.1.6.1 Layouts, blocks and templates

The XML files in the *layout* directory defines the block structure of pages. Blocks are elements of a page. The *template* directory contains phtml files, which consists of xHTML markup and, in some cases, page logic for visual elements[45].

There are two types of blocks: *Structural blocks* and *Content blocks*. Structural blocks are for page structure elements, for example header and footer. Content blocks produce the actual content with the help of phtml files. For example, there can be a content block that represents the cart element.

44 Alan Storm. *Magento for developers.*
45 Magento Inc. *Magento Designer's Guide*, p. 7.

The cart content block then refers to a phtml file that generates the actual content. A *layouts* defines these two different types of blocks into a layout package[46].

The layout files generate the html output by referring to content blocks in the layout files. For example, if the page.pthml template file wants to generate the *header* content, it can refer to the *header* content as

```
<?php echo $this->getChildHtml('header')?>
```

and fetch the content named “header” in the current layout file;

```
<block type="page/html_header" name="header"
template="page/html/header.phtml"/>
```

which will render the contents in header.phtml.

```
<MAGENTO_BASE_DIR>
+ app
    + design
        + frontend
            + base
            + default
            + your_package
                + default
                    + layout
                    + template
```

Fig. 8.3: Folder structure for a Magento theme.

As seen in Fig. 8.3, the layout and template files are stored under the app/design/frontend/package directory. An example of a complete layout/blocks/templates system can be found in the implementation chapter 8.6 Frontend development.

8.1.6.2 Packages and themes

Themes are stored in /app/design/frontend/[package_name] and contains the folders *layout, template* and *locale*. The layout and template directory have been described earlier in this chapter. The *locale* directory contains theme-specific translation files in .csv format that allows for additional customization for the theme.

A *package* is a collection of themes. As seen in Fig. 8.3, the *your_package* package contains the theme *default*. Packages and Themes can be assigned on a website, store or store view level. Themes are activated and deactivated in the administration panel.

8.1.7 Websites, stores and store views

The multi-store functionality is one of the core features of the Magento system. In Magento, a website is a collection of stores, which themselves are collections of store views. The stores share the same customer information, order information and shopping cart.

46 Magento Inc. *Magento Designer's Guide*, p. 12.

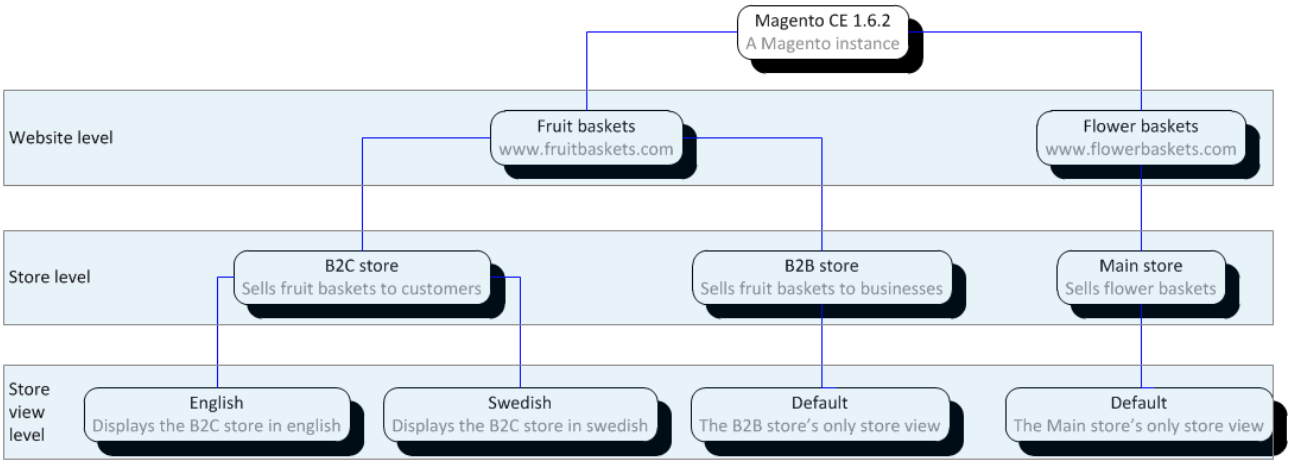

Fig. 8.4: Hierarchy of websites, stores and store views in Magento. Examples of applications are displayed in the boxes as gray text (source: the author).

Stores are collections of store views and can be setup in a variety of ways. Their main function is to provide a logical container that allows for grouping related store views together in a website.

Store Views are the actual store instances in Magento. The store views can represent different languages, design styles or seasonal variations such as a special Christmas version of a store.

For multi-store-functionality implementation in Magento, the system designer can choose to have multiple websites, stores or store views depending on how much information and configuration parameters he or she wants to share between the stores[47].

The website/store/store view relationship and hierarchy can be used to create several different types of multi-store configurations. For example

- a single website, a single store and a single store view
- a single website with multiple stores
- multiple websites with multiple stores and store views.

Fig. 8.4 shows the relationship between websites, stores and store views. The multi-store functionality of Magento can be considered as one of the platform's stronger sides, because of the flexibility and scalability in the configuration.

8.2 Requirements

To achieve the goal of demonstrating the capabilities of the Magento platform, a sample application has been developed that mimics some of the functionality of the project Mataffären.se.

Mataffären.se is a B2C (Business to consumer) web shop for groceries. It is built on the platform Wipcore eNOVA. As part of this thesis, some of the shop's features are to be re-constructed on the Magento platform. The requirements for the implementation is that it highlights the key components of developing in Magento. As stated in chapter 3.1 Goals, two modules should be implemented, as well as a store frontend.

For the frontend part, three views has been defined as necessary to achieve a useful demonstration:

47 Magento Inc. *Magento Designer's Guide*, p. 3-4.

- Home page, where the catalog, promotions and other information is presented to the user.
- Category listing, where products and/or subcategories are presented and the user can choose to view products or subcategories in more detail.
- Product page, where a product is presented in more detail such as an image, weight and other information.

The backend implements two components that are currently part of the Mataffären.se solution;

- a module that acquires product information from OPV
- a module that simulates a connection to the shipping company Widrikssons, providing order picking information and an API for data exchange.

8.3 Environment setup

To develop the Magento solution, a local and a remote environment have been created. The local environment is used for development, testing and debugging. The remote environment is a production environment, where the solution is deployed and also the environment that will be used for demonstration.

8.3.1 Local environment

The local environment consists of the following;

- A client computer running Windows 7 Enterprise
- NetBeans IDE 7.1.1, for developing and debugging of the solution
- WampServer 2.2D, which provides a web application development stack consisting of Apache, MySQL and PHP
- Xdebug, a component of WampServer that enables debugging of the solution in NetBeans.

NetBeans, WampServer and Xdebug are all licensed in a way that they are free to use (GPL) and can be downloaded and installed on the client. Details on the software used can be found on www.netbeans.org and www.wampserver.com.

8.3.2 Production environment

To provide scalability and sufficient speed for the solution, Glesys (www.glesys.com), a cloud hosting provider, has been chosen to supply the production environment for the Magento solution. Glesys provides a payment model where it is possible to instantly edit the specification of the server to match current needs, which makes the solution suitable for this project.

A server with the specifications

- disk size of 5 GB,
- 2048 MB of memory, and;
- 2 processor cores

has been created to supply the production environment, running Debian 6.0 64bit (Linux) as operating system.

Details on installing the LAMP stack is judged as out of the scope of this thesis. However, a general overview of the procedures will be provided as well as details on the installation steps that are specifically needed for the Magento solution to be able to run on the environment.

To communicate with the remote server, PuTTY[48] is used. Apache 2.2, PHP 5.4 and MySQL 5.3 are all installed via the command 'apt-get install [php/apache/mysql]' on the console. To enable URL rewriting in Magento, the module 'rewrite.so' has been activated in Apache. A default site has been enabled, located in /var/www.

After restarting MySQL and Apache, no additional configuration is needed to achieve a running web server on the production platform. In the following sections, the steps for installing and configuring the Magento solution will be described.

8.4 Installation and configuration

To install Magento, the following steps are required:

- Download Magento CE from www.magentocommerce.com.
- Unpack the contents into the public web directory of the web server (in Wamp this is c:/wamp/www, on Debian this is /var/www).
- Create a database and an associated user.
- Run the Magento installation by browsing to http://[hostname]/magento.

When browsing to the installation page, the user is presented with the *Magento Installation Wizard*, the built-in Magento installation guide. Database details and other server options are entered by the user, such as session storage and rewrite options.

48 http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

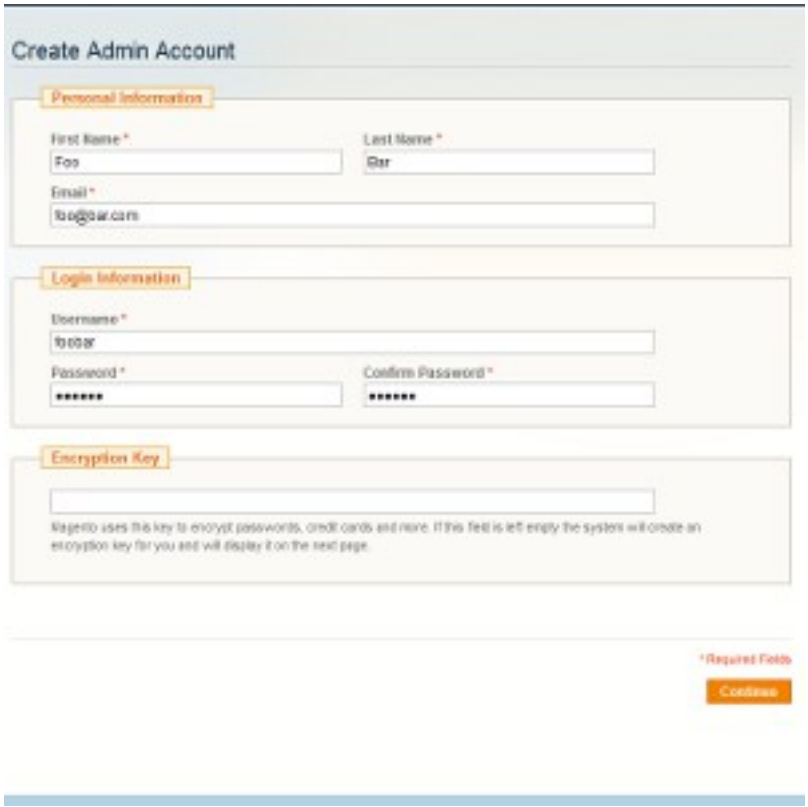

Fig. 8.5: Step 3 out of 5 of the Magento Installation Wizard.

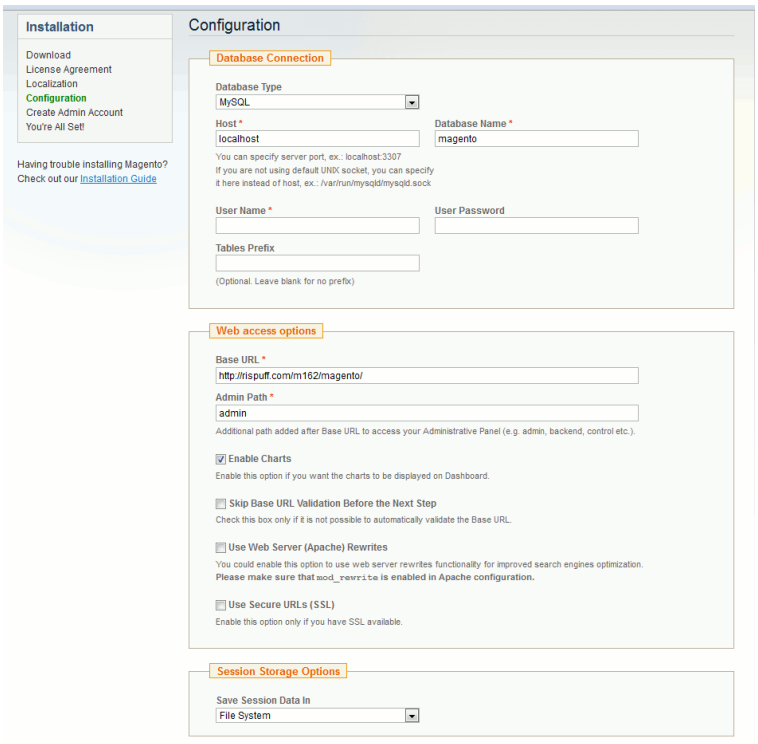

Fig. 8.6: Step 4 out of 5 of the Magento Installation Wizard.

Fig 8.5 and 8.6 shows two of the five steps of the installation. It is estimated that the installation procedure takes no more than 10 minutes for a user to complete, without previous experience. Once the Magento Installation Wizard is completed, user can access the frontend and the administration panel.

The next step is to configure the solution for the intended purposes. It is assumed that the goal of the configuration is to have a working platform that is ready to receive orders. To achieve this, the following steps are needed:

1. Configure websites, stores and store views.
2. Create attributes and attribute groups.
3. Create categories and products.
4. Install and configure payment and shipping options.

The following sections will focus on the steps 1-3 and only touch briefly on step 4, for the reason that it can be considered to be out of the scope for this theses to apply payment and shipping options. However, it is useful as a demonstration of *Magento Connect*, a system for installing third-party extensions such as payment and shipping options.

8.4.1 Configuring websites, stores and store views

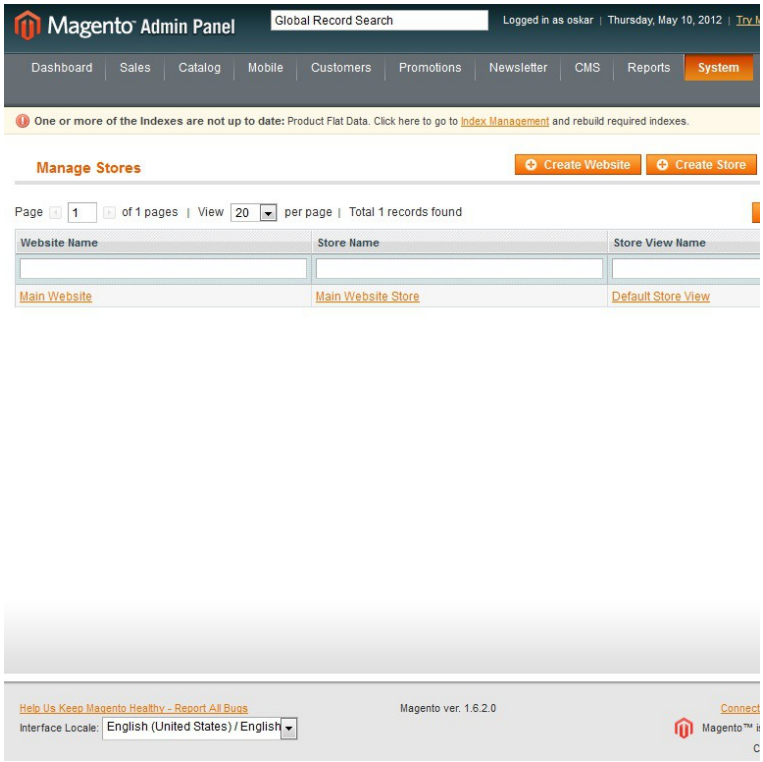

Fig. 8.7: Store configuration view.

The solution is required to have only one website, one store and a single store view. Therefore, no additional configuration is required since this is the setup that is already set after installation. Fig. 8.7 shows the settings provided as default under System → Manage stores, where a Main website, a Main Website Store and a Default Store View is provided.

8.4.2 Creating attributes and attribute groups

In Magento, a product inherits from an attribute group. An attribute group is a collection of attributes. Attribute groups are created to facilitate the creation of similar products. For example, a shop that sells clothes might want to create an attribute group called “shirts”, with the attributes “collar type”, “cotton type”, “chest size” and so forth. Attributes can be of different types, for example “text”, “integer” and “yes/no”.

A number of attributes are mandatory for products, for example “price”, “sku” and “status”. For this specific solution, the attribute group “food” has been created, which apart from the mandatory ones consists of the attributes

- “content” [text]: the contents of the food package
- “allergyinfo” [text]: information about the contents for people with allergies
- “nutrition” [text]: nutrition facts such as calories and protein
- “ean” [integer]: the EAN code of the package
- “anglamark” [yes/no]: whether or not the product is an “Änglamark” product
- “krav” [yes/no]: whether or not the product is marked as “Krav”
- “garanteradhallbarhetstid” [integer]: number of days the product is guaranteed not to out of date from the time of delivery
- “manufacturer” [text]: product manufacturer.

8.4.3 Creating categories and products

Categories are created in Catalog → Categories. For this solution, category names and the category hierarchy have been replicated from parts of the catalog on Mataffären.se.

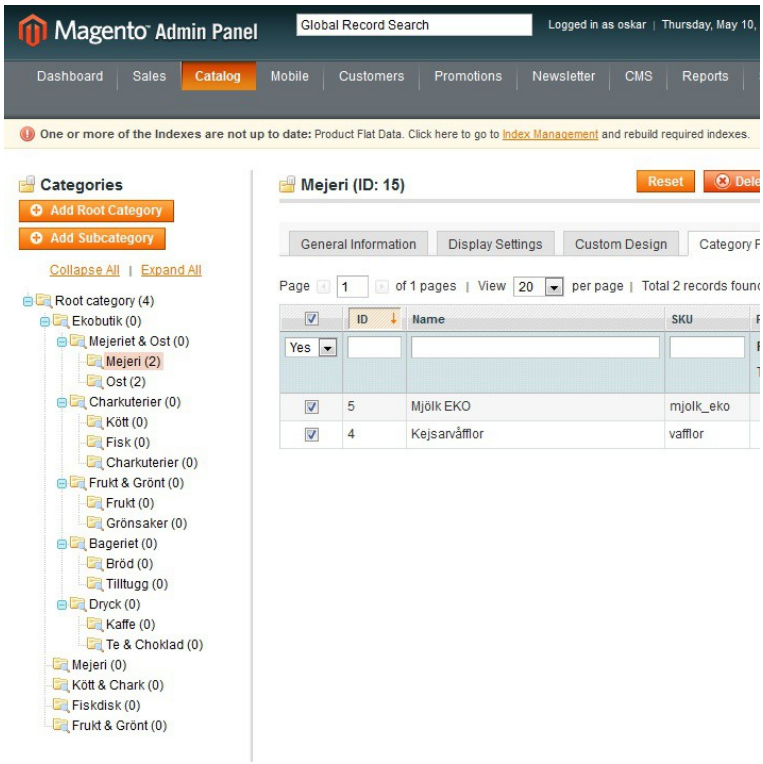

Fig. 8.8: View of the categories and the category hierarchy.

Fig. 8.8 shows the categories and the category hierarchy as well as two products belonging to the category “Mejeri”.

When categories attributes and attribute groups has been created, the user can create products inheriting from the attribute group and assign them to a category. A product must belong to a category to become visible in the shop.

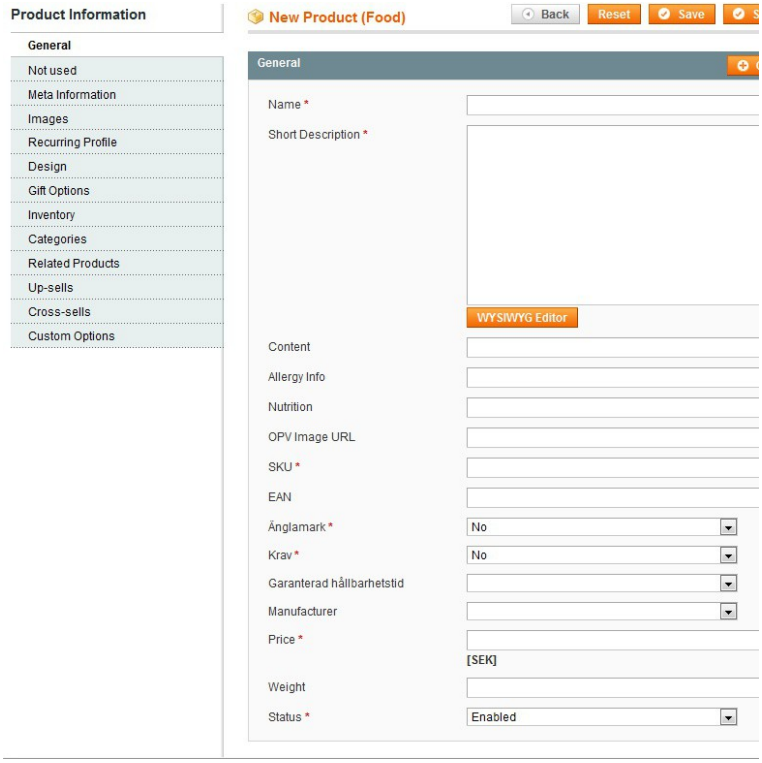

Fig. 8.9: Dialog for creating a new product. Attributes are inherited from the selected attribute group.

Product are created through Catalog → Manage products. After selecting which attribute group the product should inherit from, the dialog as seen in Fig. 8.9 is presented to the user. Once the properties for the product has been entered, user can select which categories the product should belong to in the tab “Categories”. A product can belong to multiple categories.

8.5 Backend development

This section describes briefly how the goal of demonstrating Magento backend development was achieved, without going into too much detail on the implementation. Background to the Magento architecture is provided in chapter 8.1: Developer background.

Component	Purpose
Oslind/OPVmodule/etc/config.xml	Defines the actions for the module, and that it should be available to run through the administration panel.
Oslind/OPVmodule/controller/indexController.php	Holds the action to import product data through SOAP calls.

Table 8.1: Main components of the OPV module.

The “OPV” module was created using the standard module layout as described in the background chapter. The model consists of the components described in Table 8.1. The module offers the ability for a user to, in the administration panel, click a link to trigger the action to import product data from the OPV database to the local database using SOAP calls. The module utilizes the existing

Magento API to save and update data for the products.

```
class Oslind_Opvadminupdater_IndexController extends Mage_Adminhtml_Controller_Action {
    [...]
    public function updateAction() {
        Mage::app()->setCurrentStore(Mage_Core_Model_App::ADMIN_STORE_ID);
        $client = new SoapClient('http://www.mediabanken.se/CoopConsumerWebService/CoopW
        [...]
        $products_collection = Mage::getModel('catalog/product')
                ->getCollection()
                ->addAttributeToSelect('*');
        foreach ($products_collection as $product) {
            $eanVec = $product->getEan();
            $param = array('eanVec' => array('string' => $eanVec, 'string' => $eanVec),
            $result = $client->__SoapCall('GetProductData', array('parameters' => $param
            $product->setContent($result->GetProductDataResult->Product->Content);
            $product->setNutrition($result->GetProductDataResult->Product->Nutrition);
            $product->setOpvimageurl($result->GetProductDataResult->Product->ImageURL);
            try {
                $product->save();
            } catch (Exception $e) {
                echo "Caught exception " . $e->getMessage() . "";
            }
```

Fig. 8.10: Exemple code from the update action in the OPV module's indexController.php.

Fig. 8.10 shows exemplary code from the controller action in the module.

The model for simulating export and import of packing data from the delivery agent contains a bit more code and has more components than the OPV module.

Component	**Purpose**
Oslind/Widrikssons/Model/etc/api.xml	Defines the methods that should be part of the module's API.
Oslind/Widrikssons/Model/Widorder/Api.php	Contains the methods for the module's API.
Oslind/Widrikssons/Model/Widorder.php	Contains the import/export order methods made available to the controllers of the module. This is the location for the main logic of the module.
Oslind/Widrikssons/Model/Mysql4/Widorder.php	Required for the ability to store the model entity in the database.

Table 8.2: Main components of the Widrikssons module.

```
class Oslind_Widrikssons_Model_Widorder_Api extends Mage_Api_Model_Resource_Abstract {
    [...]
    /*
     * Get or set batch nr for orders. If batch not null, user is requesting all the orc
     * Otherwise, user is requesting to set a batch nr for an order.
     */
    public function orderBatchNr($batch, $orderNrs) {
        [...]
        if (is_null($orderNrs)) {
            // Filter out the orders
            $widorders = Mage::getResourceModel('widrikssons/widorder_collection')->addI
            $widordersArray = array();
            // We use nestled arrays to separate the orders and their properties.
            foreach ($widorders as $widorder) {
                array_push($widordersArray, array(
                    $widorder->getIncrementId(),
                    $widorder->getOrderId(),
                    $widorder->getOrderBatch(),
                    $widorder->getBatchsortnr(),
                    $widorder->getSendDate(),
                    $widorder->getPickDate(),
                    $widorder->getTimestamp()
                        )
                );
            }
            return $widordersArray;
```

Fig. 8.11: Example code from the Api.php file of the Widrikssons module. The orderBathNr method is part of the module's public API and can be used to fetch properties for a model entity in the local database.

As shown in Table 8.2, a greater amount of components is required to achieve the goals of database storage of model data, and especially enabling an API for web services to allow for external processes to exchange data with the module.

Fig. 8.11 shows code from a method made available in the module's public API. The method is then specified in the module's configuration file, making it available in the public WSDL specification for the system.

8.6 Frontend development

The frontend example was created using the standard Magento frontend design procedure described previously in this document. Three templates were developed: A home page, a category view and a product view.

```
<html>
    <head><?php echo $this->getChildHtml('head')?></head>
    <body>
        <div class="mainContainer">
            <div class="header"><?php echo $this->getChildHtml('header')?>
                <div class="topLinks"><?php echo $this->getChildHtml('toplinks')?></div>
            </div>
```

Fig. 8.12: Example code from the main .phtml template file, showing the layout blocks "header" and "toplinks" being referenced from the page.xml layout file.

Fig. 8.12 shows exemple code from the home page template. The “header” block is rendered from the “header” reference in the layout file. The reference points, in turn, to a .phtml file, which contains the actual header contents.

```
<block type="page/html_header" name="header" template="page/html/header.phtml"/>
<block type="page/html" name="toplinks" template="page/html/toplinks.phtml"/>

<block type="page/html" name="topcontainer" template="page/html/topcontainer.phtml" />
<block type="page/html" name="humancontact" template="page/html/humancontact.phtml"/>
<block type="page/html" name="search" template="page/html/search.phtml"/>
<block type="page/html" name="cart" template="page/html/cart.phtml"/>

<block type="page/html" name="categories" template="page/html/categories.phtml"/>
```

Fig. 8.13: Example code from the page.xml layout file holding the references to the blocks referred to by the main .phtml file.

Fig. 8.13 shows example code from the layout file. The references “header” and “toplinks”, previously called upon from the .phtml file, link to other .phtml files whose content will be rendered on the page, as the final step of the rendering process.

8.7 Migration to production server

After developing the solution on the local environment, the solution can be migrated to the remote server. To migrate a standard Magento solution, the following steps have identified as required:

1. Transferring the files to the new site.
2. Transferring the contents of the database to the new database.
3. Editing /app/etc/local.xml to reflect the new database properties (mainly database name, user and password).
4. Setting a new host url in the database to reflect the new location of the solution.

To accomplish this, the solution has been compressed to a .tar file, transferred to the remote location and unpacked to the public web directory of the web server. Following this, the contents of the local database have been exported to a .sql file and imported into the remote database. For the solution to be able to utilize the new database, the local.xml file of the remote installation has been edited to reflect the changes.

As a final step, the solution needs to be updated with the new host, by editing the field *base_url* in the table *core_config_data*. In this case, the hostname has been updated from *localhost* to the IP

address of the production server.

When the above steps have been performed, a working solution should be up and running on the production server.

9 Conclusion

This chapter should present and analyze the evaluation results, as well as describe in short the solution to the Magento implementation.

Quality Attribute	Platforms and Architectural Services																			
	Magento CE					**Episerver Commerce**					**Wipcore eNOVA**					**Umbraco CMS with Tea Commerce**				
	Presentation	Store management	Data storage and trans.	Data access and syst. inter.	**Total (weighed score)**	Presentation	Store management	Data storage and trans.	Data access and syst. inter.	**Total (weighed score)**	Presentation	Store management	Data storage and trans.	Data access and syst. inter.	**Total (weighed score)**	Presentation	Store management	Data storage and trans.	Data access and syst. inter.	**Total (weighed score)**
Functional suitability	100	100	94	100	**98**	100	100	100	100	**100**	75	81	100	50	**78**	54	69	50	67	**61**
Performance efficiency	56	-	56	56	**56**	67	-	67	67	**67**	63	-	90	63	**79**	83	-	83	83	**83**
Maintainability	78	83	73	66	**75**	56	56	56	56	**56**	63	59	59	59	**60**	75	75	75	75	**75**
Portability	88	-	90	-	**89**	80	-	88	-	**84**	73	-	73	-	**73**	85	-	83	-	**84**
Market Readiness	99	97	-	100	**99**	91	90	-	100	**93**	80	90	-	100	**89**	84	97	-	20	**69**

Table 9.1: Summary of the evaluation results. Quality attributes who was assigned a weight of 0 for an architectural service have a dash ("-") in the evaluation matrix instead of an actual score. Total score is calculated after applying the quality computation weights defined in section 6.5.

Table 9.1 presents the score summary for the platforms. A more detailed evaluation result for each platform can be seen in chapter 7. The results show that while Magento CE 1.6.2 and Episerver Commerce have a high degree of functional suitability, they both lack in terms of performance efficiency and maintainability. Wipcore eNOVA and Umbraco CMS/TeaCommerce's strength lies in performance efficiently, while Umbraco CMS with TeaCommerce also has a high degree of portability aside from performance. Regarding market readiness, all platforms perform well except for Umbraco CMS with TeaCommerce.

The results suggest that a solution with Magento CE should be chosen when it is important to set up a new solution quickly, without having to spend a lot of time adding features that are not present by default. On the other hand, if a more customized and tailor made system is desirable, Wipcore eNOVA as well as Umbraco CMS seems like a good choice. Episerver Commerce would maybe also be a good choice for this type of system, if the high degree of functional suitability is considered more important than the low degree of maintainability.

To summarize, the evaluation shows that each platform has its strengths and weaknesses, but the comparison framework that was established has a high level of subjectivity and many of the quality attributes have an insufficient number of metrics. For future work, I would recommend revising the framework by doing a separate study on which metrics to use, and then incorporate these metrics into the existing DoSAM e-commerce comparison framework. Also, more time should be spent defining the services and components of the platforms, so that they can be evaluated separately to a

higher degree than what was possible during this evaluation. It would also be desirable to define criterias for reference systems so that performance and maintainability can be measured more accurately.

The solution presented in ch. 8 provides an overview of how to develop modules in Magento and how two modules from an existing system was replicated in Magento CE 1.6.2. The chapter provides insights into Magento development in general, and how the frontend and backend components of Mataffären.se can be implemented, specifically. In relation to this, the following section discusses the implementation process with regards to the evaluation results for Magento.

9.1 Magento implementation process in relation to evaluation results

The evaluation results for the Magento CE platform should be especially interesting since an actual implementation was part of the goal for this thesis. The evaluation results for the platform indicates a relatively high degree (a score of 90+) of functional suitability, portability and market readiness but a lower (-75) degree of performance efficiency and maintainability.

The installation and configuration process of the platform was perceived as relatively straight forward, with few steps to perform. The technical platform on which the solution had to be installed was also very common (Apache/MySQL/PHP) for web applications, and required few adjustments from default environment settings. This can be interpreted as cohesive in relation to the portability attribute evaluation result.

The functional suitability score of the services can also be seen as quite cohesive with the implementation process experience, as the components that were needed were all there to extend and utilize, such as product catalog functionality, availability of product attribute services, and administration abilities through the administration panel. However to fully assess the correctness of the evaluation score for this attribute, it is assessed that a more complete implementation process is needed, for example a project that touches all the necessary steps up to a live release, including performance test. This can also be said of the Market readiness and Performance efficiency attribute.

The Maintainability attribute can be considered as the attribute most likely to have been evaluated most out of all the attributes, since a new design was created from scratch and two new modules were implemented. The attribute should consider such factors as Modularity, Modifiability and Reliability. However it is difficult to assess these factors without a risk of a high degree of subjectivity from the developer. The only thing that can be said for certain is that the functionality that was intended could be achieved within the given time frame, the solution could be debugged using the IDE and that the framework's internal MVC implementation was utilized, which were some of the metrics with which to measure the degree of Maintainability. Weather or not the scores were accurate or not cannot be assessed by the author.

10 Discussion

This chapter describes and discusses problems that arose during the evaluation and implementation.

It was very difficult selecting and applying an evaluation method that had a suitable balance between being practical, having a high degree of objectivity and being able to apply within the given time frame of the thesis work. For the evaluation to be useful for Softronic, it had to take into account factors that lay close to the actual implementation of the platforms. On the other hand, evaluation methods in general tend to abstract things and add a lot of overhead on the evaluation procedure. Additionally, selecting attributes and defining how to measure them, while maintaining both an abstract and a practically useful perspective, has proven to be very difficult.

The original intention of the comparison was to make it relatively independent of the implementations, and only measure the abilities of the architectural foundations. Further into the work, it was clear that to make the comparison useful, certain implementation-specific attributes could simply not be left out. This moved the perspective from an abstract viewpoint further and further into an implementation-specific viewpoint. One of the consequences of this was that the Umbraco CMS system could no longer be evaluated without selecting an e-commerce implementation. This was not the original intention of use of the comparison framework, adding to the complexity of the task and forcing the work into essentially making difficult metrics choices.

For this reason, the biggest weakness of this thesis is probably the subjectivity of the metrics used, although the application and treatment of them have been according to the comparison framework's specification. Nevertheless, this is something to consider for those that might want to use the comparison framework, and especially the author's implementation of it, in the future. One might add that the subjectivity of the metrics is a bigger risk that the choice of attribute weights, since the weights are much easier to adjust later in the evaluation.

Regarding functional suitability it has been suggested by, for example, Sommerville, that functionality in general is very difficult to measure since it tends to be very dependent on the person performing the evaluation and therefore quite subjectively measured. This thesis work is no exception. The treatment of the attribute has its weaknesses, for example it takes a certain amount of metrics to achieve a somewhat reliable result, while the metrics have to be selected very carefully so that they are not too implementation-specific, in order to maintain an abstract platform-independent perspective. On the other hand, too few metrics will yield a result that is too abstract and doesn't say very much about the actual functionality of the platforms.

On some of the quality attributes, the author has had to rely almost solely on qualitative metrics, which can be said to make the evaluation even more subjective. The other side of this is that, combined with reasoning about the metrics and the measurements, usage of qualitative metrics provides a more useful overview of the platforms than measuring purely quantitative attributes such as call depth, lines of code etc. Once again it is a balance between the limited scope of the thesis, usefulness of the evaluation and maintaining objectivity.

Some of quality attribute measurements can also be considered more to be valid than others. For example, the performance efficiency attribute could only be evaluated using three metrics. To achieve a reliable evaluation result, one would have to test the platforms in a much more systematic way, using for example stress testing, load testing and spike testing. Also, the market readiness attribute should be further developed, since only some of the services were able to have the metrics applied to them. This suggests that a more thorough metrics definition process would be needed, so that metrics can be developed that reveals strengths and weaknesses to a higher degree.

It can also be argued that for the maintainability attribute, a present or non-present MVC design pattern implementation is not something that should be measured as a binary score. Some platforms may have their own way of separating views from content. Also, the level of documentation can differ between different parts of a system, meaning that, for example, developers working with the commerce area does not experience the same issues as a developer working in, for example, only the CMS area. Also, more metrics would be desirable to measure the complexity of a system, including the contents of the DLL files, and not just the number of source files. Development tools have been identified that would be suitable for measuring, for example, cyclomatic complexity and the number of methods within the source code.[49] However it was assessed that this type of measurement would require a thorough definition and setup of equally complex reference projects for each of the platforms, which was estimated to be out of the scope of this thesis. For further work, though, these measurements should be added to the comparison framework to provide a more accurate estimation of the maintainability attribute.

It can also be noted that, in many cases, the scores are the same for all the services of the platforms. This can be linked to difficulties in isolating and differentiating what the services actually do, respectively. It can be discussed weather it is correct to assign the same metric values to all services, or if the services should not be measured at all in those cases.

In all, the comparison framework can be said to be relatively specific for the task of this thesis and for the company interested in the platforms. This also applies, of course, to the Magento implementation solution. Those that want to use the comparison framework in the future, and apply it to other platforms, should definitely consider redefining parts of the comparison framework, especially the quality attribute metrics.

49 For example the Code metrics option in Visual studio, or SONAR, an open source alternative.

11 References

Alan Storm. *Magento for developers.* http://www.magentocommerce.com/knowledge-base/entry/magento-for-dev-part-1-introduction-to-magento, retrieved 2012-05-06.

Banani R., and Graham, N. (2008). Methods for Evaluating Software Architecture: A Survey. Queen's University at Kingston (Ontario, Canada).

Bergner, K., Rausch, A., Sihling, M. and Ternité, T. (2005). *DoSAM – Domain-Specific Software Architecture Comparison Model.* First International Conference on the Quality of Software Architectures, and Second International Workshop on Software Quality, LNCS 3712, pp. 4 – 20, 2005.

Dobrica, L. and Niemela, E. (2002). *A survey on software architecture analysis methods.* IEEE Transactions on Software Engineering, July 2002, Vol.28(7), pp.638-653.

Episerver. *Episerver Commerce system requirements.* http://world.episerver.com/Documentation/Items/System-Requirements/EPiServer-Commerce/System-Requirements---EPiServer-Commerce-1-R2-SP2-/

Episerver. *Episerver Labs.* http://labs.episerver.com/en/Blogs/, retrieved 2012-05-31.

Episerver. Episerver World. http://world.episerver.com/, retrieved 2012-04-30.

http://win.wipcore.se/Library/Index/Wipcore.eNOVA.Technical.Overview

HUI Research AB. *E-barometern* (2011). http://www.hui.se/statistik-rapporter/index-och-barometrar/e-barometern

ISO.org (2011). *ISO/IEC 25010:2011.* http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733.

Jajja. *SEO-test av Wipcore eNova 5.3.* http://seotest.jajja.com/sv/2011/e-handel/svar/wipcore-enova-5-3/, retrieved 2012-05-30.

Jarl, M. (2008). *Evaluation of three e-commerce platforms.* Datavetenskap och kommunikation, Kungliga Tekniska högskolan, Stockholm.

Magento Inc. *Magento Designer's Guide.* http://info.magento.com/rs/magentocommerce/images/MagentoDesignGuide.pdf., retrieved 2012-05-18.

Magento Inc. *Magento Documentation.* http://docs.magentocommerce.com/Mage_Catalog/Mage_Catalog_Helper_Data.html, retrieved 2012-05-18.

Magento Inc. *Magento Knowledge base.* http://www.magentocommerce.com/knowledge-base, retrieved 2012-04-15.

Magento Inc. *Magento system requirements.* http://www.magentocommerce.com/system-requirements, retrieved 2012-05-15.

Microsoft. *C# Programming Guide.* http://msdn.microsoft.com/en-us/library/dfb3cx8s.aspx, retrieved 2012-05-18.

Softronic AB. *Softronic.se* (2012). http://www.softronic.se

Sommerville, I. (2011). *Software Engineering.* Addison-Wesley.

Stefani, A. and Xenos, M. (2008). *E-commerce system quality assessment using a model based on ISO 9126 and Belief Networks.* Software Quality Journal, 2008, Vol.16(1), pp.107-129.

Stefani, A. and Xenos, M. (2001). *A model for assessing the quality of e-commerce systems.* Proceedings of the PC-HCI 2001 Conference on Human Computer Interaction, Patras, pp. 105-109, 2001.

TeaCommerce. *TeaCommerce system requirements*. http://www.teacommerce.dk/en/documentation/system-requirements.aspx

Umbraco. *The Umbraco documentation wiki.* http://our.umbraco.org/wiki/, retrieved 2012-05-21.

Umbraco. *Umbraco system requirements*. http://our.umbraco.org/wiki/recommendations/recommended-reading-for-it-administrators/minimum-system-requirements

van der Merwe., R. and Bekker, J. (2003). *A framework and methodology for evaluating e-commerce Web sites.* Internet Research: Electronic Networking Applications and Policy, Vol.13(5), p.330-341.

Wikipedia. *Convention over configuration.* http://en.wikipedia.org/wiki/Convention_over_configuration, retrieved 2012-05-06.

Wikipedia. *ISO/IEC 25010:2011.* http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733, retrieved 2012-04-10.

Wikipedia. *ISO/IEC 9126*. http://en.wikipedia.org/wiki/ISO/IEC_9126, retrieved 2012-04-10.

Wikipedia. *Usability*. http://en.wikipedia.org/wiki/Usability, retrieved 2012-04-10.

Wipcore. *API documentation.* http://win.wipcore.se/ApiDoc/html/ApiDoc.aspx, retrieved 2012-05-31.

Wipcore. *Library: Technical documentation.* http://win.wipcore.se/Library/List/Guideline, retrieved 2012-05-31.

Wipcore. *Wipcore eNOVA 5.1 system requirements.* http://win.wipcore.se/Articles/News/eNOVA51-release#SystemRequirements

Wipcore. *Wipcore Information Network*. http://win.wipcore.se/, retrieved 2012-05-21.

Wipcore/Enova leaks. *Kommande release for WebFoundation*. http://enovaleaks.se/2012/05/15/kommande-release-for-webfoundation/, retrieved 2012-05-31.

Zend Technologies Ltd. *Zend New BSD Licence*. http://framework.zend.com/license, retrieved 2012-04-03.

Zend Technologies Ltd. *Zend Programmer's Reference Guide.* http://framework.zend.com/manual/en/introduction.overview.html, retrieved 2012-04-03.

Zend Technologies Ltd. *Zend.com*. http://www.zend.com

Appendix A: ISO/IEC 25010:2011 Product Quality Model

Category and description	Subcategory	Subcategory description
Functional suitability - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.	Functional completeness	Degree to which the set of functions covers all the specified tasks and user objectives
	Functional appropriateness	Degree to which the functions facilitate the accomplishment of specified tasks and objectives
	Functional correctedness	Degree to which a product or system provides the correct results with the needed Degree of precision
Reliability - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.	Maturity	Degree to which a system meets needs for reliability under normal operation
	Fault Tolerance	Degree to which a system, product or component operates as intended despite the presence of hardware or software faults
	Recoverability	Degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system
	Availability	Degree to which a system, product or component is operational and accessible when required for use
Usability - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.	Appropriateness recognizability	Degree to which users can recognize whether a product or system is appropriate for their needs
	Learnability	Degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use
	Operability	Degree to which a product or system has attributes that make it easy to operate and control
	User error protection	Degree to which a system protects users against making errors
	User interface aesthetics	Degree to which a user interface enables pleasing and satisfying interaction for the user
	Accessibility	Degree to which a product or system

		can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use
Performance Efficiency - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.	Time Behavior	Degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements
	Resource Utilization	Degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements
	Capacity	Degree to which the maximum limits of a product or system parameter meet requirements
Maintainability - A set of attributes that bear on the effort needed to make specified modifications.	Modularity	Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components
	Reusability	Degree to which an asset can be used in more than one system, or in building other assets
	Analyzability	Degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified
	Modifiability	Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality
	Testability	Degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met
Portability - A set of attributes that bear on the ability of software to be transferred from one environment to another.	Adaptability	Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments
	Installability	Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment
	Replaceability	Degree to which a product can be replaced by another specified software

		product for the same purpose in the same environment
Compatibility -	Co-Existence	Degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product
	Interoperability	Degree to which two or more systems, products or components can exchange information and use the information that has been exchanged
Security	Confidentiality	Degree to which a product or system ensures that data are accessible only to those authorized to have access
	Integrity	Degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data
	Non-repudiation	Degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later
	Accountability	Degree to which the actions of an entity can be traced uniquely to the entity
	Authenticity	Degree to which the identity of a subject or resource can be proved to be the one claimed

Appendix B: Evaluation data

For metric definitions, see ch. 6.4 Quality Attribute Metrics. Some metric values are market as "-". It was assessed during the evaluation that these metrics were unable to be determined for the given service. This approach, and how it may impact the evaluation, is discussed in ch. 9 Conclusion.

Functional suitability

Architectural Service	#	Quality Metric	Mag	Epi	Wip	Umb
Presentation service	1	Ability to fully customize the checkout page	1	1	1	1
	2	Popular search terms are stored and made available for display	1	1	0	0
	3	Navigation help available (breadcrumbs etc.)	1	1	1	1
	4	Customer can create own account	1	1	1	1
	5	Store search available	1	1	1	1
	6	Electronic shopping cart available	1	1	1	1
	7	Customers can add reviews for products	1	1	0	0
	8	Customers can add products to a Wishlist	1	1	0	0
	9	Electronic payment service available	1	1	1	1
	10	Translation services available	1	1	1	1
	11	Product categorization is available	1	1	1	1
	12	Sales and discounts can be set	1	1	1	1
	13	Newsletter integration available	1	1	1	0
	14	Customer can view recently viewed, compared and added products	1	1	1	0
	15	"Send to friend" feature available	1	1	0	0
	16	Customer can be notified by email for new orders and invoices	1	1	1	0
	17	Customer can set default shipping and invoice address	1	1	1	1
	18	Designs can be set per product category and per product	1	1	1	0
	19	Customer can compare products	1	1	0	0
	20	Tags can be added to products by customers	1	1	1	0
	21	Customer can save shopping cart for later checkout	1	1	1	1
	22	Customer can use multiple shipments per order	1	1	1	0
	23	Customer can track order on personal page	1	1	1	1
	24	Guest checkout is possible	1	1	0	1
Store management service	25	Products can be grouped and sold as a bundle	1	1	1	1
	26	Support for table based rates for shipping	1	1	1	0
	27	Control multiple stores from the same installation	1	1	0	0
	28	Standard content pages can be produced and modified through the administrative interface	1	1	1	1
	29	Support for multiple currencies and tax rates	1	1	1	1
	30	Automatic site map generation	1	1	1	1
	31	Ability to add URL rewrites for individual pages and products through the administrative interface	1	1	0	1
	32	Ability to set Meta tags for products	1	1	0	1
	33	Ability to generate reports on sales	1	1	1	1
	34	Customer can use coupon codes	1	1	1	0
	35	Tier pricing available	1	1	1	1
	36	Multiple product type relations (grouped, bundled etc.)	1	1	1	0

	37	Ability to use customer groups for different pricing	1	1	1	1
	38	Invoices created separately for orders	1	1	1	0
	39	A control panel is available for store management	1	1	1	1
	40	The administrator can view, edit and create orders in backend	1	1	1	1
	41	The administrator can view, edit and create invoices in backend	1	1	1	1
	42	Administrator can track customer carts in backend	1	1	1	1
	43	Administrator can edit customer properties	1	1	1	1
	44	Order and invoice emails can be customized	1	1	1	1
	45	Downloadable product type is available	1	1	0	1
	46	Administrator can update multiple products at once (batch updates)	1	1	1	1
	47	Inventory options can be set per product	1	1	1	1
	48	Different attributes for different product groups can be set	1	1	1	1
	49	Special prices for products can be set	1	1	1	1
	50	Price between certain dates can be set	1	1	1	0
	51	User can receive an email when products become in-stock	1	1	0	0
	52	Campaigns based on customer groups	1	1	1	1
	53	Individual shipping costs per product	0	1	0	0
	54	Restricted product views based on customer group	0	1	1	0
	55	Product lists can be included on content pages	1	1	1	1
	56	Ability to set catalog promos by percentage or fixed price	1	1	1	0
Data storage and encryption service	57	SSL support	1	1	1	1
	58	PCI compliance	1	1	1	0
Data access services	59	Ability to integrate the solution with Google analytics, without having to edit source code	1	1	1	0
	60	Ability to retrieve shipping rates from external suppliers	1	1	1	0
	61	Web Services API available	1	1	0	1
	62	Ability to integrate shop with external payment suppliers	1	1	1	1
	63	Ability to install modules and plug-ins through the administrative interface, for example new payment modules	1	1	0	1
	64	Built-in support for import and export of products, orders and customers via text or xml files	1	1	0	1

Performance Efficiency

Architectural Service	**#**	**Mag**	**Epi**	**Wip**	**Umb**	**Comments**
Presentation service	1	75	75	0	50	The Magento implementation was used as a reference and load tested with 10 concurrent user sessions. Additional sources have been found that indicates a memory usage of an average Magento installation to be around 1-2 GB, which suggests that 4 GB should be a suitable number for a Magento business server. For Episerver, no performance issues have been noticed by the interviewed. The platform has been deployed on large solutions. One of the reference projects have been found to run on a server with 4 GB of primary memory, although interviews pointed towards a higher number. 8 GB is estimated as a reasonable figure for the business server of Episerver. Wipcore's way of increasing performance is to load up almost all of the contents into the primary memory. This means that the amount of memory used depends on how much data is in the database (how many products, order etc.). This will entail a high memory usage on most store instances. For example, on one of the reference projects a solution uses up to 16 GB of memory. No performance issues have been noticed by the interviewed, and the platform has been deployed on large solutions. For this reason, a score has been given based on the 16 GB figure that indicates that the platform uses up a relatively large amount of memory, however it should be noted that has not been identified as a problem by any of the developers. Regarding Umbraco, interviews have indicated that indicates that the 4.7 version of the CMS requires a minimum of 8 GB. It can be noted though that there are indications of a higher memory usage in the 5.x version.

	2	50	100	100	100	Magento CE does not feature full page cache, although this is present in the Enterprise version. There is, however, other types of caching abilities available. Episerver Commerce provides caching abilities for page requests and, at a deeper level, data cache. Umbraco provides full page caching, while Wipcore achieves a high score because of the strategy of caching all relevant content in the primary memory.
	3	30	50	90	100	Based on interview answers, it has been estimated that Magento provides, in the context of the other platforms, the weakest possibilities since the language is not strongly typed. Interviewees have expressed opinions on Wipcore such that it is assessed as more aimed at facilitating low-level access to components, thus generating a higher score. While developers felt that for Episerver, lack of documentation and weaknesses in code structure should give a lower score for this metric, no such opinions were expressed regarding the Umbraco CMS. In addition, TeaCommerce implements the XSLT standard used in Umbraco to facilitate access to content nodes.
Data storage and encryption service	1	88	50	80	50	See comments for Presentation service.
	2	50	100	100	100	
	3	30	50	90	100	
Data access services	1	88	50	0	50	
	2	50	100	100	100	
	3	30	50	90	100	

Maintainability

Architectural Service	**#**	**Mag**	**Epi**	**Wip**	**Umb**	**Comments**
Presentation service	1	90	97	10	98	5000 files were found in the /app/code/core/Mage directory. For Episerver, interviewees claims about 1500 source files, excluding DLL:s. Wipcore's API documentation claims there are 45.000 files in the API.. 1200 source files were found in the Umbraco 4.7.2 source package. TeaCommerce, though, adds functionality through DLL:s and Javascript but not source files directly. It is therefore difficult to measure the total number of files for the CMS with TeaCommerce.
	2	100	0	100	0	Magento CE implements the MVC design pattern, as well as Wipcore eNOVA (ASP.NET MVC3). Episerver CMS with Episerver Commerce does not, and neither does Umbraco 4.x, although it is introduced version 5.x.
	3	70	50	50	80	For the presentation service, a Magento Designer's guide is the only official documentation. Several interviewees expresses a lack of documentation for the commerce solutions in Episerver and Wipcore. Interviewees expresses no issues with the Umbraco documentation,, although the TeaCommerce plugin is not part of this documentation which lowers the overall score.
	4	50	50	90	100	Since PHP is loosely typed, debugging a Magento solution is assessed as more difficult than a solution written in a strongly typed language such as C#. The additional layout separation using layout files and blocks also contributes to a lower score for the platform. For both Episerver and Wipcore, interviewees have expressed that it is often difficult to debug the solutions due to the nature of the code. No such issues have been identified with the Umbraco CMS.
Store management service	1	90	97	10	98	See the comments for Presentation service.
	2	100	0	100	0	
	3	90	50	50	80	The Magento frontend documentation can be used to adapt the store management service as well. There is also additional documentation for the configuration parameters of the service available. No differences in documentation could be identified for the other platforms.
	4	50	50	75	100	It is estimated that the MVC design pattern influences the presentation service to a larger degree than the other services. For this reason, the Wipcore eNOVA is given a lower score due to interviewees expressing difficulties in debugging the solution due to the large code base and lack of documentation.

Data storage and encryption service	1	90	97	10	98	
	2	100	0	100	0	
	3	25	75	50	80	The Magento platform lacks official documentation for this service.
	4	50	50	75	100	
Data access services	1	90	97	10	98	
	2	100	0	100	0	
	3	25	75	50	80	The Magento platform lacks official documentation for this service.
	4	50	50	75	100	

Portability

Architectural Service	**#**	**Mag**	**Epi**	**Wip**	**Umb**	**Comments**
Presentation service	1	100	80	100	80	The HTML/PHP framework that constitutes the presentation service of Magento is assessed as a framework built on very common components, using standard configuration parameters on Apache. It is estimated that this impacts positively on the Portability attribute. The same can be said about Wipcore, which runs on a generic .NET configuration with IIS. Episerver and Umbraco, however, are assessed as more vulnerable to configuration changes because of their dependencies of a separate CMS framework, compared to the other platforms.
	2	70	80	80	80	In terms of backwards-compatibility, it is assessed that Wipcore eNOVA and Umbraco are the platforms that is the most tolerant for earlier generations of environments such as IIS6 and .NET 3.5. Magento requires a newly released PHP version, although it can be run on a very old version of Apache.
	3	100	100	30	100	For Magento, it is assessed that the process of deploying the presentation service as a cloud service would be fairly straight forward, since the PHP/Apache environment is offered by many cloud hosting providers. Episerver has build-in support for Amazon S3 cloud deployment, which is translated into a high evaluation score. The same can be said about Umbraco, which can easily be transferred and deployed into the Windows Azure environment using the Windows Azure Accelerator. No similar support or process information regarding cloud deployment could be found for Wipcore eNOVA.
	4	80	80	80	80	Interviewees expresses opinions about some of the platforms that can be interpreted as indications on metric values, although these opinions are assessed as not necessarily related to installing and setting up the presentation service. None of the interviewees expresses that the installation is particularly dependent on external resources such as third-party consultants. Therefore, it is estimated that all platforms share similar degrees of dependence on support and external advice.
Data storage and transaction service	1	100	80	100	80	No distinct disadvantages could be found regarding the processes of setting up the databases for the solutions.
	2	80	100	80	80	The only difference from the Presentation service is the score for the relatively high backwards compatibility of the database service (MySQL) for Magento.
	3	100	100	30	100	See “Presentation service”.
	4	80	70	80	70	The need for setting up separate databases for the CMS and the e-commerce solutions, in the cases of Episerver and Umbraco, are interpreted as having a negative impact on the score.

Market Readiness

Architectural Service	**#**	**Mag**	**Epi**	**Wip**	**Umb**	**Comments**
Presentation service	1	97	90	90	97	For Magento, tt is estimated that 3 man days is required. That evaluates as 100-3 = 97. For Episerver and Wipcore, based on interview answers, 10 man days is estimated. In Umbraco,

						the process of setting up an Umbraco CMS with TeaCommerce is estimated to take about 3 man days, seeing that the process requires fewer configuration steps that both Wipcore and Episerver.
	2	100	75	30	99	Magento CE is licensed as open source, meaning no license costs are required. License costs (list prices) have been used as metrics for Episerver Commerce, Wipcore eNOVa and Umbraco with TeaCommerce.
	3	99	99	99	99	All systems features a built-in Google Analytics integration, meaning the settings for the account can be entered directly in the administration panel.
	4	99	99	99	40	All systems except Umbraco feature the ability to automatically enable SEO-friendly url:s. The settings can be edited in the administration panel.
	5	-	-	-	-	
Store management service	1	97	90	90	97	Configuration of CRM related parameters for the applications is estimated to require an equal amount of days as the Presentation service.
	2	-	-	-	-	
	3	-	-	-	-	
	4	-	-	-	-	
	5	-	-	-	-	
Data access services	1	-	-	-	-	
	2	-	-	-	-	
	3	-	-	-	-	
	4	-	-	-	-	
	5	100	100	100	20	For all the platforms except for Umbraco/TeaCommerce, at least five payment could be found. It should be added that the Umbraco platform allows for developers to easily create their own payment plugins, however it is assessed that the platform lacks many out-of-the-box features for payment modules compared to the other three platforms.