Evaluation of Frameworks for Desktop-Like Web Applications in Pure JavaScript

Master's Thesis

Graz University of Technology
Institute for Information Systems and Computer Media (IICM)

submitted by

Thomas Billicsich

Advisor: Assoc.Prof. Dipl.-Ing. Dr.techn. Martin Ebner

26 Sep 2012

Abstract

Some recent web applications - also called rich internet applications (RIA) - try to mirrow the look and feel of the desktop platform and frameworks are an important part of the development of these desktop-like or single-page web applications. To find suitable ones is a crucial and demanding task.

This thesis presents a methodology for the selection of software components and shows a process that is adjusted to find suitable single-page web application frameworks. A central aspect is the evaluation model. An extensive evaluation model is developed based on the criteria categories: documentation, community, features, and user interface. It is enriched with enough background information to facilitate modification for employment of the method in an individual software project.

To demonstrate a practical application the current market for web application frameworks is researched. 41 candidates are found, eight are selected and an evaluation is performed. Three frameworks can be recommended for the development of desktop-like web applications.

A validation in the form of an implementation of a prototype web application generates feedback for the model and the process and presents possibilities for future improvements of both.

Kurzfassung

Einige aktuelle Web Applications, oder Rich Internet Applications (RIA), versuchen den Look and Feel von auf dem lokalen Rechner installierten Desktopprogrammen nachzuempfinden. Frameworks sind ein wichtiger Teil um die Entwicklung von diesen desktop-ähnlichen oder single-page Web Applications zu ermöglichen, aber aufgrund des großen Angebotes ist das Finden und Erkennen von geeigneten Kandidaten eine zentrale Herausforderung.

Diese Masterarbeit stellt eine Methode zur Auswahl von Softwarekomponenten vor und zeigt eine Vorgangsweise, die es ermöglicht, geeignete Frameworks zu finden. Ein wichtiger Teil dieses Prozesses ist das Evaluierungsmodell. Das hier ausführlich beschriebene Modell beinhaltet folgende Kriterienkategorien: Dokumentation, Community, Features, und User Interface. Die zahlreichen gezeigten Hintergrundinformationen helfen, dieses Modell und die Methode an die eigenen Bedürfnisse anzupassen.

Um eine praktische Anwendung zu zeigen, werden aktuelle Frameworks ausgewählt und evaluiert. Von 41 gefundenen Kandidaten werden acht ausgewählt und evaluiert. Aus diesen acht werden drei als geeignet für die Entwicklung von desktop-ähnlichen Web Applications klassifiziert.

Durch eine Validierung in Form einer Prototypenimplementierung wird das Modell und der Prozess überprüft und aus dem Ergebnis Verbesserungsvorschläge entnommen.

Contents

Chapter 1

Introduction

The growth of the world wide web and the availability of a web browser on every device of daily use have increased the importance of being able to "do" something on the web: writing texts, tracking time and expenses, or managing contacts and emails. *Web applications* make all this possible without the need for specific installations on a desktop computer. In recent years an advanced form of web applications *single-page web applications* has emerged. This class tries to incorporate behaviour that users are familiar with from their desktop computers and is therefore also called *desktop-like*.

Frameworks are an essential part of the development process for this type of applications. Accordingly, a vast amount of them is available and to find the right candidate for the support of the project at hand has become a difficult task. This work sheds light on solving the problem of making the choice for a right candidate – usually there are more than one – with the help of an evaluation and exemplifies the procedure with a practical example.

The most important aspect of the evaluation is the model the candidates are rated against. It is strongly dependent on the desired outcome and the base for a formal and reproducible process. This thesis provides an extensive evaluation model to judge JavaScript frameworks by their suitability to support the development of single-page web applications.

By finding and evaluating currently available frameworks a practical application of the model is shown and suitable candidates for single-page web application development are presented.

Web application and *framework* are terms that are often and ambiguously used. Therefore, they are clearly defined in chapters 2 and 3. This definition serves as the context for the rest of the work.

The choice for an appropriate framework depends on several formal and informal circumstances as well as technical and non-technical requirements. Previous literature in the field often ignores already gained insights, which will be presented here (chapter 4, page 17), it will also be pointed out why it

is important to allow the use of literature about the well researched topic of off-the-shelve-software to guide the process. The findings are then incorporated into a complete decision process: a stage-based methodology for finding suitable candidates (chapter 5).

The evaluation model (chapter 6) which will be defined is the base for finding candidates that may be suitable (chapter 7) and for their assessment as well (chapter 10). Before the assessment, requirements for suitable candidates are defined (chapter 8) to filter candidates that can be easily identified as unfitting (chapter 9). With the help of a suitable candidate, the model will then be validated (chapter 11) and some of its weaknesses and strenghts be pointed out for future improvement.

Chapter 2

Web Applications

Web applications have been present since the beginning of the world wide web. In recent years however they have seen a significant increase in interest fueled by expanding technical possibilities and a stronger standards body. The so called browser war between Netscape and Microsoft in the late 1990s had left the internet in a place where even web designers had difficulties to write web pages that displayed correctly in all relevant browsers [Windrum, 2004].

The main objective of JavaScript running in the browser is the manipultion of the document object model (DOM), which was differing very strongly between browsers and made implementations working cross-browsers difficult and error prone. However, in the successive years three facts established, which are the main reasons for the advanced type of web application that is the topic of this thesis: (i) competition in the browser field led to a stronger force towards meeting the existing standards and improving them in collaboration, (ii) this also stimulated significant performance enhancements of rendering engine, and (iii) libraries and frameworks that alleviated the difficulties of working with the DOM were written and improved.

The following paragraphs serve to construct a context for web applications that share desktop-like qualities to be used for the presentation of the research in this thesis; the definition is not intented as a general reference.

2.1 What is a Web Application

A web application is an application that runs in a browser. It usually is not saved locally but accessed over the Internet or an internal network. It can rely on the standard technologies HTML, CSS, JavaScript, or require the use of plug-ins like Adobe Flash, Microsoft Silverlight, or Java.

In its technologically simplest form a web applications consists of simple HTML web pages generated dynamically in response to user requests. Users deliver information about their intention with information contained in

URL paths, parameters, or cookies, the server generates a response accordingly. Most users would not feel like they are interacting with an application, but merely like surfing the web. In its technologically most complex form a web application is rendered as a single page and changes its appearance in response to external events like a mouse click or internal events like a state change. It tries to offer the same interaction possibilities and UI gestures as a native desktop-program.

An example of a simple web application is a search engine like *Bing*, *Ask* or *DuckDuckGo*[1]. Simple in this case does not refer to the back-end, which for search engines is not simple at all, but to the user interface (UI) of the front end: first a single input field, then - after a page reload - a list of results. Slightly more complicated are map services like *Google Maps* or *Bing Maps*[2]. They offer interaction by mouse drag and dynamic loading of new content. Dynamic loading refers to the retrieval of new content without a page reload. This is a central property of modern web applications.

The opposite of web applications are native applications. These are applications that require a specific platform and run time environment. Running them on other platform is usually not possible. A good example is the browser itself: although Firefox is available for OS X, Linux and Windows, it is a native application; every platform has a version specifically compiled for it.

Platform independent applications that run in a platform-near environment but can also be started on other platforms form some middle-ground between native and web applications. The most popular choice is Java. Some popular applications exist, e.g. the two integrated development environments (IDEs) NetBeans and Eclipse, but otherwise the concept has not gained significant attraction.

2.2 Properties of Web Applications

It is worthwile to concern oneself with web applications for they have several advantages compared to native applications. Advantages that bring ease to developers and users in regard to the topic mentioned below. There are also drawbacks, so for each project those two sides have to be weighed for a sound decision of the implementation strategy.

2.2.1 Advantages

platform-independency: web applications run on every platform where a browser is available, which includes all popular desktop and mobile platforms.

[1] bing.com; ask.com; duckduckgo.com

[2] maps.google.com; bing.com/maps

Drawbacks: Although the rendering engines and virtual engines are standardized to a very high degree they still differ in details of their environment. Especially the differences in the Document Object Model (DOM) resulting from HTML, CSS and JavaScript processing can be problematic. Frameworks are a very appropriate method to alleviate this obstacle.

easy deployment: web applications require no installation, no setup helper, no uninstallation wizard; updates affect all users immediately, there is no need for support of legacy versions; once a web application is online it is instantly available to all computers with an internet connection [Anttonen et al., 2011].

Drawbacks: This is very convenient for developers, it can be problematic for users who rely on a specific feature that is removed in newer versions.

increased safety: data can easily be held in a central online storage with an advanced backup system, it can be protected from loss through hardware failures, accidental deletion or theft of personal devices.

Drawbacks: The user has to rely on the developer to protect the safety and security of his data. It can also be problematic to have the data stored in the range of another legislative system.

increased security: a professional and experienced team can protect the system running the application against malware and security threats more reliably than average users can protect their system.

Drawbacks: A single security breach affects all users.

collaboration friendly: since the application is already on the internet and also available for all platforms it is easy to add social or collaboration aspects [Anttonen et al., 2011].

easy usage metrics: a web application allows the permanent tracking of user behavior, which gives the developers valuable information about used or unused features and what paths the users take within the application. This encourages improvements to usability and efficiency.

Drawbacks: This can yield problems with privacy if the data is not anonymized appropriately.

2.2.2 Disadvantages

low performance: although virtual machines gained significant increases in run-time performance in the last years they still can not compete with natively compiled code.

But: Most applications an average user runs do not require significant processing power, computational demand is high only for specific types of applications (e.g. video editing with live preview) [Anttonen et al., 2011].

no access to local file system: storage of data on a device is possible but only in the browser environment, collaboration between native and web applications can not be handles through the local file system seamlessly [Taivalsaari et al., 2008].

But: Continues access to the local file system is unnecessary for lots of applications, uploads and downloads of files and folders are still possible. In fact modern operating systems start to restrict native application's access to local resources.

limited access to host platform capabilities: on most platforms access to the peripheral and sensorial equipment (microphone, camera, gyroscope, acceleration sensor, location services) is limited.

But: For security and privacy reasons this is an advantage.

2.3 Types of Web Applications

Three types of web applications can be identified, this work is only concerned with the third one.

2.3.1 Dynamically Generated but Static Web Pages

Historically these have been the first web applications. A very popular method was the implementation as a CGI script that was invoked upon a page request. Today the back-end is often comprised of a complex database handling application written in a high-level language like Ruby, PHP, C# or Java. The page is generated on the server and sent to the client, further interaction with the web application spawns a new request and a full page reload.

2.3.2 Rich Internet Applications with Plug-ins

Although a JavaScript virtual machine was already included in all web browsers in the late 1990s the language's capabilities were limited in regard to performance and standardization and the differences between browsers very huge. The demand for web applications led to the rise of plug-in based solutions via Java applets, Flash and later Silverlight widgets. Java is a kind of open technology and free development tools are available, Flash and Silverlight are proprietary technologies of Adobe and Microsoft. These plugins aim to provide "a more intuitive, responsive, and effective user experience" [Duhl,

2003]. Their significance as a method of delivering web applications is declining steadily, as both parent companies admit to HTML5 for future development [Winokur; Foley, 2011].

2.3.3 Dynamic Web Applications with JavaScript

The introduction of the XMLHttpRequest Object by Microsoft with IE7 [Microsoft, 2012] brought a central shift to web applications. It was now possible to load new content into a web page without a full page refresh. This concept later became famous. Called *Ajax* it initially stood for asynchronous JavaScript and XML but now involves the whole concept of asynchronous – i.e. without a full page refresh – data loading.

The extreme case of this concept with only one full page load, therefore called single-page, and the rest of the data loading happening chapterially and on demand, i.e. dynamically, is the main topic of this thesis.

2.4 Desktop-Like

Although it is a very vague and unscientific term *desktop-like* will serve well to encapsulate the idea of the type of web applications that are the subject of this research work. As an example, a webmail client is analyzed and the different concepts and ideas between static web pages and desktop-like applications are emphasized. The examples in this case are based on the real-life clients of the services GMX (static) and Apple iCloud (desktop-like), simplified for the purpose of demonstration and clarity.

2.4.1 Email Presentation in a Webmail Client

Start and login to the webmail client (figure 2.1) and display of one message's content (figure 2.2).

Static: A classic webmail client presents an initial page with login text fields and after the input of the user credentials loads a page listing the contents of the user's inbox. In most cases there will be a tree on the left side showing other folders like »deleted emails« or »drafts«, the address book and maybe even a calendar; all of these as links, the subjects of the emails are links too. Every click on a link starts a reload cycle where the user has to wait for the request to get to the server, for the server to generate a response, for this response to get back to the client, and for the web browser to rerender the new site. This involves the - probably cached - reload of included CSS and JavaScript files and their re-interpretation.

Desktop-Like: A desktop-like webmail client first needs to load and start up. This may take some seconds, then the login fields are presented and af-

ter providing the credentials the user sees an interface familiar from desktop clients: a 3-column layout with an icon bar at the top; leftmost the folder tree, in the middle column the message listing of the currently selected folder and rightmost the canvas for content of the currently selected message. Initially no message is selected and the canvas is empty. Clicking a message starts a load cycle with the same roundtrip, but the server only has to provide an object containing the email *data*, the layout is completely generated on the client-side. This is quicker since no CSS or JavaScript files have to be loaded but burdens the client with more computational demand since a layout for the received data has to be generated.

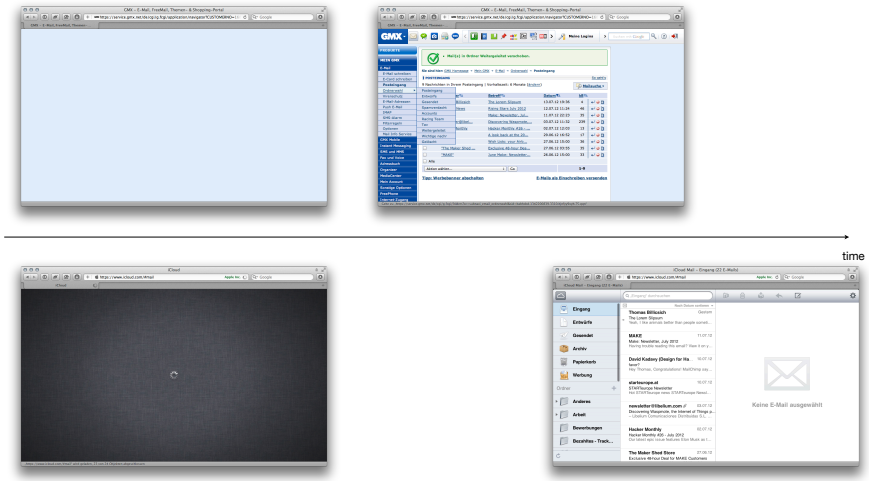


Figure 2.1: Transmitting and rendering a static web page is significantly faster than a desktop-like web application startup

2.4.2 Moving an Email in a Webmail Client

An email is to be moved from the *inbox* to another folder, e.g. *important*, *archive* (figure 2.3).

Static: In a conventional webmail client moving an email usually involves displaying it, selecting the folder to where it should be moved from a drop-down menu und then clicking a button to start the process.

Desktop-Like: In a desktop-like client an email is moved by dragging it from its current position to the desired folder.

2.4.3 Navigating Between Folders

A user starts with the contents of the inbox folder on display, changes to another folder, e.g. *important*, and then navigates back to the inbox folder.

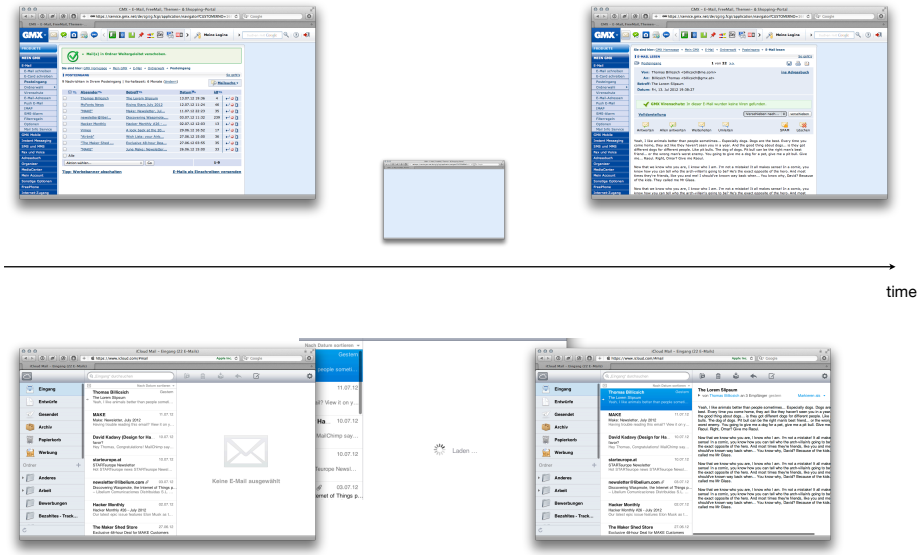


Figure 2.2: Displaying a message is faster when not a whole web page has to be transmitted and rendered

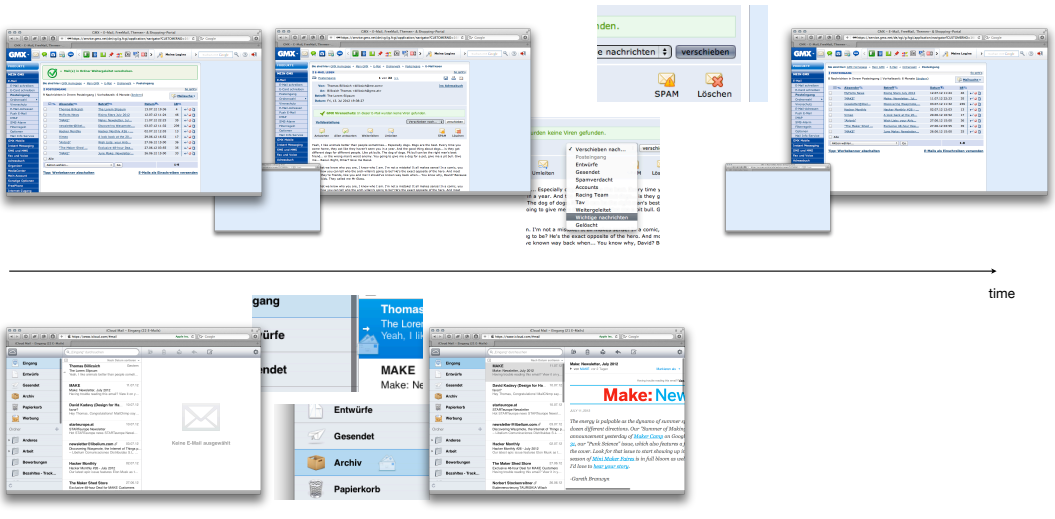


Figure 2.3: Moving in the upper sequence involves loading the message, selecting a folder, submitting, and loading the folder view again. Moving in the lower sequence is just dragging the message and dropping it on the desired folder.

	Initialization			Email Lookup		
	Server Load	Client Load	Data Transfer	Server Load	Client Load	Data Transfer
Static	High	Medium	High	High	Medium	High
Desktop-Like	Medium	Very High	Very High	Low	High	Low
	Static needs the server to generate the whole page with layout, the client needs to render the whole page. The desktop-like web application mainly consits of static files, the layout is completely generated and rendered on the client. A small dynamic part of the data loading consists of database lookups.		Static transfers the web page only. Desktop-like requires the transfer of the application, with the framework and with supplemental assets like icons.	Static is the same as on initialization. Whole page has to be generated on server and rendered on the client. Desktop-like needs the server to only look up the data and serve it, but the client has to generate and render the layout.		Static transfers the whole page, desktop-like only one data object.

	Email Move			Folder Navigation		
	Server Load	Client Load	Data Transfer	Server Load	Client Load	Data Transfer
Static	High	Medium	High	High	Medium	High
Desktop-Like	Low	High	Low	Low	High	Low
	For the static version three pages (inbox, email, inbox) have to be generated and rendered. The desktop-like version only needs to render the drag&drop and the new email, the server has to execute the move command and deliver the next email.		Three full web pages for the static version against one command to move the email location and the contents of the next email for the desktop-like client.	Again three full page loads for the static client. One listing load for the server, three times view generation and rendering for the desktop-client.		Three full web pages against the load of the listing and a refresh command.

Table 2.1: Performance comparison of different types of web applications

Static: Every-time the user changes the folder a new page on the server is generated, sent to the client, and rendered.

Desktop-Like: In a desktop-like client the contents of a folder are stored on the client-side, maybe even across sessions. This means that if the folder's content are listed for the first time the contents will be loaded and the user has to wait some time, but upon returning to the inbox the contents are shown immediately, at the same time a request will be sent to the server to see if the content has changed, if so the view will be updated accordingly.

2.4.4 Usability considerations

In addition to the differences in performance and amount of transferred data there is another important aspect: usability. The major usability improvements are:

- reduced idle time for the user while using the application
- more consistency through the elimination of the blanking of the screen while the page reloads and because the scroll position is not reset
- larger clicking target areas, since instead of a link a larger rectangle is the click area for changing folders or looking at messages

Chapter 3

Frameworks

Frameworks are an important reuse technique in object-oriented programming. They enable developers to build upon the knowledge, experience, and effort others have invested when designing and implementing a new application.

3.1 Definition

Altough frameworks are such a widely used technique a clear definition of them does not exist, or maybe it is *because* they are so ubiquitous that there exist different ideas about them.

A major source for every framework developer is the book "Building Application Frameworks" by Fayad et al. [1999]. In eight parts this book describes various aspects of frameworks. It is concerned with development, types, testing, documentation, and maintenance. One of the first mentions of the concept of a framework goes back to 1988, where Johnson and Foote described a method of reusing not only code but design and knowledge [Johnson and Foote, 1988].

These two sources point out clearly that the reuse of code alone is not enough. Frameworks are a means of reusing experience and knowledge of experts that has gone into the solution of a common and complex problem. The better and more mature the framework, the easier it is for less experienced developers to leverage the invested resources. Knowledge and experience manifest themselves in the interfaces and the structure of the components and the way they interact. A framework's main contribution to the custom application is the high-level design it imposes upon it. This is an aspect that is very important to this work. As can be seen later the architecture that the framework imposes provides a discriminating factor.

The fact that a high-level design is already pre-defined forces restrictions on developers, since they are bound to the architecture and the general idea

that the framework developers have chosen to solve the problem. However, a wise person may see this as a relieve, since the decisions that were made for and contributed to the maturing of a framework have already been tested in other applications and proved their suitability.

Quotes

"A framework is a set of classes that embodies an abstract design for solutions to a family of related problems, and supports reuse at a larger granularity than classes" [Johnson and Foote, 1988, ch1]

"The design of a program is usually described in terms of the program's components and the way they interact." [Johnson and Foote, 1988, ch4]

"A framework is a set of cooperating classes that make up a reusable design for a specific class of software." [Gamma et al., 1994, ch1.6 p26]

"a framework is a reusable design of all or part of a system that is represented by a set of abstract classes and the way their instances interact." [Fayad et al., 1999, ch1]

"a framework is the skeleton of an application that can be customized by an application developer." [Fayad et al., 1999, ch1]

d "A framework is a reusable design of a system that describes how the system is decomposed into a set of interacting objects. Sometimes the system is an entire application; sometimes it is just a subsystem." [Fayad et al., 1999, ch1.1]

"Frameworks also reuse implementation, but that is less important than reuse of the internal interfaces of a system and the way that its functions are divided among its components. This high-level design is the main intellectual content of software, and frameworks are a way to reuse it." [Fayad et al., 1999, ch1.1]

"...a framework's main contribution to an application is the architecture it defines." [Fayad et al., 1999, ch14 sb4 p345]

"A framework...imposes a design on your program, or at least on a certain problem space your program is trying to address." [Apple Inc., 2010, p126]

"collections of classes that structure a problem space and present an integrated solution to it" [Apple Inc., 2010, p129.3]

d "a framework maps out and implements an entire program structure — or model — that your own code must adapt to." [Apple Inc., 2010, p129.3]

3.2 Characteristics of Frameworks

To understand frameworks it is necessary to look beyond their presentation of code and documentation at a certain time. They are defined by:

Modularity: The skeleton application as a solution to a problem space is split up into modules with explicitly defined functionalities. Details of the implementation are hidden behind interfaces, which allows for modification of the encapsulated code to gain performance improvements or adapt to changes in the underlying hardware without global impact for example. The predefined division into modules that have a clear and defined relationship to other components of the application makes it easier to understand and maintain the software [Fayad et al., 1999, ch1 Application Frameworks].

Extensibility: Instead of developing an application from scratch software developers can rely on a proven architecture and extend the provided skeleton step by step until the solution is individualized to the desired degree and fits the requirements for the problem at hand. Furthermore, if there are extensions for specialized behavior available the developer can simply use these instead of a custom implementation [Fayad et al., 1999, ch1 Application Frameworks].

Control: In a framework application the main loop of events and their processing is controlled by the framework and developers are relieved of inventing an own system of control. The framework is responsible for handling external events originated from sources such as network communication or user interaction and decides which application-specific methods to call [Fayad and Schmidt, 1997].

Abstraction: The life-cycle of a framework usually starts as a concrete application to solve a specific challenge. Through hard work in several iterations and with domain knowledge gained by experienced developers it is abstracted to be useful to a family of related problems [Johnson and Foote, 1988; Fayad and Schmidt, 1997; Brugali et al., 1997].

Age: A framework is characterized by the way that it has been improved and adapted to extending problem fields. When actively developed its age is an important factor for stability of APIs, documentation, example code, and external contributions.

Complexity: Framework design is a very difficult and complex endavour. The framework architect has to aim towards flexibility and extensibility on one side [Gamma et al., 1994, ch1.6 p27] and towards a clear structure, ease of coding and good performance on the other side [Fayad et al., 1999, ch5.2]. Good framework design takes time and several iterations [Wirfs-Brock and Johnson, 1990].

Specialization: Frameworks are built as a solution to a family of related problems. This family is called the application domain and is an essential part of the framework concept [Korson and McGregor, 1992].

Outsourced Development: Using a framework means handing development and maintenance over to a large community or a compnay, it is a kind of outsourcing of a part of the work [Calefato and Lanubile, 2009, ch5.3].

3.3 Tasks of a Framework

The task of a framework at the application start is getting it up and running. This involves the creation and setup of the core group of objects at the program start. Some of these objects will be part of the framework and be the same for all applications, others may be partly or completely supplied by the developer. During runtime the framework is responsible for the handling of external and internal events, by deciding which internal or user provided method to invoke [Apple Inc., 2010, Starting Up].

A framework which provides support for a graphical user interface is responsible for the construction and management of this interface. Depending on the type of the framework and its intended use cases its duties may also include the saving and retrieval of objects from storage [Fayad et al., 1999, ch8 Harvesting Design].

3.4 Frameworks, Libraries and Toolkits

An important disctinction has to be made between these three types of software components. They are very often mixed, intersected or simply neglected any difference between them.

3.4.1 Framework

In this work the term *framework* is always used to denote a software as described in this chapter.

3.4.2 Library

Libraries are collections of code, they do not impose a design on the application. If necessary they get included, methods they provide are called and return a value or accomplish a specified task.

Typical library tasks in the environment of JavaScript are the handling of animation or the traversal of the DOM tree. Libraries are application independent and one can easily be switched one for another, if they provide the same API. The flow of control never enters the library [Apple Inc., 2010]. Libraries emphasize code reuse whereas frameworks emphasize design reuse [Gamma et al., 1994, ch1.6 p27].

3.4.3 Toolkit

This term is not often postulated in literature. Among the important sources for the topic of frameworks most notably Gamma et al. [1994, ch1.6 p27] and Johnson and Foote [1988, ch4.2 p13] have two very different understandings of what a *toolkit* is meant to be. Gamma et al. [1994] use the term toolkit instead of library. In this work this practice is not followed, so here toolkit stands for a set of programs that help the developer designing and implementing an application. Build tools are an example for a toolkite.

Chapter 4

State of the Art in Framework Evaluation

Framewoks are available in abundance. While this means that the probability of the existence of a suitable one is high, the question remains how this suitable one can be found. A number of techniques has been developed and this chapter sheds light on the state of the art of the steps and methodologies that aid in the process.

4.1 The Classification of Evaluation Methods

Selecting a framework has been an important part of software development since their introduction and therefore found its way into literature as well. In the context of the web however, two basic assumptions of many authors lead them to believe and complain that previous literature is scarce and the research field is unexplored [Changpil, 2012; Gerdessen, 2007]: (1) web frameworks are a separate category of framework and not comparable to enterprise or desktop frameworks, and (2) frameworks themselves are a category standing on its own within the field of software.

ad (1): Generally it is not stated why this should be the case therefore it is not appropriate to assume that it is true, when the definition of a framework (see chapter 3) holds true for web frameworks.

ad (2): A whole category of software development is based on the fact that pre-produced components are used and integrated into the custom project, the so called component based software development (CBS). This category is so common that the complexity of today's systems can not be handled without ready-made components. Research for this kind of software is available in abundance, it is called commercial-of-the-shelve software (COTS). COTS spans a broad field of application - academic, governential, corporate or military - and budget, from free to millions of dollars. Brownsword et al. [2000]

and Vigder et al. [1996] define COTS in different terms as a software that is:

- already in existence
- to be provided in many copies to multiple customers with minimal changes
- already improved upon its first design through a competitive marketplace
- has a vendor being responsible for support and maintenance
- used without source code modification
- without control over specification, schedule, or evolution by a single customer
- possibly lacking internal documentation and information about its limitations, performance or resource consumption
- usually accompanied by well developed user level documentation, customer documentation, and training

By this definition frameworks clearly belong to this category, which is confirmed by Vigder et al. [1996] and Kizzort [2002]. They go further on to establish frameworks as a distinct category within COTS, besides traditional components. *Traditional component* in this case refers to something like a word processor or an email client, applications that are incorporated into the custom software as-is or controlled from the custom code through scripting. Because of the principle of the inversion-of-control, frameworks are special in the way that the custom code interacts with them. Figure 4.1 shows the difference.

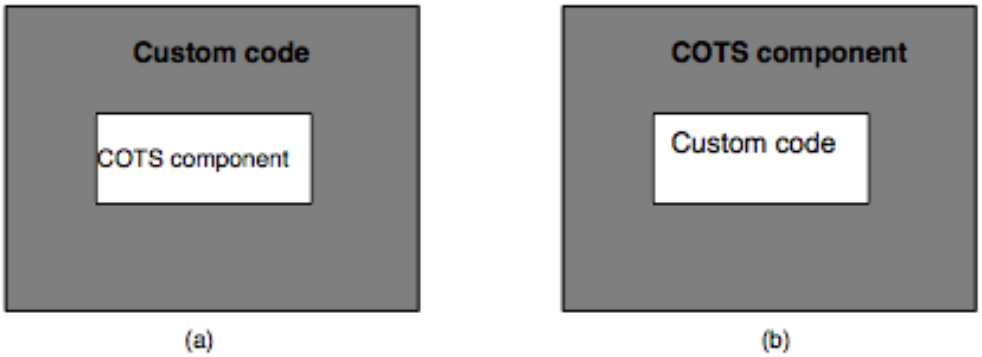


Figure 4.1: The conceptual place of custom code and COTS components for (a) traditional components and (b) frameworks [Vigder et al., 1996]

Very often frameworks are open-source software, a category that is different from commercial usually closed-source software and allows for more freedom in usage and adaption of functionality. However, studies showed that the possibility of source code modification and functionality alteration is rarely used and that usage of closed-source and open-source software hardly differs [Li et al., 2009].

4.2 Evaluation is not Decision

It is important to see the evaluation of the software packages as part of the decision process, often also called selection process, and neither as a means to itself nor as the whole process. It is a method that helps decision makers do their task [Oberndorf et al., 1997; Jadhav and Sonar, 2009]. Most decision processes follow a stage-based methodology and evaluation is only a part of it.

4.3 The Decision Process

Oberndorf et al. [1997] define the decision process as a methodology that includes the following tasks:

- determining fitness for use of those aspects of the component that are desired
- make-buy decisions
- choosing among products
- choosing among vendors
- discovering properties of the system
- validating claims
- making architectural, design, and requirement decisions

Jadhav and Sonar [2009] say that methodologies include the "factors and issues" relevant for selecting the right solution and they provide a good overview on commonly used methodologies. Another good overview is provided by Güngör Şen and Baraçli [2006]. The main stages of some of the methods these two papers mentioned are depicted in figure 4.2. They all differ in emphasis on aspects of the process or of an attribute domain but uniformly comprise evaluation as a separate stage.

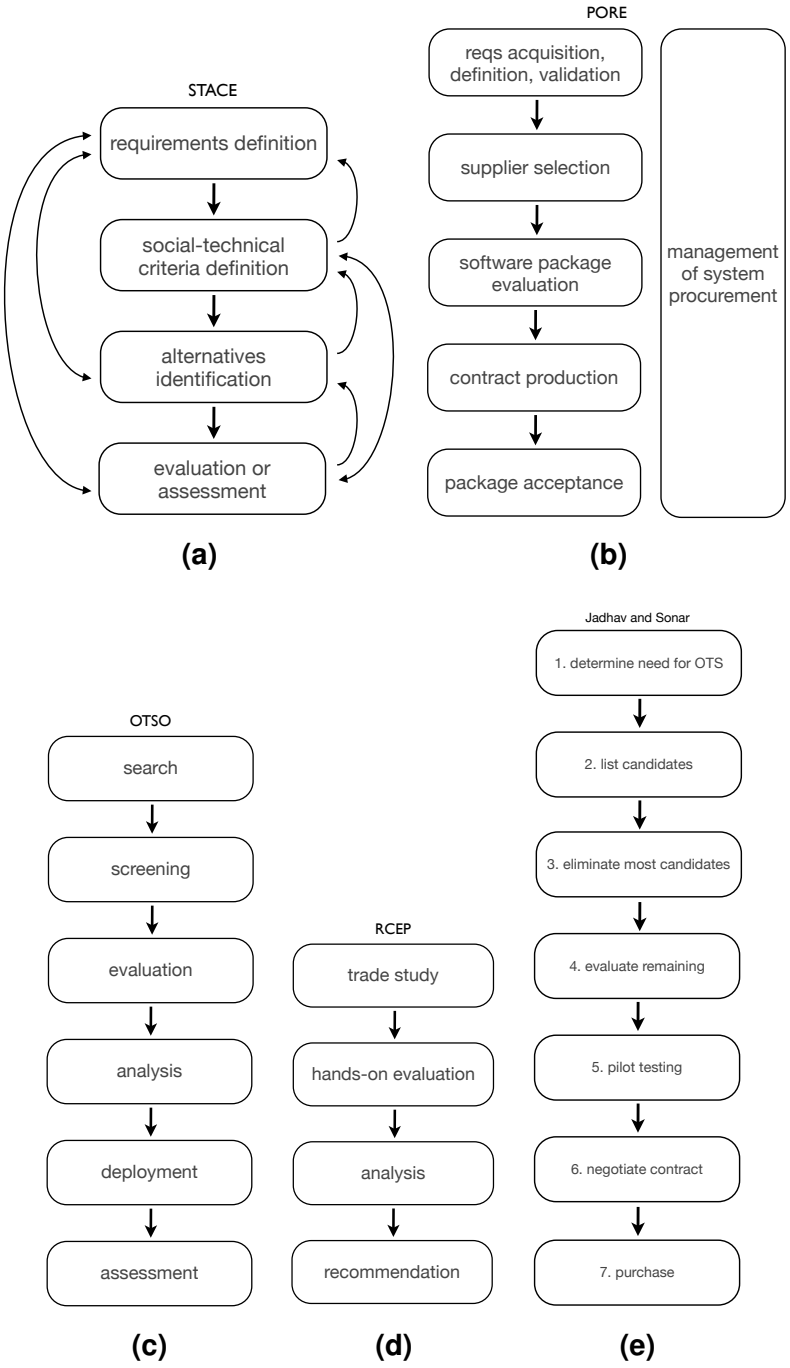


Figure 4.2: Various methods for a COTS decision process: (a) Social Technical Approach to COTS Software Evaluation [Kunda, 2003], (b) Procurement-Oriented Requirements Engineering [Ncube and Maiden, 1999], (c) Off-The-Shelf Option [Kontio et al., 1995], (d) Requirements-driven COTS product evaluation process [Lawlis et al., 2001], (e) a general process [Jadhav and Sonar, 2009]

Jadhav and Sonar [2009] propose a process based on seven stages they derived from 27 papers (figure 4.2 (e)). However, they do not mention a very important part which is present in most methodologies as a separate step: "Criteria determination and prioritization" [Güngör Şen and Baraçli, 2006], the construction of the comparison model.

The foundation of a documented and reproducible decision is a formal process. Several works point out the importance of this fact and hint at the suboptimal choice that stem from an informal decision process [Kitchenham et al., 1997; Calefato and Lanubile, 2009; Comella-Dorda et al., 2002]. They refrain from banning informal and subjective criteria like vendor reputation and familiarity, but state that they should only be used as part of a formal assessment method. Calefato and Lanubile [2009] had a first-hand experience when they relied mainly on those; they learned that "both attributes can lead to suboptimal choices, but while framework familiarity can flatten the learning curve, reputation can be totally misleading."

As approaches to a general formal process Mohamed et al. [2007] identify three ideas: (1) progressive filtering, (2) puzzle assembly, and (3) keystone identification. Progressive filtering employs an iterative approach where with every step criteria get more discriminating but also more intensive to evaluate, compensatory the number of fitting candidates decreases. Puzzle assembly is intended for the selection of several components for different parts of the system and targets their interaction. It relies on other sorts of evaluating relevant candidates for various roles and then tests combinations of these candidates mainly by developing prototypes with increasing complexity. Keystone identification realizes that there are certain non-negotiable requirements in a project, for example they may stem from a business need, software components outside of the project, or management decisions. Identifying these aspects and applying them to the candidate list can reduce the number of relevant candidates significantly. Although Mohamed et al. [2007] misinterpreted the idea of keystone identification, which Oberndorf et al. [1997] intended being a whole component comprising required aspects rather than a single pivotal criterion, their approach holds true und is used by others. Oberndorf et al. [1997] futher suggest that the presented ideas are not meant to be applied in isolation but as need arises. Within one project several switches from one evaluation idea to the other seem reasonable.

Two of theses three ideas are also found in the 7-step process developed by Jadhav and Sonar [2009] depicted in figure 4.2 (e). Step 3 would employ keystone evaluation by searching for the most discriminating requirements and step 4 would then look at the remaining candidates in details by progressively applying more refined criteria.

However, a uniform agreement can be seen on the fact that for an evaluation to be meaningful it has to be embedded in a context. The workshop conducted by Oberndorf et al. [1997] states that "evaluation for the sake of

evaluation" is without purpose. Other decision process descriptions explicitly point out that their process is strongly dependent on the context [Comella-Dorda et al., 2002; Brownsword et al., 2000].

4.4 Evaluation Model

Literature about evaluation models, also called comparison, selection or decision models, is diverse. Jadhav and Sonar [2009] find that there is no common or generic list of important attributes for an evaluation. There is even a lack of agreement on the topics that have to be covered as well as on their very rough division into functional and non-functional types [Chung and do Prado Leite, 2009].

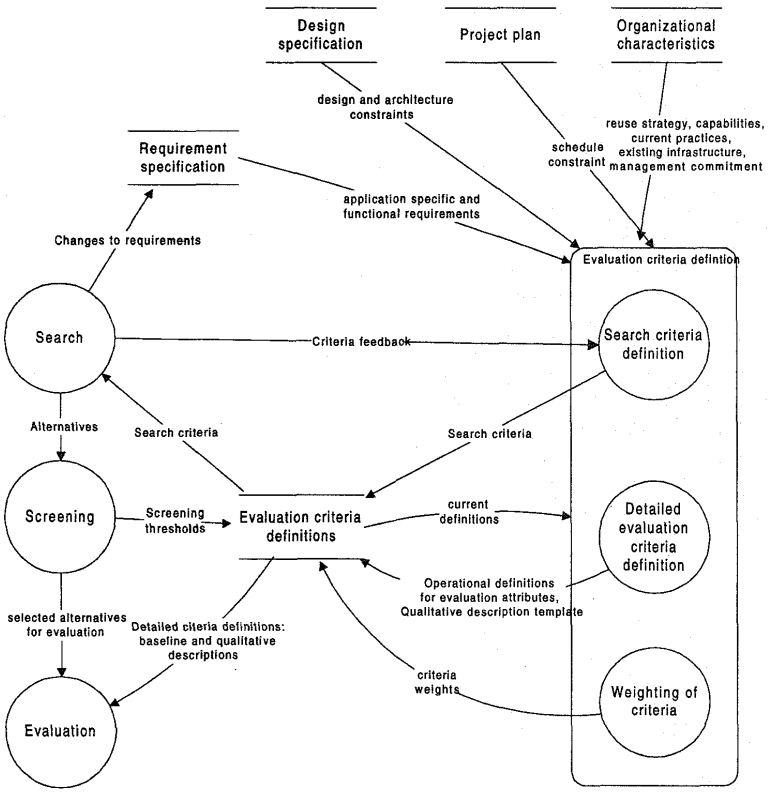


Figure 4.3: The process of defining criteria for an evaluation [Kontio, 1996]

An evaluation model consists of criteria with associated weights and scales. Kontio et al. [1996] state that the most important factor is to consider the reuse situation and to be realistic in setting expectations for criteria. They propose a categorization of categories into four areas: functional requirements, product quality characteristics, strategic concerns, and domain and architecture compatibility. They present an elaborated hierarchical comparison model, but it

can not be used here, because of the differing contexts. In another work Kontio [1996] presents an approach to the derivation of criteria. He proposes a complex process, that results in a hierarchical criteria set (figure 4.3).

Korson and McGregor [1992] on the other hand present very concrete criteria without a hierarchy – they are talking about the evaluation of a framework, although they do not make a difference to libraries. Their model has no context, it is simply a collection of "attributes that an object-oriented library of reusable components should possess" (figure 4.4).

1 Completeness; for those concepts it claims to provide, the library should provide a complete general model.
2 Abstractions; the library should be designed around a few key abstractions.
3 Standard; the design of the library should model standard knowledge in the domain.
4 Inheritance structure; the library should use inheritance to implement the generalisation/specialisation relationship.
5 Purity; the library should be designed as networks of classes without free-standing data or procedural items.
6 Coupling; the library should be designed with a low level of coupling between classes.
7 Exceptions; the library should provide a consistent and easily understood approach to the handling of errors and other exceptions.
8 Partial functions; for every partial function, there should be an 'inspector function' available to check the preconditions of the partial function.
9 Integrity of the abstraction; it should not be possible for a library user to violate the abstraction represented in a library class.
10 Complete interface; classes in the library should conform to a minimal set of standards that are enforced throughout the library.
11 Efficient; the implementation should be the most efficient available within the stated time and space parameters.
12 Consistency; the definitions and naming of items in the library should be consistent.
13 Generics; the library should provide generic classes where possible.
14 Full Implementation; the user should be aware of the degree of completeness of the implementation of the classes in the library.
15 Organisation; documentation should be organised in the manner that reflects the structure of the library.
16 Overview; document should be provided that presents an overview of the library including both content and structure.
17 Orientation; the library should provide documentation for each level of user.
18 Indexes; documentation for each library should have a minimum of three access methods: alphabetic by class name, hierarchical via the inheritance structure and a keyword facility.
19 Formal specification; the documentation should provide formal specifications for each of the components in the library.
20 Accessing tools; libraries should be accompanied by tools that assist the user in locating and viewing classes.
21 Integration tools; libraries should be accompanied by tools that assist the user in adding new classes to the library.
22 Support; commercial libraries should be supported.
23 Upgrades; receiving upgrades for object-oriented libraries will be crucial.

Figure 4.4: Criteria listing of Korson and McGregor [1992]

Colombo and Francalanci [2004] developed a hierarchical model for CRMs that is based on functional and architectural qualities, defined by the ISO 9126 software quality certification guidelines (figure 4.5). They include tables detailing every criterion by explaining the background and providing a scale and hints for the rating process.

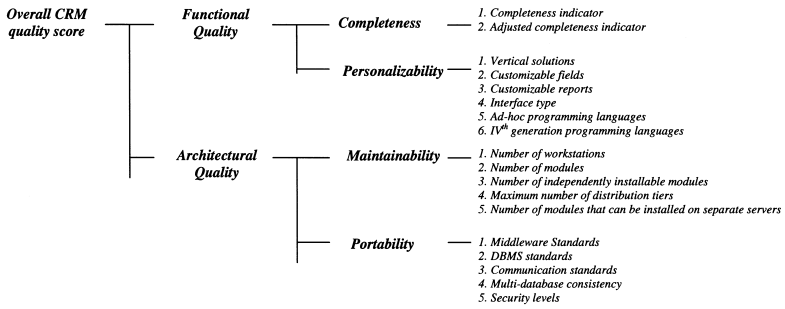


Figure 4.5: Hierarchical decision model of Colombo and Francalanci [2004]

In a thesis that is closer to the context of this work Björemo and Trninic [2010] present a compact evaluation model. Their criteria for comparing server-side web application frameworks are: documentation and learning, convention over configuration, integrated development environment, interna-

tionalization, validation, testing, additional criteria. The model is quite informal as it does not include scores.

Changpil [2012] compares web frameworks, again server-side, in regard to costs. His hierarchy includes: systems (installation, tools), design, learning, and implementation (functional, non-functional) cost, uses a scoring system and weights, but does not explain how scores are determined.

Ignacio Fernández-Villamor et al. [2008] also compare server-side web frameworks. They present clearly described criteria formulated as questions with a detailing paragraph. The model is hierarchical, the criteria are assigned to one of eight categories (see figure 4.6). Weights for the criteria are presented in the results table. Scoring is not described on a per-criteria base but altogether. They shortly state "The value of each parameter is obtained by considering (i) general degree of fulfilment and (ii) degree of integration of the approach (integrated or through third- party plugins)".

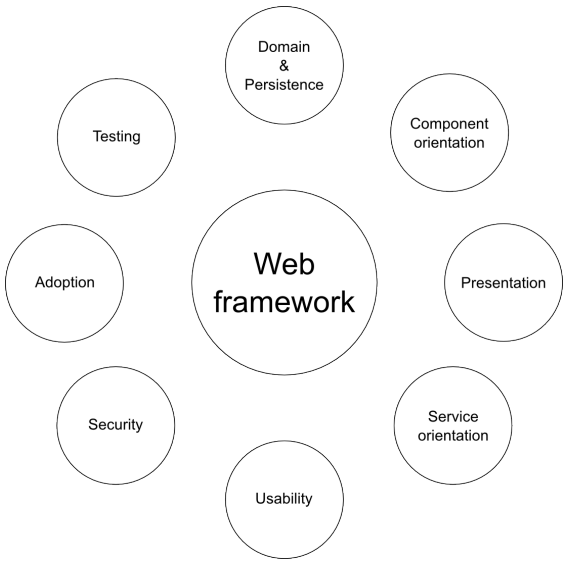


Figure 4.6: Criteria categories by Ignacio Fernández-Villamor et al. [2008]

Laakso and Niemi [2008] compare server-side Java frameworks with Ajax support. They also use WSM[1], but define their criteria via a goal based approach. They develop questions that can be answered by the goals and base their criteria on these questions (figure 4.7). Rating is done by evaluating the documentation available on the project web sites. The scores are taken from 0.5 steps out of the range including -1 and $+1$. Weights assignment is done by "project management".

A different approach is taken by Gizas et al. [2012], who rely on metrics to compare web frameworks, client-side – they do not differ between frame-

[1] weighted scoring method, see 4.5.1

Item	Decision criterion	Weight	Question (Table 3)
I1	Product maturity	0,15	6.3
I2	Activity of the community	0,1	6.1
I3	Backwards compatibility	0,05	4.1
I4	License type	0,1	5.1
I5	Security	0,1	8.1
I6	AJAX support	0,15	2.2, 9.1
I7	Scalability	0,15	7.1
I8	Level of documentation	0,05	2.1
I9	Available debugging tools	0,1	2.3
I10	Open source	0,05	6.2

Figure 4.7: Evaluation model of Laakso and Niemi [2008]

works and libraries and included both. They use various tools to generate numbers about complexity, maintainability, vulnerability, and conformance from the source code.

Saaty [1990] states that "Perhaps the most creative task in making a decision is to choose the factors that are important for that decision.". Not much literature about the evalution of JavaScript frameworks in general and about frameworks specialized for desktop-style development can be found, so there is no comparison model in existence. It has to be constructed. This is an important contribution of this work.

4.5 Evaluation Methods

Evaluation is an important part of the decision process. In a scientific sense an evaluation classifies as a multiple-criteria decision-making (MCDM) problem [Jadhav and Sonar, 2009]. More precisely it is a multiple-*attribute* decision making problem (MADM). This branch of MCDM "refers to making preference decisions (e.g. evaluation, prioritization, selection) over the available alternatives that are characterized by multiple, usually conflicting, attributes." [Yoon and Hwang, 1995]; MADM and MCDM are often used synonymously. MCDM is a well studied topic of operations research [Triantaphyllou, 2000] with well developed methods and tools for problem solution. It is also relevant in Information Retrieval. Its goals are [Jadhav and Sonar, 2009]:

- help decision makers choose the best alternative of those studied
- help sort out alternatives that seem good among the set of alternatives studied
- help rank the alternatives in decreasing order of performance

This implies that there is a distinction between methods that rank all candidates and methods that filter irrelevant candidates. In fact Roy [1996] distinguishes four different "problematics": (1) choosing one alternative, (2) sorting by assignment of alternatives to one category, (3) ranking from best to

worst, (4) description of alternatives in terms of performance on the criteria. Deciding for a software framework can be assigned to category 1 or 2.

Among the methods that have been developed in the field of MCDM according to Jadhav and Sonar [2009] the most popular for the selection of COTS components are:

- Weighted Scoring Method (WSM)
- Analytical Hierarchy Process (AHP)
- Fuzzy Based Approach

They also mention a "Feature Analysis" approach, which is quite similar to the WSM, but they show no application. An additional category of popular methods constitute the:

- Outranking Methods

4.5.1 Weighted Scoring Method

The weights that are assigned to each criterion multiplied with the criterion score and added up. For easier application criteria usually are divided into domains and the weights within each domain are normalized. The weights of the domains themselves can be normalized too, the complete score can then be presented as a percentage of accordance.

Strengths: WSM is easy to conduct, and easy to calculate. As such it is a transparent method.

Weaknesses: The WSM is a compensatory method, which means that weak scores can be evened out by very good scores. This is a characteristic that is undesireable in software selection, since a weakness in an important feature needed for implementation can not be offset by an unrelated feature with a good performance [Ncube and Dean, 2002; Morisio and Tsoukias, 1997]. In some cases this is a characteristic that is useful, since one feature can really compensate another, but WSM does not allow for a distinction of the interdependence of criteria [Kunda, 2003]. The fact that WSM is easy to use causes a tendency to "involve ad hoc procedures with little theoretical foundation" [Kunda, 2003]. Another possible weakness is the fact that the numbers the method produces can be interpreted as true differences instead of a relative ranking [Alves and Finkelstein, 2002] and that it may be difficult to assign proper weights for a large number of criteria [Alves and Finkelstein, 2002].

WSM is also called weighted average sum (WAS) [Jadhav and Sonar, 2009] or weighted sum method (WSM) [Morisio and Tsoukias, 1997].

4.5.2 Analytical Hierarchy Process

The criteria are arranged in a hierarchy, the number of layers is arbitrary. To define the weight of each criterion a pair-wise comparison of the elements within each layer is conducted. Every criterion is compared to all other criteria on its layer in regard to difference in importance and rated from "'equal important" to "definitely much more important" by assigning an odd number from 1 to 9 (equal, moderately more, strongly more, very strongly more, extremely more); even numbers are used as intermediates. Then each pair of alternatives is compared in regard to compliance to a criterion. The resulting matrices are used to compute the weights and their consistency, which is the confidence that they represent the mental model of the decision maker [Saaty, 1990], and the result. AHP is well suited for group evaluation, which means that more than one evaluator is participating.

Strengths: AHP produces a layer between rating and results, which can allow for more neutrality of the evaluation, and uses comparisons instead of absolute rating, which is easier to handle mentally. It counters one weakness of WSM, the difficulty of assigning proper weights, by providing a measurement of consistency and a hierarchy for separation.

Weaknesses: The AHP requires a significant amount of contribution from the evaluator, the number of pair-wise comparisons for one layer with N criteria and M alternatives is:

#criterion:criterion comparisons $= \frac{N*(N-1)}{2}$

#alternative:criterion comparisons $= \frac{M*(M-1)}{2*N}$

The calculation of the end result requires computational assistance, preferably in the form of a specialized software package. Another weakness is the possible reversal of order if another alternative is introduced [Kunda, 2003].

4.5.3 Outranking Method

The most popular outranking methods ELECTRE and PROMETHEE produce either a ranking or a sort order [Mousseau and Slowinski, 1998; Bouyssou, 2009; Morisio and Tsoukias, 1997].

Ranking may lead to an incomplete result because it is based on the idea that "an alternative x [is] at least as good as an alternative y if: (1) a majority of the attributes supports this assertion and (2) the opposition of the other attributes is not too strong" [Bouyssou, 2009]. Therefore alternatives may not be ranked at all (Ros [2011] provides an example).

In an order problem every alternative gets assigned to one category of a pre-defined set of categories.

Strengths: In addition to the definition of criteria weights, outranking methods allow for veto and acceptance thresholds. These prevent the com-

pensatory effect that is present in the WSM.

Weaknesses: As AHP outranking methods depend on matrix calculations and therefore require the help of a software. The main drawback of the outranking methods is the precision of the information that is needed from the decision maker a priori. Mousseau and Slowinski presented a method to overcome this problem but introduced a new: the need for existing examples [Mousseau and Slowinski, 1998].

4.5.4 Fuzzy Based Approaches

Computation intensive methods like AHP, ELECTRE or PROMETHEE are not intended to deal with the "uncertain, imprecise and subjective data" that human decision makers bring into the process [Deng, 1999]. A way of handling this obstacle is to introduce fuzzy numbers instead of crisp numbers [Deng, 1999]. Fuzzy based approaches do not define a new method but instead rely on an already developed method with the additional aspect of uncertainty.

Weaknesses: fuzzy numbers can not be unambiguously mapped to crisp numbers. The two approaches, defuzzification and the usage of fuzzy preference relations [Lee et al., 2004] produce vague results, which in some cases leads to counter-intuitive ranking and additional inconsistensies with different ranking approaches. The computational demand significantly increases [Deng, 1999].

Chapter 5

The Decision Process

This thesis will now execute a decision process as defined in section 4.3. The more specific the context the higher the quality of the result, however, the context as laid out in the chapters *Web Applications* and *Frameworks* (2, 3), although more general than advised, will serve better to build up an evaluation model that can be altered for a more specific context in practical application. The generalization makes it possible to use learnings from specialized practical applications and generate an improved model that can serve again for a base in other cases. The next stages in the decision process, evaluation and the validation of the evaluation result through a prototype, are a first scrutinizing of the model.

The decision process chosen springs from the findings presented in section 4.3. It uses the method of *progressive filtering*, the stages are:

I *Evaluation Model:* Create evaluation model

II *Candidate List:* Accumulate list of possible candidates

III *Requirements:* Specify criteria that need to be fulfilled

IV *Screening:* Filter list of candidates by judging fulfilment of requirements

V *Evaluation:* Evaluate qualifying candidates

VI *Validation:* Validate result with a prototype implementation

For the ratings in *Stage V: Evaluation* the Weighted Scoring Method (see 4.5.1) has been chosen. AHP and Outranking methods may provide more reliable results, but they require advanced software and involve a procedure with significantly increased resource demands. WSM provides the best trade-off between effort and quality of result; since the weaknesses are well known, they can be addressed by the following measures:

compensation: important criteria are listed separately as requirements and candidates that do not fulfill them are filtered out before they enter the evaluation process

tendency to informality: the process is higly formalized and only includes reproducible steps

interpretation of absolute numbers: the candidates are not presented in a ranked order but they are assigned to the pre-defined categories: recommended, unsure, not recommended

improper weights for large number of criteria: the evaluation model is structured in a two-level hierarchy with independent categories, which include a limited number of criteria

The weights will be assigned according to the importance that is expressed by the description in the evaluation model.

Chapter 6

I: Evaluation Model for Desktop-Style JavaScript Web Application Frameworks

The evaluation model is the most important part of the decision process. It is strongly tied to the intended use of the application that is to be developed with the help of the framework.

A central point of this model is to cover all stages of the development process. Beginning with the search for a suitable framework, where required criteria help to filter unsuited ones, continuing with the development and lastly considering deployment, maintenance, and further improvement. Each stage brings several requirements and to be truly usable a framework needs to perform good in all categories. A technical sound framework may be of little use if it is difficult to find documentation during development or even see its suitability upfront.

Another cornerstone is the context. As stated in chapter 4, context is important because it brings forth specific requirements. These can be formed into criteria and then be evaluated. Differences in context explain the huge differences in evaluation models of the visited literature in regard to chosen categories, criteria and weights. The context applicable to this evaluation model has been detailed in chapters 2 and 3.

6.1 Getting Started (*sta*)

Developer in search for a supporting framework should be able to determine as quickly as possible if a software is appropriate for their project. It is assumed that a developer, when in want to learn about the framework, will at first visit the project site. This is the place where the supplier is expected to

present capabilities and application domain, point out limitations, and highlight best-fit scenarios for orientation.

The information should be easy to follow and concrete; vague copy stating something in the direction ‘this framework is capable of everything and a developer’s dream’ does not help to make an informed decision that aids in a successful project execution. The introduction documents should therefore cover purpose or domain of the framework, intentions of usage, and limitations placed on applications [Froehlich et al., 2000].

sta.1 *Does the project site offer an informative introductive overview?*

An introductive overview clarifies if the framework supports the vision and requirements of the development endeavor by covering topics like: application domain, framework architecture, outstanding concepts of the framework, license, required knowledge, comparison to similar or contrasting frameworks, limits of the framework, developer tools, supporters. The introduction is searched for on the home page of the project website as text or as a direct link to a short document.

sta.2 *How straightforward is the way to get started?*

If the introduction has presented the framework as a suitable candidate the next step is to get started. It should somehow be made clear what a newcomer is supposed to do. Solutions to this problem include: download link on the project home page, one package download with installation scripts, explicit getting started section or document, a *hello world* tutorial that serves brief and coarse instructions on how to work with the framework. At this early stage of getting to know the newly software strong guidance and precise instructions are rated most positively.

sta.3 *What is the quality of the getting started tutorial or guide?*

The expectation is to have a coherent set of instructions and explanations in either one document or a consecutive series of documents. These instructions and explanations are the initial guide for: creating an application skeleton, understanding the file and directory structure the framework requires, basic handling of development tools, main concepts of the framework architecture (e.g. view/layout management, data store), class structure, deploying, structure of the documentation.

Every framework is created for a specific path and way of development, its *conventions*. If respected by the developer the framework works well. If not solutions become difficult and the implementation starts to behave in unexpected ways. The intention of the beginner’s guide should be to clarify the most basic of these conventions and show where to find description of the others if needed.

6.2 Documentation (*doc*)

Refers to the material available to all developers – fresh, intermediate and expert – helping them get informed about the purpose, the ideas and intentions upon which the framework is built, its inner workings, functionality, usage, and configuration. The material is to be provided by the supplier in any form: web pages, PDFs, books, collections of code, application examples, videos, etc. Contribution from outside can not be directly rated in a reproducible form, its influence is included in section 6.3.

Frameworks are meant to reduce development time and effort but using a framework is not effective if the learning cost is too high. A framework is defined as a high-level design including the description of its composition, which makes documentation an integral part of it [Johnson and Foote, 1988; Fayad et al., 1999] and as Fayad et al. [2000] puts it "the single most distinguishing characteristic of a high quality software product.". Janisch [2008, ch 5.4] emphasizes that a framework without documentation will not be used.

There is a big difference in the needs of documentation for libraries and complex frameworks as they are the subject of this work. Frameworks have many dependencies between classes and often the usage of one class requires a specific configuration of another class. These cases can only be communciated in a structured documentation, where they are findable. A library may be sufficiently documented with a simple API list, since components can usually be used independent of one another, this is not the case for frameworks [Fayad et al., 2000].

Failure to document edge cases or intended usage and not providing standard implementations for such circumstances leads to confusion of developers, which often leads to a high number of questions in forums [Hou et al., 2005]. This type of documentation is usually delivered in the form of guides. They combine theory – in the form of explanation of main concepts and ideas of the framework – and practice – in the form of example source code and references to more in-depth information or hints to the framework source code. They can be presented as either separate documents or as chapters of a manual. A framework needs:

- **guides** for main concepts (e.g. data storage, communication with server), complex UI widgets and testing. A framework can not be understood by just looking at its class structure and provided methods, it needs introduction to the *reasons* for designing the interface as it is, the purpose of the framework and how to use it, and the developer tools [Fayad et al., 2000; Atkinson, 1997; Hou et al., 2005; Froehlich et al., 2000].

- a set of **example applications** showcasing the usage of certain features and concepts of the framework [Fayad et al., 1999, ch21 Documenting

Frameworks] [Fayad et al., 2000]. Example applications are an important factor particularly at the beginning of the learning process, an empirical study by Shull et al. [2000] suggests that they are even more effective than structured documentation.

- an **API reference** explaining classes and methods [Fayad et al., 2000]: advices to go and look up the source code, which are given quite often in open source projects, should be the last resort as it is not what developers want [Kirk, 2005]. Furthermore it is important to explain the class hierarchy and the relations between classes that are not bound together by this hierarchy – e.g. a container class and its iterator [Korson and McGregor, 1992].

- **release notes** for new versions informing developers about deprecations and interface changes: quite often documentation or examples are not available for most recent versions, therefore the ones for olders have to be used and this requires information about the changes that happened between the documented version and the one in use.

doc.1 *Coverage of guide topics?*

The most important topics include: architecture, view system, complex widgets, model, data store, communication with server, testing, build tools.

doc.2 *Quality of guides?*

The quality of a guide is measured by: appropriateness of length[1], navigation and structure, quality of wording, provision of example source code, references to other parts of the documentation (API reference, other guides).

doc.3 *Quality of the API reference and its viewer?*

The API reference is a basic form of documentation. Some frameworks use it to communicate concepts too, but its main purpose is to present classes and their members and what both of them are intended to do. Naming conventions go a long way here by providing information through similarity; still, at least a short explaining sentence should go along every entry.

An important aspect of the API reference is its viewer. A well-arranged tree view in the sidebar, incremental search, and a content view enriched with options to hide unneeded class members is the standard.

doc.4 *Range of examples.*

[1] some concepts do not require lenghty explanations, some do

Source code examples or templates that serve as a starting point for the developement with specific application components and show their usage in connection with other parts of the framework.

doc.5 *What is the quality of documentation for new versions?*

Does the project site or the repository provide release notes or a changelog for new versions? Is there a migration guide for incorporating major changes from a new framework version into a custom application?

doc.6 *Overall structure.*

Good documentation is structured into logical sections (API, guides, examples) and has a dedicated overview page.

doc.7 *How easy is it to find out the timeliness of the documents?*

Unknowingly following guides that are outdated because they have been written for an older version and the API changed in between is among the most frustrating experiences a developer encounters. Documents should state their target version, the date they have been created and last updated, and be marked deprecated if that is the case.

6.3 Community (*com*)

To be of relevance for development a framework must be actively supported and developed [Korson and McGregor, 1992], support can come from a commercial entity or a community. It must also have arrived at a certain stage of maturity, resulting in a stable API. An important indicator for the maturity of a framework is its version history, which is usually available in open-source projects Kizzort [2002], for commercial projects only the project web site or a third-party information can be consulted. A young project is very likely to contain bugs and unstable parts in the interface, so a longer version history indicates a certain degree of maturity [Fayad et al., 1999; Laakso and Niemi, 2008].

Developers contributing to the project in various degrees of intensity form at the same time part of the framework user base as well as an equivalent to the supplier in regard to production of documentation, to maintenance, and to support. Therefore a large and active community is crucial not only to the success of the framework but also to easing the difficulty in learning and using it. For commercial frameworks a large community signifies good adoption rates and provides incentives to invest in the progress of the software.

Forums or mailing lists play a central role in the education of newbies and intermediates [Hou et al., 2005] and in the communication from framework developers to framework users. It is save to assume that a high number of active users correlates with a high number of: asked questions, answered

questions, blog posts with how-tos or best-practices, bug reports, people able to fix bugs and contribute patches. Only a large audience makes it lucrative to write a book, provide courses and professional support [Ignacio Fernández-Villamor et al., 2008; Laakso and Niemi, 2008]. Although this is such an important part its measurement is difficult. Therefore the number of results of searches of popular and academic sources will be taken as a *relative* popularity indicator.

Adoption is also reflected in applications that have been built with the help of the framework and the supplier is keen to show as a positive example. Links to real life applications, that opposed to demos and examples serve a practical purpose and have actual users, are a good sign.

Another important thing related to the community is the communication that is directed from the framework developers to it. Two standard instruments for this cause are a blog and a *Twitter* account.

com.1 *In active development?*

The framework must be actively maintained and developed. Signs of activity are releases, developer blog posts, or status updates, commits to the repository do not suffice.

com.2 *Mature?*

How long ago has the framework been officially published? Are there indications for a stabilization of the API?

com.3 *Availability of books?*

Books are an advanced form of documentation. They have a longer life-span than a website and their production is a demanding and serious endeavour. Amazon[2] is the most prominent international book store and delivers great search results thanks to its full-text search for the majority of technology books, it has therefore been selected as the source for this criterion. Dedicated books and short references in otherwise differently concerned books have to be distinguished and appropriately rated.

com.4 *Is it covered in journals or the press?*

Academic journals are not a good source for information about JavaScript frameworks, coverage is very scarce. So it seems more appropriate to measure popularity and importance to the software developer space by rating the coverage in pertinent publications of a more popular but still professional nature. As the major German publishing house for professional IT journals *Heise* (see table 7.2) has been selected. Heise allows for a quality classification by differentiating between news items, a reference in an article, or a dedicated article.

[2] amazon.com

com.5 *How popular is it on the Internet?*

A very vague criterion but may still be representative when set in relation to all others. It is represented by the number of hits for a search with the Google search engine, *Hacker News*, `quora.com`, and `stackoverflow.com`. The latter two are popular question-and-answer sites, they can be accessed for free; *Stack Overflow* is targeted at developers, *Quora* does not seem to have a specific target group. Hacker News is described in table 7.2.

Due to the lack of objective criteria thresholds, scores have to be seen in relation to all candidates[3].

com.6 *Forum/mailinglist activity.*

How active is the project forum, Google group or mailinglist? More activity means that probably more questions that might come to a new developers mind have already been asked and answered or may have even been solved by fixing the bug or adding a piece of information to the documentation.

com.7 *Activity on Twitter.*

How many *tweets* have been published in a given time frame?

com.8 *Activity on Blog.*

How many blog posts have been published in a given time frame?

com.9 *Links to real life applications on the project site.*

Does the web site link to applications that have been developed by third-parties, are actively used, and accessible, so that they may serve to demonstrate the framework capabilities?

6.4 Features (*fea*)

This category collects criteria of different types of importance and impact. The first two are critically tied to the context described in chapters 3 and 2: architecture is a central aspect of a framework, its importance can not be underestimated. The MVC architecture [Burbeck, 1992] is a commonly used architecture for native applications with a graphical user interface it is reasonable to demand the use of the same architecture in the current context; abstraction from the DOM is a special demand of the browser environment. The complicated nature of the DOM is the reason frameworks and libraries have become essential for web applications development, this criterion goes a step further. DOM abstraction serves to: (i) avoid a mental context switch

[3] grading keys are provided in a table in section 10

between view and controller or model coding (ii) encapsulate code that is directly related to presentation, i.e. will be translated to DOM elements, within the class, a core principle of object-oriented programming, and (iii) keep code that depends on the browser render engine apart from code that depends on the JavaScript virtual machine.

The next two criteria are not crucial but still very important. They refer to the model layer of the MVC architecture, with two concepts called *data store* and *bindings*. A data store is an entity that manages the provision of data on the client-side. Its loads data from the server when needed, caches it on the client-side, knows of relations between models, is able to generate queries for the server to efficiently fetch the data currently needed[4], upon a client-side query from e.g. a view, it can decide if it can return the cached data or has to send a query to the server, of maybe both. To work effectively stores require bindings. Bindings – another important criterion – are a means to mututally connect a property of two objects, e.g. an array of model objects[5] and the content property of a list view; removing an object on either end, evokes the same removal on the other, i.e. if the user removes an object from the list view it is removed from the model - and in succession from the server - if on the other hand a new object is added at the server and a query adds it to the array in the model, the object is also added to the list. In combination stores and bindings make it possible to e.g. bind a table view to an empty model array and populate this array dynamically when data from the server arrives; via the binding the table view gets notified about new data and automatically displays it.

The rest of the criteria in this section refers to properties that aid in development, some significantly, but represent functionality that can be gained manually with reasonable effort.

fea.1 *Has a Model-View-Controller architecture?*

Offers explanations about its implementation of the MVC pattern? Has classes for all modules of the MVC pattern?

fea.2 *Layout manager and HTML/CSS/DOM abstraction.*

Has a means to completely manage the layout of the application with JavaScript code or an XML/JSON description file. It offers an abstraction from the manual coding of HTML and CSS files and generates the necessary div-tags to position and resize views. It is aware of the current size of the viewport and positions the views accordingly. A very basic setup would be two panes e.g. the left for a tree view, the right for a content view. The layout manager makes it possible to grab the

[4] e.g. Person and Order are two model classes, a Person has many Orders. A store would be able to query for a specific Person with nested Orders instead of sending one query for the Person and a second one for the corresponding Orders

[5] this could be the property of a store

dividing line and resize the two views as one is accustomed to from a desktop application. A layout manager offers much more flexibility than a template engine.

fea.3 *Data store.*

Does the framework provide a data store?

fea.4 *Bindings.*

Does the framework provide a mechanism that allows the developer to connect views and model objects, so that views update their representation automatically if the underlying object's properties change.

fea.5 *Drag & drop.*

Does the framework provide support for drag & drop operations accross widgets and for custom widgets?

fea.6 *Internationalization/Localization (i18n/l10n).*

Does the framework have a structure and solution to solve the problem of translating an application to different languages.

fea.7 *Theming.*

How many themes are delivered with the framework? Is it possible to change the graphical appearance with the alteration of a central file and the exchange of graphical files?

fea.8 *User input validation.*

Does the framework allow the validation of user input in forms or before it is saved to the model?

fea.9 *Context menu.*

Is it possible to exchange the standard browser context menu (right click with mouse) with a custom menu, showing items dependent on the current mouse position within the window.

fea.10 *Tool tips.*

Do action UI elements, e.g. buttons, have an entry for a short help text that is displayed when the mouser hovers over the element for a certain time.

fea.11 *Keyboard shortcuts.*

Can key combinations, like pressing Ctrl together with a letter, be used to trigger actions?

fea.12 *Offline mode.*

Does the framework support a mode where the application can run from the browser cache, with local browser storage and without internet connection?

fea.13 *Server push.*

Does the framework support a mechanismn that allows the server to push new data into the client-side application during run-time. An example would be the update of live results of a sports game?

fea.14 *Browser history management.*

Can the URL be updated to reflect the state of the application, so that it can be bookmarked (also called: browser history manipulation or deep linking).

6.5 User Interface (*uif*)

User interface refers to the parts of the program that the users see and directly interact with. A framework has to support the developer in producing a consistent look & feel, adhering to standards of the web, providing a good usability for the user, composing an easy adaptable layout, and translating the interface to different languages. Users would not feel a complicated class structure or an insufficient documentation, but they instantly experience the clumsiness of quickly self-implemented views.

Text fields, radio buttons, tree views, tables, these are examples of common interface elements that users have an expectation of how they can interact with. The developer must be able to rely on the framework for providing the correct behavior. This is very specific to the desktop-style context, since web applications that are spread over several pages have more moderate demands and can use standard HTML elements. User interface guidelines or references to guidelines of other platforms or frameworks are a good supplement Janisch [2008].

This category tests for the degree of support by checking the availabilty of common widget types.

uif.1 *Repository of UI elements (widget library)?*

Which of the following widget types are present: label, input field, button, progress bar, activity indicator, radio button, checkbox, combo box (drop down menu), number spinner (field restricted to numbers with up/down buttons), slider, menu, tabs, modal view, alert, split view, tree view, grid/collection view, charts, calendar, date picker, map view, rich text editor?

6.6 Development Setting (*dev*)

Although a big part of the usability of a framework for a developer is related to the quality of the documentation, there is a significant amount of other points that influences the effectiveness and efficiency of using a framework. These include: development support via tools, independency of systems, and license issues.

In addition to the standard arguments for the introduction of automated testing into the development process the special circumstances of the JavaScript run-time environment make testing and catching errors before execution even more important because: (i) the whole client-side JS application may stop or not even begin execution upon an error without any indication of where the error happened, (ii) client to server logging of errors and exceptions is difficult and unreliable.

Deploying a JavaScript application can be as simple as copying all implementation files to a web server. This, however, is inefficient and unfavorable for desktop-like applications as it further increases the already high initial loading time unnessecarily: the higher number of source files forces accessing browsers to spawn additional requests and may lead them to running into their upper request limit; comments and white space increase the size of the data that needs to be transmitted. Reduction is the process that deletes unused code parts from the source files and merges several files into one; minification or obfuscation strips comments and whitespace from the source files and shortens names of variables or functions for additional data amount savings. These are standard procedures that a deployment tool should be able to handle. Other points are additional configuration options and the possibility to build and deploy the reduced and minified application with a single command.

Boundaries of a financial budget are a common problem in software development, but especially in the academic context and for small business it is important to have frameworks available that permit experimentation with commercially exploitable software without the necessity to pay fees upfront. Even more welcome is the possibility to benefit from a development for free and still be able to run a business.

dev.1 *Code generation tools?*

> Is a script or a binary application available that takes an application name and sets up the required directory structure with appropriate file and class names? Is a generator provided that takes a name and a type of class (e.g. model, controller) and creates the standard implementation files and additional (e.g. helper, testing) files?

dev.1 *Built-in testing?*

> Is there a standard convention and a tool that allows for unit and other types of testing? Is there a documentation?

dev.1 *Deployment support?*

Support for reduction, minification, copying of build files to a server?

dev.1 *Client-server communication protocols?*

What data formats are supported for the communication with the server-side? JSON and XML are the most important standards.

dev.1 *Independence of a development platform.*

Development is done on desktop operating systems. The most popular are: OS X, Linux, and Windows.

dev.1 *Independence of a server platform.*

Is the deployed product tied to a specific server platform or can it be adapted to communicate with any standard (e.g. REST) compliant service?

dev.1 *What license models are provided?*

Is the deployed application usable in open-source applications? How much does it cost to use the framework in a commercial application?

Chapter 7

Stage II: Candidate List

Names and web addresses of frameworks were accumulated by conducting a market research on the Internet. Using only the Internet as a place to search for the emergence of candidates should not limit the findability of software that is inherently bound to it.

Over the course of approximately one year – between June 2011 and June 2012 – the sources listed in table 7.2 were followed. Items related to web frameworks were investigated and classified. The term *web framework* is more often found describing a server-side framework and in recent years server-side frameworks get improved with an often called "Ajax component". Frameworks enhanced in such a way enable developers to write enriched applications that are less dependent on a full page reload and are able to load data dynamically. These frameworks are easy to distinguish from frameworks for desktop-like web applications since they are mostly written in a different language, functionality accessible via JavaScript is an addition. They have not been added to the list of relevant frameworks or otherwise recorded.

Other types of frameworks where applications are written in another language and then compiled to JavaScript code – the *Google Web Toolkit* is a well-known example – are not of interest to this thesis either.

A special type of JavaScript framework that has gained popularity in recent years are *micro frameworks*. The popular site `micro.js` currently lists around 250 of them. They describe micro frameworks as "pocketknives" that do "one thing and one thing only". Hence they are highly specialized, small (below 5kB), and not suitable for desktop-like web applications.

Finally 41 candidates were added to the framework list. Their names, the URL of their project site, and the version number at the date of examination are listed in table 7.1.

Name	URL	Version	Date
ActiveJS	activejs.or	n/a	19/06/2012
Agility.js	agilityjs.com	0.1.2	19/06/2012
AmplifyJS	amplify.js	1.1.0	26/06/2012
AngularJS	angularjs.org	1.0.0	19/06/2012
AppJS	appjs.org	0.0.11	19/06/2012
Backbone.js	backbonejs.org	0.9.2	19/06/2012
batman.js	batmanjs.org	0.9.0	19/06/2012
Bindows	bindows.net	4.1.1	19/06/2012
Cappuccino	cappuccino.org	0.9.5	19/06/2012
Choco	github.com/ahe/choc	n/a	19/06/2012
CorMVC	bennadel.com/projects/cormvc-jquery-framework.htm	n/a	19/06/2012
DHTMLX	dhtmlx.com	3.0	26/06/2012
Dojo Toolkit	dojotoolkit.org	1.7	19/06/2012
Eyeballs	github.com/paulca/eyeballs.js	0.5.17	19/06/2012
Ember.js	emberjs.com	0.9.8.1	19/06/2012
Ext JS	sencha.com/products/extj	4.1.0	25/06/2012
Glow	bbc.co.uk/glow	1.7.7	19/06/2012
Google Closure Library	developers.google.com/closure	r1376	22/06/2012
JavaScriptMVC	javascriptmvc.com	3.2.2	22/06/2012
jQuery	jquery.com	1.7.2	22/06/2012
Knockback.js	kmalakoff.github.com/knockback	0.15.3	22/06/2012
Knockout.js	knockoutjs.com	2.1.0	22/06/2012
Luna (Asana)	asana.com/luna	n/a	19/06/2012
MooTools	mootools.net	1.4.5	22/06/2012
MochiKit	mochikit.com	1.4	22/06/2012
Mojito (Yahoo!)	developer.yahoo.com/cocktails/mojito	0.3.27	22/06/2012
Prototype	prototypejs.org	1.7	22/06/2012
qooxdoo	qooxdoo.org	2.0	25/06/2012
Rialto	rialto.improve-technologies.com/	1.1.5	25/06/2012
Rico	openrico.org	2.1	25/06/2012
Sammy.js	sammyjs.org	0.7.1	25/06/2012
script.aculo.us	script.aculo.us	1.9.0	25/06/2012
SmartClient	smartclient.com	8.2	25/06/2012
Spine.js	spinejs.com	1.0.8	25/06/2012
SproutCore	sproutcore.com	1.8.2	25/06/2012
Spry	labs.adobe.com/technologies/spry	1.6.1	25/06/2012
UIZE	uize.com	n/a	25/06/2012
underscore.js	underscorejs.org	1.3.3	25/06/2012
Wakanda	wakanda.org	1.0	25/06/2012
YUI!	yuilibrary.com	3.5.1	25/06/2012
zepto.js	zeptojs.com	1.0rc1	25/06/2012

Table 7.1: The complete list of frameworks, accumulated via market research.

Name	Comment	Web Address
heise online News	Newsticker of the publishing house of Germany‘s most important magazines related to professional IT and computer technology.	heise.de/newsticker
Ars Technica	US American online magazine for science and computer technology	arstechnica.com
Stack Overflow	Community question and answer site. Popular among professional and amateur software developers of all kinds	stackoverflow.com
Hacker News	Community news site of silicon-valley based startup funding company for web technology, startup, funding and management advice	news.ycombinator.com
Wikipedia	de:Freies_Webframework, en:List_of_JavaScript_libraries, en:List_of_widget_toolkits, en:List_of_Ajax_frameworks	wikipedia.org
Google	Searches releated to JavaScript, frameworks and web application	google.com
Amazon	Online book store	amazon.com
Other	Other sites with listings and occassional news on web frameworks	bestwebframeworks.com ajaxian.com todomvc.com

Table 7.2: Sources that were used to compile the framework list

Chapter 8

Stage III: Requirements

To filter out clearly unsuitable candidates very early in the process some criteria from the evaluation model will be selected and defined as *required*. Every candidate in the list will be checked against these *requirements* and needs to fulfill them to remain for the next stage of the process.

8.1 Derivation of Requirements

The criteria for this step of the process were selected from the evaluation model (chapter 6) with the aim to achieve a sub model that is effective at sorting out unsuitable candidates while at the same time being easy to rate.

The set of criteria defined as requirements has to fulfill the following objectives:

1. rating must be possible in short time
2. performance judgement must be binary for easy rating
3. the number of candidates must be significantly reduced

Requirements have either been derived from the importance indicated in the detailed description of the evaluation model, or they have been chosen with regard to the context of single-page web applications. Iteratively the list was expanded until the desired reduction was accomplished.

8.2 Requirements Listing

from Evaluation Model

The following criteria have been extracted from the evaluation model descriptions:

Basic Documentation: *the candidate's project site must point to an API reference, at least one tutorial, an example application, and a first steps guide with instructions for the installation.*

The documentation section of the evaluation model points out clearly that documentation is an indispensable part of a framework.

Activity: *the candidate must be actively maintained and developed. This needs signs of activity (releases, developer blog posts, status updates) within the last year, commits to the repository do not suffice.*

Bug fixes and feature additions are a necessity for such a complex software as frameworks are.

Maturity: *the first public version of the candidate must have been released more than a year ago (i.e. before July 2011).*

An immature framework has an unstable API and is more likely to contain severe bugs.

Framework: *the candidate must not be identifiable as not being a framework according to the definitions of chapter 3.*

The development of single-page web applications requires the support of a framework, a library is unsuitable.

Widget Library: *the candidate must provide a library of composable views with complex widgets like a tree view, table view, grid view or similar.*

To ensure consistent, quick, and error reduced user interface development.

from Context

The following requirements have been chosen from the evaluation model with regard to the context provided in chapters 2 and 3:

Layout Manager/DOM abstraction: *the candidate must provide a layout manager. It must be possible to build a UI with JavaScript or XML/JSON without coding it in an HTML file directly using HTML entities like* `div`.

Direct manipulation of the DOM is complex and performance intensive. Single-page web applications frameworks offer specialized modules to handle these tasks.

Model-View-Controller: *the candidate must offer a complete stack of classes for all levels of the MVC.*

This is the standard pattern for GUI desktop applications and desktop-like web applications.

Independent of a development platform: *development must be possible on all major desktop platforms (OS X, Linux, Windows).*

It is appropriate to be able to develop platform independent applications on a platform of choice.

Server Agnostic: *the candidate must be a client-side framework and must not rely on a specific server back-end.*

Web applications should operate independently of the server-side platform.

Chapter 9

Stage IV: Screening

The candidates accumulated in chapter 7 were checked against the requirements listed in chapter 8. The screening process is an important stage to efficiently identify candidates that are unsuitable for the development of single-page web applications and should therefore be excluded from the subsequential stages of the process.

9.1 Screening Execution

The number of accumulated candidates (41) is high when taken into consideration that the screening process is carried out by a single person. However, the requirements are specifically selected to be judged easily and quickly.

The analysis is based on information provided by the candidates' project websites and – if available – the source code repository. This may mean that a feature that is technically available in a candidate has been classified as absent, however this is in line with the explictly declared need for documentation: a feature that is hard to find lacks proper documentation.

The abbreviated results of the examination are listed in table 9.1, eight candidates were accepted for the evaluation. The complete table with more detailed information about each candidate can be found in appendix A.

9.2 Screening Result

The subsequent results summary present only the main reason a candidate was rejected from further evaluation. It does not mean that all other requirements have been fulfilled.

Activity

The following candidates have been classified as no longer actively maintained and developed because of missing indication of the contrary on their project website or repository:

- ActiveJS
- Choco
- CorMVC
- Eyeballs
- Glow
- MochiKit
- Rico
- Spry

Library

The following candidates have been classified as a library, because of their explicit declaration or because of the description of their purposes and capabilities:

- AmplifyJS
- AngularJS
- jQuery
- Prototype
- script.aculo.us
- underscore.js
- zepto.js

Layout Manager/DOM Abstraction

The following candidates failed to present a method that relieves the developer from managing the DOM and lack a class or module that allows the management of the visual interface:

- Agility.js
- Backbone.js
- batman.js
- Ember.js
- Google Closure Library
- JavaScriptMVC
- Knockback.js
- Knockout.js
- MooTools
- Rialto
- Sammy.js
- Spine.js
- UIZE
- Wakanda
- YUI!

Maturity

The following candidates have seen their first commit within the last year and have therefore been classfied not mature:

- AppJS
- Mojito (Yahoo!)

Other

One candidate was rejected for a reason not listed among the requirements: Luna is not a public framework. Its description indicates that *Asana*, the company behind it, uses it internally for development, but public access is not granted.

Name	Accepted	Reason
ActiveJS	No	no longer maintained
Agility.js	No	layout manager/DOM abstr., widget library
AmplifyJS	No	library
AngularJS	No	library
AppJS	No	immature
Backbone.js	No	layout manager/DOM abstr., widget library
batman.js	No	layout manager/DOM abstr., widget library
Bindows	Yes	
Cappuccino	Yes	
Choco	No	no longer maintained
CorMVC	No	no longer maintained
DHTMLX	Yes	
Dojo Toolkit	Yes	
Eyeballs	No	no longer maintained
Ember.js	No	layout manager/DOM abstr., widget library
Ext JS	Yes	
Glow	No	no longer maintained
Google Closure Library	No	layout manager/DOM abstr.
JavaScriptMVC	No	layout manager/DOM abstr., widget library
jQuery	No	library
Knockback.js	No	layout manager/DOM abstr., widget library
Knockout.js	No	layout manager/DOM abstr., widget library
Luna (Asana)	No	not public
MooTools	No	layout manager/DOM abstr., widget library
MochiKit	No	library, no longer maintained
Mojito (Yahoo!)	No	immature
Prototype	No	library
qooxdoo	Yes	
Rialto	No	layout manager/DOM abstr., MVC
Rico	No	no longer maintained
Sammy.js	No	layout manager/DOM abstr., MVC
script.aculo.us	No	library
SmartClient	Yes	
Spine.js	No	layout manager/DOM abstr., widget library
SproutCore	Yes	
Spry	No	no longer maintained
UIZE	No	layout manager/DOM abstr.
underscore.js	No	library
Wakanda	No	layout manager/DOM abstr., MVC, OS independency
YUI!	No	layout manager/DOM abstr.
zepto.js	No	library

Table 9.1: The complete list of frameworks with the result of the screening and reasons for their exclusion from further scrutinizing

Accepted

The following candidates have been accepted, since they fulfilled all requirements:

- Bindows
- Cappuccino
- DHTMLX
- Dojo Toolkit
- Ext JS
- qooxdoo
- SmartClient
- SproutCore

9.3 Remarks

A smooth and efficient execution of the screening was inhibited by a high variation of quality in project self-descriptions. Few projects present explicit and concise background information. Architecture, application domain, or UI capabilities often are not mentioned at all or packed in ambiguous language. Generally it seems that it is unknown to most developers that libraries and frameworks are two disctinct concepts.

Another attribute that has unexpectedly proved to be difficult to ascertain is the activity state of a project. Frequently the project website is filled with misleading information about a next version, a road map, or announcements of an imminent release, but results are unseen. A planned retirement has only happened in one of the eight cases: the maintainers of *MochiKit* explained that the project is “feature complete” and no longer in active development. Some projects that are marked *no longer maintained* have seen commits to the repository so it seems that a core group of developers is still present, but maybe resources are too scarce to sustain proper support and communication. Deducting from these circumstance it is advisable to look beyond the official statements to assess the possible future of a project.

Chapter 10

Stage V: Evaluation

The screening of stage IV showed that eight candidates qualify for the evaluation (see table 9.1). What follows is an introduction of every candidate, the evaluation results presented by category, and a final discussion.

10.1 Candidates

10.1.1 Bindows

The Bindows framework is a product of the company *MB Technologies* based in Georgia, USA, with a development center in Sweden. The company provides software solutions for governments, military and large corporations. MB Technologies calls Bindows the "leading object-oriented platform for developing AJAX enterprise applications" and claims it is used by a majority of Fortune 500 companies. The project site stresses that it can replicate a "Windows Look-and-Feel"[1].

Version 1.0 was released on Feb 24, 2004, the current major version is 4 and was released on Feb 13, 2009. The evaluated version is 4.1.2 with an unknown release date, 4.1 was released on Mar 31, 2010, thus is more than two years old.

Topic	URL
Company	`mb.bindows.net`
Initial Release	`mb.bindows.net\news\bindows\10\release.html`
Version Information	`bindows.net\bindows\changelog.txt`

Table 10.1: Sources for Bindows (last accessed: Aug 29, 2012)

[1] jugding from the provided screenshots "Windows" refers to Microsoft Windows XP and Vista

10.1.2 Cappuccino

Cappuccino is an open-source community driven framework that is based on its own language: Objective-J. Objective-J is a strict superset of JavaScript, modeled after Objective-C. It uses a very similar syntax and the same message passing model, which itself is heavily inspired by Smalltalk. Transformation to JavaScript happens at run-time transparantly for developer and user. The Cappuccino and Objective-J APIs are in large parts a copy of Apple's Cocoa framework and its open-source sibling GNUStep[2].

The projects have been initiated by three graduates from the University of Southern California: Ross Boucher, Tom Robinson, and Francisco Tolmasky. They were open-sourced on September 4, 2008 with version number 0.5. To showcase its capabilities *280 North*, the company founded for the development of Cappuccino, released *280 Slides* a presentation application similar to Microsoft's PowerPoint or Apple's Keynote, which was critically acclaimed as being an impressive example of a browser application. After 280 North was aquired by *Motorola* in Aug 2012 Cappuccino and Objective-J development was continued by the community. 280 Slides was shut down in the end of 2011. The current and evaluated version of Cappuccino is 0.9.5 released on Nov 16, 2011; version 0.9 was released on Feb 22, 2011.

Topic	URL
Company	`mb.bindows.net`
History	`arstechnica.com/apple/2008/06/cocoa-on-the-web280-north-objective-j-and-cappuccino`
Initial Release	`cappuccino.org/discuss/2008/09/04/announcing-cappuccino`
Current Status	`groups.google.com/d/topic/objectivej/HLc6mcZbGqQ/discussion`

Table 10.2: Sources for Cappuccino (last accessed: Aug 29, 2012)

10.1.3 DHTMLX Suite

DHTMLX is a commercial product of DHTMLX, a company based in Saint Petersburg, Russia. It has a strong emphasis on widgets while still providing an MVC architecture. According to the DHTMLX website 30% of the Fortune 500 companies are among their customers. All DHTMLX components are available separately, the product comparable to other framework is the DHTMLX Suite.

DHTMLX claims to have been building the library since 2003, the earliest information about a release is a blog post from Jul 20, 2005 describing the

[2] gnustep.org (last accessed: Aug 29, 2012)

release of the component *dhtmlxTree*. DHTMLX then went on to add more widgets and began releasing it as a suite, presumably in 2007. The current release is 3.5, but since it was released after the cutoff date the release evaluated here is 3.0 from Oct 31, 2011.

Topic	URL
History	`dhtmlx.com/docs/services.shtml`
Early Release	`dhtmlx.com/blog/?m=200507`

Table 10.3: Sources for DTHMLX (last accessed: Aug 29, 2012)

10.1.4 Dojo Toolkit

The Dojo Toolkit is a highly modular project backed by the Dojo Foundation, a non-profit organization based in California, USA. The Dojo Foundation was formed by Alex Russell, Dylan Schiemann, and David Schontzler while working at *Informatica* in 2004 to support the Dojo Toolkit but today is host to several open-source projects related to web technologies. It is sponsored by companies and private persons. *Dijit* is the widget library of the Dojo Toolkit and presented as the main factor for application development.

Version 0.1.0 was released on Aug 30, 2005. The current version is 1.8, which was released on Aug 15, 2012, which is after the cutoff date. The release evaluated here is 1.7.2 from Feb 16, 2012.

Topic	URL
Company	`dojofoundation.org`
History	`dojotoolkit.org/reference-guide/1.8/quickstart/introduction/history.html`

Table 10.4: Sources for Dojo Toolkit (last accessed: Aug 29, 2012)

10.1.5 Ext JS

Ext JS started life as a full open-source project and is now the product of *Sencha*, a company based in California, USA. It was created in 2006 by Jack Slocum as an extension to YUI[3]. Slocum's intention was to extend YUI with widgets and enhanced animation handling. It quickly became popular and was renamed to Ext JS, with the release of version 1.1 it had no longer any dependency on an external library. To be able to generate monetary support for a business the license model was changed to a dual licensing system with a GPL license and a commercial license. In June 2010 it was announced that

[3] Yahoo User Interface library, yuilibrary.com (last accessed: Aug 29, 2012)

Ext JS and the two other JavaScript libraries jQtouch and Raphaël were joined in one company named Sencha. Currently Slocum is no longer affiliated with Ext JS or Sencha. Sencha claims that 50% of the Fortune 100 companies "rely on Sencha technologies". Ext JS started to support an MVC architecture with version 4.0.

The current major release of Ext JS is 4, it was released on Apr 26, 2011. The evaluated version is 4.1.1, 4.1 was released on Apr 20, 2012.

Topic	URL
History	`web.archive.org/web/20061018033829/http://www.jackslocum.com/yui/index.php`
History	`yuiblog.com/blog/2006/10/10/ten-questions-slocum`
History	`web.archive.org/web/20090614072938/http://jackslocum.com/blog/2007/08/01/ext11`
Company	`sencha.com/company`

Table 10.5: Sources for Ext JS (last accessed: Aug 29, 2012)

10.1.6 qooxdoo

qooxdoo is an open-source framework sponsored by the company *1&1*. It has been initiated by 1&1 in 2004 and was registered as an open-source project at *Sourceforge*[4] in Jan 2005. qooxdoo is offered in four variants for enhancing websites, mobile, applications, and server. Currently it offers a dual licensing model were both licenses permit commercial usage in closed-source projects. qooxdoo – quite uniquely among open-source projects – has a dedicated core developer team working full-time under the lead of Andreas Ecker.

The current and evaluated version of qooxdoo is 2.0.1 released on Jul 3, 2012.

Topic	URL
History	`qooxdoo.678.n2.nabble.com/I-m-a-Ext-LLC-Jack-Slocum-victim-and-I-m-thinking-about-converting-to-Qooxdoo-td783372.html`
History	`qooxdoo.org/project/developers`

Table 10.6: Sources for qooxdoo (last accessed: Aug 29, 2012)

[4] sourceforge.net (last accessed: Aug 29, 2012)

10.1.7 SmartClient Ajax Platform

The SmartClient Ajax Platform (SmartClient) is a product of the privately held company *Isomorphic Software*, based in California, USA. Isomorphic Software sees enterprises "from banking through telecom to defense" as the target group of its frameworks and services.

According to its own description the SmartClient SDK was released in 2001, more information on the history or development of the project is not available. It is available as an open-source edition with a commercially usable license and commercial editions. The current and evaluated version is 8.2, which was released on Dec 5, 2011.

Topic	URL
Company	`smartclient.com/company`

Table 10.7: Sources for SmartClient Ajax Platform (last accessed: Aug 29, 2012)

10.1.8 SproutCore

SproutCore is a community-backed framework, which was started by Charles Jolley at the company Sproutit, based in California, USA. Development started in 2007 with the goal to rewrite the company's email management application so it can be run in a browser. On an architectural level its – like Cappuccino – connected to Cocoa but unlike Cappuccino not a copy, instead it is "built around the Model-View-Controller programming model that Cocoa uses". SproutCore was used by Apple in 2008 to build the web application collection for MobileMe, it included an email client, basic contact management and a calendar, these have been updated and are still part of the iCloud service today.

In 2010 a debate arose about the future of SproutCore 2.0 which would have brought significant changes to the whole framework; the result was a split. SproutCore 2.0 was renamed to Ember.js. Development on the legacy version of the framework continued, coordination of development and community has shifted from Charles Jolley to Tyler Keating. The most recent and evaluated version of SproutCore is 1.8.2, released on May 10, 2012.

10.2 Evaluation Results

The scores of the results are presented in percentages, every category was normalized so that the contained criteria accumulate to 100% and then the category weight is multiplied. Behind the presented percentages though, is a system of five grades in with a 25% interval. Only in cases were it seemed

Topic	URL
History	arstechnica.com/apple/2008/06/sproutcore-rich-web-apps-in-javascript-no-flash-needed
History	web.archive.org/web/20070830190206/http://www.sproutcore.com/blog
MobileMe	appleinsider.com/articles/08/06/16/apples_open_secret_sproutcore_isvcocoa_for_the_web.html
Change	blog.sproutcore.com/changes-to-sproutcore

Table 10.8: Sources for SproutCore (last accessed: Aug 29, 2012)

highly unappropriate to decide for one grade an intermediate value was given. The results in full detail can be found in appendix B.

10.2.1 *sta* Getting Started

As a main source of information the project web site served. Every framework was downloaded, installed, and tested. The more elaborate getting started tutorials were not implemented, they were just scrutinized.

Category Weight: 10%

Bindows

The text offered on the Bindows project home page fails to present meaningful information. It emphasises phrases like "better than desktop", "richest windowing", "most powerful", or "best-in-industry" instead of information about concepts and technologies. The almost sole concrete information is the replication of the "Windows look-and-feel", this communicates a clear target group.

The path to getting started with development is not straightforward. The *Downloads & Purchase* section of the menu contains a link to a document which requires personal data input. After submitting, a link is sent via email, then the actual download process can start.

The zipped package contains release notes, some samples, a directory filled with all HTML files of the API named with the containing class, and the framework. The getting started document points to the getting started section on the website.

The getting started section on the Bindows project site is similar to the documentation site of other frameworks. The main pointer goes to the manual titled *Bindows - User's Manual*. Other than that a very basic *Hello World* tutorial can be found in the tutorial section of the *Developer Resources*. Bindows offers no dedicated getting started document, the first sections of the manual

seem to be intended for this purpose. They cover installation, architecture, directory structure, and layout. The document has been last updated in 2005.

Score: 38%

Cappuccino

The Cappuccino project home page presents an unmistakable statement about the objectives of the project: "desktop-caliber applications that run in a web browser". The introductive overview is found under the link *Read all about Cappucino and Objective-J* and clearly expresses the framework's goal of helping to develop desktop-like applications, the license, the source of the APIs, the complete abstraction from the DOM, HTML and CSS, and shortly explains differences to other popular JavaScript frameworks like Prototype and jQuery.

A direct link to the download of the most recent version is placed on the home page, a dedicated getting started lead is missing. However, the downloaded package contains a readme with a short explanation pointing to the *Downloading & Running the Sample Application* tutorial on the website, and showing the installation of the tools. The basic getting started tutorial shows how to modify a *Hello World* app.

The more enhanced getting started tutorial is an incomplete series of documents titled *Building the Scrapbook App*. Two parts are available, there are hints to more, but they do not exist yet. The series provides detailed instruction on the building of the layout and the setup of drag & drop, and with completion of the second document a working image editor has been implemented, but it misses out on links to in-depth information, comments about the framework architecture, or a troubleshooting section.

Score: 63%

DHTMLX

The introductive overview is found on the home page of the DHTMLX suite. It presents a widget list, learning resources, and the basic idea of the widget, the data store and the server communications concept. The download can be initiated directly from this page. A getting started guide is available under the name *DHTMLX. Start Building Web Applications Today*. It is not directly accessible from the front page and has to be found in the documentation section. It covers the creation of a simple contact management application, but is superficial and hard to follow. An obstacle to comprehension is the wording; the guide was not written by a native speaker, probably by a Russian person[5].

[5]the author has spent a year in Prague and feels confident to recognize the English of a Slavic speaker (e.g. articles for nouns are frequently missing, since Slavic languages do not have articles), furthermore the DHTMLX office is located in Saint Petersburg

In addition the guide is concentrating on the creation of the layout, requires the usage of the PHP version of the dhtmlxConnector for the server connection - tutorials of other frameworks use fixtures - and fails to describe patterns or ideas behind the framework and link to more in-depth information.

Score: 35%

Dojo Toolkit

The introductive overview is found on the project home page and shortly mentions the MVC architecture, bindings and forms. A large part of the site is dedicated to the detailed presentation of the data grid, the rich text editor, and the calendar widget.

Not only is a dedicated getting started link missing but also a comprehensive document. A section titled *Getting started* is available on the *documentation* page but it is aimed at designers wanting to enhance their website, not at application developers.

Dijit is the part of the library that is promoted as being important for application development but the document *Beyond Dojo's Core: Dijit and DojoX* is not more than an explanation of what the Dojo Toolkit is capable of and fails to present how this can be achieved or where that information can be found.

Another document *Application Controller* makes quite a clear statement: "Dojo does not express an opinion on how you should assemble applications out of the components it provides. It has all the nuts, bolts and moving parts, but no blueprints."[6].

Score: 18%

Ext JS

The introduction is found on the project home page and offers a short overview of the follwing features: UI widgets, documentation, development tools, charts, framework component model, browser compatibilty.

A dedicated getting started link or button is missing, tutorials for beginners have to be looked for in the *Documentation* or *Learn* section. Downloading the GPL version is straightforward, the commercial evaluation version requires an email address.

A first steps guide *Getting Started with Ext JS 4.0* is available as well as a three-part tutorial *Architecting Your App in Ext JS 4*. The tutorial provides a basic understanding not only of the inner workings of the framework and the

[6] dojotoolkit.org/documentation/tutorials/1.7/recipes/app_controller, retrieved: Aug 13 2012

directory structure of an application but also of the design of an implementation for a web application. It provides a real life example of a user interface (a Pandora[7] app) and walks through ideas of how this UI can be implemented. The tutorial also touches on model, fixtures, and data store concepts. The third part of the tutorial concludes with an outlook on the next parts, which are missing. Customizing components or deploying is not covered here, but mentioned in the getting started document.

Score: 83%

qooxdoo

The project home pages offers a clear overview of the four types of frameworks and libraries, describing the target group and most important features. The rest of the page presents the project backer, license, the general idea, and the toolchain, enriched with links to more in-depth information. Clicking on *Desktop* offers a short source code example, links to demos, getting started, documentation and a direct download of the framework package.

The getting started link takes the aspiring developer to an extensive tutorial that guides the creation of a twitter client. The instructions cover all of the basic knowledge that is needed for development, show best practices, mockups, screenshots, code listings, and links to in-depth information. After the tutorial the developer seems to have a solid understanding of the main patterns and concepts behind the framework and should be able to start developing small web applications.

Score: 90%

SmartClient

The project home page offers an introductive text covering coding, testing, data management, mobile applications, server integration, and UI widgets, but in a superficial and unspecific way. It neither names any components nor does it include links to more in-depth information. There are however links to a demo page, which showcases various aspects of the widgets, to the download section, which requires a registration, and to the quick start guide.

The quick start guide is an up-to-date PDF document, which states right at the beginning the version for which it is intended (v8.2 November 2011). It presents an overview of: architecture, installation, requirements, deployment, layout system, data binding, and server communication. The quick start guide is not a tutorial, it is divided into chapters that cover a single aspect separated from others, it has the character of a manual.

Score: 62%

[7] pandora.com (last accessed: Aug 29, 2012)

SproutCore

The SproutCore project home page offers a very short introductive overview; the six mentioned points, HTML5, native experience, MVC architecture, scalability, performance, theming are accompanied by one sentence.

However, SproutCore and its tool chain are easy to install with the provided package or with an already set up Ruby system. The getting started section offers a good tutorial divided intro three parts which guides through the development of the first application, a Todo-App. The tutorial covers the whole process from generating an application to deploying it, describes basic ideas of the development with the framework, points to important documentation sources for in-depth information. It explicitly mentions best practices for developing with SproutCore, the directory and file structure of the framework, and main concepts behind the framework. It is up to date, since having been rewritten for the newest SproutCore release in March 2012. This can be conveniently derived from its revision history, which can be found on the bottom of the pages.

Score: 82%

Conclusion

The main goal of a framework's getting started material should be to inform a developer so that a decision can be made if the framework is worth a more detailed investigation. This goal is met by quite well by qooxdoo, Ext JS, and SproutCore, closely followed by SmartClient and Cappuccino. Bindows, DHTMLX, and Dojo lack the necessary focus and structure. DHTMLX and Bindows seem very concentrated on their widget system leaving out the important rest.

(a) Bindows

(b) Ext JS

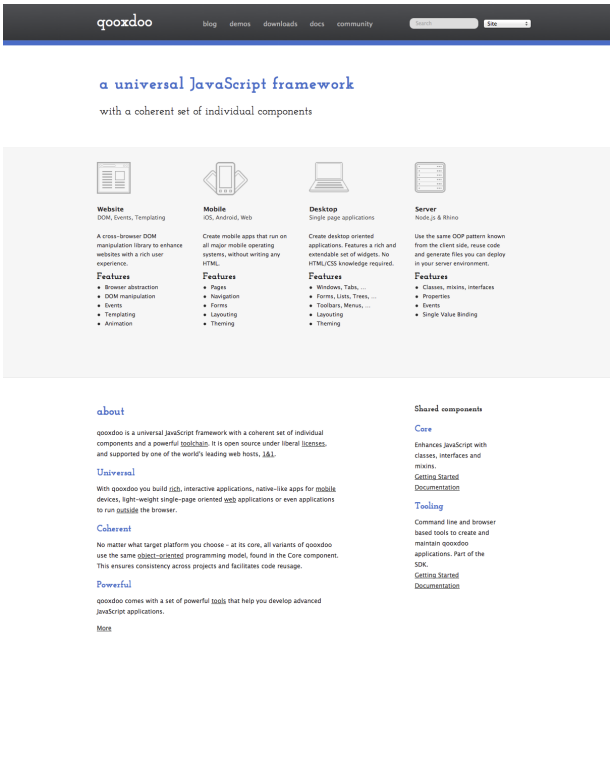
(c) Cappuccino

(d) DHTMLX

Figure 10.1: The top 1500px of the project home pages at a width of 1203px

(a) Dojo Toolkit

(b) SmartClient



(c) qooxdoo

(d) SproutCore

Figure 10.2: The top 1500px of the project home pages at a width of 1203px

10.2.2 *doc* Documentation

The documentation was rated by exploring the material on the websites only. It was tried to find as much documentation as possible, often with the help of the *Google* search engine, if something could not be assessed as up-to-date, or seemed particulary old it was not taken into account.

Category Weight: 25%

Bindows

The Bindows documentation consists of the manual, the API, and a tutorials page.

The manual is a PDF document that has been created in 2004 and last updated in 2005, some parts are still marked "TBD"[8]. Coverage is acceptable, the topics are: framework architecture, directory structure, drag & drop, charting, internationalization, and tree view. The quality is good, the text includes ideas of the framework, pitfalls, and example source code. Outstanding is the fact that the PDF is not searchable. It seems as if the copy text has been obfuscated in the background, only source code can be searched for and copied. This, combined with the fact that neither the table of contents nor the index has in-document links, presents a huge obstacle for efficient usage of the document.

More information is not available, tutorials are present but they are very short and consist mostly of example source code. The Bindows online API reference consists of the two index pages, with alphabetical and hierarchical sorting, and the corresponding class description pages. The offline version is not found in the *documentation* but in the *example* folder, it has a sidebar with those two indexes as tabs, no search box, the content view shows basic information about the class: a short paragraph of description and a list of properties and methods.

Source code examples are not available. Information about the new major version (4) and an upgrade process is only a marketing page describing new features, but release notes are publicized.

Bindows is very special in regard to the *last updated* date shown in the footer of all pages of the website. The one of the front page for instance was changed to a more recent date several times during the course of the creation of this work, the page content however stayed the same. It may be the *last updated* date for the whole website, but significant changes like new blog posts or new menu entries have not been noticed. If *last updated* refers to small changes like error corrections it is misleading to display a new date on every page, if no changes have been made and just the date was changed this is deceptive. Which of these two it is can not be decided without in-depth

[8] probably: to be delivered

investigation, which will not be conducted here, nevertheless this is influences the rating because it is unprofessional.

Score: 37%

Cappuccino

The documentation of Cappuccino is limited. Guides are only provided for layout management, drag & drop, adding undo, and debugging. No descriptions for general architecture, storage, testing. Because of the close relation to Cocoa, which has an enormous amount of documentation and books, some of this could be alleviated by references to parts of this documentation but this opportunity is not used.

The API reference is provided in a standard viewer, some different sortings for the classes and its members are available, so is an incremental search field on the top right. A lot of classes and methods are scarcely commented.

The immature state of the documentation reflects the framework version number below 1.0. Due to the absence of a major revision change an update document is not necessary, description of new versions are thorough but only findable through a general search engine[9]. Information about the timeliness of the content is only available for the API referece, the website or any of the tutorials are completely unmarked.

Score: 37%

DHTMLX

All documentation that DHTMLX offers is presented on the *Documentation* page.

Documentation is available for every complex widget, for the server-side part of the framework, storage, and some basics. The quality of the documentation is inferior. The general idea behind the framework is not explained at all, the sections specific to widgets link to documents that link to further documents and then provide no indication where to continue. It is difficult to assess the coverage and quality of the coverage. Generally the documents are very short and contain mainly source code, quite similar to the tutorials of Bindows. The widget descriptions cover a lot of ground but are difficult to navigate.

A complete API reference is not available, only separated according to type of widget. Guides for upgraders are available for some parts of the frameworks, comprehensive release notes are not provided. Information about the timeliness of the content is not provided.

Score: 28%

[9] tested with Google

Dojo Toolkit

The Dojo Toolkit documentation includes the reference guide - a kind of manual - the API reference, and a long list of tutorials.

The table of contents of the reference guide is divided into six sections: quickstart, dojo, dijit, dojox, utilities, and miscellaneous. These six sections have no further subsections and contain approximately 1000 entries on one HTML page. This qualifies as a listing but not as a guide. The reference guide has a second view for its content, the *startpage*. It links to overviews for dojo, dijit, dojox, util, release, and developer notes. These overviews offer a second layer of headers with occasional short explaining paragraphs, but they too have the character of a listing. Opening a link from an overview presents a content page with links to previous and next topics that are completely different from the ones presented in the overview listing. The content pages are of varying type and quality, some are short guides, some are like an API reference, some contain code examples. It seems that a lot of ground is covered but it is difficult to find what one is looking for.

The tutorials page is quite similar, in that it shows a large amount of tutorials, but has almost no content structure. Most of the tutorials seem to be targeted at web designers who want to enhance a website with some JavaScript. Documentation targeted at application developers can not be distinguished.

The API reference itself has a left sidebar with a tree view but lacks a search field. It features some icons that show whether elements in the tree are objects, classes, or functions, in the method list of the content view every entry is accompanied by a function icon, and a property has an icon describing its type instead of a proper type declaration. The explaining purpose of the icons would be better served with a consistant naming convention (e.g. `_BidiSupport`, `_Calendar`, and `_Contained` are a function, an object, and a class repectively).

Examples can not be found. Extensive and commented release notes for every major version are provided, information about last updates to the documentation not.

The Dojo Toolkit documentation has to be classified as unstructured and inhomogenious.

Score: 29%

Ext JS

The Ext JS documentation is divided into a *Learn* and a *Documentation* section. The latter is an application of a typical API reference viewer style, but the sidebar offers tabs for the API, guides, videos, and the extensive example suite. This makes the page the dedicated place for nearly all available documentation, since the *Learn* page lists some of the guides and only some

additional FAQs. It also includes an incremental search box that is placed at the top right and searches through the content of all tabs.

The guides cover all major topics: upgrading, class system, MVC architecture, layout management, storage, drag & drop, testing, and some components. The quality is good, they contain useful copy text, source code examples and links to the API.

The API is represented by a tree in the sidebar that can be sorted by package or by inheritance. The content view shows an overview bar at the top with the number of configs, properties, methods, events, and CSS mixins of the displayed class. Hovering over an entry reveals the complete list, a click scrolls the page to the desired position. The bar has a text field that allows incremental filtering of all class members and a drop down menu that allows to select the display of public, protected, private, inherited, accessor, deprecated, and removed class members. This viewer provides the most straightforward and useful experience of all candidates.

Moreover, the documentation includes an extensive collection of examples with their source code showcasing basic and advanced capabilities of the framework, mostly related to widgets. Included as well is a detailed update guide to from version 3 to version 4 and extensive release notes, the guides contain not only the latest improvement date but also the name of the author.

Score: 84%

qooxdoo

The qooxdoo documentation page provides four fields with the follwing headers: *In a Nutshell*, *Manual*, *API*, and *More*.

The manual provides a overview page with clearly separated entries for the different editions of the framework. The coverage is very good; the topics are: detailed introduction to the framework, class system, data binding, widgets, layout management, theming, client-server communication, debugging, performance, testing, developer tools. The topics are split up into separate pages but a navigation system which allows the continuous consumation of the content is present. The quality of the manual is good, it contains useful copy text, example source, links to other parts of the manual und the API.

The API reference viewer features a left-hand sidebar with a tree view and a content view. Within the tree icons help distinguish between packages, classes, static classes, and mixins. The viewer also provides incremental search with optional filters for types of searched components (package, class, property, etc.). It provides the possibility to change the visibility of regular, inherited, protected, private and internal properties and methods, quite similar to the reference viewer of Ext JS, although with a little bit less functionality.

Examples with source code are provided in the demos section, some of them are summarized in a separate list and enriched with a short explanation

and links to releated documentation. The demo browser features an extensive collection of use cases with commented source code. qooxdoo provides a detailed overview of changes for every new version and a short but straightforward migration section for major revision upgrades. The project site shows last update time and person for every page, but not for the manual.

Score: 84%

SmartClient

The SmartClient documentation consists of the PDF quick start guide, the reference viewer, and the example set. The quick start guide includes the topics: overview (architecture, capabilitites), installation, coding, layout management, data bindings, client-server communication. It is - according to the declaration on the front page - up-to-date and features good copy, code examples, links to APIs, and in-document references, but they are too short.

The reference viewer has a left-hand sidebar with a tree view and a non-incremental search box. The viewer is a SmartClient application that does not use the native scrolling behaviour of the web browser for overflowing `div` tags, but a self implemented version. This leads to problems with performance and scrolling sensitivity. It contains additional guides and the API reference. The guides cover: architecture, debugging, internationalization, are short but informative, similar to the quick start manual.

The API viewer itself is slightly more complicated than in other frameworks: the content view has two tabs *Overview* and *Instance APIs* which requires an extra mouse movement and mouse click to get to the information about class members. The tree is sorted according to concepts (e.g. *Grids*, *System*, *Drawing*). An alphabetical or hierarchical sorting is not possible.

The example suite is extensive, covers a lot of UI aspects and provides source code for all simple examples. Release notes are provided for every major version, the quickstart guide shows it last improvement date, but information about editor or creation date of the guides is not provided.

Score: 56%

SproutCore

The SproutCore documentation consists of an index page for the guides and the API reference viewer.

The guides cover: class system, bindings, view system, store, theming, testing, build tools. The quality is very good, they are extensive, refer to the same app, include notes and pitfalls, the copy is helpful, the example code also includes the name of the file it should go into, and a changelog is included at the bottom of the page, so the up-to-dateness can be easily ascertained.

The API viewer is basic. The left-hand sidebar has an incremental search field, but only a flat alphabetical listing of all classes. The content view is a standard page, additional comments on classes are present.

The page also features a showcase section where all widgets are presented and the relevant source code can be seen. In addition to the changelog for the guides, the repository contains a detailed changelog for all releases.

Score: 73%

Conclusion

Theoretical background and reality correlate quite well in the area of documentation. The standard repertoire of documentation comprises: a dedicated page, an API reference, and guides for central framework concepts with explanatory text including code listings and links to more information.

The usual way of presenting an API reference is a dedicated viewer with a side bar on the left containing the class tree as well as a search-as-you-type field and a main content field showing details about the currently selected class or method. Bindows is the only framework that does not provide an online API viewer just an offline version, DHTMLX does not provide a coherent API reference at all. qooxdoo and Ext JS on the other hand offer an advanced viewer that allows the alteration of visibility of various class member types.

The presentation form of guides is more differentiated: Bindows, and qooxdoo provide a manual; Ext JS, Cappuccino, and SproutCore collect links to guides on a list; SmartClient has both; Dojo and DHTMLX have no dedicated place. The quality of the guides themselves is equally diverse. SproutCore can serve as a role model in this regard, the guides are well written, easy to follow, enriched with small notes and caveats, and they all refer to the same application, which helps in organizing the mental model.

Extensive example suites are uncommon, despite their significant contribution to a beginner's learning experience. Only Ext JS, SmartClient, and qooxdoo offer a relevant collection for quick reference. This may be an area that is underestimated in its significance.

10.2.3 *com* Community and Presentation

The criteria evaluated in this category are very time dependent so to ensure comparability, care is taken that every criterion is evaluated for all frameworks at once. All results were retrieved around July 27, 2012. Most of the results will not be reproducible though, since they are dependent on complex algorithms and crawling bots, and rely on changing data sets.

The scores for searches on Amazon are defined as follows: no results (0%), mentioned in a single book (25%), mentioned in several books (50%), availability of dedicated books (100%). Matches on Heise are classified in a

similar manner: no results (0%), mentioned in a news items or articles (25%), dedicated articles are available (100%).

The forum activity is judged by looking at the frequency of postings within the last week and month, the scores are: several posts per day (100%), several posts per week (75%), several posts per month (50%), not much more than one post per month (25%).

Where search terms are likely or noticeable triggering lots of false positives, care is taken that these are filtered out e.g. with the addition of the boolean search term `AND(jsORjavascriptORlibraryORframework)` or, if the number of results is very low, by manual selection. Terms affected by their meaning in common language are: cappuccino[10] and dojo[11]. For other reasons problematic terms were again Dojo, the name of the foundation behind the Dojo Toolkit and home to other projects; Ext JS, which is often written ExtJS and since June 2010 closely associated with its then formed parent company Sencha; and SmartClient, which is a technology term in use for products by Microsoft and other companies.

On Stack Overflow the feature to tag every post is used consistently by the community, so this mechanism was employed. Therefore the number of hits does not represent matching results but posts marked with the tag corresponding to the framework.

The number of blog posts is measured in *number of blog posts between Jul 01, 2011 and Jul 26, 2012*, so is the number of tweets.

Activity and maturity are a requirement for every candidate that is still under investigation they are not further rated.

Category Weight: 25%

Bindows

Bindows has a very small community. A search on Amazon reveals only one book where Bindows is mentioned, on Stack Overflow Bindows does not even have a tag of its own; Heise shows no match. This is also reflected on the forum, which is not well frequented. Communication from the supplier towards the community is lacking as well.

Real life apps are presented in the *Screenshots* section of the Bindows website. It contains some screenshots of Bindows applications acommpanied by a company logo. Explanations are missing here but some are given on another page *Case Studies*, the information there however is limited to marketing language. The applications mentioned on the *Screenshots* page can not be accessed and it is unclear if they are still in existence.

Score: 11%

[10] a type of coffee drink

[11] a Japanese term related to martial arts

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
1	0	0	118k	4	n/a

Table 10.9: Search hits and posts for Bindows

Cappuccino

The Cappuccino web framework is quite popular, still, it has no dedicated books, but is only mentioned in some books[12]. It has good scores at Hacker News, where it has the second most mentions. The communication from developers towards the community is limited. Tweets are quite frequent but not a lot of entries to the blog have been posted.

Cappuccino uses a Google group for discussion. The activity there is average.

The website features a list of real life applications on the *Demos* page. Among them *Mockingbird* is a capable application for creating user interface mockups, another example is a commercial time tracking tool. These applications give a real sense of the capabilities of the framework.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
718	117	1250	5.800k	7	45

Table 10.10: Search hits and posts for Cappuccino

Score: 48%

DHTMLX

DHTMLX is not very popular either. It is mentioned in one book on Amazon and few hits on Hacker News or other sources, but it has tag of its own on Stack Overflow. The communication from the DHTMLX company to its community is good, the team posted frequently to twitter and to the blog.

DHTMLX does not show real life examples, it hosts a long list of customers without testimonials or explanation of how or why they are using the product. It is not possible to validate the list.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
5	56	67	873k	19	50

Table 10.11: Search hits and posts for DHTMLX

Score: 39%

[12] search terms: “cappuccino javascript”, “cappuccino web framework”

Dojo Toolkit

The Dojo Toolkit has an impressive community, but it is diverse. Due to its nature as a highly modular project not only application programmers but a lot of web designer are interested in Dojo.

An objective search on Amazon is quite difficult, searching for "Dojo" brings up many books about martial arts, so the term "Dojo Toolkit" was used. This leads to close to 100 results, including dedicated books.

In all other areas of the Internet criteria and also in regard to forum activity Dojo is leading the field. The team behind Dojo is actively communicating the progress to the community, Twitter is the main channel, but also blog posts have been frequent.

The Dojo project site does not provide any information about real life applications apart from an uncommented list of nine companies titled *creating apps with Dojo*.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
844	3396	2730	48.700k	21	287

Table 10.12: Search hits and posts for Dojo Toolkit

Score: 85%

Ext JS

The popularity of Ext JS is comparable to Dojo's. Dedicated books are available, some directly targeted at web application development with version 4; on Heise only some news articles can be found. Ext JS has a massively used Twitter stream, but this account is a Sencha account and hence used for communication for all of its products, it has therefore not been rated. The blog is similarly mixed, but it was possible to filter the posts unrelated to Ext JS. The web site has its own forum which is frequently used.

Links to real life applications are not provided but the site offers a spotlight section in its blog. In there, companies that use Ext JS explain why and for what application they are or were using it. It provides tips for new developers and in the comments section questions can be asked and are answered by the developers in the spotlight. This helps to get a sense for the area of application of the framework.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
90	6073	2000	33.500k	28	n/a

Table 10.13: Search hits and posts for Ext JS

Score: 74%

qooxdoo

Amazon reveals that qooxdoo is mentioned in some books about JavaScript and it has a dedicated book in existence. It is also regularly mentioned on Heise and articles cover different aspects of the framework. Its popularity on other platforms is average, but activity on the mailinglist is very good. The developer team is outstanding in its communication towards the community. The blog features a weekly update and regular explanative posts, this consistency is unique among all candidates. Twitter has been used quite frequently too.

qooxdoo offers quite a long list of real life applications in the section *Community/Real-life Examples*. Nevertheless, out of the ten top-listed applications only one is accessible, the others link to pages in Ukrainian or French or do not exist any longer, or maybe both[13]. The accessible application loads and is functional, it features data display in a tree and a table view.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
53	146	55	357k	83	51

Table 10.14: Search hits and posts for qooxdoo

Score: 67%

SmartClient

SmartClient seems not to be mentioned in any book. The search for "SmartClient" yields some results, but they are connected to the Microsoft software component. The popularity in general is quite low, no mentions on Heise, few on other platforms. The site's own forum is very well frequented, lots of traffic stems from the team itself who is quite quick on answering questions from its customers. Communication from the company towards its customers on public channels is below average, a twitter account can not be found.

Isomorphic Software has collected information about real life usage of its products on the *Our Customers* page, which is quite superficial. The long list of corporation names is similar to the DHTMLX listing, the spotlight section below gives some more details, names of the companies and the products using the framework are listed with a short summary or a customer quote, still not very concrete information.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
7	80	36	137k	15	n/a

Table 10.15: Search hits and posts for SmartClient

Score: 23%

[13]The author does not speak any of those languages, so determination is not possible

SproutCore

Searching for SproutCore reveals some books but searching inside them does not yield any matches, only one is clearly speaking of the framework. There are only some mentions on Heise, but SproutCore is very popular according to the other sources; activity on the Google group is average. The communication from the core team towards the community is good too, especially tweets are very frequent.

SproutCore has a small selection of working real life examples right on the front page. It is not commented, some take the user to the apps right away, some require a login, some are not reproducibly made with SproutCore. From the presented applications *GestiXi*, a kind of online shop management software in French, *Hubbub*, a social neighbourhood lending service, and *Insightify*, a survey creation tool, show the flexibility and capabilities of the framework with trees, charts, table views, menus, maps, and custom UI widgets.

Hacker News	Stack Overflow	Quora	Google	Blog Posts	Tweets
667	195	195	513k	33	156

Table 10.16: Search hits and posts for SproutCore

Score: 55%

Conclusion

Despite the objectivity problems the criteria seem to give a good sense of the community behind a project. The impression is that the commercial frameworks Bindows, DHTMLX, and SmartClient have not managed to build up a significant community, despite their age, that the relatively young projects Cappuccino and SproutCore have already aroused a lot of interest, and that especially Dojo is extremely popular; Ext JS and qooxdoo are closing up.

Some books that come up on Amazon are simple collections of Wikipedia articles, available as a print on demand, these are misleading and are sorted out. In general books are uncommon for the investigated kind of web application frameworks, maybe the criterion should be rethought. Heise does not seem very concerned with this type of software too, this can be another point of improvement for the model. Another metric that could be useful is the number of followers on Twitter. As side note to Quora, which could not be asked directly but is queried via the google search term addition `site:quora.com`.

The source for the number of twitter posts is the service at `tweetstats.com`. It offers a graphical overview for the number of tweets for every month the account has been active. These numbers are added up. If a Twitter account is not findable, this criterion is not rated.

Real life applications proved hard to assess. Many projects link to applications that could not be accessed directly. In the best case they required a free registration, this is done for a maximum of two applications; in other cases the web page is in a language other than English or German, and screenshots were not provided or inconclusive, so it is not possible to determine if the related company is really using the framework. Often a developer is cited confirming a positive experience with the framework, but this may refer to a pilot project or a prototype that has never been published. Therefore only usable and accessible applications were rated positively; outdated links and unaccessabile applications lead to a slightly negative score, as a sign of sloppy maintenance on the side of the project site responsibles. Short customer quotes were ignored, due to not being reproducible, extensive customer spotlights with reproducible real life background were conceded a positive influence.

10.2.4 *fea* Features

JavaScript provides a very flexible environment, so it is quite likely to be able to find an expert who – provided with an advanced knowledge of working with the framework – knows how a feature can be included. However, this thesis is mainly concerned with the experience a beginner is offered. Therefore the assessment concentrates on the availability of documentation for a feature, it needs to be either explained in a guide or the API reference, or at the minimum in a blog post on the project site or from a person that is a contributor to the project. The quality of a feature can not be evaluated one the surface, this can only be accomplished by implementing a custom application.

Providing a layout manager and abstraction from HTML, CSS and the DOM is a requirement for all the candidates that remained in the evaluation process to the current stage, it is not rated again.

Category Weight: 20%

Bindows

Bindows does not seem to have a data store. Some model classes are described in the manual, but they are tied to widget types, a general description is not available. The assumption is supported by the confirmed lack of bindings, in an official blog post they are announced for version 4.5[14].

Drag & drop as well as internationalization (i18n) are described in the manual; description of keyboard shortcuts, theming, and the configuration of the context menu can be found in how-tos or blog posts, tool tips in the API reference. Form validation is only provided on a very basic level, the API

[14]MB Technologies : Bindows 4.5 beta; mb.bindows.net/news/Bindows45beta.html; retrieved 2012-09-01

reference of some widgets mentions a property that can be assigned a regular expression validator. Support for an offline mode, server push, and browser history management is unclear.

Score: 50%

Cappuccino

Support for a data store in Cappuccino could not be confirmed. As of 0.9 Cappuccino supports bindings, this is stated in a blog post. Detailed documentation is not provided. Drag & drop received a thorough guide in the getting started document, theming and tool tips are mentioned in a blog post, support for all other features is unclear.

Score: 33%

DHTMLX

DHTMLX features a data store, however the documentation is very short and only explains basic usage, background information is not available. The same is true for bindings: standard use cases are covered, but information on concepts or technical details are left out.

In DHTMLX most features that can be deployed per widget are provided, others that need application wide support are not. Input validation is found in forms, for offline usage a proxy class is provided and server push is supported as well. Drag & drop and i18n however are only provided within a widget, there is no documentation for an application wide handling. Theming is limited, three themes are built-in and their color can be changed. Support for tool tips, keyboard shortcuts and browser history management is unclear. The context menu is available as a separate widget.

Score: 45%

Dojo Toolkit

The Dojo Toolkit has a data store, its documentation however is spread over several documents - two of them can be found on the tutorials page *Dojo Object Store*, *Data Modeling for MVC Applications* others in the reference guide or the API reference - they also mention bindings and validation. Drag & drop is extensively described in a blog post of the contributing company *Sitepen*. I18n, theming, and tool tips are a part of the reference guide. Dojo is delivered with four built-in themes. Support for context menus, keyboard shortcuts, and server push is unclear. Documentation for offline capability is present but marked outdated, browser history management was added in 1.8[15].

[15] released after the cutoff date

Score: 60%

Ext JS

Ext JS has a data store since version 3.0, it is explained in a separate guide, so are drag & drop, theming, and keyboard shortcuts, validation is a part of the model description. Bindings, the context menu, and browser history management have a short API reference, tool tips a detailed one. Solutions for offline application handling are described in a video. Support for server push is unclear.

Score: 78%

qooxdoo

Guides for bindings and the data store are placed in the documentation for the *Core* framework, offline storage is described in there as well. Further guides are available for drag & drop, i18n, theming - the framework has four built-in themes - input validation, and keyboard shortcuts. Tool tips and browser history management are documented in the API, the guide for the history functionality is marked outdated. Support for server push is unclear.

Score: 90%

SmartClient

The data store in SmartClient is implemented in `DataSources`, they are explained in the quick start guide together with bindings. Other guides are available for i18n and theming. The API reference is the place for short information about context menus, the offline mode, and history management. Barely adequate information is provided for drag & drop, where only source code examples can be found, and for validation, which is occassionaly mentioned in the quick start guide and the API reference. Server push is mentioned but it only works with Java servers. Support for tool tips is unclear.

Score: 76%

SproutCore

SproutCore has an advanced data store and it is documented in detail in the guide *Records*, together with bindings. Drag & drop, input validation, keyboard shortcuts, and browser history management have a detailed API reference with explaining text. History is handled by `SC.routes` its reference is a little bit less detailed. Theming is discussed in a dedicated guide, i18n in a blog post titled *Localization* on the SproutCore blog, tool tips have a blog post as well. Support for context menus, offline usage, and server push is unclear,

general articles can be found that confirm an offline mode but documentation is missing.

Score: 78%

Conclusion

Despite the clear-cut nature of the criteria it was quite difficult to ascertain the features' availability. The structured documentation of many frameworks is missing key information and therefore various other places of the project website have to be consulted. Key functionality is not available in all frameworks.

10.2.5 *uif* User Interface

The rating in this category is concerned with elements that are an integral part of the framework. Plug-ins and other user contribution are not taken into account.

Category Weight: 10%

Bindows

Bindows presents all of its widgets in a showcase app. The provides coverage is very good, all essential elements are present, a speciality of Bindows is a highly promoted gauges library. Missing widgets are the map view and the rich text editor.

Score: 80%

Cappuccino

The Cappuccino project site neither lists all widgets nor does it provide a showcase application. Judging from an examination of the real life apps and the API reference the list of unavailable elements includes: rich text editor, charts, calendars, and maps.

Score: 60%

DHTMLX

As DHTMLX started out as a widget library its coverage of widget types is very good, still a map view is missing.

Score: 90%

Dojo Toolkit

The widget collection of the Dojo Toolkit is extensive, everything is covered except for a map view.

Score: 90%

Ext JS

The Ext JS widget collection is complete. For almost all widgets source code examples can be found in the examples collection.

Score: 90%

qooxdoo

The demo page has a link to a widget browser, that showcases the qooxdoo widget collection. Not available are charts and maps.

Score: 80%

SmartClient Ajax Platform

The feature explorer on the SmartClient project site is extensive and provides examples for all widget types, except for a map view.

Score: 90%

SproutCore

The link to the widget showcase is very prominently placed in the top bar of the project site. Widget types that are missing: text editor, chart, date picker, and maps.

Score: 60%

Conclusion

The user interface category does not show a lot of diversification between candidates. Coverage of the standard widgets is given for all frameworks.

Although some of the more advanced widget types are not provided by a framework directly, a blog or forum entry discussing their disposal with the help of an external library can easily be found. To account for this the weights do no grant them a significant influence on the total result.

10.2.6 *dev* Development Setting

Category Weight: 10%

Bindows

Support for deployment or code generation is not mentioned at all. Built-in support for testing is unclear. One document describes testing with an external tool, but this document has been last updated in 2004. Bindows can communicate with a server via XML or JSON. Of all evaluated frameworks Bindows is the only one with no open-source license. It is stated that free licenses are available under the requirement to get in contact with the sales team.

Score: 14%

Cappuccino

Cappuccino's support for code generation or testing is unclear. Deployment tools are provided in the form of scripts to be run with the Rhino engine, they offer command line help and an online documentation in a blog post on the Cappuccino blog.

It is unclear what protocols are supported. The license is LGPL, which allows free commercial development.

Score: 45%

DHTMLX

It is unclear if DHTMLX provides any development tools, nothing is mentioned on the site. The server communication protocols include CSV in addition to the standards JSON and XML. DHTMLX is offered with a free license and commercial licenses. The commercial licenses have "some additional features", which remain unspecified, the free license is the GPLv2 which requires developed applications to be open-source.

Score: 28%

Dojo Toolkit

The Dojo Toolkit is fairly good equipped with developer support. A separate guide and suite of JavaScript scripts for testing and the build system, not for code generation; they run on Node.js or Rhino.

Communication with the server is supported via JSON and XML and the Dojo Toolkit can also be used in commercial closed-source projects.

Score: 89%

Ext JS

Ext JS provides no code generation, but testing is discussed in a dedicated guide. For deployment Sencha offers the JSBuilder a build tool written in Java, server communication is supported for JSON and XML.

Ext JS started out as a free commercially usable library but this is no more, open source applications under GPLv3 are still supported and encouraged, but for commercial usage a license needs to be purchased.

Score: 76%

qooxdoo

The qooxdoo SDK is a collection of Python scripts. They support application creation, developments builds – in case new files have been created – testing, and deployment.

Communication is supported via JSON and a JSON-RPC protocol that requires a specific backend preparation, which is described for popular backends like Rails, Java, Perl, Python, and PHP. qooxdoo allows free commercial development by selecting one of two licenses: LGPL or Eclipse Public License.

Score: 89%

SmartClient

No support for code generation, the documentation is specific about testing and deployment though: a short notice recommends an external testing tool, deployment should be done by manually copying the development folder to the production server. These advices do not qualify as developer tools.

Server and client can communicate via XML or JSON. The SmartClient is licensed commercially or under LGPL, but the quick start guide concludes with an encouragement to purchase a license; open source development does not seem to be welcome.

Score: 34%

SproutCore

SproutCore is the only framework that provides a scaffold[16] generator. It also has support for testing and deployment. Server communciation is restricted to JSON. SproutCore is free for commercial use.

Score: 93%

[16] Scaffolding refers to the creation of implementation and test files for a class and filling them with generic code. It is very prominent in Ruby on Rails.

Conclusion

The importance of support for proper developer tools seems to be quite low for many framework suppliers. Altough all frameworks offer some kind of free version, the only commercial candidate that really seems to embrace its open source option is Ext JS. The companies behind Bindows, DHTMLX, and SmartClient promote the free variant as possibility to evaluate, but it does not seems as if they were really keen on seeing a lot of applications implemented that way.

10.3 Results Summary

Following the idea of the decision process as explained in 4.3, the evaluation is a part of the decision process and delivers a suggestion. Therefore the result is not presented as a ranking with crisp numbers, but as a classification of the candidates into the three categories: *recommended*, *unsure*, and *not recommended*. The total scores can be obtained from table 10.17, all ratings are detailed in appendix B.

Classification	Candidate	Score
+	qooxdoo	82%
+	Ext JS	80%
+	SproutCore	71%
o	Dojo Toolkit	60%
o	SmartClient	53%
o	Cappuccino	45%
-	DHTMLX	41%
-	Bindows	35%

Table 10.17: Total scores and classification of evaluation candidates

10.3.1 Recommended: Ext JS, qooxdoo, SproutCore

Although qooxdoo received the most points, Ext JS feels slightly more professional and mature, probably thanks to the effort of its vibrant community. Still, both seem to be equally well suited for the development of single-page web applications. qooxdoo's strength lies in its coherent architecture, the straightforward getting started experience, and the open and dedicated development team; Ext JS offers an impressive documentation viewer, an extensive example suite, and it has a company in its background that relies on the functionality of the framework and therefore secures future investments.

SproutCore with its explicit domain of single-page applications can be recommended as a good choice too. Although it lacks some advanced UI

widgets and features, it seems to be built on a solid foundation and the fact that a major company like Apple relies on it for their consumer web applications is reassuring. Despite its young age it has a strong community, and the documentation is well maintained and focused on the desktop-like aspect.

Common to all three candidates is the embrace of innovation and the feeling that they are not trying to copy the desktop-experience but adapt it to the web.

10.3.2 Unsure: Dojo Toolkit, SmartClient, Cappuccino

The Dojo Toolkit has an incredibly large community and is mentioned articles, books, and all over the Internet. The functionality it provides is impressive as well, but it is by its nature more a library than a framework. The high modularization may provide benefits for computational resource demand and loading time, but it makes it difficult to present a consistent structure. This has repercussions in the software architecture and in the documentation. For single-page web applications the Dojo Toolkit might not be the best choice.

Almost in contrast the SmartClient Ajax Platform offers solid and structured functionality and documentation, but lacks a significant community. It seems to be mainly targeted at projects that are willing to use the SmartClient Server, documentation for other back-ends is scarce, teams in large corporations who have the duty of implementing a platform-independent version of a native software, as screenshots and documentation suggest. Innovative or customized user interface concepts or catering to the open-source software community does not seem to be in the interest of Isomorphic Software.

Despite its relatively low score Cappuccino is not seen as a framework that should be not recommended. Although the evaluation results list several features and UI widgets as not available, most of them are present in the framework, this is proved by the real life applications. However, Cappuccino has to mature and strongly improve its documentation. It is founded on the proven architecture of the Cocoa framework, but it is not certain that this architecture is also a good choice for web applications. In addition, towards the end of the evaluation it has been discovered that the Wiki of the Github repository hosts a huge amount of information. An earlier discovery would not have altered the result, since it is clearly stated that the project site is seen as the central hub, and links from the project site to the Wiki do not exist in this case, nonetheless it shows that Cappuccino has a potential underneath that may lead to strong improvements in the near future.

10.3.3 Not Recommended: Bindows, DHTMLX

In regard to architecture and structure of documentation DHTMLX is quite similar to the Dojo Toolkit. However, it has no community to compensate for

this predicament. Its character of a widget library and the lack of an overall cohesion of the components make it inappropriate for the development of single-page web applications.

Bindows may be technologically advanced as touted on the project site or not, the presentation by MB Technologies and the information that Bindows is surrounded with severly lacks quality, shown by the confusing getting started experience, lack of proper documentation, and the non-existing community. The fact that Bindows is proud of its replication of the look and feel of an outdated operating system speaks for itself in regard to attitude towards innovation and progress. Bindows can not be recommended for single-page web application development.

Chapter 11

Validation

11.1 Description of the Validation Process

To validate evaluation model with criteria and weights that led to the results reported in chapter 10, a small web application with the character of a prototype has been implemented.

The implementation process itself can serve as a validation because it touches several categories of the model: getting started for the obvious reason of starting development, look-ups in the documentation to get an understanding of the needed facts, searches for answers in the community to get information not present in the documentation, and development setting for the implementation process itself. For the prototype itself the feature and user interface category were most important, although not all criteria were covered.

11.2 Prototype Design

As an example application that serves at least some useful purpose while still being quite minimal in regard to its feature set and implementation cost a program to compare ratios was conceived.

It shows a list with named ratios (e.g. the golden ratio or the ratio of a modern TV screen) in a sidebar on the left and the visualization in the form of colored, semi-opaque rectangles on the right side. The tool does not compare sizes, only ratios, this means that the ratios 1:2 and 2:4 are represented by rectangles of the same size.

Additional features are: expanding the list with new ratios through an entry form, changing the visibility of ratios in the list, zooming of the rectangle view. A UI mockup is shown in figure 11.1. The distinction between the widgets can easily be seen; the main parts of the interface are: the list, the form, and a custom visualization widget.

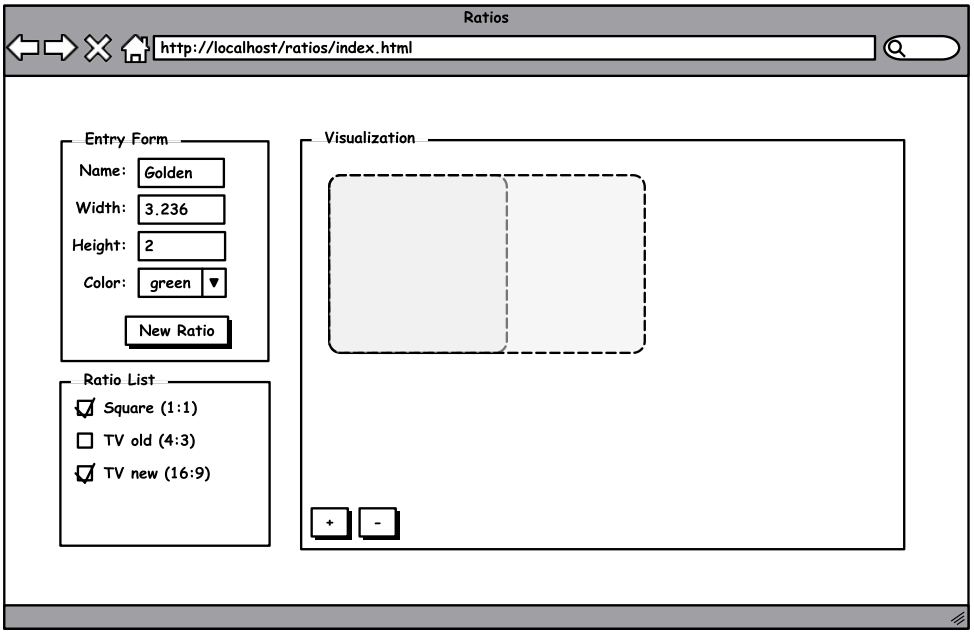


Figure 11.1: Mockup of the arrangement of the user interface elements for the ratios application

Deducting from the visual components the architecture is quite simple: in addition to the widgets a model is needed, the visual representations are then connected to the model via bindings, the form does not need a binding. The controllers responsible for hooking up models and views are not represented in the architecture (figure 11.2).

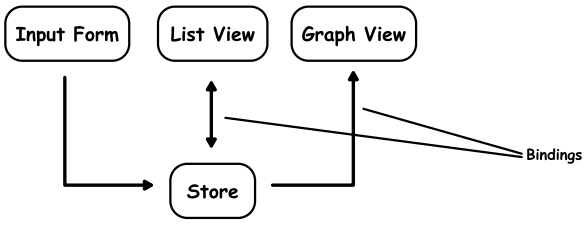


Figure 11.2: Coarse grained architecture for the ratios application

Within the model a single class is needed `Ratio` a list of its members is shown in table 11.1.

11.3 Framework Selection

The described design is based on the MVC pattern and it should be possible to implement it in any of the frameworks. If this were a real life situation in the current stage the best performing candidates would be selected and their

Ratio	
Name	*String*
Width	*Number*
Height	*Number*
Color	*Color*
Visible	*Boolean*

Table 11.1: Ratios has a single model class

aptitude tested by *prototyping* or *pilot testing* 4.3. As described in chapter 5, this work, however, needs only one selected candidate to validate the model. From the recommended frameworks qooxdoo is selected. Generally being equal to Ext JS, its more permissive license scheme seem more attractive.

11.4 Prototype Implementation

11.4.1 Getting started

To get started with the development process the easiest way was chosen: clicking *getting started* on the project web site. This presents the extensive *getting started* tutorial and a link to the *hello world* tutorial which presents a quick overview for all necessary steps ranging from app creation to deployment. It is very informative and helps to get a sense for the whole process. Explanation of the development tools, logging and debugging, helps to get a sense for the workflow for finding errors later on in the implementation process.

The app created in the *hello world* tutorial is just a template app and can also serve for the start of the *getting started* tutorial. The mockup of the app to be created in the tutorial (figure 11.3) shares important aspects with the mockup of the prototype app: it too contains a list, an input area and an add button. So it was decided to name the app “ratios” and not “twitter” as suggested, because it could be expected that the *getting started* app could be iteratively transformed into the prototype.

The first objectives were adding a window, assigning a layout to it, and filling it with widgets. These instructions were consequently accompanied with links that referenced detailed information in the API or the manual. When following the links and skimming over the provided information, a solid understanding of the layout system and the basic properties of the used widgets can already be built up at this early stage. In addition, while accessing the documentation regularly a new developer automatically gets a feeling for the documentation structure and where to find which information.

The tutorial continued by adding behaviour to the UI. The event system is

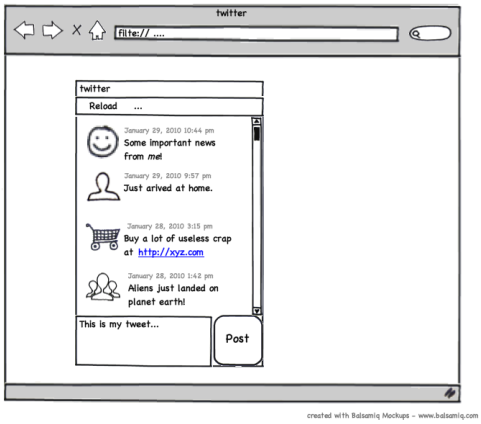


Figure 11.3: The mockup of the qooxdoo getting started example shares some properties with the mockup of the prototype (Source: qooxdoo website, getting started tutorial)

explained and this new knowledge directly brought to use by implementing the toggling of the enabled state of the button "post" depending on content in the text area.

In its third part the tutorial adds communication with the Twitter service. Again the developer is guided with links to more information about properties of objects, the binding system, and models in qooxdoo. Lastly the class responsible for communication and the list are connected via bindings.

After two and a half hours of learning time, significant knowledge has been built up and the application already communicates with a third-party service (see figure 11.4).

Conclusion: the experience shows that the getting started experience has not been underestimated in the evaluation model, it may be even ranked higher. The constant linking to more in-depth information leads to a good understanding not only of the linked information itself, but also of the structure of the documentation, which is the most important factor.

11.4.2 Transforming the *getting started* app

There is no interest in posting something to Twitter with the prototype – which would concede the next part of the tutorial – so it is decided to start moving the implementation in the direction of the *ratios* design.

At this stage first problems arise, because the model was not as good documented as initially thought. In fact the whole data store concept is not as evolved as it seemed during the evaluation: the central point is that qooxdoo does not explicitly require the creation of model classes. Instead they are created during run-time from a prototype object. This might be comfortable but it is very different from concepts in other frameworks where every complex

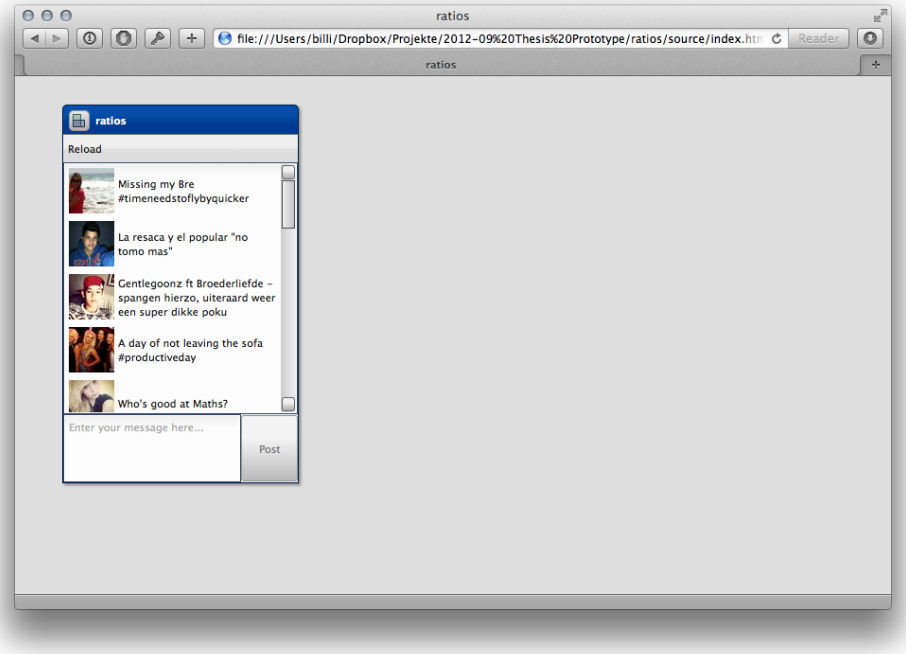


Figure 11.4: The *getting started* application after two and a half hours.

class needs explicit definition. Altough this does not present a current problem, it may lead to complications later on, when some advanced functionality of a model class is desired. To continue the store was rewritten as a service that transforms a prototype object into a model class that is then presented in the list. The sole member of the model class at this time is `name`.

With the help of the event system and the knowledge gained about layouts the UI is transformed to be able to alter the list by adding and removing names.

The next step was to add the visualization window and connect it to the model. The experience here is in line with the experience above, as problems with the binding system came up. Although there are suggestions in the documentation on how to find errors if bindings do not work as they should, it seems as if such a basic setup as the current (array is an object property binded to an external array, the object is listening to changes in this array) should have a guidance of some kind.

As mentioned in the evaluation model description bindings are a central concept of the MVC architecture for web applications and so they should be documented in more detail. Some examples that use bindings can be found in the example collection but they are not labeled specifically as binding examples but serve as examples for other concepts or widgets. Searching on the forum did not yield any results either.

The problem has been solved by listening for changes to the object property itself and then setting up the listenting for changes within the property.

The state of the application at the end of this stage is depicted in figures 11.5 and 11.6.

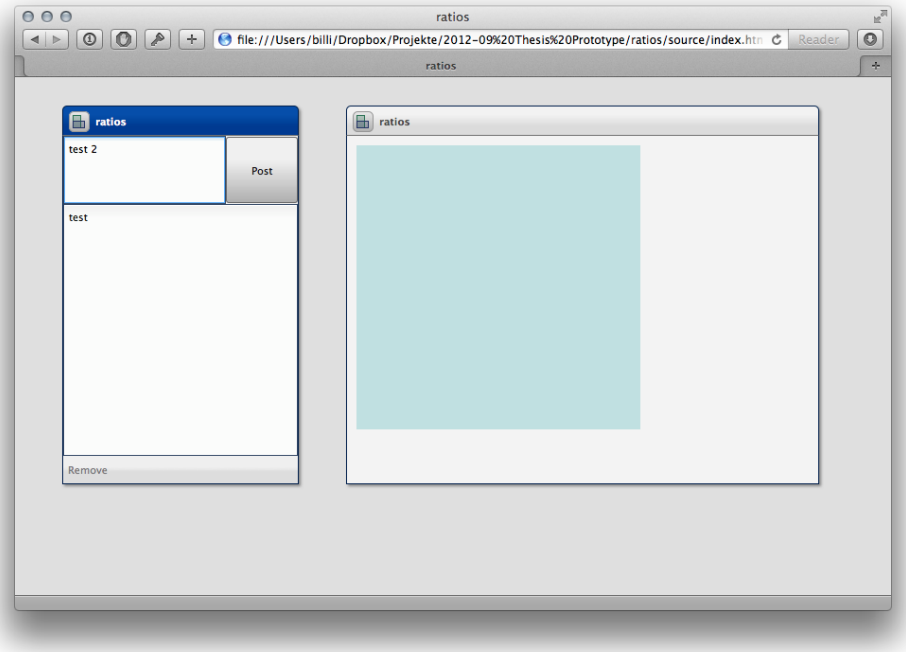


Figure 11.5: The application can alter the list and bindings with the vizualization widget work

Conclusion: As the application and implementation aspects move away from the *getting started* tutorial the first problems occur due to missing documentation. It seems that the assessment of it should be refined, the lack of results for this question in the community may signify that this category was also rated incorrectly. Another possibility, due to the simple nature of the problem, is that is simply stemmed from a personal misunderstanding.

On the other hand, the creation of the customized widget was easy and smooth, because of the documentation provided and because of good example cases. For these criteria the evaluation model seems adequate.

11.4.3 Improving the input possibilities

To add usable functionality to the application the next step was exchange the text area for some form fields. In qooxdoo's example collection several examples for the usage of forms are available so the implementation is straightforward. Validation is very easy to add, with the help of the documentation and the example code (figure 11.7).

However, at this stage the aforementioned problem with models comes up. In "ratios" the model generates a class from a prototype and the form

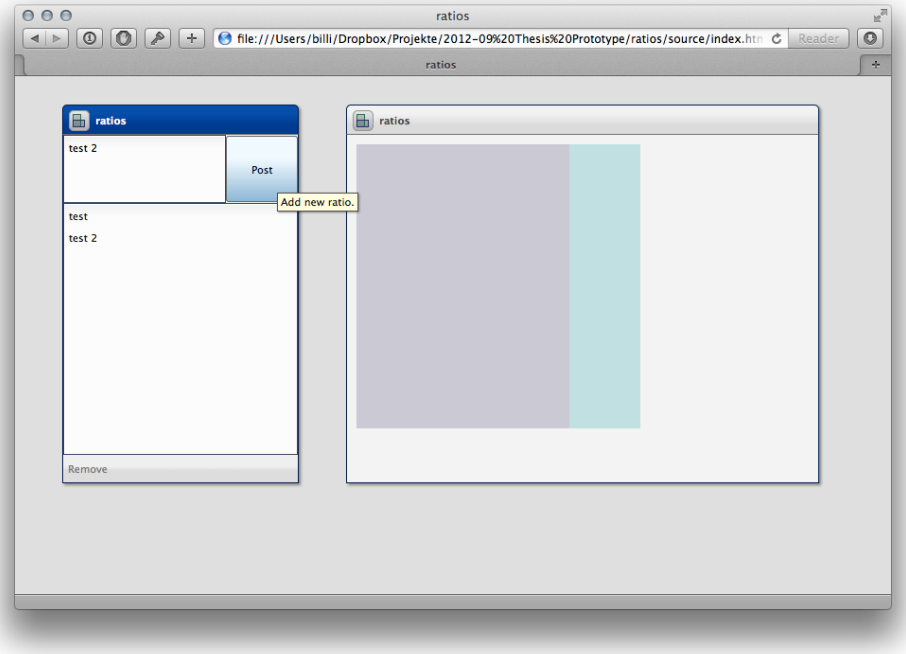


Figure 11.6: The application can alter the list and bindings with the visualization widget work

generates another class from the form fields. The interesting detail here is that these two classes only differ by capitalization of its members, otherwise they are identical, but the model class from the form can not be given an attribute where it supports the "bubbling up" of change events, a property necessary for bindings to work. The documentation is very limited in this case. A thorough explanation of models or an example that shows how to best handle a case where a customized model class is needed, are missing.

Conclusion: The evaluation model should be revised so that documentation of core functionality is inspected more deeply.

11.4.4 Refinements

The last stage of the implementation process brought mixed results as well. The goal is to refine the display of ratio names in the list, so that the width and the height visible, to change the two window appearance to a "normal" appearance by putting both parts on the main background, and to provide a zoom function.

Zooming was done with the event system and easy and straightforward. So was the change of the UI with the help of examples and the understanding of the layout gained in the *getting started* tutorial.

The problems were again connected with bindings, in this case the addi-

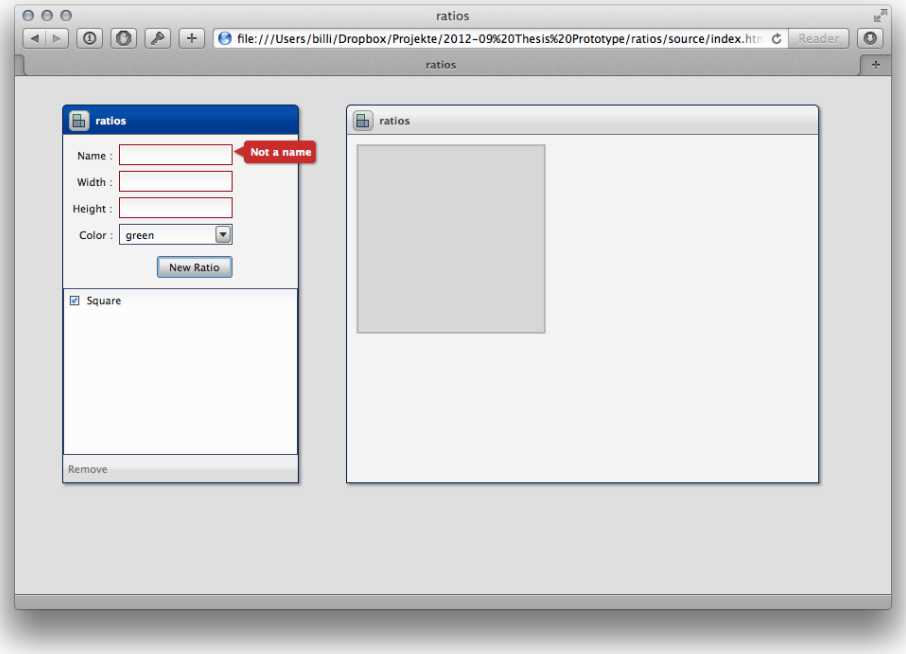


Figure 11.7: Form validation works very well, translating the form model to the model used in the rest of the application proved difficult

tional involvement of a controller complicated the situation. Complex widgets like a list often have a delegate that is responsible for their management, qooxdoo offers such delegates in the form of controllers. However, this concept is not explained in detail in the documentation. The *list controller* offers an option to change a value coming from the model before it is displayed. This is done with a custom converter. For using this converter the API lists some parameters that the converter is provided when called, and says that a controller is required for a certain parameter to be provided. Since the concept of controller is not completely clear, this problem could not be solved in a clean way. The finished prototype is shown in figure 11.8.

11.5 Conclusions

It is appropriate and convenient to learn solutions for problems that are closely connected with widgets (arrangement, configuration, special behavior and the like) from small code examples. However, central concepts like data store, model, bindings, controllers, etc. need explicit guides. The evaluation model does not differ between those two categories and delivers misleading results in this regard, the emphasis on examples should be increased.

It was also only learned while implementing that the model of qooxdoo

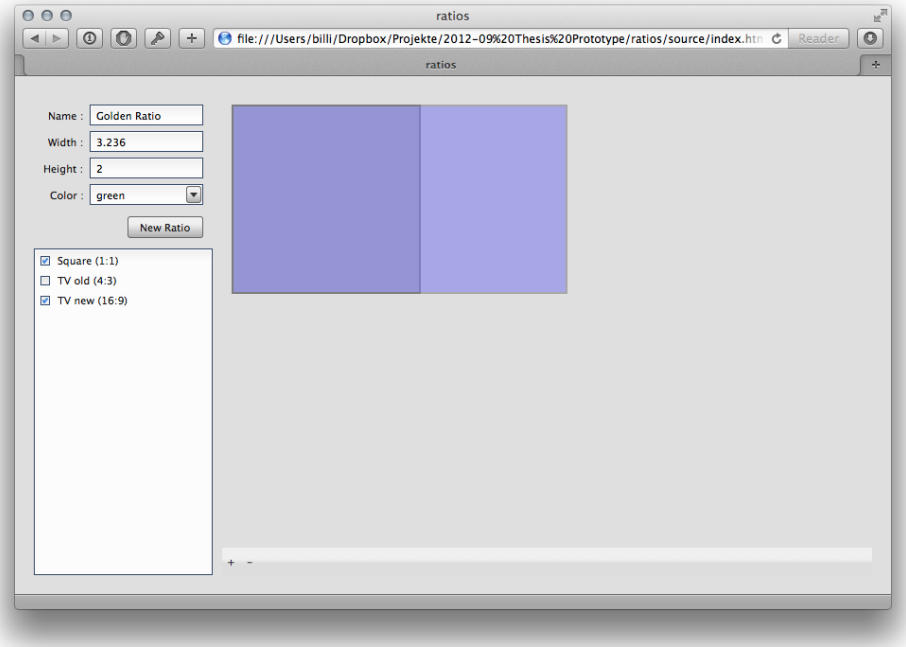


Figure 11.8: The finished prototype

is limited and does not know about relations or is not able to post to a server. This is an essential concept and the evaluation model should put more emphasis on this kind of functionality.

It is also advisable to limit the number of candidates for the evaluation phase. It seems doubtful to increase resources by adding personal to the evaluation as this impedes the comparability of results. When the same person sees all candidates the results should be more constant.

Very positively noted are the build tools, during development they helped in finding syntax errors and the deployment option is very important for good performance. This could also be more emphasised in the evaluation model.

Superficial features like tool tips play only a minor role and if a solid understanding of the framework is achieved they can be easily implemented by oneself. Then again the probability is high, that someone in the community already did that. This is some part of the evaluation model that could be deemphasized.

Chapter 12

Conclusion

This thesis presented a methodology to find suitable candidate frameworks for the implementation of desktop-like web applications. The developed decision process (chapter 5) includes the creation of an extensive evaluation model (chapter 6), a market research, and an evaluation with progressive filtering. The evaluation model can be used and improved in future evaluations, the market research provides an overview on the current web application framework landscape, and the evaluation yields recommendations for suitable frameworks.

41 frameworks could be found (chapter 7). After a screening stage (chapter 9) 8 of them have been scrutinized in detail (chapter 10), and three were classified as *recommended for desktop-like/single-page web application development*. These three are: qooxdoo, Ext JS, and SproutCore.

The concluding validation (chapter 11) showed that the quality of the result is satisfying. However, suggestions for improvement do exist. The evaluation model is in large parts adequate for its purpose, but model and process are not a good fit, for the amount of criteria is extensive and the filtering stage left too many candidates in the process.

The proposed solution is to split the evaluation stage. The process would then include two stages between the screening and the evaluation. These intermediate stages should be used to further reduce the number of candidates by finding suitable criteria and applying them to the candidate set.

Suitable criteria should be derived from the project that has to be implemented with the help of the framework. For communication intensive applications it would be recommended to rate the data store and binding system, for applications with an innovative user interface support for drag & drop or the flexibility of the widget system might be more important. Focus on a specific sub-set of criteria in this stage should allow an in-depth judgement without exceeding personal resources.

A reduction of the number of candidates for the evaluation stage would

allow a more precise assessment of two important criteria: the coverage of guide topics and the quality of guides (*doc.1* and *doc.2*). The results of the validation recommend an improvement in this area.

Another category with weaknesses is *community*, here yielding sources for books and articles have to be found (*com.3* and *com.4*).

All other categories seem well balanced. However, the model generally emphasizes information surrounding the framework code (documentation, communication, presentation) and is less focused on its inner workings. Altough this focus is supported by literature it may still be worthwhile to look at metrics like performance for further improvement.

It would also be interesting to see the effects of different evaluation methods on the resulting numbers and classification. Outranking methods and AHP are valid alternatives to the Weighted Scoring Method used here. If they yield clearly differing results this might also help in improving the model.

Bibliography

Alves, Carina and Anthony Finkelstein [2002]. *Challenges in COTS decision-making: a goal-driven requirements engineering perspective*. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 789–794. SEKE '02, ACM, New York, NY, USA. ISBN 1-58113-556-4. doi:10.1145/568760.568894. http://doi.acm.org/10.1145/568760.568894. (Cited on page 26.)

Anttonen, Matti, Arto Salminen, Tommi Mikkonen, and Antero Taivalsaari [2011]. *Transforming the web into a real application platform: new technologies, emerging trends and missing pieces*. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 800–807. SAC '11, ACM, New York, NY, USA. ISBN 978-1-4503-0113-8. doi:10.1145/1982185.1982357. http://doi.acm.org/10.1145/1982185.1982357. (Cited on pages 5 and 6.)

Apple Inc. [2010]. *Cocoa Fundamentals Guide*. Apple Inc. (Cited on pages 12, 14 and 15.)

Atkinson, Steven [1997]. *Cognitive deficiencies in software library design*. In *Software Engineering Conference, 1997. Asia Pacific ... and International Computer Science Conference 1997. APSEC '97 and ICSC '97. Proceedings*, pages 354 –363. doi:10.1109/APSEC.1997.640192. (Cited on page 33.)

Björemo, Martin and Predrag Trninic [2010]. *Evaluation of web application frameworks*. Master's Thesis, Chalmers University of Technology. (Cited on page 23.)

Bouyssou, Denis [2009]. *Outranking Methods*. In Floudas, Christodoulos A. and Panos M. Pardalos (Editors), *Encyclopedia of Optimization*, pages 2887–2893. Springer US. ISBN 978-0-387-74759-0. http://dx.doi.org/10.1007/978-0-387-74759-0_495. 10.1007/978-0-387-74759-0_495. (Cited on page 27.)

Brownsword, L., T. Oberndorf, and C.A. Sledge [2000]. *Developing new processes for COTS-based systems*. *Software, IEEE*, 17(4), pages 48–55. (Cited on pages 17 and 22.)

Brugali, Davide, Giuseppe Menga, and Amund Aarsten [1997]. *The Framework Life Span*. *Commun. ACM*, 40(10), pages 65–68. ISSN 0001-0782. doi:10.1145/262793.262806. http://doi.acm.org/10.1145/262793.262806. (Cited on page 13.)

Burbeck, Steve [1992]. *Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC)*. Technical Report. (Cited on page 37.)

Calefato, Fabio and Filippo Lanubile [2009]. *Using frameworks to develop a distributed conferencing system: an experience report*. *Softw: Pract. Exper.*, 39(15), pages 1293–1311. doi:10.1002/spe.937. http://dx.doi.org/10.1002/spe.937. (Cited on pages 14 and 21.)

Changpil, Lee [2012]. *An Evaluation Model for Application Development Frameworks for Web Applications*. Master's Thesis, Graduate School of The Ohio State University. (Cited on pages 17 and 24.)

Chung, Lawrence and Julio do Prado Leite [2009]. *On Non-Functional Requirements in Software Engineering*. In Borgida, Alexander, Vinay Chaudhri, Paolo Giorgini, and Eric Yu (Editors), *Conceptual Modeling: Foundations and Applications*, *Lecture Notes in Computer Science*, volume 5600, pages 363–379. Springer Berlin / Heidelberg. ISBN 978-3-642-02462-7. http://dx.doi.org/10.1007/978-3-642-02463-4_19. 10.1007/978-3-642-02463-4_19. (Cited on page 22.)

Colombo, Enzo and Chiara Francalanci [2004]. *Selecting CRM packages based on architectural, functional, and cost requirements: Empirical validation of a hierarchical ranking model*. *Requirements Engineering*, 9, pages 186–203. ISSN 0947-3602. http://dx.doi.org/10.1007/s00766-003-0184-y. 10.1007/s00766-003-0184-y. (Cited on pages 23 and a.)

Comella-Dorda, S., J.C. Dean, E. Morris, and P. Oberndorf [2002]. *A Process for COTS Software Product Evaluation*. In *COTS-based software systems: First International Conference, ICCBSS 2002, Orlando, FL, USA, February 4-6, 2002: proceedings*, volume 2255, page 86. Springer-Verlag New York Inc. (Cited on pages 21 and 22.)

Deng, Hepu [1999]. *Multicriteria analysis with fuzzy pairwise comparison*. In *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99. 1999 IEEE International*, volume 2, pages 726 –731 vol.2. ISSN 1098-7584. doi:10.1109/FUZZY.1999.793038. (Cited on page 28.)

Duhl, Joshua [2003]. *White paper: Rich internet applications*. Technical Report, IDC. (Cited on page 6.)

Fayad, Mohamed and Douglas C. Schmidt [1997]. *Object-oriented application frameworks*. *Commun. ACM*, 40(10), pages 32–38. ISSN 0001-0782. doi:10.1145/262793.262798. http://dx.doi.org/10.1145/262793.262798. (Cited on page 13.)

Fayad, Mohamed E., David S. Hamu, and Davide Brugali [2000]. *Enterprise frameworks characteristics, criteria, and challenges*. *Commun. ACM*, 43(10), pages 39–46. ISSN 0001-0782. doi:10.1145/352183.352200. http://doi.acm.org/10.1145/352183.352200. (Cited on pages 33 and 34.)

Fayad, Mohamed E., Douglas C. Schmidt, and Ralph E. Johnson [1999]. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. 1 Edition. John Wiley & Sons. ISBN 0471248754. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0471248754. (Cited on pages 11, 12, 13, 14, 33 and 35.)

Foley, Mary-Jo [2011]. *Microsoft: Our strategy with Silverlight has shifted*. http://www.zdnet.com/blog/microsoft/microsoft-our-strategy-with-silverlight-has-shifted/7834. (Cited on page 7.)

Froehlich, Garry, Amr Kamel, and Paul Sorenson [2000]. *Exploring O-O framework usage (poster session)*. In *Proceedings of the 22nd international conference on Software engineering*, pages 783–. ICSE '00, ACM, New York, NY, USA. ISBN 1-58113-206-9. doi:10.1145/337180.337639. http://doi.acm.org/10.1145/337180.337639. (Cited on pages 32 and 33.)

Gamma, Erich, Richard Helm, Ralph E. Johnson, and John Vlissides [1994]. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional. (Cited on pages 12, 13 and 15.)

Gerdessen, Anton [2007]. *Framework comparison method*. Master's Thesis, University of Amsterdam. (Cited on page 17.)

Gizas, Andreas, Sotiris Christodoulou, and Theodore Papatheodorou [2012]. *Comparative evaluation of javascript frameworks*. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 513–514. WWW '12 Companion, ACM, New York, NY, USA. ISBN 978-1-4503-1230-1. doi:10.1145/2187980.2188103. http://doi.acm.org/10.1145/2187980.2188103. (Cited on page 24.)

Güngör Şen, Ceyda and Hayri Baraçli [2006]. *A Brief Literature Review of Enterprise Software Evaluation and Selection Methodologies: A Comparison in the Context of Decision-Making Methods*. In *5th International Symposium on Intelligent Manufacturing Systems*, pages pp.874–883. (Cited on pages 19 and 21.)

Hou, Daqing, K. Wong, and H.J. Hoover [2005]. *What can programmer questions tell us about frameworks?* In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on*, pages 87 – 96. ISSN 1092-8138. doi:10.1109/WPC.2005.47. (Cited on pages 33 and 35.)

Ignacio Fernández-Villamor, José, Laura Díaz-Casillas, and Carlos Á. Iglesias [2008]. *A comparison model for agile web frameworks*. In *Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*, pages 14:1–14:8. EATIS '08, ACM, New York, NY, USA. ISBN 978-1-59593-988-3. doi:10.1145/1621087.1621101. http://doi.acm.org/10.1145/1621087.1621101. (Cited on pages 24, 36 and a.)

Jadhav, Anil S. and Rajendra M. Sonar [2009]. *Evaluating and selecting software packages: A review*. *Information and Software Technology*, 51(3), pages 555 – 563. ISSN 0950-5849. doi:10.1016/j.infsof.2008.09.003. http://www.sciencedirect.com/science/article/pii/S0950584908001262. (Cited on pages 19, 20, 21, 22, 25 and 26.)

Janisch, Gerhard [2008]. *Analyse von Rich Internet Application Frameworks am Beispiel einer Thesaurusverwaltung*. Master's Thesis, FH Hagenberg. (Cited on pages 33 and 40.)

Johnson, Ralph E. and Brian Foote [1988]. *Designing Reusable Classes*. *Journal of Object-Oriented Programming*, 1(2), pages 22–35. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.8594. (Cited on pages 11, 12, 13, 15 and 33.)

Kirk, Douglas Samuel [2005]. *Understanding Object-Oriented Frameworks*. PhD Thesis, University of Strathclyde, Glasgow. (Cited on page 34.)

Kitchenham, B., S. Linkman, and D. Law [1997]. *DESMET: a methodology for evaluating software engineering methods and tools*. *Computing & Control Engineering Journal*, 8(3), pages 120–126. (Cited on page 21.)

Kizzort, B. [2002]. *Selection of components for OTS component-based systems*. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 6, pages 6–2651 – 6–2659 vol.6. doi:10.1109/AERO.2002.1036106. (Cited on pages 18 and 35.)

Kontio, J. [1996]. *A case study in applying a systematic method for COTS selection*. In *Software Engineering, 1996., Proceedings of the 18th International Conference on*, pages 201 –209. doi:10.1109/ICSE.1996.493416. (Cited on pages 22, 23 and a.)

Kontio, Jyrki, Gianluigi Caldiera, and Victor R. Basili [1996]. *Defining factors, goals and criteria for reusable component evaluation*. In *Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research*, pages 21–. CASCON '96, IBM Press. http://dl.acm.org/citation.cfm?id=782052.782073. (Cited on page 22.)

Kontio, Jyrki, Show-Fune Chen, Kevin Limperos, Roseanne Tesoriero, Gianluigi Caldiera, and Mike Deutsch [1995]. *A COTS Selection Method and Experiences of Its Use*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.20.7769. (Cited on page 20.)

Korson, T. and J.D. McGregor [1992]. *Technical criteria for the specification and evaluation of object-oriented libraries*. *Software Engineering Journal*, 7(2), pages 85 –94. ISSN 0268-6961. (Cited on pages 14, 23, 34, 35 and a.)

Kunda, Douglas [2003]. *STACE: Social Technical Approach to COTS Software Evaluation*, chapter 5, pages 64–84. Lecture Notes in Computer Science, Springer, Berlin. http://www.springerlink.com/content/6kyyjwew7hbadrwe. (Cited on pages 20, 26 and 27.)

Laakso, Tuukka and Joni Niemi [2008]. *An evaluation of AJAX-enabled java-based web application frameworks*. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, pages 431–437. MoMM '08, ACM, New York, NY, USA. ISBN 978-1-60558-269-6. doi:10.1145/1497185.1497278. http://doi.acm.org/10.1145/1497185.1497278. (Cited on pages 24, 25, 35, 36 and a.)

Lawlis, Patricia K., Kathryn E. Mark, Deborah A. Thomas, and Terry Courtheyn [2001]. *A Formal Process for Evaluating COTS Software Products*. *Computer*, 34, pages 58–63. ISSN 0018-9162. doi:10.1109/2.920613. http://dl.acm.org/citation.cfm?id=619063.621716. (Cited on page 20.)

Lee, Hsuan-Shih, Pei-Di Shen, and Wen-Li Chih [2004]. *A fuzzy multiple criteria decision making model for software selection*. In *Fuzzy Systems, 2004. Proceedings. 2004 IEEE International Conference on*, volume 3, pages 1709 – 1713 vol.3. ISSN 1098-7584. doi:10.1109/FUZZY.2004.1375439. (Cited on page 28.)

Li, Jingyue, R. Conradi, C. Bunse, M. Torchiano, O. Slyngstad, and M. Morisio [2009]. *Development with Off-the-Shelf Components: 10 Facts*. *Software, IEEE*, 26(2), pages 80 –87. ISSN 0740-7459. doi:10.1109/MS.2009.33. (Cited on page 19.)

Microsoft [2012]. *About Native XMLHTTP*. http://msdn.microsoft.com/en-US/en-us/library/ms537505.aspx. (Cited on page 7.)

Mohamed, Abdallah, Guenther Ruhe, and Armin Eberlein [2007]. *COTS Selection: Past, Present, and Future*. In *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, pages 103 –114. doi:10.1109/ECBS.2007.28. (Cited on page 21.)

Morisio, M. and A. Tsoukias [1997]. *IusWare: a methodology for the evaluation and selection of software products*. *Software Engineering. IEE Proceedings- [see also Software, IEE Proceedings]*, 144(3), pages 162 – 174. ISSN 1364-5080. doi:10.1049/ip-sen:19971350. (Cited on pages 26 and 27.)

Mousseau, V. and R. Slowinski [1998]. *Inferring an ELECTRE TRI Model from Assignment Examples*. *Journal of Global Optimization*, 12, pages 157–174. ISSN 0925-5001. http://dx.doi.org/10.1023/A:1008210427517. 10.1023/A:1008210427517. (Cited on pages 27 and 28.)

Ncube, C. and N.A.M. Maiden [1999]. *PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm*. In *International Workshop on Component-Based Software Engineering*, page 1. (Cited on page 20.)

Ncube, Cornelius and John Dean [2002]. *The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components*. In Dean, John and Andrée Gravel (Editors), *COTS-Based Software Systems*, *Lecture Notes in Computer Science*, volume 2255, pages 176–187. Springer Berlin / Heidelberg. ISBN 978-3-540-43100-8. http://dx.doi.org/10.1007/3-540-45588-4_17. 10.1007/3-540-45588-4_17. (Cited on page 26.)

Oberndorf, Patricia, Lisa Brownsword, Ed Morris, and Carol Sledge [1997]. *Workshop on COTS-Based Systems*. Technical Report, Defense Technical Information Center OAI-PMH Repository (United States). (Cited on pages 19 and 21.)

Ros, Jordi Canals [2011]. *Introduction to Decision Deck-Diviz: Examples and User Guide*. The Decision Deck Project. http://www.decision-deck.org/diviz/tutorials.html. (Cited on page 27.)

Roy, Bernard [1996]. *Multicriteria Methodology for Decision Aiding, Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht. (Cited on page 25.)

Saaty, Thomas L. [1990]. *How to make a decision: The analytic hierarchy process*. *European Journal of Operational Research*, 48(1), pages 9 – 26. ISSN 0377-2217. doi:10.1016/0377-2217(90)90057-I. http://www.sciencedirect.com/science/article/pii/

0377221790900571. ¡ce:title¿Desicion making by the analytic hierarchy process: Theory and applications¡/ce:title¿. (Cited on pages 25 and 27.)

Shull, Forrest, Filippo Lanubile, and Victor R. Basili [2000]. *Investigating Reading Techniques for Object-Oriented Framework Learning*. *IEEE Trans. Softw. Eng.*, 26(11), pages 1101–1118. ISSN 0098-5589. doi:10.1109/32.881720. http://dx.doi.org/10.1109/32.881720. (Cited on page 34.)

Taivalsaari, A., T. Mikkonen, D. Ingalls, and K. Palacz [2008]. *Web Browser as an Application Platform*. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pages 293 –302. ISSN 1089-6503. doi:10.1109/SEAA.2008.17. (Cited on page 6.)

Triantaphyllou, Evangelos [2000]. *Multi-Criteria Decision Making Methods: A Comparative Study*. Springer. (Cited on page 25.)

Vigder, Mark, W. M. Gentleman, and John Dean [1996]. *COTS Software Integration: State of the Art*. Technical Report. (Cited on page 18.)

Windrum, Paul [2004]. *Leveraging technological externalities in complex technologies: Microsoft's exploitation of standards in the browser wars*. *Research Policy*, 33(3), pages 385 – 394. ISSN 0048-7333. doi:10.1016/j.respol.2003.09.002. http://www.sciencedirect.com/science/article/pii/S0048733303001434. (Cited on page 3.)

Winokur, Danny []. *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5*. http://blogs.adobe.com/conversations/2011/11/flash-focus.html. (Cited on page 7.)

Wirfs-Brock, Rebecca J. and Ralph E. Johnson [1990]. *Surveying current research in object-oriented design*. *Commun. ACM*, 33(9), pages 104–124. ISSN 0001-0782. doi:10.1145/83880.84526. http://doi.acm.org/10.1145/83880.84526. (Cited on page 13.)

Yoon, K. Paul and Ching-Lai Hwang [1995]. *Multiple Attribute Decision Making*. SAGE Publications. (Cited on page 25.)

Appendix A

Framework List

The following pages list all candidates found during the market research described in chapter 7 (page 43).

	ActiveJS	Agility.js	AmplifyJS
URL	activejs.or	agilityjs.com	amplify.js
Version Number	n/a	0.1.2	1.1.0
Version Date	26/10/2010	17/12/2011	08/11/2011
Examination Date	19/06/2012	19/06/2012	26/06/2012
short description	ActiveJS is a JavaScript application framework that provides local and REST based data modeling and pure DOM view construction with back button and history support. • No external dependencies • Does not modify built in objects • Exports only 5 globals • Framework agnostic, designed to be used with Prototype, jQuery, etc	Agility.js is an MVC library for Javascript that lets you write maintainable and reusable browser code without the verbose or infrastructural overhead found in other MVC libraries. The goal is to enable developers to write web apps at least as quickly as with jQuery, while simplifying long-term maintainability through MVC objects.	AmplifyJS is a set of components designed to solve common web application problems with a simplistic API. Amplify's goal is to simplify all forms of data handling by providing a unified API for various data sources. Amplify's store component handles persistent client-side storage, using standards like localStorage and sessionStorage, but falling back on non-standard implementations for older browsers.
dependency		jQuery	
accepted	abandoned	no	no framework
is framework		yes	
application domain			
OS independent		☑	
server agnostic		☑	
MVC		☑	
HTML/CSS abstraction		☐	
Layout Manager		☐	
Widget Library		☐	
basic documentation			
API reference		☑	
tutorials		☑	
example applications		☑	
first steps		☑	
installation instructions		☐	
Comment			

	AngularJS	AppJS	Backbone.js
URL	angularjs.org	appjs.org	backbonejs.org
Version Number	1.0.0	0.0.11	0.9.2
Version Date	13/06/2012	15/06/2012	21/03/2012
Examination Date	19/06/2012	19/06/2012	19/06/2012
short description	HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.	Build Desktop Applications for Linux, Windows and Mac using HTML, CSS and Javascript	Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.
dependency			underscore.js, json2.js, jQuery/ Zepto
accepted	no framework	no	no
is framework			yes
application domain			web applications
OS independent			☑
server agnostic			☑
MVC			☑
HTML/CSS abstraction			☐
Layout Manager			☐
Widget Library			☐
basic documentation			
API reference			☑
tutorials			☑
example applications			☑
first steps			☑
installation instructions			☑
Comment	different approach: enhancing HTML, no framework	says: „Attention: AppJS is under heavy development. API changes a lot until we bump version to v0.1.0". Too young for serious development, but very promising	

	batman.js	Bindows	Cappuccino
URL	batmanjs.org	bindows.net	cappuccino.org
Version Number	0.9.0	4.1.1	0.9.5
Version Date	02/04/2012	16/05/2012	16/11/2011
Examination Date	19/06/2012	19/06/2012	19/06/2012
short description	Batman.js is a framework for building rich web applications with CoffeeScript or JavaScript. App code is concise and declarative, thanks to a powerful system of view bindings and observable properties. The API is designed with developer and designer happiness as its first priority.	Bindows is a Software Development Kit (SDK) for writing robust and secure Rich Internet Applications. The Bindows platform provides rich functionality for thin Web clients. Bindows applications require no end-user downloads - true zero-footprint (no Java, Flash, plug-ins or ActiveX are used).	Cappuccino is an open source framework that makes it easy to build desktop-caliber applications that run in a web browser.
dependency	node.js		
accepted	no	yes	yes
is framework	yes	yes	yes
application domain	rich web applications	rich internet applications	desktop-caliber web applications
OS independent	☑	☑	☑
server agnostic	☑	☑	☑
MVC	☑	☑	☑
HTML/CSS abstraction	☑	☑	☑
Layout Manager	☐	☑	☑
Widget Library	☐	☑	☑
basic documentation			
API reference	☑	☑	☑
tutorials	☑	☑	☑
example applications	☑	☑	☑
first steps	☑	☑	☑
installation instructions	☑	☑	☑
Comment			

	Choco	CorMVC	DHTMLX
URL	github.com/ahe/choc	bennadel.com/projects /cormvc-jquery-framework.htm	dhtmlx.com
Version Number	n/a	n/a	3.0
Version Date	07/09/2010	21/12/2009	31/10/2011
Examination Date	19/06/2012	19/06/2012	26/06/2012
short description	A delicious Javascript web framework made in Belgium! Choco brings the MVC to the client side! You like Javascript and you want to develop rich internet applications? You also know that HTML & CSS are powerful? Cappuccino & Sproutcore don't feel like web development anymore? Thanks to Choco, you'll be able to easily develop maintainable web applications. A Choco app consists of only one HTML page, all the interactions are managed by Javascript. Your UI only uses HTML and CSS!	CorMVC is a jQuery-powered Model-View-Controller (MVC) framework that can aide in the development of single-page, web-based applications. CorMVC stands for client-only-required model-view-controller and is designed to be lowest possible entry point to learning about single-page application architecture. It does not presuppose any server-side technologies, or a web server of any kind, and requires no more than a web browser to get up and running.	dhtmlxSuite is a rich JavaScript library that delivers a complete set of UI components
dependency			
accepted	abandoned	abandoned	yes
is framework			yes
application domain			professional web apps
OS independent			☑
server agnostic			☑
MVC			☑
HTML/CSS abstraction			☑
Layout Manager			☑
Widget Library			☑
basic documentation			
API reference			☑
tutorials			☑
example applications			☑
first steps			☑
installation instructions			☑
Comment			• specialized for .NET, PHP and Java back-ends

	Dojo Toolkit	Eyeballs	Ember.js
URL	dojotoolkit.org	github.com/paulca/eyeballs.js	emberjs.com
Version Number	1.7	0.5.17	0.9.8.1
Version Date	16/02/2012	23/06/2011	22/05/2012
Examination Date	19/06/2012	19/06/2012	19/06/2012
short description	Dojo saves you time and scales with your development process, using web standards as its platform. It's the toolkit experienced developers turn to for building high quality desktop and mobile web applications. From simple websites to large packaged enterprise applications whether desktop or mobile, Dojo will meet your needs.	eyeballs.js is a slim javascript library designed to sit on top of a javascript framework, such as jQuery or Prototype.	a framework for creating ambitious web applications
dependency	node.js/Rhino		
accepted	yes	abandoned	no
is framework	yes		yes
application domain	simple website to large enterprise app		ambitious web applications
OS independent	☑		☑
server agnostic	☑		☑
MVC	☑		☑
HTML/CSS abstraction	☑		☑
Layout Manager	☑		☐
Widget Library	☑		☐
basic documentation			
API reference	☑		☑
tutorials	☑		☑
example applications	☑		☐
first steps	☑		☑
installation instructions	☑		☑
Comment			was SproutCore 2.0

	Ext JS	Glow	Google Closure Library
URL	sencha.com/products/extj	bbc.co.uk /glow	developers.google.com/closure
Version Number	4.1.0	1.7.7	r1376
Version Date	20/04/2012	07/07/2011	10/11/2011
Examination Date	25/06/2012	19/06/2012	22/06/2012
short description	Ext JS 4 is the next major advancement in our JavaScript framework. Featuring expanded functionality, plugin-free charting, and a new MVC architecture it's the best Ext JS web application development platform yet. Develop incredible web apps for every browser.	Glow is a JavaScript library which gives you... • Simplified DOM manipulation, event handling, animations, etc • A versatile set of user interface widgets • Clear and comprehensive documentation • BBC Browser Support Standards compliance	The Closure tools help developers to build rich web applications with web development tools that are both powerful and efficient.
dependency			
accepted	yes	abandoned	no
is framework	yes		yes
application domain	cross-platform applications		
OS independent	☑		☑
server agnostic	☑		☑
MVC	☑		☑
HTML/CSS abstraction	☑		☑
Layout Manager	☑		☐
Widget Library	☑		☑
basic documentation			
API reference	☑		☑
tutorials	☑		☑
example applications	☑		☑
first steps	☑		☑
installation instructions	☑		☑
Comment	commercial	last Github commit on 30/06/2011 (on forks just one commit 'removed white space' on Oct)	

	JavaScriptMVC	jQuery	Knockback.js
URL	javascriptmvc.com	jquery.com	kmalakoff.github.com /knockback
Version Number	3.2.2	1.7.2	0.15.3
Version Date	25/01/2012	21/03/2012	02/06/2012
Examination Date	22/06/2012	22/06/2012	22/06/2012
short description	JavaScriptMVC is an open-source framework containing the best ideas in jQuery development. It guides you to successfully completed projects by promoting best practices, maintainability, and convention over configuration.	jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.	brings Knockout.js magic to Backbone.js
dependency	jQueryMX, StealJS, FuncUnit, DocumentJS		knockout.js, backbone.js, underscor
accepted	no	no framework	no
is framework	yes		yes
application domain	rich web applications		web applications
OS independent	☑		☑
server agnostic	☑		☑
MVC	☑		☑
HTML/CSS abstraction	☐		☐
Layout Manager	☐		☐
Widget Library	☐		☐
basic documentation			
API reference	☐		☑
tutorials	☑		☑
example applications	☑		☐
first steps	☑		☑
installation instructions	☑		☐
Comment		probably most popular JS library, offers UI widgets, but not under a single hood	no HTML/CSS abstraction, initial commit: 01/11/2011

	Knockout.js	Luna (Asana)	MooTools
URL	knockoutjs.com	asana.com/luna	mootools.net
Version Number	2.1.0	n/a	1.4.5
Version Date	07/05/2012		26/02/2012
Examination Date	22/06/2012	19/06/2012	22/06/2012
short description	Simplify dynamic JavaScript UIs by applying the Model-View-View Model (MVVM) pattern	When writing a complex, highly-reponsive web application, there are all kinds of really difficult programming tasks that you end up doing over and over again for every feature you want to write. [...] So we built Luna, an in-house end-to-end framework that automates the busy work of writing rich web applications to an unprecedented degree.	MooTools is a compact, modular, Object-Oriented JavaScript framework designed for the intermediate to advanced JavaScript developer. It allows you to write powerful, flexible, and cross-browser code with its elegant, well documented, and coherent API.
dependency	e.js		
accepted	no	no	no
is framework	yes	n/a	yes
application domain	dynamic JavaScript UIs		web applications
OS independent	☑		☑
server agnostic	☑		☑
MVC	☐		☑
HTML/CSS abstraction	☐		☐
Layout Manager	☐		☐
Widget Library	☐		☐
basic documentation			
API reference	☑		☑
tutorials	☑		☑
example applications	☐		☑
first steps	☑		☐
installation instructions	☑		☐
Comment		proprietary and in-house only, no further information available	

	MochiKit	Mojito (Yahoo!)	Prototype
URL	mochikit.com	developer.yahoo.com /cocktails/mojito	prototypejs.org
Version Number	1.4	0.3.27	1.7
Version Date	27/11/2008	15/06/2012	16/11/2010
Examination Date	22/06/2012	22/06/2012	22/06/2012
short description	MochiKit is a highly documented and well tested, suite of JavaScript libraries that will help you get shit done, fast. We took all the good ideas we could find from our Python, Objective-C, etc. experience and adapted it to the crazy world of JavaScript.	Mojito is a sweet (and minty!) MVC application framework built on YUI 3 that enables agile development of Web applications. Mojito allows developers to write client and server components in the same language (JavaScript), using the same framework. Because Mojito applications are written in JavaScript, they can run on the client (in the browser) and on the server (with Node.js). In addition, Mojito has built-in support for internationalization, testing, and building documentation.	Prototype is a JavaScript Framework that aims to ease development of dynamic web applications.
dependency		YUI 3, node.js	
accepted	no framework	no	no framework
is framework		yes	
application domain		MCV web applications	
OS independent		☑	
server agnostic		☑	
MVC		☑	
HTML/CSS abstraction		☐	
Layout Manager		☐	
Widget Library		☑	
basic documentation			
API reference		☐	
tutorials		☑	
example applications		☑	
first steps		☑	
installation instructions		☑	
Comment	titles itself 'feature complete'	Not mature enough, initial commit on 30/03/2012	

	qooxdoo	Rialto	Rico
URL	qooxdoo.org	rialto.improve-technologies.com/	openrico.org
Version Number	2.0	1.1.5	2.1
Version Date	21/06/2012	10/04/2012	03/05/2009
Examination Date	25/06/2012	25/06/2012	25/06/2012
short description	Create desktop oriented applications. Features a rich and extendable set of widgets. No HTML/CSS knowledge required. • Features • Windows, Tabs, … • Forms, Lists, Trees, … • Toolbars, Menus, … • Layouting • Theming	Rialto (R ich I nternet A pplication T oolkit) is ajax based cross browser javascript widgets library. Because it is technology agnostic it can be encapsulated in JSP, JSF, .Net, Python or PHP graphic components. The purpose of Rialto is to ease the access to rich internet application development to corporate developers. Ideally a Rialto developer have neither need to write or understand DHTML, Ajax or DOM code.	
dependency	node.js/Rhino		
accepted	yes	no	abandoned
is framework	yes	yes	
application domain	single page applications	corporate web applications	
OS independent	☑	☑	
server agnostic	☑	☑	
MVC	☑	☐	
HTML/CSS abstraction	☑	☑	
Layout Manager	☑	☐	
Widget Library	☑	☑	
basic documentation			
API reference	☑	☑	
tutorials	☑	☑	
example applications	☑	☑	
first steps	☑	☑	
installation instructions	☑	☑	
Comment		Does not provide model or controller classes.	

	Sammy.js	script.aculo.us	smartclient
URL	sammyjs.org	script.aculo.us	smartclient.com
Version Number	0.7.1	1.9.0	8.2
Version Date	21/01/2012	23/12/2010	05/12/2011
Examination Date	25/06/2012	25/06/2012	25/06/2012
short description	a small framework with class	script.aculo.us provides you with easy-to-use, cross-browser user interface JavaScript libraries to make your web sites and web applications fly.	SmartClient provides an open DHTML/Ajax client engine, rich user interface components, and metadata-driven client-server databinding systems, for rich GUI, zero-install web applications.
dependency	jQuery		Java
accepted	no	no framework	yes
is framework	yes		yes
application domain	JavaScript applications		business web applications
OS independent	☑		☑
server agnostic	☑		☑
MVC	☐		☑
HTML/CSS abstraction	☐		☑
Layout Manager	☐		☑
Widget Library	☑		☑
basic documentation			
API reference	☑		☑
tutorials	☑		☑
example applications	☑		☑
first steps	☑		☑
installation instructions	☑		☑
Comment	from the documentation: 'It does this without the overhead of a large base framework or a single method or system for building models or views. If you want something more fully featured I highly suggest checking out SproutCore or Cappucino.'		commercial

	Spine.js	SproutCore	Spry
URL	spinejs.com	sproutcore.com	labs.adobe.com/technologies/spry
Version Number	1.0.8	1.8.2	1.6.1
Version Date	06/06/2012	10/05/2012	25/02/2008
Examination Date	25/06/2012	25/06/2012	25/06/2012
short description	Spine is a lightweight framework for building JavaScript web applications. Spine gives you an MVC structure and then gets out of your way, allowing you to concentrate on the fun stuff, building awesome web applications.	SproutCore is an open-source framework for building blazingly fast, innovative user experiences on the web.	Spry is a JavaScript-based framework that enables the rapid development of Ajax-powered web pages. Not a JavaScript guru? No problem. Spry was designed to feel like an extension of HTML and CSS, so anyone with basic web-production skills can create next-generation web experiences by adding the power of Ajax to their pages.
dependency	node.js	Ruby	
accepted	no	yes	abandoned
is framework	yes	yes	
application domain	JavaScript MVC applications	fully functioning applications	
OS independent	☑	☑	
server agnostic	☑	☑	
MVC	☑	☐	
HTML/CSS abstraction	☑	☑	
Layout Manager	☐	☑	
Widget Library	☐	☑	
basic documentation			
API reference	☑	☑	
tutorials	☑	☑	
example applications	☑	☑	
first steps	☑	☑	
installation instructions	☑	☑	
Comment			

	UIZE	underscore.js	Wakanda
URL	uize.com	underscorejs.org	wakanda.org
Version Number	n/a	1.3.3	1.0
Version Date	n/a (but regular commits to github)	10/04/2012	15/03/2012
Examination Date	25/06/2012	25/06/2012	25/06/2012
short description	develop dazzling web sites with rich client side interactivity	Underscore is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Prototype.js (or Ruby), but without extending any of the built-in JavaScript objects. It's the tie to go along with jQuery's tux, and Backbone.js's suspenders.	One open and complete solution for all your Web and mobile business apps.
dependency			
accepted	no	no framework	no
is framework	yes		yes
application domain	dazzling web sites		business apps
OS independent	☑		☐
server agnostic	☑		☐
MVC	☑		☐
HTML/CSS abstraction	☑		☑
Layout Manager	☐		☐
Widget Library	☑		☑
basic documentation			
API reference	☑		☑
tutorials	☑		☑
example applications	☑		☑
first steps	☑		☑
installation instructions	☑		☑
Comment			• development on Linux is not possible, wakanda studio is available only for Mac and Windows. • it is not server agnostic since it depends on the wakanda server as back-end

	YUI!	zepto.js
URL	yuilibrary.com	zeptojs.com
Version Number	3.5.1	1.0rc1
Version Date	04/05/2012	09/04/2012
Examination Date	25/06/2012	25/06/2012
short description	YUI is a library of JavaScript utilities and controls for building richly interactive web applications using techniques such as DOM Scripting, DHTML, and Ajax.	Zepto is a minimalist JavaScript library for modern browsers with a largely jQuery-compatible API. If you use jQuery, you already know how to use Zepto.
dependency		
accepted	yes	no framework
is framework	yes	
application domain	single-page JavaScript applications	
OS independent	☑	
server agnostic	☑	
MVC	☑	
HTML/CSS abstraction	☑	
Layout Manager	☐	
Widget Library	☑	
basic documentation		
API reference	☑	
tutorials	☑	
example applications	☑	
first steps	☑	
installation instructions	☑	
Comment	YUI offers an extensive array of functionality including a widget library and a component named _App Framework_ that is specifically targeted at web applications. After some in-depth research the conclusion is that version 2 of the framework had a layout manager, by transitioning to version 3 it was eliminated though. It may be introduced again, the _App Framework_ is in beta status, but the description text of the app framework states that "If you've used DocumentCloud's excellent Backbone.js framework, many of the classes and APIs provided by App Framework components will look familiar to you". Backbone.js has already been classified as unsuitable.	

Appendix B

Detailed Evaluation Result

Criterion	Weight	Bindows	Cappuccino	DHTMLX	Dojo Toolkit
URL		bindows.net	cappuccino.org	dhtmlx.com/docs/products /dhtmlxSuite/index.shtml	dojotoolkit.org /features/desktop
Version Number		04.01.0001	0.9.5	3.0	1.7
Version Date		16.05.2012	16.11.2011	31.10.2011	16.02.2012
Twitter name		n/a	cappuccino	dhtmlx	dojo
Blog URL		blog.bindows.net	cappuccino.org/discuss	dhtmlx.com/blog	dojotoolkit.org/blog
Repository		self-hosted	Github	self-hosted	Github
Examination Date		Jul/Aug 2012	Jul/Aug 2012	Jul/Aug 2012	Jul/Aug 2012
forum URL		forum.bindows.net	groups.google.com /forum/?fromgroups# !forum/objectivej	forum.dhtmlx.com /viewforum.php?f=2	dojotoolkit.org/community
short description		Bindows is a Software Development Kit (SDK) for writing robust and secure Rich Internet Applications. The Bindows platform provides rich functionality for thin Web clients. Bindows applications require no end-user downloads - true zero-footprint (no Java, Flash, plug-ins or ActiveX are used).	Cappuccino is an open source framework that makes it easy to build desktop-caliber applications that run in a web browser.	dhtmlxSuite is a rich JavaScript library that delivers a complete set of UI components	Dojo saves you time and scales with your development process, using web standards as its platform. It's the toolkit experienced developers turn to for building high quality desktop and mobile web applications. From simple websites to large packaged enterprise applications whether desktop or mobile, Dojo will meet your needs.
Getting started	**10 %**	38 %	63 %	35 %	18 %
overview	**20 %**	25 %	75 %	75 %	25 %
straightforward start	**30 %**	25 %	75 %	50 %	25 %
get started document	**50 %**	50 %	50 %	10 %	10 %
Documentation	**25 %**	37 %	37 %	28 %	29 %
guides: coverage	**20 %**	50 %	25 %	75 %	50 %
guides: quality	**20 %**	50 %	50 %	25 %	25 %
API ref	**20 %**	25 %	50 %	10 %	25 %
examples	**15 %**	10 %	10 %	10 %	10 %
new version doc	**5 %**	50 %	50 %	50 %	75 %
up-to-dateness transp	**5 %**	10 %	10 %	10 %	50 %
overall structure	**15 %**	50 %	50 %	10 %	10 %
Community	**25 %**	11 %	48 %	39 %	85 %
books	**25 %**	25 %	50 %	25 %	100 %
refs on heise.de	**5 %**	0 %	0 %	0 %	100 %
refs on Hacker News	**5 %**	10 %	100 %	25 %	100 %
refs on Quora	**5 %**	0 %	75 %	10 %	100 %
Google hits	**5 %**	10 %	75 %	50 %	100 %
tagged on StackOverflow	**10 %**	0 %	25 %	10 %	75 %
forum/mailinglist activity	**20 %**	5 %	50 %	100 %	100 %
tweets in last year	**10 %**	n/a	50 %	50 %	100 %
blog posts in last year	**10 %**	10 %	10 %	25 %	25 %
real life apps on web site	**5 %**	10 %	90 %	0 %	0 %
Features	**20 %**	50 %	33 %	45 %	60 %
data store	**15 %**	0 %	0 %	50 %	50 %
data bindings	**15 %**	10 %	50 %	50 %	50 %
drag & drop	**10 %**	100 %	100 %	10 %	100 %
i18n/l10n	**10 %**	100 %	0 %	10 %	100 %
theming	**10 %**	100 %	100 %	25 %	100 %
form validation	**10 %**	50 %	0 %	100 %	100 %
context menu	**5 %**	100 %	0 %	100 %	0 %
tool tips	**5 %**	75 %	100 %	0 %	100 %
keyboard shortcuts	**5 %**	100 %	0 %	0 %	0 %
offline mode	**5 %**	0 %	0 %	100 %	0 %
server push	**5 %**	0 %	0 %	100 %	0 %
history management	**5 %**	0 %	0 %	0 %	0 %
User Interface	**10 %**	80 %	60 %	90 %	90 %
standard widgets	**60 %**	100 %	100 %	100 %	100 %
rich text	**10 %**	0 %	0 %	100 %	100 %
charts	**10 %**	100 %	0 %	100 %	100 %
date picker	**10 %**	100 %	0 %	100 %	100 %
map	**10 %**	0 %	0 %	0 %	0 %
Development Setting	**10 %**	14 %	45 %	28 %	89 %
code generation	**10 %**	0 %	0 %	0 %	0 %
testing	**30 %**	0 %	0 %	0 %	100 %
deployment	**20 %**	0 %	100 %	0 %	100 %
server communication	**15 %**	90 %	0 %	100 %	90 %
license	**25 %**	0 %	100 %	50 %	100 %
Total Weighted Score		35,11 %	44,50 %	40,90 %	60,16 %

Criterion	Weight	Ext JS	qooxdoo	SmartClient Ajax Platform	SproutCore
URL		sencha.com/products/extj	qooxdoo.org	smartclient.com /product/smartclient.jsp	sproutcore.com
Version Number		4.1.0	2.0.1	8.2	01.08.0002
Version Date		20.04.2012	03.07.2012	05.12.2011	10.05.2012
Twitter name		Sencha	qooxdoo	n/a	SproutCore
Blog URL		sencha.com/blog/category/extjs	news.qooxdoo.org	blog.smartclient.com	blog.sproutcore.com
Repository		self-hosted	sourceforge	self-hosted	Github
Examination Date		Jul/Aug 2012	Jul/Aug 2012	Jul/Aug 2012	Jul/Aug 2012
forum URL		sencha.com/forum /forumdisplay.php ?6-Ext-Open-Discussion	qooxdoo.org/forum	forums.smartclient.com /forumdisplay.php?f=13	groups.google.com /forum/?fromgroups# !forum/sproutcore
short description		Ext JS 4 is the next major advancement in our JavaScript framework. Featuring expanded functionality, plugin-free charting, and a new MVC architecture it's the best Ext JS web application development platform yet. Develop incredible web apps for every browser.	Create desktop oriented applications. Features a rich and extendable set of widgets. No HTML/CSS knowledge required. • Features • Windows, Tabs, … • Forms, Lists, Trees, … • Toolbars, Menus, … • Layouting • Theming	SmartClient provides an open DHTML/Ajax client engine, rich user interface components, and metadata-driven client-server databinding systems, for rich GUI, zero-install web applications.	SproutCore is an open-source framework for building blazingly fast, innovative user experiences on the web.
Getting started	**10 %**	83 %	90 %	62 %	82 %
overview	**20 %**	75 %	90 %	35 %	50 %
straightforward start	**30 %**	75 %	90 %	75 %	90 %
get started document	**50 %**	90 %	90 %	65 %	90 %
Documentation	**25 %**	84 %	84 %	56 %	73 %
guides: coverage	**20 %**	90 %	90 %	75 %	75 %
guides: quality	**20 %**	75 %	75 %	50 %	90 %
API ref	**20 %**	90 %	90 %	25 %	50 %
examples	**15 %**	90 %	90 %	90 %	75 %
new version doc	**5 %**	90 %	90 %	50 %	50 %
up-to-dateness transp	**5 %**	25 %	75 %	50 %	90 %
overall structure	**15 %**	90 %	75 %	50 %	75 %
Community	**25 %**	74 %	67 %	23 %	55 %
books	**25 %**	100 %	100 %	0 %	75 %
refs on heise.de	**5 %**	25 %	100 %	0 %	25 %
refs on Hacker News	**5 %**	50 %	25 %	10 %	100 %
refs on Quora	**5 %**	90 %	10 %	10 %	75 %
Google hits	**5 %**	90 %	25 %	10 %	25 %
tagged on StackOverflow	**10 %**	100 %	25 %	10 %	25 %
forum/mailinglist activity	**20 %**	50 %	75 %	75 %	25 %
tweets in last year	**10 %**	n/a	50 %	n/a	75 %
blog posts in last year	**10 %**	50 %	100 %	25 %	50 %
real life apps on web site	**5 %**	75 %	25 %	10 %	90 %
Features	**20 %**	78 %	90 %	76 %	78 %
data store	**15 %**	100 %	100 %	100 %	100 %
data bindings	**15 %**	50 %	100 %	100 %	100 %
drag & drop	**10 %**	100 %	100 %	75 %	90 %
i18n/l10n	**10 %**	100 %	100 %	100 %	90 %
theming	**10 %**	100 %	100 %	100 %	100 %
form validation	**10 %**	50 %	100 %	50 %	90 %
context menu	**5 %**	50 %	100 %	50 %	0 %
tool tips	**5 %**	100 %	50 %	0 %	75 %
keyboard shortcuts	**5 %**	100 %	100 %	100 %	75 %
offline mode	**5 %**	100 %	100 %	50 %	0 %
server push	**5 %**	0 %	0 %	25 %	0 %
history management	**5 %**	50 %	50 %	50 %	75 %
User Interface	**10 %**	90 %	80 %	90 %	60 %
standard widgets	**60 %**	100 %	100 %	100 %	100 %
rich text	**10 %**	100 %	100 %	100 %	0 %
charts	**10 %**	100 %	0 %	100 %	0 %
date picker	**10 %**	100 %	100 %	100 %	0 %
map	**10 %**	0 %	0 %	0 %	0 %
Development Setting	**10 %**	76 %	89 %	34 %	93 %
code generation	**10 %**	0 %	50 %	0 %	100 %
testing	**30 %**	100 %	100 %	0 %	100 %
deployment	**20 %**	100 %	100 %	0 %	100 %
server communication	**15 %**	90 %	60 %	90 %	50 %
license	**25 %**	50 %	100 %	80 %	100 %
Total Weighted Score		79,76 %	81,59 %	53,49 %	70,85 %

List of Figures

List of Tables

d

Acknowledgements

Ich bedanke mich ganz herzlich bei Martin, meinem Betreuer, für seine Geduld, das Durchhaltevermögen und seine Unterstützung nicht nur bei dieser Arbeit, sondern auch in anderen Teilen meines Studiums.

Keineswegs selbstverständlich ist die Selbstverständlichkeit mit der meine Eltern, Andrea und Karli, immer hinter mir stehen und ständig alles geben um meine oft zweifelhaften Vorhaben zu ermöglichen.

Genauso kann ich, schon länger als es mir möglich ist mich zu erinnern, mich auf alle anderen in meiner Familie verlassen. Ganz besonders meine Schwester Sophie, die Schwestern meiner Mutter Christa, Dorothea und Anna und meine Oma Margarethe haben mich bedingungslos unterstützt.

Gerade recht zeitig um diese jüngste schwierige Zeit täglich teilen und ertragen zu dürfen, trat eine besondere Frau in mein Leben. Dir, Lisa, blieb im letzten Jahr wohl wenig erspart. Trotzdem haben Deine Entbehrungen, Nachsicht und Liebe nicht gereicht. Du musstest auch noch Deinen verdienten Urlaub opfern; ich hoffe, das einmal wieder gut machen zu können. Du bringst Freude in mein Leben, ich liebe Dich.

Zu guter Letzt nicht völlig unbeteiligt waren meine guten Freunde. Rami, Leo, Gernot, Erhard, Nozi, Stefan, Luki, Sandro, Tase, Max, Fokker, Flo. Sie waren und sind immer gerne bereit Ablenkung und Aufheiterung zu spenden.

Vielen Dank Euch allen, ohne Euch
hätte ich es nicht geschafft,

Thomas und Billi

f

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Place/Ort	Date/Datum	Signature/Unterschrift