

SaaS Interface Design

Designing web-based software for business purposes

By

M.T. Hoogvliet

Date

July 4th 2008

Email

info@mthoogvliet.nl

Web

<http://www.mthoogvliet.nl>

© 2008 M.T. Hoogvliet

Description

Research into designing interfaces for SaaS (Software-as-a-Service) applications.

Synopsis

The key elements in designing an interface for a web-based software application delivered as a service consist of basic web-based software guidelines set up in this document, relating to establishing a productive environment on the web, on the one hand. The key to successfully transform the web in a productive environment is building an effective working space. In designing such an application one must emphasize on the user losing the web-behaviour he is used to; quick navigation. Let your web-based software application behave as the main object of attention. On the other hand these basic guidelines must be expanded with specific SaaS guidelines relating to call-to-action, overall application behaviour, and interface consistency. The foundations of a SaaS are the web-based software guidelines also applicable to standard or free web-based software. In addition to that the SaaS specific guidelines can make the difference in designing a profitable SaaS application.

Note: This document was originally submitted as part of a student thesis in Communication and Multimedia Design, Rotterdam University (HRO), The Netherlands.

Contents

1	Introduction	5
1.1	<i>Motivation & Relevance</i>	5
1.2	<i>Main Question</i>	8
1.3	<i>Goals</i>	8
1.4	<i>Methods</i>	8
2	Context and Definition	9
2.1	<i>Software as a Service</i>	9
2.1.1	SaaS: The business model	9
2.1.2	Types of SaaS	9
2.1.3	SaaS vs. ASP	10
2.1.4	How to define a SaaS application?	11
2.1.5	Arguments for using SaaS	12
2.2	<i>The web as a platform</i>	12
2.2.1	Web 2.0: A brief history	13
2.2.2	The RIA & web-based software	14
2.2.3	Web applications/Thin Clients vs. web-based software	15
2.2.4	Technologies.....	15
2.2.5	The benefits of web-based software	16
2.3	<i>Interaction Design</i>	17
2.4	<i>A user-centric design approach</i>	18
2.4.1	General user interface design principles	18
2.4.2	The benefits of usability	20
2.4.3	Usability testing	20
3	The Twilight Zone	22
3.1	<i>Desktop and Web clashing together</i>	22
3.2	<i>Desktop and Web vs. web-based</i>	22
3.3	<i>Desktop vs. SaaS</i>	24
3.4	<i>General web-based vs. SaaS</i>	25
4	Guidelines	27
4.1	<i>The evolution of usability</i>	27
4.2	<i>Web-based design considerations</i>	27
4.2.1	Alan Cooper	27
4.2.2	Luke Wroblewski	28
4.2.3	Chris Loosley	28
4.2.4	Jess McMullin & Grant Skinner	28
4.3	<i>Web-based design principles</i>	31
4.4	<i>SaaS design considerations</i>	33
4.5	<i>Web-based design principles enhanced for SaaS</i>	35
5	Conclusion	37

6 Bibliography	38
<i>Papers</i>	38
<i>Books</i>	38
<i>Articles</i>	39
<i>Websites</i>	39
Acknowledgments	40

“Stop thinking about the web as a collection of pages. What you’ve been calling a page is really a window in a niche of thought. Assume that your user will interact with your website as intimately as he might interact with a word processor. This implies, of course, that your website will boast software as rich and intricate in its algorithms as a word-processor. That’s a lot of work. But it’s the future.”

*Chris Crawford, 2003
The Art of Interactive Design*

1 Introduction

Once again, a new buzzword has risen: SaaS, short for Software as a Service. SaaS delivers software on the Internet, instead of on your own computer. But maybe that sounds a little abstract. To put it in a very catchy chant: “Are you sick of buying expensive software packages, now there is SaaS!”

More concrete; SaaS is a software business model. Software is not offered as a product, but as a service. Software runs on a webserver and the user uses it through an Internet connection. The user only pays for using the software instead of for owning it.

Desktop software and Internet applications are going through changes; the two are merging together. Desktop functionality and richness in interaction is now possible on the web. SaaS establishes a model to build a business on these advances. I will define SaaS more in chapter two.

The SaaS idea of using software on the internet lets you think about a fundamental change in how we use our computers nowadays; the complete integration of the Internet in your operating system. When all software runs on a webserver eventually you will stop needing the noisy, extra-large computer case below your desk. Never again will you encounter the infamous ‘blue-screen of death’ (that is, for the Windows users among us), because you simply won’t have any hardware at home except for your monitor and Internet connection. All you will have to do is log in to your personal online desktop. You will have no software installed; everything runs from a webserver. Automatically all your files and settings are also stored remotely.

At this time we are not speaking of a whole different approach to using software (yet). I don’t doubt that the operating systems of the future will be largely web-based, but for now SaaS is ‘just’ a business model which popularity is growing among software developers and IT companies. However, it takes the first steps into a growing area of digital business with a value highly potential.

1.1 Motivation & Relevance

“What if somebody put a word processor on the web? If you don’t have a word processor of your own, you just go to that website and type your document. (..) this may strike you as silly, after all, the web is so slow that you’d spend most time of your life waiting for the latest edited version of your document to come down the wire. (..) what happens when the internet is so fast that it can bounce back a page faster that you can type a single character?”¹

The Internet has evolved the last few years. From pure information to interaction with its users, to the users interacting with each other through it. Most users browse the Internet through a high-speed broadband connection nowadays. *Note: Broadband means a transmission of data at several megabits per second, opposed to the 28 and 56 kilobits per second modems of the nineties.*

¹ C. Crawford, The Art Of Interactive Design, 2003, p. 41.

In the US over 45% of Internet connections were high-speed broadband in 2004. In 2006 that number had grown to 65% and to 83% in august 2007.

Asian countries as South Korea are even further with a broadband penetration of 89% at the end of 2006. Europe is a proud third. In the Netherlands, for instance, the percentage was 70% at the end of 2006.²

“The combination of ever-more-abundant bandwidth, increasingly powerful processors, and inexpensive storage is broadening the choices for designing, deploying, and using software.”³

The fast connection speeds, the evolution of the Internet and the user’s growth in feeling of comfort in, and acceptance of, the digital world gives a lot of space for new digital business opportunities. We humans have shown a remarkable ability to absorb technological advances for centuries. The SaaS businessmodel and the Internet technologies that make SaaS possible are now part of those advances.

“Software as a service represented approximately 5 percent of business software revenue in 2005 and, by 2011, 25 percent of new business software will be delivered as SaaS”.⁴

“We believe eventually all business software will be offered as a service”⁵

“SaaS will definitely breakthrough in 2008. It will be used on a large scale and the benefits will become clearly visible.”⁶

“Worldwide 2,4 Billion Euros are spend on Software as a Service, an amount that is expected to grow with an average of 18 percent each year.”⁷

“One of the words most used in the ICT industry was Software-as-a-Service in 2007, it is expected that 2008 will also be a ‘SaaS’ year”⁸

“As SaaS gains mainstream acceptance, it is becoming an important disruptive force in the software industry”⁹

The quotes above illustrate that SaaS and web-based software is becoming quite a hot topic, both in the IT industry as a whole and among large companies. Microsoft and Adobe also see the benefits of the SaaS business model (which I will discuss later on in

² Websiteoptimisation.com, August 2007, <http://www.websiteoptimization.com/bw/0708/>

³ Microsoft Corp., 2008, <http://www.microsoft.com/serviceproviders>

⁴ Gartner, Inc, October 2006, <http://www.gartner.com/it/page.jsp?id=496886>

⁵ IBM, Saascon congress, 2006, source: M. de van der Schueren (Chairman ASP-Forum)

⁶ Retail Vista, SaaS, 2007 hype or (h)(n)ot?, 2007, p.1

⁷ P. Vermeulen, IDC Opinion “De business case voor Software as a Service”, 2006, p.1

⁸ Computable, March 2008, http://www.computable.nl/artikel/ict_topics/storage/2271254/1277017/saas-breekt-definitief-door-in-2008.html

⁹ Business Week, Software as a Service Myths, April 2006, p.2

this thesis). And, of course, it is important to develop while the world in which you operate develops.

The growing popularity of web-applications has revealed a lack of effective guidelines for their design and implementation. Luke Wroblewski defines the problem in his paper “Design considerations for web-based applications” as following:

- Existing guidelines for Web usability hinder web-based application usability since they are primarily based on interactions within a browsing metaphor.
- Interface design guidelines for client applications, on the other hand, do not address the conventions of web users, limitations of the web environment, nor the new possibilities that the web has to offer, such as AJAX and other web 2.0 technologies.

The guidelines for these two separated disciplines are used for multimedia products, which are a combination of, and a crossover between both. There is a grey area where the two are merging, where existing guidelines don’t apply anymore. The one can’t be used for the other just because we think we know how to apply them, or because a combination of two disciplines seems the same as one or the other.

Douglyss Giuliana describes the problem in the following metaphor: “You wouldn’t send a boxer in a karate match just because he said he knew how to kick, would you?”¹⁰

‘The user interface demands for Software as a Service (SaaS) are very distinct from enterprise applications and other software models.’¹¹

Within web-based software design we find the specialism of SaaS design. A SaaS is a web-based software product, but with specific properties (see chapter 2.1). Software delivered as a service over the Internet is becoming more and more popular and appealing to major, medium and small companies and there are few, if any standards. Designing and developing a SaaS application without clear guidelines is guessing what will work. That takes time, practice and perseverance. Many companies, especially smaller ones, don’t have the luxury to do that. If there isn’t a recipe, we need one. New guidelines for this specific area of web-based software need to be developed.

¹⁰ D. Giuliana, User Interface Design Explained, 2002, p.3

¹¹ Catalyst Resources, 2008, <http://www.catalystresources.com/saas/>

1.2 Main Question

Now we arrive at the central question of this thesis:

What are the key elements in designing an interface for a web-based software application delivered as a service?

In the chapters below I will answer the following parts of this main question:

How to define a SaaS application?

What are the key elements in designing general web-based software?

How does a SaaS application differ from general web-based software?

On what points does a SaaS application typically needs to appeal to a user?

1.3 Goals

SaaS and web-based software design falls in between the existing guidelines for designing interfaces because two, formerly separated, disciplines (Desktop & Web) are merging together.

In this thesis I will approach SaaS mainly from the application's user perspective. What do SaaS users need, want and what is most effective for them? After that I will translate this in an approach for designing interfaces for software delivered as a service. I will outline a set of guidelines for designing a web-based application within a SaaS business model to form a more solid foundation for interface design on this growing area of digital business. The guidelines will not be a strict rulebook to what works and what doesn't, it will be an advice and a set of considerations relating to web-based software design with an emphasis on deliverance as a service.

1.4 Methods

Proper web-based software design comes from an understanding of what makes web-based software different from standard websites and traditional desktop applications. Proper SaaS design comes from an understanding of what makes a SaaS application different from a standard or free web-based software application.

Thus, I will analyse the changes that led to this design dilemma. Miscellaneous aspects of literature will be compared to analyse interface design guidelines for web and desktop and how those are applicable to Web-based software design. I will combine them in an advice and a set of guidelines. For the answer to my main question I will analyse the few established web-based software design guidelines. From that, I will distillate and enhance the guidelines especially for web-based software delivered as a service.

2 Context and Definition

In the parts below I will define the terms that lie at the basis of the subject of Software as a Service and interface design. I address and explain the main parts to give a better understanding of the motivation and goals of this thesis.

2.1 Software as a Service

In response to the adoption and growing popularity of SaaS, Microsoft Chairman Bill Gates stated in an internal memo in 2006: *“This coming ‘services wave’ will be very disruptive. (...) Services designed to scale to tens or hundreds of millions will dramatically change the nature and cost of solutions delivered to enterprises or small business”*.

If even the richest man in the world worries about SaaS it must really be something else!

2.1.1 SaaS: The business model

SaaS should not be seen as a new form of building software or architecture solution. SaaS is a business model, which establishes a way of dealing with and delivering software other than the paved road of software supplying. SaaS is about delivering web-based software over the Internet, the user runs the application in a browser and only pays for using the software instead of for owning it.

The definition of SaaS according to Gartner reads:

“SaaS is software that is owned, delivered and managed remotely by one or more providers. The provider delivers an application based on a single set of common code and data definitions, which are consumed in a one-to-many model by all contracted customers, at any time, on a pay-for-use basis, or as a subscription based on usage metrics.”¹²

The time where clients paid lots of money for a software package seems to be over. The concept of SaaS avoids the known problems of desktop software; system compatibilities, installation difficulties, manual updating, etcetera. The customer-focused and customer-driven approach is appealing to customers and therefore also to companies.

2.1.2 Types of SaaS

There are two main types of SaaS: Hosted application management or hosted AM and Software on demand.

¹² Gartner Inc., IT Service Management, best practices, part 4, 2006 (from Mansystems, Servicemanagementapplicaties in het SaaS-concept, 2007)

Hosted AM contains standard applications hosted as a service. Clients acquire an application and host it with a service provider who is responsible for the use, maintenance, and management. The hosting can be done by the company who developed the application or by a third party. The following points are distinctive for Hosted AM:

- The application is hosted as a service and is installed and managed by a service provider.
- The application is financed by a traditional one-time acquisition of licenses and a continuing maintenance contract. The license stands apart from the hosting contract.
- Most contracts leave room for expanding the software with new parts. This additional configuration doesn't require new software code and improves flexibility.

With Software on demand the application, services and support are designed specifically for distribution via the Internet. Most software on demand suppliers embrace a web service strategy; all customers share the same public infrastructure. The following points are distinctive for Software on demand:

- The software is developed for network use and is only supplied via network. The application is never installed on the client's server.
- Licenses and hosting costs are combined in one returning periodic amount.
- There is little option for customizing applications for specific clients. Expanding the application is possible for as far as the suppliers configuration options extend.

Hosted AM has more in common with ASP, a hosting model of the nineties (see 2.3.3), because, in contrary to Software on demand, the power of the web is not fully used. The software is hosted as a service, not fully supplied as a service. Hosted AM seems to be a step between traditional software selling and Software on demand. Therefore the rich user experiences and architecture of web-based software in combination with a fully web-powered business model are only applicable in the Software on demand variation of the SaaS business model.

2.1.3 SaaS vs. ASP

Although SaaS is presented as a new business model that is not quite a fact. SaaS is another version of the failed Application Service provider, or ASP, and other hosting models of the past.

According to M. de van der Schueren (Chairman of the dutch ASP-forum) SaaS is just a marketing term; in fact, ASP and SaaS are basically the same¹³. The basic principle of hosting applications externally was re-branded because of the previously failed attempts. The important thing is that now the environment seems to be ready for, and reacting positively on, SaaS developments. So, why did the business model failed to be a success then and is now becoming a succes?

¹³ Computable, January 2007, http://www.computable.nl/artikel/ict_topics/loopbaan/1816677/1458016/asp-en-saas.html

ASP and hosting companies in the web 1.0 era failed for two reasons. First, they did not fundamentally change the architecture of their software (like we do with our now available web 2.0 Flash, AJAX and XML techniques, true genius!), but simply sold applications to organizations that did not want to house them on their own systems. The hosting costs proved to be too much for the ASPs to withstand. Second, only a small segment of the market was willing to outsource their application needs. They considered their business applications a strategic asset and preferred to have them under their own roof.

But times have changed; the boundaries are gone. Today's economic and competitive pressure makes nearly any form of outsourcing fair game. The evolution of the Internet and the central place it takes, also in business, make companies more flexible and open to the possibilities of the web. This, together with technical advances, decreased cost of investment and increased reliability, makes SaaS more attractive than ASP and hosting models of the past.

2.1.4 How to define a SaaS application?

In the broadest sense of the word a SaaS application is a web-based software application empowered by a SaaS business model. That means that if we only observe the application itself, it has little difference with a regular or free web-based software application. It is the business model that counts and what makes it SaaS. But because of that business model and the way a SaaS application is used and developed, design-wise there are substantial differences from a standard web-based software application. I will return to this in chapter four.

Gerard Blokdijk defines SaaS software in his book "SaaS, 100 Succes Secrets"¹⁴ according to four basic characteristics. The software should:

- Be multi-tenacity

The same programme must be able to be used by different customers, from single individuals to large corporations.

- Incorporate shared services

It should be able to link up to other services available online. For instance, when a SaaS company offers web statistics as a service, it should be able to access other programs that enable it to monitor the activities on the website, database and many others. The customer should be able to link services together as a whole.

- Have a feedback mechanism

To increase user friendliness the software should always have a built in feedback mechanism to help customers, and also the software itself, report problems or difficulties encountered while using the program.

- Be pay-as-you-use only service (the most important one and defining characteristic of SaaS)

¹⁴ G. Blokdijk, SaaS, 100 Succes Secrets, 2008, p.78

The program should not bind the customer to the service for a longer period of time. Suppose that the application does not please the user or its use is no longer needed, the client must be able to quit immediately without loss of investment.

Software as a service is a combination of the use of benefits of web-based software (discussed in 2.2.5) and the benefits of a more customer friendly and flexible business model of delivering that software.

2.1.5 Arguments for using SaaS

Business arguments for developing and offering SaaS as a company:

- SaaS is customer-focused and customer-friendly
- SaaS can be easily and frequently updated, bugs can be fixed and customers are assured that they constantly get the latest version
- Improved customer support, service and feedback
- Software is quicker and easier to market
- More attention can be given to developing software instead of to delivering and operating it, therefore increasing the value of the software

Client arguments for using SaaS:

- The costs of investing are relatively small
- Fast implementation, no installation
- Improved customer support, service and feedback
- Shared responsibility for support infrastructure
- More predictable and expected ongoing costs
- Lower risk factors and more stable security

Clearly, SaaS provides convenience to both customer and company. It is expected that the trend is going towards further growth and overall acceptance. In the future we may even be using business and at home software as described in my introduction.

2.2 The web as a platform

The web has been long since the platform for doing business. Web technologies continue to evolve to deliver new user experiences and increased utility. Web-applications are another step in that process.

2.2.1 Web 2.0: A brief history

Tim O'Reilly and Dale Dougherty of O'Reilly Publishing first used the label "Web 2.0" in 2004. The story goes that the two were brainstorming about what today's successful Internet companies had in common with web companies of the late nineties. Their ideas led to a conference – Web 2.0 – (October 2004) and gave birth to "Web 2.0" as an expression. Since then, it is widely accepted and turns up over 82 million hits on Google (June 2008), a number increasing every day.

So the birth of the expression Web 2.0 can be easily traced back. What's more difficult is to pinpoint the change that we call 2.0 and all the things that led to that change. Let alone capture it in a timeline or something like that.

Developments established a change in using the internet, after a while time someone suddenly noticed the change, gave the whole of it a nice big sticker a shouted it across the street. Actually it isn't more than just that.

But if you name something it can be used. Luckily Tim and Dale did that so we can reduce developments to a common denominator and everyone will know what we are talking about.

Tim O'Reilly summarized the term to the following short definition in 2006:

"Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more people use them."¹⁵

Although it is widely used, the meaning of web 2.0 is subject to debate. Therefore, I will try to define it further by naming a few cases out of the era that we now call Web 1.0 opposed to some that are clearly Web 2.0.

- Personal Homepage vs. Social Network Profile
- Mp3.com vs. iTunes
- CNN.com vs. news RSS-feeds
- HTML vs. XML
- Reading vs. Writing
- Owning vs. Sharing
- Dial Up vs. Broadband

The cases represent the web becoming a more social medium; the user can participate now, instead of only watching from the sideline. Without a vast and widely accepted definition it is best to describe the expression this way. Because Web 2.0 isn't an object or area, it is a metaphor for a change, an antagonism.

¹⁵ Tim O'Reilly, 2006, <http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html>

Ultimately, as SaaS, Web 2.0 is an industry buzzword. Like any buzzword, its usefulness can be debated because they are easily misinterpreted and not used correctly. Web 2.0 contains internet phenomena like blogging, content filtering, social networks & communities, geotracking & mapping, instant messaging & voice messaging, online business software (yes, that's the one we want), online storage, peer to peer content, photosharing, podcasting, relationship management, social commerce & bookmarking, RSS feeding, portals, video storage, web analytics, wiki, word processing & note taking and so on and so forth.

The expression contains so much that it is obvious and quite understandable that the meaning is subject to debate. All above phenomena originate from the growing power of the Internet in bandwidth, more powerful processors and the users feeling of comfort amidst these developments. And of course, the users do not just sit staring out of the window feeling comfortable; they actually make use of the Web 2.0 possibilities. For every venture there seems to be a market in the new digital world. In "The Long Tail" Chris Anderson describes the enormous growth of the Internet and the change that establishes in our business models. The unlimited storage space provides a near limitless choice to consumers. Tomorrow's markets belong to those who can take advantage of the enormous offer creating an ever-growing demand. That is one of the main reasons why the Internet is such an empowerment to business nowadays.

2.2.2 The RIA & web-based software

The term "Rich Internet Application" was introduced in the paper "Macromedia Flash MX—A next-generation rich client" by Jeremy Allaire in March 2002. According to him desktop offers media-rich power and web offers deployment and content-richness. Rich Internet Applications are the best of both worlds.

RIAs come in many forms and shapes. The area between classical desktop applications and classical websites has become fuzzy and confusing. Rich Internet Application is a term used for many different forms of applications more or less connected with the Internet while the user uses them. The range extends from desktop applications with extended online functions to fully web-based applications.

Different sorts include (in order from more desktop-related to more web-related) Rich or Fat Client/Internet-enabled applications, Smart Clients and Web-applications/Thin Clients.

Desktop applications connected to the Internet, Internet-enabled applications or Fat Clients are applications that make use of network support, but also run offline, perhaps with limited functionality. Examples are the e-mail client Microsoft Outlook or the music and video player Apple iTunes, of which the latter has a strong focus on the Internet with its integrated store. These applications need to be installed on the user's machine.

Web-applications/Thin Clients rarely have to be installed by the user. They can be started and loaded via network and are usually personalized by login. Examples are online agenda application Google Calendar and e-mail client Yahoo! Mail. Applications of this kind very often run in a web-browser.

The common characteristic aspect is that RIAs benefit from the best of traditional desktop and web application functionality, thus are a combination of both. The borders between the two are becoming more and more blurred.

As you see the term Rich Internet Application contains a number of different sorts of applications. Therefore I prefer the term web-based software when we speak of an application with desktop functionality, but which is purely web-based. That name also gives more association with desktop software. After all, the two are growing nearer. In the following parts I will focus on applications that are purely web-based and have the intention of replacing desktop software in functionality and workflow, because those applications and, most of all, their design and usability is the main subject of this thesis.

2.2.3 Web applications/Thin Clients vs. web-based software

The web was originally used to help researchers share documents as static pages of linked text in HTML. From there, web pages quickly evolved to more complex structures of text and graphics, with plug-ins to play audio and video or to stream other multimedia content. Scripts inserted in HTML can make use of a lot of browser functionality on the users computer. GUI interface elements such as rollover effects, pull-down menus, drag and drop, sliders and form validation are possible, thus creating a more lively & desktop-like user experience. These script possibilities, while they surely enhance the users interaction with a web page, do not change the fundamental model in which an application runs on the server and reacts to user clicks. With web-applications the user clicks, waits for the server to handle the input and then gets a response page. As a traditional web-page, a web-application is compiled of multiple pages. The difference lies in the existence of a larger process in which the individual pages represent steps. But at the basis, a standard web-application relies on the traditional model of the web as used in the beginning of static HTML.

The biggest drawback with this model is that all interaction must first pass through the server, which requires data to be sent to the server, the server to respond, and the page to be reloaded on the client-side of town.

A Web-based software application, on the contrary, makes use of client side technology. A Web-based software application can circumvent the process of downloading response pages for many actions with executing instructions on the users computer. It has an intermediate layer of code between the user and server, a client engine. This layer acts as an extension of the browser and is often largely responsible for the correct rendering of the application's interface and for the communication with the server (the application can interact with the server without an action from the user so the user doesn't have to wait). The client engine also makes more functionality and technical advancement possible in building Web-based software applications.

When a user first starts the application, the client engine must be downloaded before the user can begin using it. The best and most widely accepted example is Macromedia's series of Flash plug-ins.

2.2.4 Technologies

Below I will quickly touch on the main technologies used building RIA's to better understand the workings and difference between pure web, desktop and web-based software.

- AJAX (Asynchronous Javascript and XML)

AJAX is combination of existing technologies, including Javascript, XML, XSLT and DOM. Together these technologies drive application richness. Examples of applications

driven by AJAX are Google Maps and Gmail. The main benefit of AJAX is that it can be developed without additional tools, plugins or runtimes because it is based on standard browser technology.

- Java (Java applet, Java Web Start)

Java is a programming language that provides a platform independent rich user interface library called “Swing”. Java applications run within the Java Runtime environment, a downloadable plug-in. Furthermore, there is Java Web Start (a thin client application), which enables Java applications to run directly within the browser. An example of a Java driven web application is Wurm Online (a 3D massively multiplayer online fantasy game, www.wurmonline.com).

- Flex & Flash powered applications

Flash is often associated with banner ads, animation and video (mostly coded in actionscript 2). However, Flash is also very usable as a web application technology. Adobe has been expanding the capabilities of Flash to enhance application functionalities with its release of Adobe Flex 3. The code of a Flash/Flex driven RIA is mostly written in actionscript 3, examples are PicNik (basic online photo editing, www.picnik.com) and SlideRocket (online presentation software, www.sliderocket.com). For using such an application, a Flash Plug-in must be downloaded (Adobe Flash Player).

- Silverlight

Microsoft Silverlight is a browser plug-in that makes in-browser animation, audio and video playback possible, features that characterize a Rich Internet Application. Vector based animation is possible with Silverlight, but the main point is that it can display multimedia content integrating audio and video in the browser. Silverlight is comparable to AJAX because it uses XML to load dynamic content. Silverlight is also comparable to Flash because of the support of vector based animation and integration of audio and video. The downside of Silverlight is that essentially it offers nothing new (it competes with Adobe Flash, Adobe Flex, Adobe Shockwave, JavaFX, and Apple QuickTime) and both developers and users are much more familiar with Flash and the series of Flash plug-ins.

- AIR (Adobe Integrated Runtime)

Adobe AIR is a cross-operating system runtime environment using Flash, Flex, HTML and AJAX. AIR makes desktop deployment of internet technologies possible. The downside is that a RIA deployed in a browser does not require installation, while one deployed with AIR requires the application be packaged, digitally signed, and installed on the user’s system. The main advantage is the flexibility of working with your own data instead of having to upload files to the web server. An example of an AIR based application is eBay Desktop (desktop.ebay.com). Because of the use of Flash, an Adobe Flash Player plug-in is the main requirement.

2.2.5 The benefits of web-based software

Point-wise there are four main arguments for using web-based software opposed to desktop software.

- Application installation is not required; users access the application on the web (only plug-in installation may be required, e.g. Adobe Flash Player and Active X).

- Updates and upgrades to newer versions are automatic.
- Any computer with an Internet connection can become an access point to an application, no matter what operating system is installed.
- The risk of viral infection is greatly decreased when running an application on the web instead of an executable.

2.3 Interaction Design

“Interaction: a cyclic process in which two actors alternately listen, think and speak.”¹⁶

In “The Art of Interactive Design” Chris Crawford compares human-computer interaction with the most commonly and natural form of interactivity; conversation. Two people alternately listen to what the other has to say, think and then react, thus building a better understanding of one another.

It is important that both actors perform these three tasks well. Imagine having a conversation with someone who keeps drifting off due to insignificant reasons while you are bulging out the most wonderful of ideas. The first of three steps, listening, then is disrupted and makes the entire conversation useless. On the other hand, if you are having a conversation with someone listening closely to what you are saying, cracks his brain and then reacts in an intelligent way you can both get something out of that. For instance, it can bring a solution to a problem that kept you awake for weeks or give you new inspiration for an idea you had.

Interactivity is best regarded as a chain; one weak link even breaks up a cycle of links made of the strongest of metals. If the flow of tasks gets disrupted by one element the entire experience is broken down. Every step needs the same attention. This idea is applicable for using an interactive product, and also, for having a conversation. At the basis, good design therefore means general attention to keeping the user’s flow going while using an application.

Metaphorically a computer listens with its keyboard and mouse, thinks with its processor and then speaks with pixels, colour, frame rate and sound (one also may speak of input, process and output). These assets can be used to create a way of intelligently interacting with a machine, so that the machine can be helpful to a user and assists him building something (and if the user is happy, he will not hesitate to pay us lots of money, but of course that is not main goal).

The discipline of interaction design is about realizing a proper human-computer interface. The user stands central, everything resolves around him. It is a sad truth that the contemporary digital industry does not have a good understanding about how to best serve their users. Most software products are even designed without usability tests.

¹⁶ C. Crawford, The Art of Interactive Design, 2003, p. 5

2.4 A user-centric design approach

“Know Thy User!”¹⁷

Make your user happy and your products will be a success!
Then why are so many digital products so difficult and unpleasant to use?

Developers have a different view of an interactive product, a different set of skills and often enforce their own desires rather than those of the end users, because often they have developed a certain blindness to application utility when observed by a digibetic end-user.

Only the users know what they need and what they want when it comes to interactive products. And the only way to find out what the users need and want is to ask them.

If a designer wants to improve the chance of creating an intuitive, efficient and effective interface, the user must be put at the center of the design approach.

“Know thy user” parallels *“know thy opponent”* in sports.¹⁸ Different opponents have different strengths, different skill sets and different strategies; not every match can be approached the same way. The understanding of the users, their tasks and goals must be the headstone of the design process.

This means the design process should precede the programming phase. Bug tests then lead back to the programming phase. In the same way user tests lead back to the design phase and then again forward to the programming phase. An ongoing circle than comes into being, the process repeats itself until the product gets trough the usability testing. At last, launch the rocket and make some good money!

2.4.1 General user interface design principles

“A well-designed user interface is based on principles and a development process that centers on users and their tasks”¹⁹

There have been numerous authors (e.g. Cooper, Tidwell, Crawford, Nielsen) writing about usability and forming principles and/or guidelines. The seven principles below are from Douglyss Giuliana’s paper “User Interface Design Explained” and I consider them a simple and concrete basis for thinking about usability and making design decisions. They are basic, specific, short and contain every aspect of thinking about the usability of an interface. I summarized them point-wise.

¹⁷ A. Cooper, About Face 2.0, 2003, p. 3

¹⁸ D. Giuliana, User Interface Design Explained, 2002, p.5

¹⁹ Microsoft Corp, 1995, (D. Giuliana, User Interface Design Explained, 2002, p.3)

1. Consistency

Consistency allows users to use the knowledge they already have in learning a new task. This means a similar layout, terminology, interaction and navigation as applications already used by the users or as the industry standards.

Also an application has to be consistent within itself, that means a similar look throughout all windows, a consistent use of metaphors and navigation methods, common control placement, alignment, grouping and the use of standard terminology in words and icons. This way, the user has to spend less time getting started and has more time being productive.

2. Redundancy

Redundant cues are signs with a specific shape, color and lettering, such as traffic signs. With use of redundancy a designer builds an interface partially on the recognition of familiar symbols. Imagine all traffic signs being the same shape and colour. Driving would certainly be more difficult because it would require more concentration. The same goes for using an interface. To increase the likelihood of recognition a designer should use multiple redundant cues in an interface. For example, when an error occurs, the error can be displayed on the screen along with a beep. Be careful with sound though, keep it short and subtle so it doesn't disrupt the flow of the user.

3. Forgiveness

Even if we have had the ultimate eureka-moment and designed the perfect interface, a user is going to make a mistake. But that same interface also helps the user avoid making that mistake. For instance, by enabling buttons or fields only when appropriate and providing users with

reminders about the effects of their choices. However, when mistakes do occur, it is important to be forgiving. Users must be free to undo their actions, especially those that are destructive.

Users like to test an interface. They will click on each button to see what it does. To enable a user to learn by trial-and-error, let them explore risk-free and enable them to cancel or undo a series of actions.

4. Feedback

When a user pushes a button, clicks on a menu item or selects text an interface should provide feedback immediately. Users need this information to know what they are doing and if the application received their input. Feedback can be visual, auditory, or both. If an action takes more than a few seconds to complete, give a user an idea of how long it will take and keep them up to date about the process.

5. Simplicity

Two factors should be balanced carefully in a design; functionality and simplicity. It is easy to make an interface intuitive by keeping it plain and simple, that way we clearly show the user what and how much functionality is available.

But the more functionality, the less simplicity. A designer can consider giving users options to hide or show information they don't need or access frequently. This is a reasonable option, but keep in mind that functions should be easy to find when needed. Furthermore, limit the use of art or decoration that has no contribution to the information.

6. Interaction

The user should be in control of the system, is the actor, instead of only reactor. Allow users to personalize

the system with their own preferences. When appropriate, give a user access to setting as defaults, colours, fonts and other options.

The perfect interface I mentioned earlier lets users directly manipulate the data and objects. To make interaction as natural and logical as possible, enable interactions such as drag-and-drop. Directness of an interface prevents confusion and irritation.

7. Directness

2.4.2 The benefits of usability

The reasons for attention to usability are simple; without attention to it the users are less productive, therefore it requires more time for them to finish a project, the users require more training and support and the products developed are generally less attractive to customers.

At the deepest level, successful interactivity demands that you offer ideas to your users.²⁰ The best applications don't get their users confused and hopeless, even don't let them just do their work, but let them rise to new heights and make them realize their ambitions and dreams. So, organize your application as a vast, closed, complete and consistent working model and let your users test it (again, again and again)!

2.4.3 Usability testing

Formal usability testing requires a dedicated room with hidden video cameras, software that records user activity, microphones, detailed test scripts and established metrics. However, although less professional, a test can be as simple as a computer, a user and an observer with a sharp eye, pencil and a piece of paper and give a lot of feedback for improving one's software application.

The goal of usability testing is simply to watch your user interact with your brand new perfect interface. I bet it will not turn out so perfect when tested by multiple test-users with a variety of different skill sets, attitudes and work habits. It is important to see how well different users can work with the software.

Good usability testing enables a designer to change the things in an interface he takes for granted but are not so obvious for a user. I mentioned it earlier; developers have a different view of an interactive product. User tests open a designer's eye to overlooked details or errors quietly slipped in. You may even have to change larger parts of the interface or screen-sets which turn out illogical.

Jakob Nielsen's components of usability²¹ point out a set of criteria to test an interface when user testing. According to Nielsen usability is defined by five quality components:

- Learnability

²⁰ C. Crawford, The Art of Interactive Design, 2003, p. 42

²¹ J. Nielsen, Alertbox Column: Usability 101: Introduction to Usability, 2003, <http://www.useit.com/alertbox/20030825.html>

How easy is it to accomplish basic task on the first encounter of an interface?

- Efficiency

Once users are familiar with the interface, how quickly can they perform tasks?

- Memorability

When users return after a period of not using the interface, how easily can they re-establish proficiency?

- Errors

How many errors do users make, how severe are the errors and how easily can they recover?

- Satisfaction

How pleasant is it to use the interface?

Finishing programming an application is often seen as the end station of a development process. Usability testing is not performed at all, or half-heartedly without much attention. One must realize it is really one of the most important steps in creating good software. It has to be integrated and planned as a main part. Because how can you create something that works well if you don't know if it works for the people you designed it for? Good usability testing therefore means redesigning and redeveloping of what you thought you had finished. But that's small offer for success, right?

3 The Twilight Zone

In this chapter I will analyse the enrichment of the web and why web-based software interfaces oblige designers to come with a new approach to designing and developing web-based software applications. I will focus on the differences in behaviour and use between Desktop and Internet powered applications, a basis to later form interface guidelines specific for the latter.

3.1 Desktop and Web clashing together

The two worlds of desktop applications and websites have grown historically. Their initial purposes were not to have the possibility of merging in the future. Their roots are now inevitably growing together, resulting in a design dilemma. Desktop functionality in a web jacket; Tom Noda and Shawn Helwig summarize the situation in the following metaphor: “Imagine trying to delivering electricity through water pipes!”²²

Web-based software design can be approached successfully when we understand what makes it different from traditional web-design on the one hand, and traditional desktop application design on the other hand.

3.2 Desktop and Web vs. web-based

With desktop applications users can enjoy extremely comfortable interactivity and responsiveness. Input forms can be evaluated directly, alerts can notify users on preventing errors and so on. User’s actions can be observed the whole time.

Generally desktop applications are also available offline, even when it concerns an internet-enabled application. Desktop applications can make use of asynchronous communication; events offer the system to update the user interface at any time, even without input from the user. The request-and-response model is not used because there is no server distance to bridge.

A designer controls every pixel on the screen when designing a traditional desktop application. He can make sure that the interface looks exactly the same on the users screen. He knows what operating system he is designing for, of what fonts it disposes, how large the effective space of the screen will be and he can work by a system vendor’s style guide for design-rules. Guidelines for designing desktop applications can be found in the Apple Human Interface Guidelines (2008) and the Windows Vista UX Guidelines (2007).

Pure static HTML has a very poor user experience. Interaction is limited to the page-to-page concept, which only provides feedback when the next page is loaded: a ‘synchronous’ interaction model. For instance, all text fields in a web form can only be evaluated after pressing a submit button and receiving feedback on the next loaded page.

²² T. Noda & S. Helwig, Rich Internet Applications, November 2005, p.1

The interaction on a classic website primarily involves searching and following links. A navigation model of such a website consists of pages sequentially or hierarchically organized. Users are taken from one page to another, with the possibility of jumping steps by searching. In a classic website, design principles focus primarily on the organisation, content and clarity. The classic web was just not meant for interactivity like we know it nowadays. A number of well-developed guidelines for web usability and design can be found in *Prioritizing Web Usability* (2006) by J. Nielsen and *Designing Web Navigation* (2007) by J. Kalbach.

The main difference between the classic web and desktop software is that in desktop application design the focus lies mainly on the behaviour of the application, whereas in web the focus lies on the content and organization. On the web, users move between pages very quickly. It is a rare occasion for users to spend more than a few minutes on a single page. According to Jakob Nielsen because of the quick navigation users feel that they are “using the web as a whole, rather than a specific site”.²³

Desktop software, on the contrary, is mostly viewed as the main application. Users stay within the same application for longer periods of time. Of course, there are different categories within desktop software. Alan Cooper describes them in his book “About Face” (summarized):

- Sovereign Posture

Programs that are used full screen and monopolize the user’s attention for a longer period of time. E.g. Adobe Photoshop.

- Transient Posture

A program that comes and goes, presenting a single high-relief function. E.g. Calculator.

- Deamonic Posture

A program that runs in the background and of which the user doesn’t notice that it is running. E.g. a Printer Driver

- Auxiliary Posture

A program that blends characteristics of sovereign and transient programs. Continually present like a sovereign posture program, but only performing a supporting role. E.g. the Windows taskbar or a clock program.

These postures generally are meant for desktop software, but with the arrival of the RIA and the many different forms it takes web and desktop have collided. Rich desktop-like interfaces are now possible on the web. The web seems to become an equal to desktop when it comes to fluent interactivity and responsiveness. Screens are loaded and presented dynamically. These applications are, like desktop-applications, heavily transactional and can be thought of half-website, half desktop-application. Such a software application also needs to monopolize the user’s attention for a longer period of time. A web-based software application could therefore be viewed as a sovereign posture program.

²³ J. Nielsen, Alertbox column, *The Difference Between Web Design and GUI Design*, May 1997, p.2

Again I point out the difference Luke Wroblewski defines in his paper “Design considerations for web-based applications”²⁴:

- Existing guidelines for Web usability hinder web-based application usability since they are primarily based on interactions within a browsing metaphor.
- Interface design guidelines for client applications, on the other hand, do not address the conventions of web users, limitations of the web environment, nor the new possibilities that the web has to offer, such as AJAX and other web 2.0 technologies.

Web and Desktop go hand in hand in web-based software design, because they are a combination of both. Often a web-based software application is a part of a website, or runs in a browser but supports desktop functionality. I mentioned above that in web the focus lies on the content whereas in desktop the focus lies on the behaviour of the interface. Therefore we must make a contraction of that statement. Content drives the interactivity. If the interactivity is fluent but the content remains unclear users are expected to abandon their efforts. In web-based software design the focus therefore lies on the behaviour of the application *supporting* the content.

In addition to that, interactivity in web-based applications within the same category is often unique. For example, a Customer Relationship Management application (CRM) of one company can be highly different from another in interface and interaction, because there are no standards to web-application design and not every company shops at the same software-developer.

3.3 Desktop vs. SaaS

Many sources describe the technological challenges of using or deploying a SaaS model when compared to traditional business software installed on the company’s server. But what distinguishes a SaaS application from a desktop-one on use and design level?

More must be done than only finding a technological solution to repackage a traditional desktop application for deliverance over the web. According to Paul Giurata from Catalyst Resources a SaaS application “*must do at an application strategy and design level, what traditional on-premise applications do not*”²⁵.

To develop a successful SaaS application, it must fulfil the following two demands:

- They need to provide an application user experience that is seductive, targeted and engenders loyalty.
- They must integrate a constellation of self-service tools and users experiences that enable customers to do things they are not normally exposed to, such as purchasing, customization, billing, provisioning and monitoring.

The main difference is that with traditional in-house software the user only focuses on the application itself. The application is stand-alone, installed on the users system and

²⁴ L. Wroblewski, Design Considerations for web-based applications, 2001, p.1

²⁵ P. Giurata, Application strategy and design for a profitable SaaS, p.1

the IT department or network administrator takes care of the management, monitoring, updates and all other tasks related to using software in a business environment.

A SaaS application also focuses on this primary application part, but incorporates tasks that all together make up the SaaS application and business model. These tasks include registering, paying for the software, using support and customizing the application. A SaaS application is much more specifically directed to the customer instead of a traditional desktop application directed at the mass. This results in a higher value of design and interaction. For instance, when the registering procedure of a SaaS application is not intuitive, fluent or clear enough, a potential client abandons his efforts instantly. A case like that immediately results in the service provider not making money. Desktop packages, on the contrary, are sold and installed in large numbers. In a business situation a user is more or less obliged to use that particular piece of software when installed. In an at home situation the user is at least more determent to learn how to use the software, because he has paid a certain amount of money for owning the package.

A SaaS application does not consist of a single user experience like a traditional desktop application. SaaS combines multiple user experiences and each of those experiences must be designed with the same user interface and procedure. This is the idea of a “constellation of services”²⁶.

Furthermore, SaaS applications are accessed over the Internet; therefore users will perceive the use of the application similar to that of a web page. They will transfer their knowledge and previous experiences with websites to the application. However, when a SaaS application is used, we want the user to experience it as the primary active application. To avoid the quick navigation behaviour users are familiar with in classic web pages a SaaS application must be used full-screen and requires the user’s full attention. In his book “About Face” Alan Cooper also describes the behaviour of sovereign posture web applications very shortly. Sovereign posture web applications “*should and do strive to be nearly indistinguishable from their desktop cousins*”²⁷. About this Luke Wroblewski says the following; “*weblications most often do not occur independent of a web presence. In order to achieve a unified user experience, a sense of ‘place’ needs to be maintained within the website*”²⁸. A good example of a near exact desktop copy in a browser is the Microsoft Outlook Web Access client, which behaviour is almost exactly the same as the desktop version of Microsoft Outlook although it is constrained by browser technology.

3.4 General web-based vs. SaaS

On an application and interface design level a standard web-based software application may not be very different from a SaaS application, because it works with the same server- and client-side technology and interface behaviours. Essentially a SaaS application is just a standard web-based software application, but (and there is a large but here) with a lot of added SaaS-specific functions. If a web-based software application is empowered by a SaaS business model the application must contain a number of services

²⁶ P. Guirata, Application strategy and design for a profitable SaaS, p.7

²⁷ A. Cooper, About Face 2.0, 2003, p. 483

²⁸ L. Wroblewski, Design Considerations for web-based applications, 2001, p. 2

that enable a company to make money with the application and a client to use it. These include registering, billing, support and the other examples I mentioned above.

With SaaS the user interface is much more important than with a standard or free web-based software application, because a company's business depends on it.

A user is free to choose for your SaaS application or not, and even when he does he can just as easily quit using without having seriously invested. The customer-driven and customer-focused approach of SaaS obliges a SaaS application to be completely customer friendly. The power lies with your user with SaaS, when you don't serve him you will not make money. It is as simple as that. Therefore, the business of a SaaS provider depends on the user experience of the application as a whole.

Because a SaaS application itself is a web-based software application guidelines can be derived from web-based design guidelines. But they need to be enhanced on points specific for SaaS relating to call-to-action, overall application behaviour, and interface consistency in all services next to the application itself.

4 Guidelines

In this chapter I will outline guidelines for web-based software. Both out of aspects of the few literature available, the differences I mentioned in the above chapter as well as my own research. From that I will derive specific SaaS principles.

4.1 The evolution of usability

“Web-sites are effective places to get information you need, just as elevators are effective places to get to a particular floor of a building. But you don’t try to do actual work in elevators.”²⁹”

Web-based software products are meant to let a user be productive. That does not seem to fit in the transient environment of the web. It is difficult to supply a user with a working environment in a medium in which he normally would quickly jump from page to page. Designers need to find ways of dealing with the boundaries of the web-browser, while web-based software continues to evolve enabling productive power on the Internet.

4.2 Web-based design considerations

Below, I will discuss several considerations specific for web-based software per author.

4.2.1 Alan Cooper

In his book “About Face” Alan Cooper that the difference between desktop and web-based software applications in behaviour should be made as little as possible. The design of such an application is best approached as if it were a desktop application. However, a designer needs a clear understanding of the limitations of the browser (even while the technological boundaries are ever expanding) and must keep those in mind at all times. Sovereign web-based software applications should be full-screen applications and can be just as densely populated with buttons and sliders as a desktop application. They need to make use of different panes or screen regions to group related options. Furthermore, users need the feeling that they are working in a specialized environment on the web, because they are so used to the web’s quick navigation possibilities. A designer must make the user settle down and handle a web-based software application with the same main attention a desktop application needs. It is a possibility to hide all browser controls, providing an effective full-screen browser area, this gives the user a more desktop-like feeling, but decreases flexibility and multitasking while using the application.

²⁹ A. Cooper, About Face, 2003, p.483

4.2.2 Luke Wroblewski

In his paper “design considerations for web-based applications” Luke Wroblewski describes the willingness of a user to learn how to use a web-based application, because it is likely to be used more intensively and more frequently than a traditional web-page. This is very different from the rapid surfing from web-page to web-page. Users want the services that web applications have to offer to increase productivity and comfort while working. This results in a general positive attitude towards web-applications, an enormous advantage for a designer of an web application interface. However, solutions must be found for averting inapplicable knowledge users transfer from the use of a traditional web-page to the web application. Furthermore, during delays users need to be accommodated with a possibility to multitask, and when they return they need to be informed about the loading progress. With increasing Internet connection speeds, loading delays will become shorter, but nonetheless this is an essential part of the software experience.

4.2.3 Chris Loosley

In Chris Loosley’s paper “Rich Internet Applications: Design, Measurement and Management Challenges” he discussed a few web-based design considerations shortly. A website containing a web-based software application should be simple and natural, not pull the attention away from the application itself. It must be easy to learn, predictable and consistent. RIA technology may allow developers to create more intuitive and desktop-like interfaces on the web, but it will not guarantee a better experience than a traditional web application when exaggerated. About this J.J. Garrett says: “The more power we have, the more caution we must use in exercising it”.³⁰ Furthermore, as broadband penetration continues to increase, the success of rich user experiences will continue to grow. Customers will expect websites to keep up with the developments and neglects the ones that fall behind. Therefore, it is important to keep up with the latest technology as a company.

4.2.4 Jess McMullin & Grant Skinner

Jess McMullin and Grant Skinner converted Jakob Nielsen’s 1994 usability heuristics for desktop³¹ especially for RIA’s.³² I have summarized their conclusions point-wise.

³⁰ J.J. Garrett, Seminal Ajax Paper, 2005, <http://www.adaptivepath.com/ideas/essays/archives/000385.php>

³¹ J. Nielsen, 1994, http://www.useit.com/papers/heuristic/heuristic_list.html

³² J. McMullin & G. Skinner, 2003, http://www.boxesandarrows.com/view/usability_heuristics_for_rich_internet_applications

1. Visibility of system status

RIAs should display clearly when background processing requires the user to wait, e.g. when the application is preloading, but also during the user's whole interaction with the application, e.g. displaying the sequential steps or progress through different tasks.

2. Match between system and the real world

The RIA should reflect the user's real world (e.g. speak in verbs, nouns and concepts familiar to the user instead of language from computerland), so that the user's learning process speeds up. Use of RIA technology and related terms like rollover, timeline, actionscript and remotings are best avoided, both in the application itself as in popups or notifiers like error messages.

3. User control and freedom

When a user makes a mistake, he must be able to go back and forth within an application to correct. Therefore, a RIA should support undo and redo. Though the internet does not automatically support this process, data loss is not an option when using an online application. A RIA should include code to support this functionality and letting the user undo and redo.

4. Consistency and standards

Users need a safe, known environment. They should not have to wonder if different words, situations or actions use the same thing. A RIA should follow platform conventions. In developing a RIA, a designer should follow interface standards like those of Windows or Apple. Note: herein also lies the dilemma, why use desktop standards for a platform independent web-based application running in a platform dependent

browser? This is a clear example of missing guidelines for a new type of software.

5. Error Prevention

You can design a good error message, but even better is the overall prevention of the user making errors. In designing RIAs this means a careful consideration about the behaviour of input fields in the application (e.g. recognizing different input formats like xx-xx-xx or xx.xx.xx, saving repetitious data and auto-filling fields). Furthermore, avoid destructive functions like "Delete all data".

6. Recognition rather than recall

Make information, objects and actions visible instead of the user having to remember where certain information can be retrieved. In RIAs, be careful with rollovers, cause they tend to slow the user down.

7. Flexibility and efficiency of use

Enable accelerators like keyboard shortcuts for experts, so that the system can be effectively used by a wide group of users. Furthermore, enable customization of the application, both in functionality as in colour schemes or fonts used.

8. Aesthetic and minimalist design

Design the application as minimal as possible, application functionality is the main factor. Limit the use of animations and make the branding as subtle as possible.

9. Help users recognize diagnose, and recover from errors

Error messages should be expressed in normal language (no computer codes). Reference to “missing objects” will only frustrate users. Since RIAs run on the Internet, we have the possibility to immediately connect with a online help service. Help can be provided by chat, video conferencing or remote manipulation of the application.

10. Help and documentation

It is better to have no help or documentation at all, simply because it is not required for a user to understand the application. RIAs can provide simple and concise instructions, prompts, and cues embedded in the application itself. Though, if it extended information is required, the application should contain a simple help and documentation area.

4.3 Web-based design principles

These guidelines should be seen as a recommendation, based on various aspects analysed in literature. These recommendations should be enforced with user testing, because most web-based software products are unique, and their interaction patterns and interface behaviours are often unique as well.

1. Draw a clear border between the shell and the core

A web-based software application mostly resides within a website. Before a user accesses the application he first browses through the website containing the application. To distinguish the website (the shell) from the application (the core) a clear border must be drawn, both in design and behaviour. The transition from website to application must be clear to the user, for example by launching the application full screen automatically when accessed. We need the user to lose the “web-feeling”; the transition from website to application and design of the application need to establish that.

2. Open the application full-screen size

We want a user to experience the web-based software application as if it were an installed desktop application. This gives the user the

feeling of working in a specialized environment and improves the workflow. Opening the application

full screen is automatically is a good option. However, we want the user to have the possibility of multitasking while working. Therefore operating system basics like the Windows taskbar or the Apple dock must be accessible at all times. Note: offer the possibility to abort full screen view and work on a custom browser size.

3. Hide browser controls

As stated, a web-based software application resides within a browser. Users perceive the browser as the

application and the web-based software application running in the browser as the content. We want the user to look at the web-based software application independent from the browser; as the main application. Technology to run web-based applications without a browser is in its infancy (e.g. Adobe AIR, see chapter 2). In the meantime, we must find solutions for increasing web-based software’s independency from a browser. Hiding browser controls is a considerable option. When the application runs full screen and browser controls are hidden the application is more likely to be the user’s central point of attention.

Furthermore, the browser’s back and forward buttons can be a danger to application utility and the user losing data (especially in Flash/Flex based applications). Note: be careful with hiding controls; give the option to show the controls when desired.

4. Do not use multiple browser-windows or tabs

We don’t want information in one window get lost behind others and we don’t want a user to get confused by multiple windows or tabs. Websites have the tendency to open links in other windows or tabs. That behaviour must not be transferred to web-based software. Like most desktop applications we want the user to work in one main work-area. Therefore, use only one main window.

5. Make use of known web elements & conventions

Users tend to transfer their knowledge of the web to the web-based software application. Build on that knowledge. Make use of web elements and conventions like underlined links, coloured clickable area's and use radiobuttons and checkboxes as utilized online to increase a user's familiarity with the interface on the first encounter.

6. Use constant values for fonts, tables and all other visual elements

Make use of a consistent layout, cross-browser and cross-platform. Web-based software applications have the possibility to be used by the same user on different places, different computers and different operating systems. We must make sure that the application looks the same on all. Establish a stable interface that users can rely on.

7. Use roll-over interface elements

As desktop applications often do, use roll-over interface elements. Dropdown menus lined up in the upper left corner of the screen further increase the number of desktop similarities.

8. Use overlays

Use overlays for functions as login, register and other parts besides the main application like help or about. Overlays prevent switching screens. Therefore, overlays don't disrupt the workflow as much as loading another page. Furthermore, a user's data or project can still be visible behind the overlay. This increases comfort and a feeling of security ("my project is still there") while working.

9. Animate, display and time loading delays

Inform your user about the progress made when loading. Animate and display that the application is loading and give an estimate about how long it is going to take. A user must be able to multitask during loading-times.

10. Enable undo and redo

As in desktop applications, undo and redo must be enabled in web-based software applications for establishing a desktop-similar workflow. It can be a drop-down menu option and shortcuts (e.g. CTRL-Z) can be enabled.

11. Enable application customization

Give users the option to customize the application's settings, keyboard shortcuts, colour schemes, sound alerts, etcetera. That data can be linked to the user's login. Personalizing the application increases a user's familiarity with it and helps building a personal working environment, as in most desktop applications.

12. Limit the use of animations and artwork

Animations and art can be aesthetically pleasing, but mostly draw the attention away from the application itself. While the web is known for its frills, desktop applications are not. Of course visual design is important, but ask yourself where form and function meet. Slick design can be tempting, but at all times let function be your main point of interest.

4.4 SaaS design considerations

Below, I will discuss several considerations specifically for designing a SaaS application. They arise from the considerations and guidelines above, but focus on SaaS and its specific properties. These considerations are based on Paul Giurata's paper "Application strategy and design for a profitable SaaS".

- Purchasing/Sign-up

The purchase experience of a SaaS application is completed online. There is no third party or store between the client and the company. Therefore, a SaaS application can be profitable "with annual revenues as low as \$250-\$1000 per year".³³

Purchasing becomes a large part of the product itself and should be designed along with it. Purchasing or signing up to a SaaS application is the main barrier for a user to start working (and the SaaS provider to make money). To eliminate that barrier, options available must include full-access trials, occasional usage and a sign-up experience tested on usability the same way as the application.

- Provisioning

Traditionally provisioning is done by administrative professionals within a organisation. With SaaS, provisioning is exposed to the user and must be simplified. Setting up accounts, adding and deleting users must be free of errors and efficient. As purchasing, provisioning becomes a part of the application and should be given the same attention as the application itself.

- Support

Support is one of the key elements in designing a SaaS application. When a user makes an error, he must be able to immediately recover from it. Both in setting up an effective help and support area within the application as in enabling personal customer support. A SaaS application, as a matter of course, must be designed to avoid a user making errors. But errors do occur. Help and support can be completely integrated in the application so the user's workflow gets interrupted as little as possible. Implant visual cues and give hints to what to do next.

- Adoption and Penetration

Building an ongoing revenue stream revolves around users getting to try your SaaS application. Social networking plays a large role in adoption. Therefore, built-in tell-a-friend options and community based adoption enables users to spread the word.

- Revenue Model

With traditional software the manufacturer gets paid up front, even when the user isn't satisfied and ends up not using the product. With SaaS, customer loyalty is complete based on a satisfactory user experience. If the sign-up process is difficult to complete, a

³³ P. Giurata, Application strategy and design for a profitable SaaS, p.5

user will quit the process and there is no subscription. If a user does has completed subscription, but isn't satisfied about the product's value they will stop using it nonetheless. Therefore, a successful SaaS application is completely based on a well designed interface and workflow. Establish on-going loyalty and revenue by setting the user as the headstone of your application.

- Development & Updates

Because the application runs on a central server a SaaS application can be constantly updated, ensuring the user they always dispose of the latest version. This way software can be easily innovated and users can see their feedback quickly incorporated in the software. When the user's feedback and developing the software go hand in hand, a developer can ensure the software will continue to improve. Therefore, SaaS development cycles should be periodic and "managed to no more than 90-180 days to enable incorporation of feedback and refinements"³⁴

³⁴ P. Giurata, Application strategy and design for a profitable SaaS, p.7

4.5 Web-based design principles enhanced for SaaS

All web-based software guidelines also apply to SaaS. However, a SaaS application needs to be further enhanced on specific points that differ SaaS from standard web-based software. Out of the considerations described above I have set up SaaS guidelines as an extension to general web-based software guidelines point-wise.

1. **Expand the design focus to all services next to the core application**

As stated, an application hosted as a service consists of multiple services next to the core application itself, because those services together enable the SaaS business model. This includes registering/subscribing, paying for the software/billing, using support and customizing the application. The idea of a “constellation of services” obliges a SaaS developer to expand the focus when developing software. Make sure all tasks and screen-sets a user encounters while using your SaaS application are designed with the same attention and are friction free.

2. **Enable users to evangelize**

SaaS subscriptions are mostly sold per-user. The users are the key to building a profitable SaaS application. Give them the option to tell other users about the application. In the best case, send-a-friend and word of mouth brings you free publicity, blog-posts and many new customers. Evangelizing users can establish the wave of attention you need for an ongoing revenue stream.

3. **Establish a strong call-to-action**

Next to the expansion of the design focus one of the strongest assets in hosting a SaaS application is the call-to-action. Convince your user of your application’s value by establishing a

call-to-action tested thoroughly on user experience. Make sure the core values of the application speak on the

first encounter of the home page. Members are the key to making money.

4. **Enable testing sessions and free-of-charge trials**

Convince users further of your application’s benefits by offering testing sessions and trial periods. When a user can access the full power of the application before actually becoming a member he can make an objective decision about becoming a member or not. When your application is powerful and worth trying, it will increase confidence and trust and helps building a loyal customer group.

5. **Integrate help and support in the application itself**

Help and support can be integrated in the SaaS application itself. When a user runs into a problem while using the application he or she can immediately be linked to the appropriate help or tutorial part. Direct customer assistance is also an option; help can be provided by chat, video conferencing or remote manipulation of the application. Proper help is a part of offering good service.

6. **Let user feedback and software developments go hand in hand.**

Ask your users for feedback and process this feedback in the application on a periodic basis. This way, a developer can establish an ongoing circle. The software will

continue to improve, better serving the users. Furthermore, involving the user established a personal bond. This

further increases customer loyalty and comfort.

5 Conclusion

When I started this thesis I had formulated an abstract idea of what SaaS design might contain. Many sources described the technical challenges of SaaS and the benefits of the business model opposed to the paved road of software supplying. But there were little sources describing actually designing an interface for a web-based software application, let alone specifically about designing an interface for a web-based software application delivered as a service. I stumbled upon many dilemmas relating to what makes a SaaS application different from a standard web-based software application design-wise, the basis to form guidelines for SaaS.

During the course of this thesis I realized that the basis to found the guidelines for web-based software and for SaaS is desktop similarity. The key to successfully transform the web in a productive environment is building an effective working space. In designing such an application one must emphasize on the user losing the web-behaviour he is used to; quick navigation. Let your web-based software application behave as the main object of attention.

SaaS is defined by not only delivery via the Internet, but by subscription and periodic payment. Especially these properties differ SaaS from standard web-based software. As stated, a SaaS user experience resolves around a constellation of services, all equally important in establishing a positive user experience, a successful SaaS application, a loyal customer base and, essentially, revenue. This lays the foundation for designing and developing a web-based software application delivered as a service.

In the answer to my main question stated in chapter one:

“What are the key elements in designing an interface for a web-based software application delivered as a service?”

The key elements in designing an interface for a web-based software application delivered as a service consist of both the basic web-based software guidelines set up in chapter 4.3, expanded with the specific SaaS guidelines in chapter 4.5. The foundations of a SaaS are the web-based software guidelines also applicable to standard or free web-based software. In addition to that the SaaS specific guidelines can make the difference in designing a profitable SaaS application.

The IT industry may be migrating to other forms of delivering software and other ways of establishing that technically. However, we must continue to serve the user in designing applications while new business ventures arise and technical possibilities are expanding.

6 Bibliography

Papers

- Wroblewski, L., Rantanen, E.M.: *Design Considerations For Web-Based Applications*. Santa Monica, CA: Human Factors & Ergonomics Society, 2001
- Giuliana, D.: *User Interface Design Explained*, 2002
- Bovee, R.: *Servicemanagementapplicaties in het SaaS-concept*, Mansystems BV, 2008
- Allaire, J.: *Macromedia Flash MX-A next-generation rich client*. Macromedia, Inc. March 2002
- Kintera, Inc.: *Rent or Buy: Why Nonprofits Need Software as a Service*. 2007
- Voelker, K.: *Web 2.0*. MGT 407 Intro to MIS Business Relationship, Research Paper, 2007
- Kaplan, J.: *Software-as-a-Service Myths*. BusinessWeek Online, April 17, 2006
- Vermeulen P.: *De Business Case voor Software as a Service*. IDC Information and Data, August 2006
- Vroom, M.: *SaaS 2007 hype or (h)(n)ot?* Nedfox BV, RetailVista, March 2007
- Wroblewski, L., Ramirez, F.: *Web Application Solutions: A Designer's Guide*.
- Noda, T., Helwig, S.: *Rich Internet Applications, Technical Comparison and Case Studies of Ajax, Flash and Java based RIA*. Best Practice Reports, UW E-Business Consortium, University of Wisconsin-Madison, November 16 2005
- Loosley, C.: *Rich Internet Applications: Design, Measurement and Management Challenges*, SLM Technologies, Keynote Systems, 2006
- Giurata, P.: *Application Strategy and Design for a Profitable SaaS*. Catalyst Resources, 2007
- Schelling, J.A.: *Social Network Visualization*. Graduation thesis Interaction & Visual Interface Design, C&MD Rotterdam, 2007
- Moritz, F.: *Rich Internet Applications: A Convergence of User Interface Paradigms of Web and Desktop Exemplified by JavaFX*. Masterthesis Digitale Medien, University of Applied Science Kaiserslautern, January 2008

Books

- Hoekman, R. jr.: *Designing The Obvious*. New Riders, Berkeley, CA, 2007
- Crawford, C.: *The Art Of Interactive Design*. No Starch Press, San Francisco, CA, 2003
- Cooper, A., Reimann, R.: *About Face 2.0, The Essentials of Interaction Design*, Wiley Publishing, Inc. Indianapolis, IN, 2003
- Tidwell, J.: *Designing Interfaces, Patterns for Effective Interaction Design*. O'Reilly Media Inc, Sebastopol, CA, November 2005

Blokdijk, G.: *Software as A Service – SaaS 100 Success Secrets*. 2008

Anderson, C.: *The Long Tail, Why the Future of Business is Selling Less of More*. Nieuw Amsterdam Uitgevers, 2006

Articles

Garrett, J.J.: *Ajax: A New Approach to Web Applications*. Seminal Ajax Paper, 2005, <http://www.adaptivepath.com/ideas/essays/archives/000385.php>

McMullin, J., Skinner, G.: *Usability Heuristics for Rich Internet Applications*, Boxes and Arrows, July 2003, http://www.bboxesandarrows.com/view/usability_heuristics_for_rich_internet_applications

Nielsen, J.: *The Difference Between Web Design and GUI Design*. Jakob Nielsen's Alertbox Columns, May 1997, <http://www.useit.com/alertbox/9705a.html>

Nielsen, J.: *Usability 101: Introduction to Usability*. Jakob Nielsen's Alertbox Columns, August 2003, <http://www.useit.com/alertbox/20030825.html>

O'Reilly, T.: *Web 2.0 Compact Definition: Trying Again*. O'Reilly Radar, 2006, <http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html>

Van Heur, R.: *Alle Drempels Zijn Weg: ASP en SaaS*. Computable, January 2007, http://www.computable.nl/artikel/ict_topics/loopbaan/1816677/1458016/asp-en-saas.html

Van Heur, R.: *SaaS breekt definitief door in 2008*. Computable, March 2008, http://www.computable.nl/artikel/ict_topics/loopbaan/1816677/1458016/asp-en-saas.html

Websiteoptimisation.com.: *Worldwide Broadband Penetration*, August 2007, <http://www.websiteoptimization.com/bw/0708/>

Websites

CatalystResources, User Interface and Rapid Application Design for Financial Services & Software as a Service (SaaS), <http://www.catalystresources.com>

Gartner Consulting IT Inc., www.gartner.com

Microsoft Service Providers, <http://www.microsoft.com/serviceproviders>

Salesforce, Succes On Demand, www.salesforce.com

Acknowledgments

I would like to express my gratitude towards the following people.

My two companions in graduating: Sander & Matthijs, for emailing me nonsense every day, keeping my spirits up and listening when I ran into problems.

My friends for the “nights off”, clearing my head and, every now and then, helping me with dilemmas (even when they hadn’t got a clue of what I was talking about): Bouke, Robin, Dennis, Emmie, Fenneke, Christiaan, Corstiaan & Leonard.

My supervisor at C&MD; Ayman van Bregt, among other teachers that shaped my vision and of whom I learned a lot during the past four years at the Hogeschool Rotterdam: Carolien van der Akker, Bas Leurs, Alexander Bakker, Deanne Herst, Barend Hendriks & Michel Penterman. Furthermore, my C&MD classmates who inspired, motivated and brought up the best in me both in classes and projects: Harro, Lennart, Niels, Cheyenne, Rick, Peter, Jan & Marco.

The people at my graduation company Pixelindustries, for giving me the opportunity to write about this interesting topic and putting faith in my skills as a designer. Thanks go out to my supervisor Rodger Buyvoets and my colleagues Johan, Deniel, Danilo, Koos, Febrian, Marcel, Sam & Rogier and to owners Ray, Dave & Matthijs.

I would like to thank Tony DeYoung at CatalystRecources for his valuable input and ideas. He really helped me settle the matter in the core of this thesis.

Last, but definitely not least, my parents and brother Carl for their ongoing support and encouragement.