User-centred design and the next generation OPAC – a perfect match?

HENRIK LINDSTRÖM and MARTIN MALMSTEN
National Library of Sweden, LIBRIS Department.
Box 5039, SE-102 41 Stockholm, Sweden.
(*email: henrik.lindstrom@kb.se, martin.malmsten@kb.se*)

Abstract

The shortcomings of today's OPACs have been widely debated over the last few years. It is clear that the OPAC has been falling behind in the general development of web applications. We now see a new generation emerging, but how do we make sure that we do not lose ground again? In order to keep and possibly attract new users, we need to offer applications that comply to their expectations. This paper suggests a combination of iterative development and user-centred design as a way to develop systems that will meet the constantly changing demands on functionality, at the same time as supporting usability. The paper gives a short introduction to iterative software development and user-centred design. A case study of the development of the new version of LIBRIS, the Swedish National Union Catalogue, is used as an example of how these methodologies can benefit from each other in practice. Pros and cons with the two methodologies and how they were adopted in the LIBRIS-project are also discussed.

KEYWORDS: iterative development; agile software development; usability; user-centred design

1 Introduction

Many of the present library systems have been in use for several years, and little development and improvement has taken place. These systems have not only fallen behind in terms of functionality, but also suffer from overly complex and library specific solutions that most non-expert users do not find themselves familiar with. Today one expects user friendly, intuitive and aesthetically pleasing interfaces. The users have also adopted new search-behaviours from other services, such as Google, that highly influence how they expect a library system to work.

These are all well known issues, which have been discussed for several years in conferences, papers, email lists etc. Many listings of the shortcomings of the OPAC have been posted, as well as suggestions for improvements (Schneider 2006, Bates 2003, Calhoun 2006, numerous threads on lists such as NGC4Lib[1], etc.). We know what we would like to see in the next generation of OPACs, but how do we turn this into a usable system, and how do we make sure that we in five years are not back in the same position of outdated systems?

[1] NGC4Lib is a mailing list for discussions about the Next Generation Catalogs for Libraries. http://dewey.library.nd.edu/mailing-lists/ngc4lib/ (2008-03-31).

On December 19 2007 a new version of LIBRIS, the Swedish National Union Catalogue, was released after just over a year of development. The release was preceded by half a year with a public beta version, which was used for continuous development and testing. Not only did we develop the application itself, we also tested and adopted new methodologies for software development.

During the project an agile, iterative, development methodology evolved, in combination with a user-centred design (UCD) process. We argue that these two methodologies allow you to meet the users' demands and expectations today, as well as keep up with, and adapt to the constantly changing world of web applications. It is the purpose of this paper to briefly present these two methods and how they can benefit from each other.

We are not the first ones to point to the benefits of combining agile development and UCD. For example, Patton (2002) describes how interaction design successfully was added to an agile software development project and Gulliksen et al. (2003) discusses how agile methods could be adapted to fit a user-centred design process.

2 Usability

In the introduction we discussed the importance of designing for usability. But what do we mean by a "usable" system? ISO 9241-11 defines usability as:

> The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use (ISO 1997).

In the case of an OPAC this implies that it can be usable for a librarian, who is familiar with library systems, whereas the usability arguably is quite poor for a fist-time, non-librarian user. Expert users (e.g. a librarian) and non-expert users (e.g. a undergraduate student) may differ in goals and satisfaction (the librarian wants an exact and exhaustive answer, whereas the student is content by finding the course book), to what degree they can perform the task efficiently (the librarian is familiar with the system and knows how to perform a search, whereas the student may struggle finding what he or she is looking for, perhaps because the system differs in design and functionality from other systems he or she is used to) and so forth.

Designing a usable OPAC, therefore, in many respects is a task of designing a tool for at least two quite different user groups. For example the non-expert users may need a simpler and more supportive system that guides them to functionality they do not know of, or are uncertain of how it works. However, this does not necessarily imply a conflict in design as long as the transparency and supportiveness of the system is provided for in a manner that is not obstructing the workflow of an expert user. But designing for usability is generally speaking a matter of finding compromises. And if you take a closer look at both the "expert" and the "non-expert" user groups you probably find them to be quite heterogeneous within themselves. However, it is important to understand who the users are and in which contexts they are going to use the system.

3 User-centred design

User-centred design (UCD) can be defined in many ways, but all definitions are characterized by a focus on the user, and on that the user's perspective should be incorporated in all stages of the design process. Donald Norman describes UCD as "a philosophy based on the needs and interests of the user, with an emphasis on making products usable and understandable" (Norman 2002, p.188). By this definition, actual user involvement is not a part of UCD by

necessity. However, involving the users in the user-centred design process is a common way of ensuring that their needs and interests are being met.

The user-centred design process has also been formalized in the ISO-standard 13407 *Human-centred design processes for interactive systems* (ISO 1999). The standard describes UCD as an iterative process consisting of five steps, depicted in figure 1, and also states the following key principles:

- The active involvement of users and clear understanding of user and task requirements.
- An appropriate allocation of function between user and system.
- Iteration of design solutions.
- Multi-disciplinary design teams (ISO 1999).

Much could be said about each of these principles, but we will only briefly comment on the second and the fourth, since the other two should be quite self explanatory. "An appropriate allocation of function between user and system" means that it is important to determine which aspects of a task should be carried out by the system and what should be carried out by the user (Maguire 2001). It is for example desirable to limit the amount of tedious time-consuming routine work for the user, and instead let him or her focus on the aspects that require the users' expertise. UCD is collaborative in its nature and implies the involvement and exchange of different competences, hence the last principle of multi-disciplinary teams (Maguire 2001). We will return to the benefits of working across disciplines later in this paper.

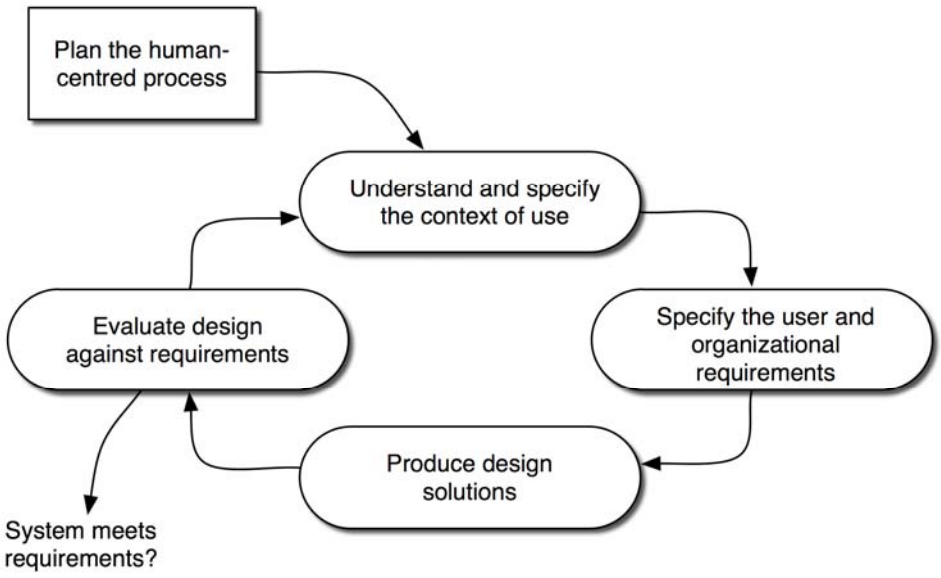

FIGURE 1. The human-centred design process, ISO-13407 (ISO 1999)

In figure 1 we see a schema over the iterative user-centred design process as defined in ISO-13407. In order to make the users an active part of each step in this process there is a number of empirical methods, such as:

- Interviews
- Surveys
- Workshops
- Focus groups
- Field studies
- Usability testing

The different methods each have their strengths and weaknesses. For example, interviews can give in-depth, qualitative data, but are resource-consuming to carry out on a large number of users. Surveys, on the other hand, can be distributed to a large number of subjects, but with more quantitative data as result. However, if we carry out a survey *and* conduct interviews to study the one same thing we can eliminate some of the weaknesses and at the same take advantage of the strengths of each method. This is known in social science as *triangulation*, a concept based on the idea that if you use more than one method to study one phenomenon you increase the validity of the study (Jick 1979).

The choice of methods depends on what we are studying and at what stage in the design process we are working. Generally speaking one can say that interviews, surveys and field studies are well suited for the first two steps (see fig. 1), i.e. when specifying and understanding who the users are, in what context they are going to use the system, and for what purposes. It is perhaps needless to say that interviews and surveys are not very well suited for producing actual design solutions. Hence for the last two steps, when designing and evaluating we need different methods such as focus groups, workshops and prototyping.

4 Agile and iterative development

Iterative approaches to development have been around since the early days of software development but have not been formalized until the last few decades. Iterative development is a central part of models such as Rational Unified Process, the Dynamic Systems Development Method and Agile methods, e.g. Extreme Programming and SCRUM (Basili & Larman 2003).

There are many reasons why the classical single-pass waterfall model is still popular, perhaps especially amongst managers; it gives you a feeling of knowing exactly what the end result will be, how much it is going to cost, etc. However, at the same time as the advantages of iterative development are becoming better documented and more widely spread, the shortcomings of the single-pass, document driven model are becoming clearer. Many of the iterative methodologies have, in fact, evolved from bad experiences of water-fall projects (Basili & Larman 2003).

The biggest problem of the water-fall model is failure to adapt to discoveries made during the design and implementation phase. There is little or no room for adjustments once the analysis phase is over. Also, the demand for constantly evolving products and services can be hard to meet with such a rigid model. If the duration of a project runs over a significant amount of time, chances are that the desired end-result changes during the project. Simply put: the waterfall model does not take into account the fact that conditions and goals change over time.

Agile software development is a generic term for a number of methods using *iterations* to avoid the problems described above. One iteration usually last from one to four weeks and optimally comprises planning, analysis, design, coding and testing. One of the main points is that evaluation is done many times during the project, which enables change in direction and/or redesign at an early stage, saving time and resources (Schwaber & Beele 2002). The key principles of agile software development are found in the so called Agile Manifesto. It states that priority should be given to:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan (Agile Alliance 2001)

Since iteration in design is a key principle of UCD, there is much to gain by employing an iterative approach for implementation as well. It makes the developers more aware of the changing nature of the project, but also lets them give feedback to the design process. A focus on communication (sitting together in one room, not communicating via paper, etc) also lowers the participation threshold.

5 Case study: LIBRIS – the Swedish National Union Catalogue

In this section we will describe the development of a new search service for the Swedish National Union Catalogue, LIBRIS. The different methods we have used for the user-centred design process are described and exemplified as well as the model for the iterative development. The focus is on these two aspects and therefore, this section is not a report on the entire LIBRIS-project. It should also be pointed out that the user-centred design process as described here leaves out many important aspects concerning interaction design and user experience. For example we do not touch upon visual design and communication.

In the project the methodology was not formalized from the start, but rather it evolved with the project. Especially the iterative and agile development is something that we have adopted as much by experience as by actually studying documented methodologies.

5.1 How we worked user-centred

Even though we have not followed the ISO-13407 standard to the letter, we have worked according to many of its concepts and principles (see section 3 User-centred design). Firstly, the design process has been iterative, as described in more detail under the section about iterative development; secondly, the project group has been multi-disciplinary, including experts on:

- Design (visual and interaction)
- Engineering (system architecture, databases, interface programming etc.)
- Human computer interaction (HCI)
- Format (metadata)

We have also, as far as possible, included the users in every step of the design process, even though the degree of involvement and in what ways the users have participated have varied during the project. This reflects that the design process is an ongoing and changing process. We will, however, in the next section discuss what we could have done differently. The main components, in terms of empirical methods, of our user-centred design process consisted of a survey, a focus group/workshop, and usability testing. We have also conducted interviews with researchers and library colleagues and worked in close collaboration with a reference group.

When talking about user-centred design we have sometimes come across the misconception that it would imply some sort of "least common denominator solution" or "design by majority vote". However, this is not the case. The role of the project group is by no means diminished when users are being involved. On the contrary, the work of conducting and analysing tests, focus groups, workshops, surveys etc. make great demands on the development team. And in the end it is up to the team to make informed decisions on how the system best can meet user preferences and demands.

5.1.1 *Survey*

The ISO-13407 standard states that the first step in the design process is to understand and specify the context of use. In order to get this background information we conducted a user survey and a focus group/workshop. The user survey was publicly accessible from the start page of the old version of LIBRIS, and during seventeen days in November 2006 we got 873 replies. The survey consisted of multiple choice questions in combination with open questions. The selection was uncontrolled and therefore the results can not bee seen as representative for LIBRIS' users. Rather, the survey aimed at gaining general background information and finding out what some of our users expect from a new version of LIBRIS. One of the most valuable outcomes of the survey was the suggestions for improvements and new features that showed up in answers to the open questions.

5.1.2 *Focus group*

The focus group/workshop was conducted in November 2006 and aimed at gaining some more qualitative data. The group consisted of seven persons: one pensioner, three doctoral students and three undergraduate students. The participants were recruited by postings at libraries, universities and email send lists.

The workshop consisted of two parts; the first part concerned search behaviour and was conducted with a scenario-based method. The scenario was about an undergraduate student who had to find literature for a paper. The participants were divided into two groups with the task of coming up with a strategy for the student.

The second part of the workshop consisted of a design task, were the participants were to come up with features they would like to see in a personal login area, and how this area should integrate with the rest of the system. The groups presented their suggestions for each other and we discussed pros and cons of the different designs. Even though the login area did not become part of the release the workshop was valuable for getting an understanding for the users and also serves as background material for a coming project that will design and implement a personal area.

In general the focus group was a good method for getting qualitative data, and for going into detail with some key concepts. However, the composition of the group is very important, both when it comes to profile and personality of the participants. It is also important to have a good moderator who is able to make sure that the discussion does not go off topic nor get stuck on one subject.

5.1.3 *Usability testing*

During the project we have had two more extensive test periods. The first was conducted on a so called vertical prototype, i.e. the functions we wanted to test were almost fully implemented, whereas the rest of the interface was inaccessible. The other test version was the aforementioned beta version.

These tests were controlled and took place in the office premises of the National Library. The subject was seated by a computer and the test leader orally presented the tasks. We used a so called think aloud methodology were the subject is being instructed to talk out loud about how he or she perceives the system and how he or she reasons when performing the tasks. The tasks were alternating open ended and more specific. The screen was filmed with a dv-camera which also recorded the sound. In connection with the test we also conducted an interview. The interviews had two purposes: firstly to get some background information about occupation, age, accustomedness to new web services, etc, and secondly to discuss in more detail how the subject had experienced the system. This also gave the opportunity to ask about desirable functions and other things that were not part of the test itself.

There are many methodological problems with controlled tests such as those we conducted: the formulation of the tasks strongly influences the subjects, lack of real

motivation to perform tasks, unfamiliar environment, stress from being observed, etc. On the other hand you have the possibility to pinpoint certain parts of the system and functions that you want to test.

In order to get some real use data we conducted a more naturalistic test in a library environment. The test took place at a university library and we asked visitors to conduct their searches in the beta version LIBRIS in stead of the local OPAC or the old version of LIBRIS. The task was therefore in large determined by the subjects and there were a real motivation for searching. This test mainly aimed at studying how users approached the system when seeing it for the first time, and how the basic search functionality worked. We also conducted tests on a couple of librarians to see how the system was used by expert users.

One important question regarding usability testing is how many test subjects you need. This depends on what you are testing and how heterogeneous the users are. However, Nielsen (2000) argues that when it comes to discovering usability problems it is sufficient with quite few test subjects (he mentions five) in order to discover most of the problems. He also advocates running many small tests throughout the design process, rather than having one big test period. We used four subjects for the prototype tests, seven for the controlled beta tests and eight for the naturalistic tests conducted in the library. We feel that these subjects were sufficient to test the basic functionality and work flow of the system, though we could have used a couple of more subjects for the prototype tests. What we feel is lacking, in terms of testing, is a longitudinal study that examines how the system is being used by accustomed users and how it functions as a working tool.

5.2 How we worked with iterative development

Our focus on flexibility extended both to the physical environment and to our development environment. Turnaround time in any part of the project needed to be short, be it making changes to the server configuration, fixing a bug or redesign part of the interface. It was vital that the different competences needed for these tasks were part of the project team.

5.2.1 *Flexible development environment*

In order to facilitate rapid communication and knowledge sharing between project members, a separate project room was acquired. This room was big enough to hold all project members, including any consultants and was also used for project meetings and reference group meetings. It was, in essence, the heart of the project. However, the room was not available to us exclusively. This meant that we had to find a way to construct and deconstruct the project room, sometimes on a daily basis. Increased knowledge sharing also lead to increased security due to the fact the project itself did not stand or fall by one single project member.

It was also important to get the right tools for the job when it came to development. We needed to be flexible, both in terms of software and hardware. We used Open Source tools almost exclusively and applications were deployed on Apache Tomcat servers.

The project team included members with experience in Unix maintenance which meant short turnaround time for server related bug fixing and reconfiguration. When needed, changes could often be made on-the-fly.

Since almost all development was done in Java, we could choose any platform for developers. However, since the application was deployed on hardware from Sun Microsystems a Unix-based development environment was considered optimal. We chose iMacs from Apple Inc. running Mac OS X. The iMac has the advantage of being in one piece (computer + display) and thus can be effortlessly moved, meaning that we could assemble or disassemble the project room in a matter of minutes.

We had, by design, almost no external dependencies in day-to-day communication on people or hardware outside the project group. This lead to low information latency, making most decisions fast and easy to make. An external consultant was used for one task: building

browser independent CSS. Since it was vital for us to understand and learn about the work of the consultant, he was integrated into the project rather than being an external partner. The consultant was in essence a part of the development team, albeit for a shorter time.

5.2.2 *Iterative development*

As mentioned before, we did not have a clear view of any particular iterative development method before starting the project. We did have enough experience to understand, that in order to realize our goal of continuous development, we would have to adopt some kind of iterative approach. However, in retrospect, we realize that we have fulfilled most of the individual parts of the Agile Manifesto (see section 4, Agile and iterative development):

- Individual interactions over processes and tools – the shared project room made communicating easy. Our development process was deliberately malleable from the start.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiations – meetings with the reference group throughout the project.
- Responding to change over following a plan – we revised our plan continually to meet changing requirements during the project.

The first version that could be used to test some functionality was produced two months after the project started. After seven months of development, about half-way through the project, we released the first version of the public beta version of the service. This gave us, in addition to user input, invaluable data concerning performance and statistics.

Since the iterative approach encourages continuous change, there is a risk of the codebase becoming unstable or that the same problem is solved in different ways depending on when it was implemented. Also, there is a tendency when working against a deadline to favour quick fixes in one place, and to not propagate these changes throughout the code. This is natural since the "gold standard" changes during the progress of the project.

To normalize the codebase we planned a three week long code review phase into the project. In the code review the whole codebase was put under scrutiny and discussions were made about the best way to solve a certain problem. Parts of the code were then refactored[2] to meet the requirements. Steps were also taken to improve the overall readability of the code.

All this, at this point somewhat theoretical, flexibility and security was put to the ultimate test one month before launch of the beta version. The computers used for development were stolen at the same time as two developers, including the technical lead, were at a conference in another country. It took the remaining project members less than two days to have everything up and running again (mostly time waiting for new computers to arrive). This proves that enough knowledge was spread among the project members. Even though strictly this has nothing to do with iterative development, we are convinced that the constant cycle of plan/design/implement/deploy helped iron out any hidden problems.

6 Discussion

The methodology outlined in this paper requires engagement from the project group; there is no room for hiding or not doing ones share. This can of course be strenuous at times, but in general we believe that it leads to devotion and a greater sense of satisfaction. Furthermore

[2] Code refactoring is revision and restructuration of source code intended to make it more streamlined and to increase its readability.

the tight collaborative way of working ensures that any problems that may arise are noticed at an early stage.

By working user-centred you get an understanding for the user, and thereby an understanding of why and for whom you are developing the system. This, we have found, is highly motivating, regardless if you are doing visual design, writing code or working with metadata specifications; if you know your users, you do not do the job by halves. Agile methods have been criticized for not taking usability into consideration, with a too narrow focus on deliverable code (Gulliksen et al. 2003). However, even though for example SCRUM does not say anything about UCD, we do not see anything inherent in the methodology that prevents a focus on usability (Schwaber & Beele 2002).

At the same time as the UCD process ensures an understanding for the users, the agile development model ensures that you can work in tighter loops. The two methods are so to speak reinforcing each other: the latter makes for faster development of functional prototypes, which are more easily communicated and tested, thus giving you better input for the next iteration, and so forth.

While UCD leads to an understanding of the users, an interdisciplinary project group gives an understanding for ones colleagues and their different competences. This is also, in our opinion, almost a prerequisite for working iteratively in tight loops. Our experience from other projects is that external dependencies slow down the process. In the LIBRIS-project we could implement new features in very small and effective iterations within the project group, doing everything from specifying and implementing metadata transformations to designing and implementing the interaction in the user interface. Furthermore, this could be done without having problems with delays from external consultants or development teams, sending specifications and bug-reports back and forth. And by communicating direct with each other the amount of intermediary paperwork and lengthy mail conversations is reduced.

It is important to stress the need for management support and trust when working with agile development. Since there is no final specification of requirements to show up front, before the project starts, you have to convince management that the methodology and process themselves will lead to a good product.

6.1 What could we have done differently?

Of course, there is no such thing as a perfect methodology; especially when trying new approaches you are bound to do some things backwards and run into difficulties now and then. However, during the LIBRIS-project problems and drawbacks have been surprisingly absent. That is not to say that the methodology could not be improved. Sometimes the work has been a bit unstructured and perhaps ad hoc. This is not necessarily a problem with the methodology itself, but rather a lack of planning and of making sure that each and every design decision is well founded. We also feel that, now at the end of the project, there is a lack of documentation. Surely there has been quite a lot of paperwork, but some parts of the system are quite obscure for anyone other than the person who wrote the code. This is also a result of lack of competence transfer between the programmers. Agile methods such as eXtreme Programming stress the importance of pair programming, and this is something that we have to work more with in coming projects.

The user-centred design process could be improved in many aspects, not least in making the users even more involved in the design process. During the project, a few persons had most of the contact with the users, and it would be interesting to conduct more collaborative design workshops with the entire project group and the users working together. The methodology of making the users take active part in the design process is called participatory or cooperative design, and is also known as "the Scandinavian model" (Bødker et al. 2000). Even though we had, from experience quite a good knowledge of the context of use of the system, we could also have done more when it comes to specifying and understanding this

context, e.g. by doing field studies. We also feel we could have devoted even more time for testing, especially by doing more naturalistic tests with real tasks in real contexts.

7 Conclusion

The fact that we at an early stage in the project had a version of the system with all essential parts integrated, in combination with the six months of beta-testing, made us confident that the system would work. The continuous user tests and user-centred design process also gave us confidence that the system would be well received by the users.

Furthermore, the iterative development model in combination with an efficient release-process put us in a position were we can easily update and add new features to the live version. This thus allows us having a service that can meet the continuously changing demands on web services in general, and on library systems in particular.

Finally, we would like to conclude that working with user-centred design in combination with iterative development is a *better*, *faster* and *cheaper* way of software development, compared to traditional models. Better – the product being released at the end is a more up-to-date and bug-free version than had we worked with a more traditional approach. Faster – it is our conviction that with traditional methodology we would not have finished on time, or at least not with the same amount of features implemented. Cheaper – if the same number of persons is able to do a better job in a shorter amount of time, it is a more cost-effective way of getting the job done.

References

AGILE ALLIANCE (2001). *Manifesto for Agile Software Development*. Webpage available at: http://agilemanifesto.org/ (2008-03-05)

BASILI, VICTOR R. & LARMAN, CRAIG (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer*. 36(6), p.47-56. doi:10.1109/MC.2003.1204375. Available online at http://www2.umassd.edu/SWPI/xp/articles/r6047.pdf (2008-02-28)

BATES, MARCIA J. (2003). *Improving user access to library catalog and portal information.* Final report (version 3). Available at: http://www.loc.gov/catdir/bibcontrol/2.3BatesReport6-03.doc.pdf (2008-03-31)

BØDKER, EHN & SJÖGREN, SUNDBLAD (2000). Co-operative Design – perspectives on 20 years with 'the Scandinavian IT Design Model'. *Proceedings of NordiCHI*. Available at: http://cid.nada.kth.se/pdf/cid_104.pdf (2008-03-28)

CALHOUN, KAREN (2006). *The Changing Nature of the Catalog and Its Integration with Other Discovery Tools*. Prepared for the Library of Congress. Unpublished. Available at: http://www.loc.gov/catdir/calhoun-report-final.pdf (2008-03-31).

GULLIKSEN, GÖRANSSON, BOVIE, BLOMKVIST, PERSSON & CAJANDER (2003). Key principles for user-centred systems design. *Behaviour & information technology*. 22(6), p. 397-409

ISO (1997). *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs). Part 11 – guidelines for specifying and measuring usability.* Geneva: International Standards Organisation. Also available from the British Standards Institute, London.

ISO (1999). *ISO 13407: Human-centred design processes for interactive systems*. Geneva:

International Standards Organisation. Also available from the British Standards Institute, London.

JICK, TODD D. (1979). Mixing Qualitative and Quantitative Methods: Triangulation in Action. *Administrative Science Quarterly*, 24(4), p.602-611. doi:10.2307/2392366

MAGUIRE, MARTIN (2001). Methods to support human-centred design. *Int. J. Human-Computer Studies*. 55, p. 587-643. doi:10.1006/ijhc.2001.0503.

NIELSEN, JAKOB (2000). *Why You Only Need to Test With 5 Users*. Webpage available at: http://www.useit.com/alertbox/20000319.html (2008-03-18)

NORMAN, DONALD A. (2002). *The design of everyday things*. New York: Basic Books. ISBN: 0-465-06710-7

PATTON, JEFF (2002). Hitting the target: adding interaction design to agile software development. *OOPSLA 2002 Practitioners Reports*, p. 1-ff. doi:10.1145/604251.604255. ISBN: 1-58113-471-1

SCHNEIDER, KAREN G. (2006). *How OPACs Suck, Part 2: The Checklist of Shame*. Blog entry available at: http://www.techsource.ala.org/blog/2006/04/how-opacs-suck-part-2-the-checklist-of-shame.html (2008-03-31)

SCHWABER, KEN & BEEDLE, MIKE (2002). *Agile software development with scrum*. Upper Saddle River, NJ: Prentice Hall. ISBN: 0-13-067634-9