A Usability Refactoring Process for Large-Scale Open Source Projects: The ILIAS Case Study

Agnes Lisowska Masson
Human-IST Research Center
University of Fribourg
Fribourg, 1700, Switzerland
agnes.lisowska@unifr.ch

Timon Amstutz
Teaching and Research, iLUB
University of Bern
Hochschulstrasse 6
3012 Bern, Switzerland
timon.amstutz@ilub.unibe.ch

Denis Lalanne
Human-IST Research Center
University of Fribourg
Fribourg, 1700, Switzerland
denis.lalanne@unifr.ch

Abstract
This case study presents efforts to introduce and encourage consistent application of usability and user interface design principles in an existing large-scale open source project. We present the project in question, the ILIAS learning management system, the challenges involved in incorporating usability into an open source project, the novel solution we proposed, a set of tools called the Kitchen Sink, and the steps that were needed to have the solution accepted and used by the ILIAS community. We conclude with a discussion of the lessons learned and an assessment of the overall success of our efforts.

Author Keywords
Usability; Open Source Software; Open Source Communities; Learning Management Systems.

ACM Classification Keywords
D.2.2 Design Tools and Techniques, H.5.2 User Interfaces, K.3.1 Computer Uses in Education.

Introduction
Poor usability is an oft-cited complaint when discussing open source software and its adoption. Efforts to introduce usability work and usability engineering

methodologies are often met with resistance for a number of reasons including developers' lack of understanding of usability [2, 8, 10], a lack of resources to focus on usability issues [2, 8], the general attitude towards contributions that are not code [2, 8, 9] and the internal organization and politics of the community [2, 9].

As part of the training and support team for ILIAS[1] - an open source learning management system - at a major Swiss university, one of the authors has been actively involved in the ILIAS community for a number of years by proposing features, being involved in testing, and attending ILIAS related conferences and events. Contact with regular users of the system (professors and students at the University) made him acutely aware of the usability issues that users encountered on a regular basis. At the same time, contact with and participation in the community gave him insight into both the development process and the attitudes and perspectives of the developers involved. As someone with an interest in usability, his experience put him in a unique position to try to find a way to integrate usability principles and practices in a more systematic and concrete way into the ILIAS project. Consequently, we launched a project to explore what the most effective way to do this might be.

The ILIAS Open Source Community

ILIAS, which has been in development since 1997, is an extensive and complex open source learning management system providing functionalities such as access control, ways to input and display content, communication tools and organization of user groups. It focuses on two user groups in particular – professionals (namely teachers who use ILIAS to provide content for external use) and consumers (the students who use the content). It has around 189 registered installations world-wide, accounting for approximately 1,000,000 users. However, due to the inherent nature of open source software, the number of actual users exceeds these figures by far.

The community is structured following the onion metaphor [4] in which the inner layers of the onion, populated by smaller groups of members, have more power than the outer layers, which tend to have more members. In the case of ILIAS, 3 out of around 30 core developers provide almost 80% of the code, while there are around 100 active community members, and thousands of users in the outermost layers (Figure 1).

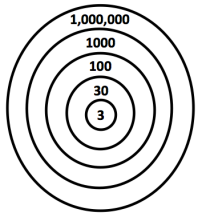

Figure 1: The ILIAS community represented through the onion metaphor. 3 main developers, around 30 core developers, 100 very active developers, 1,000 active users (teachers), 1,000,000 passive users (students).

Any new feature proposed for inclusion in ILIAS must be approved during the Jour Fixe (JF) – a bi-weekly meeting at which all decisions regarding code and features are made and in which only the core developers and the product manager can participate.

[1] http://www.ilias.de

This means that not every member of the community has the same role, power and influence over the evolution of the project. It is particularly important to note that the core developers and the product manager essentially have complete control over the project, creating an unbalanced and developer-centric structure.

Challenges for Usability and UI design

An examination of the literature [2, 3, 5, 8, 9 and 10 among others] revealed five issues in particular which seem to contribute to poor usability and UI design in open source software projects. These issues are described below, with a particular focus on how they manifest themselves within the ILIAS project.

Lack of trust

Like most open source communities, the ILIAS community functions as a meritocracy in which code submissions are almost always the indicator of 'merit'. Submitting code therefore becomes the most effective way to earn the trust and respect of fellow community members. This makes it very hard for those who do not contribute code, such as usability engineers and interface designers, to obtain a position within the community through which they can have a more powerful influence on the evolution of the software (i.e. a place in the JF meetings).

Lack of resources

In most cases, those who participate in open source projects do so on a voluntary basis, motivated by personal interests in the topics or technologies involved, and tend to focus on what is of interest to them. Resources in terms of labour are therefore not necessarily evenly distributed across project needs.

Even when a project does have a budget, such as is the case with ILIAS in which a handful of developers work on the project full time and receive a regular salary, the size of the budget limits investment to the development of functionalities rather than usability aspects of the software. Testing and bug reporting are almost exclusively done on a voluntary basis and are usually done by actual end-users of the software.

Lack of expertise

Many open source projects lack direct involvement of usability engineers and UI designers, and to the best of our knowledge ILIAS is no exception. The reasons that are most commonly given for this is that a) there are fewer usability and UI experts than there are developers, so the lack is to be expected, and b) that the distributed and collaborative nature of open source projects makes the application of traditional usability processes and methods extremely difficult.

Lack of guidelines and taxonomies

Usability and UI guidelines and taxonomies provide not only a reference resource for how to design and implement elements, but can also help structure discussions about usability problems and solutions, creating a common vocabulary to facilitate discussion and a common basis on which to build solutions. However, open source projects tend to lack both.

The taxonomy of modules and services within ILIAS includes UI elements in only a rudimentary manner – the name of the PHP class used to draw them. This leads to nomenclature problems within bug reports and discussions that can result in ambiguities or duplicate work. A simple example from ILIAS are navigational breadcrumbs, which are sometimes referred to as

breadcrumbs (56 instances in bug reports), and sometimes as *locators* (29 instances in bug reports).

Similarly, while usability and accessibility guidelines exist within ILIAS [6], they are very basic and decentralized (some are found in the feature wiki, others in bug reports), making it difficult to find them and to maintain a complete and coherent set.

Lack of tightness
Most open source software is full of features that have varying degrees of usefulness. This feature-bloat is commonly attributed to two factors. The first is that a large number of features can be considered a selling point for the software. The second is that features are often contributed as the result of personal motivations of developers or specific demands of individual 'clients' of a software. In most cases, features are never removed after they have been included since developers don't want to throw away code that they have spent time and effort developing, particularly when there is no incentive to do so. The consequence of this on usability is that any modifications such as usability improvements to one feature or functionality may cause 'ripple effects' that can adversely affect related features or functionalities. Given the fact that ILIAS has been in development for almost 20 years, it certainly falls prey to feature-bloat and problems of ripple effects since specifications or guidelines produced for a component have to be equally applicable in all of the different contexts in which that component is used.

Our Goals

Faced with the above issues, we had two primary goals in the project. The first was to provide the ILIAS community with a set of tools, which we dubbed the Kitchen Sink, which would:

- Encourage developers to trust and see the value of the work of usability engineers and UI designers.
- Act as a medium through which usability engineers and designers could actively and effectively contribute to the project.
- Provide a clear taxonomy of UI components within ILIAS as well as clear and effective guidelines for how to use them.
- Provide automated UI tests for at least some of the guidelines to address the issue of limited resources.

The second goal was to successfully introduce the Kitchen Sink into the community and encourage its adoption in the long term.

The Kitchen Sink

The Kitchen Sink that we created is composed of 5 key components, shown in Figure 2 and described below.

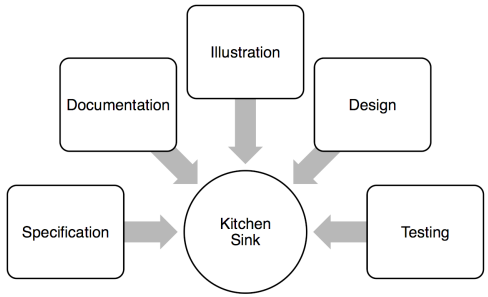

Figure 2: The components of the Kitchen Sink.

The *specification* refers to a taxonomy which would centralize all of the UI components in ILIAS and specify

how they are named, when they should be used, the relationships they have with other components, how they should look, and how to interact with them.

The *documentation* provides specific information about how each component can be used, including the implementation state and examples of best practice regarding how to implement the component, as well as background information such as the theoretical underpinnings of the component.

The *illustration* shows a concrete visual example of a particular component once it has been implemented.

The *design* lists the 'less variables' used for customizing the component.

The *testing* provides rules for each component, which include how strictly the rule must be adhered to (may, should, must) and, for a subset of rules, ways to automatically test whether the rule has been correctly applied in a new feature.

How each of these components in its final form was designed and created will be described in the 'Process' section, as the process and the final results are tightly intertwined.

The Process

We decided to follow a participative design approach, in which we considered the developers in the community as our clients, and worked closely with them to develop a number of aspects of the Kitchen Sink. We also tried to follow the tenets outlined in the Agile Manifesto [7], both as a way to structure our own work, and as a way to use a process that might be familiar to developers.

Phase 1: The Pitch

The idea for the Kitchen Sink (KS) was introduced to the ILIAS community for the first time during the annual Advisory Council meeting in February 2015. In this meeting one core topic is chosen that will be given special support throughout the following year. Although the KS was not chosen as the core topic, it did get a considerable number of votes and interest was expressed, including potential funding prospects.

Encouraged by the interest generated in the KS, we presented it at the Jour Fixe about a week later. The reaction at the JF was also positive, but concerns were raised about the effort that would be required to successfully complete the project. Two suggestions were also made which were not part of our original proposal: to closely bind the KS with existing guidelines and to include examples of existing ILIAS classes.

Phase 2: Getting to work

Once we had a green light from the JF to start working on the project, we gathered a select group of developers who had supported the idea from the start and began a series of discussions with them.

One of the primary results of this work was the implementation of a static webpage (example: supplementary material KSPrototype1.png) which contained all of the most important UI components in ILIAS and which would serve as an example of how those components should be used. We considered this webpage to be an early prototype of the Kitchen Sink.

We made two mistakes during this stage of the project. The first was that we fell prey to feeling the need to introduce significant changes to prove the 'merit' of our

work and justify our project in order to gain the trust and respect of the community.

The second mistake was that we were overly focused on the 'build projects around motivated individuals' principle of the Agile manifesto, and not enough on the 'deliver working software frequently' principle. The result was that while we had a number of productive discussions with our chosen set of developers, the prototype wasn't shown to the rest of the community until later than anticipated. When it was finally presented at the ILIAS developer conference in September 2015 it was met with mixed reactions.

While there was clear interest in the project and the fact that something was being done to help improve the UI, a number of concerns were raised about the prototype itself. Most notably:

- There was skepticism about the big changes.
- There was doubt that funding could be found.
- There was criticism about the slow pace thus far.
- There were complaints that components only worked for some modules and services but not others.
- There was disagreement about the proposed process, specifically worry that it could slow down development of the software in general.

Our overall experience at the developer conference left us in a bit of doubt about the future of the project, but a veteran power user who had been working with ILIAS for over 15 years and who had done some usability work within the project in the past offered to commit to the project if we were willing to make some changes in the disputed ideas and concepts. We were grateful for the support that this person expressed and accepted their offer, launching into the next phase of the project.

Phase 3: Starting Over

Working with the veteran user had a number of advantages, not the least of which was that given her long-time experience and in-depth knowledge of the project, she was able to quickly pinpoint where proposed rules might cause significant ripple effects and propose alternative solutions. This type of insight was invaluable for reducing the time and effort that went into creating the next version of the prototype.

However, it also meant that we would have to change our initial approach and let go of some of our original ideas and concepts, which was much harder to accept. In hindsight though, this experience was particularly enriching because it put us in the same position as that into which we might be forcing some developers – having to change their original designs and ideas into which they had invested time and effort – which gave us a greater sense of empathy.

Two dramatic changes were made in this phase of the project. The first was to change the approach itself from a more traditional, holistic, top down approach in which we proposed major redesigns at the global level, to a bottom up, more atomistic, approach, starting with small refactoring of existing UI structures. While it is often advised to follow a more holistic approach in usability and UI design work we felt that in our case, given the obstacles to gaining trust within the community, the amount of legacy content, the limited resources and the large end-user population that would be particularly affected by the changes, smaller refactoring was a more practical and judicial choice.

Creating a more consistent UI through small steps at the component level would in and of itself already increase the coherence and usability of the software, and would seem like a less daunting task for the developers, thereby increasing the chances of the changes being accepted and applied within the community.

The first step was therefore to catalogue the existing UI components of ILIAS, and to make concrete proposals for improving the usability of those components at the micro, rather than the macro level. This resulted in a taxonomy containing over 130 entries, with a total of 361 guidelines for how to improve those components.

The next step was to develop a template for KS entries. The purpose of the template (supplementary material KSTemplate.png) was two-fold. On one hand it would be used to make clear which information was needed in order for a new UI element to be added to the KS and to be considered and approved during the JF. On the other hand, it would introduce a much-needed coherence for how components should be described, making it easier to understand how to use and implement the component in practice. The template was presented at a full-day workshop in November 2015, and received a positive reaction.

Phase 4: Acceptance

In December 2015, the Kitchen Sink was officially accepted as a standing item in the Jour Fixe meetings. This means that a fixed time slot is available at each JF meeting to discusses new proposed KS entries.

Over the next two months, 11 new entries were accepted, while only 2 were rejected. Although this number may seem small, there are two important and related points that must be considered, and which in our minds make this slow pace acceptable. The first point is that on average the discussion of a single entry takes around 30 minutes. The second is that these discussions are themselves a valuable contribution to the community. They help to sensitize and inform developers about UI design and usability principles in general, which we feel will in turn make it easier to pass new entries in the future and give developers some preliminary tools to themselves be able to spot potential usability problems as they are working, rather than waiting for user feedback once code has already been developed.

Phase 5: Future Work

Discussions in two further workshops, held in January and February of 2016, revealed that the developers would like a closer integration of the Kitchen Sink with their code and their daily output. As a result, two new sub-projects emerged. The *Centralizing UI Components* sub-project uses the generated taxonomy and guidelines to refactor the classes that render the UI. The outcome will be a framework that allows developers to ask for a UI component to be rendered by passing the required data, without them needing to know what the component actually looks like. The outcome of the second sub-project, *KS-Integration in Layout and Styles*, will allow designers to generate a new skin for ILIAS by only changing a few design related variables of a specific UI component directly through the frontend.

We have also started to create automated testing to complete our Kitchen Sink and to help deal at least

partially with the problem of lack of resources. We have developed automatic tests for 69 of the 361 guidelines that have been proposed thus far within the project. Most of these 69 guidelines deal with style, use, accessibility, or composition. The automated testing aspect of the project is still in the early stages of development, but details about the work done thus far can be found in [1]. We do not pretend that the automated testing is an ideal solution, and it should by no means diminish the role that usability experts should play within a project. As our own work has shown, the rules that can be automated tend to be relatively simple and black and white, while many usability questions are much more complex and grey. Moreover, the automated testing itself would have to be maintained and updated as the project evolves, which also requires resources, posing somewhat of a chicken and egg problem.

Lessons Learned and Discussion

Over the course of the project, we learned a number of important lessons that may be of use for others who would like to attempt a similar endeavor:

- Deliver frequently updates about progress to the developers – this reassures them and gives them a chance to flag potential problems early on.
- It helps to have a veteran community member who knows the project inside out to help point out potential problem areas and ripple effects early on.
- Keeping things simple is crucial for enabling less involved community members to participate in discussions.
- A slow pace can be the key to success - it gives a larger part of the community the opportunity to participate, and to digest and accept guidelines.
- Although a holistic (top down) approach to usability and UI design is preferable, an atomistic (bottom up) approach may be needed to make it easier to gain trust…smaller changes are easier to understand and accept, and their consequences are easier to predict, making the process more reassuring.
- Allowing for easy integration of the work into the established development process of the community helps facilitate acceptance and uptake.

Outwardly, it may not seem that the project was very successful. There has been no grand redesign of ILIAS. The quantifiable results of just over a year's work do not seem particularly impressive - 130 entries for UI components governed by 361 guidelines, only 11 of which have been officially accepted, and only 69 of the 361 guidelines can be automatically tested thus far. However, we did manage to surmount one of the most difficult obstacles - gaining the trust of the developers and the community. Not only has the Kitchen Sink been added as a standing item at the JF meetings and its accepted entries are now binding for the whole project, but, as a direct result of his work on the Kitchen Sink project, the author who initiated the project (described in the introduction) has been asked to be a member of the Technical Board of ILIAS, participates regularly in the JF meetings and is now in a position to have a big influence on ILIAS development, even though he still doesn't do much coding. This to us is perhaps the biggest success of the project. There is finally a usability advocate within the community in a position to make a difference.

References

1. Timon Amstutz. 2016. *Usability Refactoring in Large Scale Open Source Projects – A Case Study.* Master's Thesis, University of Fribourg, Fribourg, Switzerland.
2. Morten Sieker Andreasen, Henrik Villemann Nielsen, Simon Ormholt Schrøder, Jan Stage. 2006. Usability in open source software development: opinions and practice. *Information Technology and Control*, 35, 2: 303-312.
3. Calum Benson, Matthias Muller-Prove, and Jiri Mzourek. 2004. Professional usability in open source projects: GNOME, OpenOffice.org, NetBeans. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems* (CHI EA '04). ACM, New York, NY, USA, 1083-1084. DOI=http://dx.doi.org/10.1145/985921.985991.
4. Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2. Online. http://firstmonday.org/article/view/1207/1127
5. Jim Hall. 2014. *Usability themes in open source software*. PhD Dissertation. University of Minnesota, MN, USA.
6. ILIAS Development Guide. 2016. Retrieved December 29, 2016 from http://www.ilias.de/docu/goto_docu_lm_42.html
7. Robert C. Martin. 2003. *Agile software development: principles, patterns and practices*. Prentice Hall, Upper Saddle River, NJ, USA.
8. David M. Nichols, Michael B. Twidale. 2003. The Usability of Open Source Software. *First Monday* 8, 1. Online. http://firstmonday.org/article/view/1018/939
9. Mikko Rajanen and Netta Iivari. 2015. Power, Empowerment and Open Source Usability. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (CHI '15). ACM, New York, NY, USA, 3413-3422. DOI: http://dx.doi.org/10.1145/2702123.2702441
10. Michael B. Twidale, David M. Nichols. 2005. Exploring usability discussions in open source development. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05)*. https://doi.org/10.1109/HICSS.2005.266