

Launchpad's Quest For A Better And Agile User Interface

Martin Albisetti

Canonical Ltd, User Experience and Design
martin.albisetti@canonical.com
<http://www.canonical.com/>

Summary. Introducing change into an established team with an aging code base is always a challenge. This paper is about my experience in shifting the Launchpad teams' focus to a more user-friendly mindset, evolving the existing processes to encourage constant improvements in the user interface and teaching developers to see their work from the users' perspective. We went from nobody owning the user interface to the developers feeling strong ownership over it, and I describe the ups and downs we went through as a team, what worked, what didn't, and what we could have done better.

1 Introduction

When I started working with the Launchpad team I was tasked with designing and rolling out a new interface using cutting-edge technology on a well established product and team. The existing processes and team structure made it very hard to roll out big changes while also ensuring consistency as time went by.

While the general designs and work flow changes were being fleshed out, I started to drive some change to the existing processes, enabling me to be successful at an objective that would take a year to accomplish, and unexpectedly, beyond that.

The project was 4 years old and had over 500 dynamic pages with different templates and layouts that had been left untouched at different points in time. The goal for the next year was to make the application easier to use, even enjoyable. I also had to make the UI consistent across the board, take the project from static HTML pages into the wonderful world of in-line editing, streamlined work-flows and predictable interactions. In parallel, fundamental features that had been developed were going completely unused and we needed to turn that around. A re-usable AJAX infrastructure had to be developed from the ground up, new features needed to be designed and delivered, and general navigation issues needed to be addressed.

However, this story isn't about the success of the roll out of a new interface, but rather the success in the process changes that evolved during that year and how the project went from nobody feeling ownership over the user interface, to the developers taking strong ownership.

2 Project background

Some changes in the process were particularly challenging because the Launchpad team is a fully distributed team of 34 people, across 9 different timezones, grouped into 6 sub-teams that focus on specific parts of the application. This involves a lot of people clustered virtually into small teams, so communicating project-wide changes is always hard.

It's an open source web-based project that serves as a tool for other open source projects to collaborate and manage their projects. It is also the key infrastructure behind Ubuntu, the popular Linux distribution, which means there are many stakeholders involved, at times with contradicting requirements.

The application is pretty large with more than 350.000 lines of Python and Javascript code, over 2 million visitors a month and does daily rolls out to an edge server which half of the most active users use. Roll-outs to production happen every 4 weeks, so changes are delivered to users on a frequent and regular basis.

3 The road to UI ownership

3.1 Nobody owns the UI

The project had an amazing team of highly skilled and passionate software engineers. They focused on solving hard engineering problems while delivering features to users as quickly as possible. Everyone involved was a strong believer in agile processes, and proposals to evolve existing processes come up all the time with a general good predisposition to change.

While the team uses it's own application to develop, nobody uses all the features, and many times not even the ones they develop as they don't relate directly to the work they do.

One side effect of everyone being so narrowly focused and driven to release as early as possible, was that both the small details and the overall result frequently got overlooked. It's hard for a developer to justify on his own spending an extra day working on a feature so it's consistent with a different part of the application. That is, of course, if they know of the existing pattern at all.

The lack of consistency and polish created a general feeling with users that the application was hard to use, slow, and changed too often. Developers felt frustration as well since a lot of their hard work wasn't being appreciated.

The end result of this was that nobody felt enough ownership over everything that users interacted with, but rather specific features or pages, and focus was more on functionality rather than usability. Nobody felt like they owned the user interface.

The first step was to introduce user interface reviews at the point where the code was proposed for inclusion in the main code base. This mirrored the existing process of code reviews, where all code had to be reviewed by someone else before it was added to the code base, although this had a minor twist to it: they were optional. Each developer decided at that point if they needed a review

or not. As you will see in a graph later on, this was not a very popular service to take advantage of. It turns out that developers don't normally like to go back and change code that is well tested and already proposed for inclusion.

While this was going on, myself with others outside of the team started to work on the general plan for the UI for the following year. It was mainly a set of interactions built as movies so we could present them individually to each sub-team and ensure that the end result would be consistent. It was also the time where I talked to existing users a lot in order to understand what their frustrations were, and try to prioritize the changes in a way that would address the biggest amount of users.

Some informal user testing was also performed, accompanied by reviews of existing work flows. I also started using more parts of the application in order to get a better sense of what our users were feeling.

3.2 I own the UI

Having spent a few months planning and researching, I was quickly centralizing most of the information involving features, work-flows and usage patterns.

The whole team had a non-development 2-week sprint where I presented the general plan, short movies illustrating examples of what we wanted to do and the technology that was chosen to achieve it. We also started figuring out how we were going to achieve this on a technical level. The movies were very effective in conveying what we were aiming for. They were developed in Flash, showed a specific interaction on a portion of the page, and varied from 5 to 30 seconds in length. After that sprint, the team started working.

The fast pace at which development started happening made it very hard for me to ensure everyone was going in the right direction, so the optional UI reviews were turned into gate-keeping tool by making them mandatory. So for every piece of code that needed to be done, it had to have a code review signing off it was a good approach at fixing the problem, and a similar sign-off that the change that it introduced affected the user experience in a positive and consistent way.

A UI review would happen, for example, when a developer wanted to introduce a new feature that would let users subscribe to a certain event, and get notified via email when something changed. The review focused on where on the page this option was presented, with what text and icon, and the content of the email sent. The new option needed to be introduced in a way that made it clear what it would do. To address this, I would provide recommendations on how to do this, and work with the developer until we were both happy with the outcome. For example, here's a quote of something that would happen preparing for a UI review:

“Preparing for the UI review I discovered I neglected to put the batch navigator links into the page template. I've done that and updated one of the factories to allow a release to be re-used.”

While it was a very effective at helping in the overall objective, it created some friction and frustration with developers who had tight deadlines and code they deemed ready to send, but were pushed back and in some cases needed days of work in order to be re-proposed. One example of this was a feature that had been delayed due to some unexpected complications in the code, and when it was proposed it had already doubled the time allocated to work on it, and yet the UI was not in an acceptable state. When I requested some changes to it the developer got very frustrated and raised the issue with his manager in order to get an exception.

Some of the frustration was addressed with negotiation, trading certain flexibility today, in exchange for some more complex work in the future. Another approach was to offer multiple solutions to the same problem, and work out together which would be the best balance between amount of work and best user experience. It can be a mix of both as well - "do this simple change today, but commit to developing the more complex one in a certain time-frame". Another reason for the frustration was that I sometimes became the bottleneck in the process, simply delaying work from sending it due to lack of time.

A key tool to remove friction also came inspired from what was done with code: pre-implementation meetings. Developers would bring up on a call or an email what work they were going to do, and we would work out together how it would look and behave. This made the UI review process flow much quicker as the biggest potential problems were usually addressed before the work was started.

Since all the conversations around UI were happening in a central place where I was always involved, it became easier for me to re-use existing patterns with new code, and gave me a place to plug into the process to slowly increase the general quality of the UI and raise the bar of what a minimum acceptable user experience was.

Five months into the process I had managed to sustain a reasonable speed in UI review turnarounds, as well as provide support in developing features, but this didn't completely address the bottleneck. The team was planning to focus entirely on the UI for 1 month. There was no reasonable way for one person to be able to deal with that amount of incoming change, and on top of everything, my vacations were coming up! We needed a new plan that would allow us to continue with the process that had already started to produce good results, but was scalable.

3.3 They own the UI

While the process evolved, some developers had taken a special interest in UI patterns, rules of thumb for interactions and general design issues. The interest was so great that they were sometimes pointing out things I had missed in my own reviews with other developers or even the ones I did for them. It took me by surprise at first, but I quickly realized that I had found the solution to the heavy dependency on my day-to-day availability, and started looking into interesting places to go on vacation.

I picked 3 of the developers who had shown the most interest and good intuition, and talked them into experimenting with doing UI reviews for other developers, with help and support from myself. Interestingly enough, this mentoring process maps to the same process the team has for code reviews, where a new developer gets assigned a mentor that oversees all their reviews and makes sure they understand what they need to look for and how to ask the right questions in order to become an approved code reviewer.

The mentoring phase was very interesting. Developers had gained so much experience with UI reviews from the other side of the table that it took them only a few days until they were covering all the major issues by themselves.

What was really unexpected was what happened when we decided to start to bring in more developers into this new “meta-team”. Against my better judgment, I brought in a few developers who had not been good at delivering UI but for some reason had an interest in participating in this experiment. After performing only 2 UI reviews, the same developer who had a hard time with changes to the interface, started to deliver code that showed thought and polish in the user experience. I was puzzled until one of them told me that doing a single UI review had completely changed the way he saw his work. Seeing the work of others from the perspective of somebody who doesn't care how brilliant the code is, but rather what was being used by people, seemed to have a profound impact on developers.

September was the month in which every single developer was going to work only on the user interface, to update current UI to the new design and layout that we had been building for months. Seeing how well developers had started to take over my daily tasks, I decided it was the perfect time to get lost in Europe on vacations for 2 weeks. Sometimes an image is worth a thousand words, so here's a graph of the number of UI reviews performed per month, separating my reviews from the ones from the developers:



Fig. 1. This shows the number of UI reviews performed per month, distinguishing reviews done by developers from the reviews done by me

That cycle of work was a success. All the changes that they had set out to make were done. By the end, the developers knew more about the pages and patterns that I had developed than I did. They had dealt with all the problems

that had arisen from going from pictures and examples to actual code, had discussed them, made decisions, and even done a few iterations.

Developers started to feel very strongly about their UI, a quote extracted from a review portrays this well. Note the use of passionate words like “I am unhappy”, and the person taking the advice agreeing that the extra work made it better:

Developer reviewing UI: “I create two branches in dev, then proposed a merge. I ran make sync_branches. The toggle works well, *but I am unhappy with the spacing between it and the download link. We use <ul class="horizontal"> in many places to normalize the space between user actions on a page (1.5em). I am not sure that this can be reused in the <div>. Can you experiment with the markup/CSS to make the spacing consistent with the rest of Launchpad?* ”

Developer proposing the code: “I've done this, and attached an incremental diff. I agree this is a bit nicer. ”

The trend of developers performing more UI reviews than user experience people continues to hold, the team feels it's a process that adds more value than it costs, and other teams across the company have started to adopt it.

In the future, I'd like to start working with all the teams to start performing cross-team UI reviews to build more shared knowledge across different applications and increase coverage of the reviewers available. I believe the process described here enables communication at the right times, and allows user experience practitioners to be able to hook into the development process in a way that fits in well with Agile software development.

4 Conclusion

This story covers a year and a half, quite an extensive period of time. If we had set out to implement this process from the beginning, I'm certain it would have taken only a few months, and some initial experience with other teams seems to indicate that adoption can be achieved in a shorter period. If you would want to implement this process, I would recommend:

- Don't recommend UI reviews, make them mandatory
- Be involved at the management level, it helps to be in line with the overall project strategy, and it's a good forum to bring up big changes (strong leadership can sometimes make up for lack of management involvement)
- Make sure that you can generally give immediate feedback. Having a very quick turnaround lowers the barrier for a developer to bring up issues before they get to the review stage
- The person who reviews the code, should not review the UI. They will understand the complexity behind the implementation and have a hard time proposing changes

- The initial adoption of the process will likely create friction, be prepared to deal with it and provide quick solutions to the problems created
- Be aware that you will need to negotiate: First bring up the problems with the proposed UI, then and propose more than one solution to the problem. The negotiation step also helps to gradually raise the bar of what an acceptable user experience is
- Pre-implementation calls and/or meetings removes friction created in reviews. Encourage these as much as you can, it will help everyone
- Build the specification of the UI with the developers. It will give them ownership over the feature and enables the experience to evolve beyond your initial concept

Something I regret not having done was documenting good patterns across the application as they came up in reviews. It would of made developers even more self-sufficient, and even would of reduced the time I had to spend in the end documenting the existing patterns.

Another success of this process was that it helped maintain the UI to an acceptable level through time and avoided the need for a big re-design, which at that point had been previously undertaken every year.

5 Acknowledgements

I would like to thank Charline Poirier for her unvaluable help in writing this paper, Francis J. Lacoste and Christian Reis for being so supportive and helping me drive this process, Angela Martin for all the encouragement and Elizabeth Whitworth for her guidance in writing this paper.