Increasing support performance through documentation optimization

A case study

By Noah Lloyd-Edelman

<u>Introduction</u>

Imagine you're a company with an incredible product you wish to share with the world. You've been polishing your code, growing your customer base, and refining your marketing campaign. People are starting to use your product, but you want it reach a wider audience - and not just in the English-speaking markets.

Transifex's support staff was having trouble keeping up with its growing customer base. They were spending too much time answering questions rather than improving the platform and developers were often distracted by support-related material. Our goal was to help customers help themselves by intelligently updating the support material and making sure we were answering our customer's actual needs.

<u>Enter Transifex</u>

Transifex was created to help internationalize digital content by creating a platform to easily manage, collaborate, and of course translate digital content. As a developer, you can hook your product into the platform, upload your content, collaboratively translate it, then export the finished files as quickly as you can translate them. There's no need to depend on a translation agency because experienced third-party translation vendors are built into the platform. With a multitude of translation utilities and supported file formats, most content can be directly imported and exported from the user's platform. Transifex can even collect and localize websites with a single JavaScript snippet.

Transifex (based in Menlo Park, California and Athens, Greece) was formed in 2007; and has been steadily growing since then.

<u>Jumping in</u>

Internationalization was new territory for me. With Transifex, I needed to understand both the ins and outs of the internationalization industry with a high degree of detail, as well as the software and support platform. Before I even thought about sketching an idea or pushing a pixel, I dove into the deep end of Transifex's day-to-day activities. I lived in Greece for a month, camping out with the developers, crashing product meetings, and learning as much as I could about the company and product.

Transifex to-do list:
1. Interview Transifex employees extensively
2. Create a Django application using the Levenshtein formula (it compares the similarity of two words)
3. Begin answering support tickets and calls
4. Begin participating in engineering sprints

<u>Understanding the problem</u>

Internationalization is a complex process that can be difficult to explain, especially when each solution is unique for each company and individual. The Transifex team was excited about its international audience, but it did not clearly explain the processes involved, nor the options.

Customers needed to better understand what the company was offering and how they could go about using those resources.

I gathered an incredible amount of insight from the insanely knowledgeable development team. I also realized the customer care and sales reps were super in touch with how prospects and customers perceived and used Transifex. They were able to help me understand what was currently working and what areas needed my help.

Understanding the problem:
- How do we demonstrate the benefits of presenting a product to an international audience?
- How can we make the complexities of Transifex easy to understand?
- How quickly and efficiently can we get users up-and-running?
- How can we inject personality into the documentation?
- How do we exhibit Transifex's reliability and care?

A fragmented foundation

Because of its open source roots, Transifex has been built by hundreds of developers. Instructions and code were written during different parts of the product's life-cycle and documentation was updated sporadically to include new features and large updates. It was largely unmaintained though — the result was fragmentation of the product.

Troubled areas:
- There were no analytics installed on the documentation website
- The varied writing styles of its many authors was inconsistent and hard to follow
- Many sections of the documentation contained outdated material
- The search engine rarely returned a relevant results

Creating goals

After my initial information gathering session, I met with the other customer support agents and we laid out a set of guidelines to follow and goals we wanted to achieve:

- Work in short, agile sprints
- Handle technical debt first, then focus on stability, and then improvement
- Create and test hypotheses, with measurable outcomes, and share the results each week

Our goals were as follows:
1. Begin collecting analytics for an analysis baseline
2. Update all documentation pages, utilizing a uniform and friendly writing style (with many examples)
3. Establish stronger communication between the Athens and California office
4. Consolidate all company knowledge into the documentation
5. Maintain a current FAQ, based on support feedback
6. Improve the search engine's heuristics

Roles and team members

I worked in California with Transifex's sales and logistics teams. My main duties included researching, installing, and monitoring analytics, updating the websites codebases, answering support tickets and calls, editing documentation, writing guides, attending sales and company meetings, and occasionally updating Transifex code.

Elena and Marilena were in Athens (a nine hour time-zone difference) and were the other two "customer success" managers I worked with. Elena was an editor, and spent most of her time answering tickets, wrangling developers, and was my main point of contact. Marilena worked directly with the developers and spent most of her time working deep inside the platform's code.

Creating a design baseline

Research

I interviewed the developers and product managers to learn about the company's roadmap for the next several months. This allowed us to publish support material as soon as features were launched, and to make our own changes to the software when there were lulls in development.

To see what support alternatives existed, I began researching frameworks and scouting other company's platforms. I also read as many style guides and case studies as I could get my hands on. We were working with a limited budget, so reengineering our current systems was preferred to buying an entirely new one.

Strategy

While I was learning the ropes at Transifex, I began to focus some of my attention on reorganizing and rewriting all the documentation in a unified style, based on the style guide we created. This would give us a solid baseline of writing which was up-to-date with the software. While working on this, started to test different combinations of analytic metrics to see what we should be tracking. I also started cataloguing common problems and bugs with solutions and their occurrence counts per week.

In addition to keeping our documentation up-to-date, we were tasked to publish content for new features and changes. Documentation and communication were crucial so that both branches of the company were aware of updates being made to the product and providing insight into the troubled areas of the software. We knew creating an extensive online knowledge base would help insure the continued health of the entire system.

Personas

Based on our user's self-reported titles and repeated questions, we found most of them could be categorized into developers, translators, or managers. The developers were usually very self-sufficient, had a deep understanding of the platform and the processes behind localization. They usually left suggestions, reported bugs, or had technical questions about the platform and its workflows. The managers were quite often ran small development teams and were the most interested in workflows, best localization practices, and learning about platform features. Translators were very eager to jump in, and often focused on the interface, interaction with other users, and project functionality. Between these three characters, we had a wide variety of perspectives we had to consider when creating content.

Whenever writing new content we would always ask:
- Who is this page for?
- How do we know they need it?
- What might prompt a user to take this action?
- What problem does this page solve for the user?
- What is the primary action we want users to take on this page?
- How will we know that this page is doing what we want it to do?

Solved Problems

Process refinement for remote work
The designers and engineers all sat in the same room, working in teams that cycled members each day. When a task was being addressed, we would convene as a group to review sketches, designs, and prototypes. At first, we worked fast and loose. We would discuss their strengths and weaknesses, make decisions, and get back to work. This would happen several times a day.

Our original workflow was shaped like a waterfall. That is, my team would wait for one part of the process to finish before starting the next. We would never architect anything until we had a full list of requirements. We wouldn't design guides without the screenshots, and we most certainly wouldn't have written anything without thoroughly discussing it with the product managers.

When I moved back to California, the nine-hour time zone difference slowed my productivity to a crawl. We did checkins at least every other day, but they weren't enough to keep our previous momentum. We learned from this bottleneck, thankfully. To combat this we switched to a second-person approval method. We would discuss a task while digitally sharing the content as we were working on it. This allowed peers to leave feedback and suggestions, see what other members were working on, all while allowing everyone to work at their own tempo.

To increase efficiency even more, we started using version control for our messaging and collaboration. This allowed the developers to contribute without breaking their normal workflow, and the branching and backup functionality it provided was invaluable.

Using analytic insight to reduce support traffic
It took roughly a month before I gathered a basic analytic dataset. Afterwards, I found the following metrics were the useful to track:

- Search terms used
- Number of returning visitors
- Percent of search term refinements
- Time spent on website after search
- Average search depth
- Total time spent on site
- Search engine traffic volume
- Traffic source changes (referrals, most visited pages, etc)
- User's navigation path through the documentation

To gain further insight, I compared these trends with consistently repeating ticket issues. This made me aware of which sections of the documentation were most active and, thus, crucial to maintain. Other useful and time-saving strategies we used involved keeping a current FAQ and adding an embedded solution suggester to the support-ticket submission form.

I experimented with a several different analytic frameworks such as Crazy Egg (heat maps), Mixpanel (for behavior funnels), and Google analytics.

Search engine optimization

The first search engine had a very simple heuristic. It turned every page into a text “blob” and counted the number of times the search term appeared in each one. This rendered terms like “language” or “translation” unusable since the target pages were lost in the noise.

Initially, I tweaked the search algorithm to give more weight to carefully worded titles, but when benchmarking the heuristic results against engines like Google or Swifttype, there was no competition. We made the switch to Google and our 4% search traffic slowly began to rise to a more standard 40%.

While gutting the site for the new search engine, I took the time to update the design, responsiveness, navigation, some pesky CSS bugs, all while upgrading the framework and libraries.

Communicating product changes to the company

The development team (in Athens) worked in the same room and could solve most of their problems with a simple conversation. Although useful, this destroyed any chance of a followable paper trail and often ended up leaving the California office in the dark. Updates came via word of mouth or our monthly all-hands meetings. Unfortunately, there were a few cases where sales staff were stumped mid-call because the interface had been updated the night before.

Our solution for communicating customer-facing changes happened during a company hackathon. Without getting too into details, all feature changes to the software were stored in a code branch. After the branch had been reviewed and approved, it was merged into the main branch which held the live software. This merge included a useful comment explaining what it did and changed. This comment was then scraped and sent to the company. This allowed everyone to tune into the product’s heartbeat, and have an easy-to-read product history. If you were on holiday, you could check the chain of change emails rather than reading the code line by line or asking all the developers what had happened.