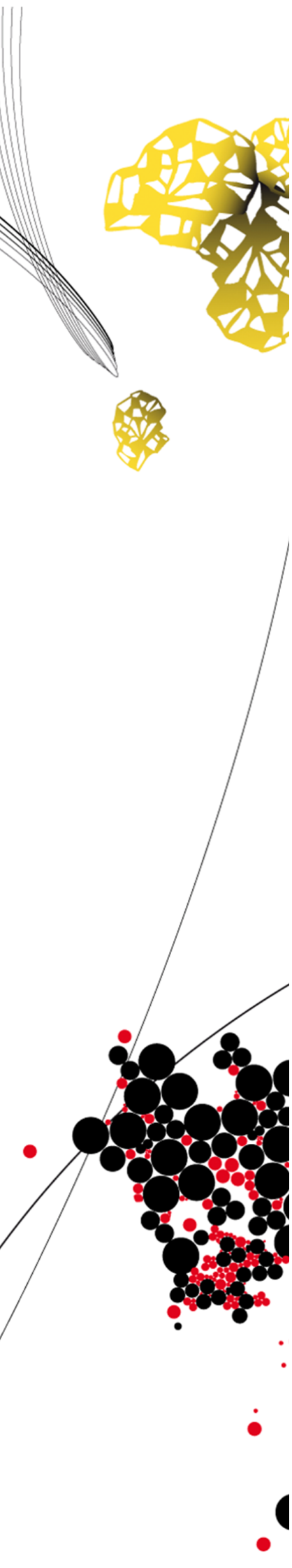

UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Marrying GUI and Git: How Authoring Tools for E-Learning Can Benefit from Version Control

Simon Kreiser

M.Sc. Thesis
November 2015

Supervisors:
Prof. Dr. Mariët Theune
Julian Ringel

Telecommunication Engineering Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The following thesis investigated the possible application of a proven paradigm from the realm of software development, called version control, to the process of authoring e-learning content. Version control is especially useful in settings of ever-changing text-based data that is created by a large number of people. Therefore, it fits the context of content authoring quite well.

But version control is notoriously hard to use, even for tech-savvy people. For this reason, this project's goal was to develop a simple user interface for version control which could be employed inside a content authoring tool, that is used by non-technical audience.

The interface design was based on task and requirements analysis as well as insights gained from previous research on the usability of version control. During the course of the project the interface was tested twice in two separate usability studies. The results informed the final design proposal presented at the end of the thesis.

The research has shown that version control can be vastly simplified and thus benefit a non-technical user-base as well. The project's intention is to kick off a discussion on the viability of version control outside the realm of programming and make those features available to a wider audience.

Contents

List of acronyms

CAT	Content Authoring Tool
CLI	Command Line Interface
DL	Display Language
GUI	Graphical User Interface
HTA	Hierarchical Task Analysis
LL	Learn Language
PM	Project Manager
QA	Quality Assurance
VCS	Version Control System

Chapter 1

Introduction

With the emergence of massive open online courses the e-learning industry has witnessed a sharp growth in recent years [**?**]. In 2012, as much as one third of all enrollments in the US were registrations for online classes [**?**]. In addition, 77% of US companies offer advanced professional training based on e-learning to their employees [**?**]. But it is not just professional or academic education that has driven the growth of e-learning. More and more people are turning to online courses for pursuing personal interests as well. One area that is particularly popular in this regard is online language learning [**?**] [**?**].

The increasing popularity of online learning has also increased the demands on content creators and their tools. The systems that have been quietly powering most e-learning platforms are mostly invisible to its end-users. They are often referred to as *learning management systems* or *authoring systems* or *tools* and have vastly simplified the creation, management and distribution of educational material. Some systems even track the learning progress and gather detailed usage statistics thereby allowing educators to optimize the learning content iteratively. But, with the complexity of learning content and number of contributors increasing quickly, these systems also run into limits. Most of these systems were not built to be used by a large number of contributors at the same time. Furthermore, most lack the ability of keeping track of changes made to learning content or to quickly revert changes. These problems together with a growing user base and increasing expectations on e-learning platforms necessitate the development of a new kind of authoring system more sophisticated than its predecessors.

This thesis investigates how language learning content could be created and managed more easily by applying a proven paradigm from the realm of software development, called version control, to a Content Authoring Tool (CAT) used at one of the European market leaders in the space of online language learning, called Babbel. Special emphasis is put on fostering collaboration between different professionals and keeping track of an ever-changing and expanding course structure. The

goal is to exploit version control's benefits and make them work for a non-technical user group.

1.1 Babbel

Babbel is an interactive language learning service that was founded in 2007 and employs about 350 people as of Fall 2015 [**?**]. Babbel users can choose one of 14 different languages to learn. Each course is created uniquely for one language combination, which consists of a user's native language, called Display Language (DL), and the language he or she wants to learn, called Learn Language (LL). Education experts, linguists and language teachers working for Babbel are responsible for managing and improving existing courses as well as creating new ones. All courses can be accessed on the desktop as well as on iOS and Android devices [**?**]. A speech recognition system helps users to practice their pronunciation. The service is subscription-based and customers can choose among 1- to 12-month payment plans [**?**]. The research described in this thesis was conducted over the course of half a year at the Babbel headquarters in Berlin.

1.2 Motivation

With the accelerating adoption of e-learning and rising demands in regards to the user experience of e-learning platforms, the pressure on content creators to deliver high-quality learning content is increasing steadily. But, in order to deliver content that meets these expectations, authoring tools are needed that facilitate collaboration and quality assurance. It seems that many authoring tools were built under the assumption that creating new learning content is a mostly solitary task. But today, especially at Babbel, this is no longer the case. To create a new language course a number of experts are needed. Until a new course is released, there are at least 5-6 individuals who have participated in the production of the course, usually even more. The authoring tool currently used at Babbel does not facilitate this workflow very well and has a couple of inherent problems:

- *Collaboration is inconvenient:* Concurrent editing is almost impossible and content needs to be partially locked so that contributors do not interfere with each other.

- *Reverting changes is difficult:* A change, after it was saved, cannot be easily reverted. Instead it needs to be undone manually.

- *Changes are not tracked:* The system is more or less blind to what was changed, by whom and when. There is no history, which would allow inspecting past changes.
- *Quality assurance:* Because changes are not tracked, Quality Assurance (QA) professionals need to do more work than necessary, just because they do not know what has changed exactly. Sometimes new content is also overlooked because of that.

1.3 Research Questions

The problems described above have one commonality: they are not exclusive to content authoring, but frequently occur during large software development projects as well. Most developers mitigate these problems by using a Version Control System (VCS). Initially, version control adds a small overhead to the software development workflow, but this usually pays off when projects become more complex [**?**].

Despite its obvious benefits, version control has not been widely adopted outside of the programming realm. This could be partially due to its usability issues [**?**] [**?**]. This thesis investigates if and how authoring tools could benefit from incorporating version control and how such a system could be made more user-friendly. The research question therefore reads as follows:

How can version control improve the authoring process for e-learning content?

The research question can be further divided into several sub-questions:

1. Which features can be simplified or omitted while preserving the usefulness of version control?
2. How can the learnability of VCSs be improved in order to make them more attractive to a non-technical audience?
3. How can the overhead, usually introduced through a version control system, be reduced?

1.4 Methodology

In order to answer the research question stated above this research project adopts a rather practical approach. An interface for a new content authoring tool will be designed incorporating version control. The interface will be tested with real users,

the linguists at Babbel, in order to ensure its usability and to determine whether version control can enrich the process of authoring language lessons.

The research project consists of two major phases: the analysis phase and the design and prototyping phase. During the first phase a literature review is conducted as well as two analyses. The literature review examines research concerned with the usability of popular version control software as well as examples where version control was applied outside of the programming realm. The subsequent analyses will be centered around the future users of the system and how it is used in the context of creating language courses. Therefore, a task analysis and a user requirements analysis are performed. These ensure that user needs and goals will be properly addressed during the following design phase.

Based on the insights gained during the first phase a prototype is designed that incorporates the most important version control features needed for content authoring. The prototype is semi-interactive and will be tested with a handful of users during a task-based usability study. This first study is followed by a focus group meeting during which users discuss the problems they experienced. The focus group puts special emphasis on the version control terminology since finding alternatives to the technical and abstract terms used in most version control systems is one key for improving the usability. The results of the focus group as well as the first usability study are then used to design an improved version of the interface. This new design is again tested during another usability study. The findings of this last study will inform the final design of the authoring tool as well as help answer the research question described above.

1.5 Thesis Structure

The next two chapters, Chapter 2 and 3, will introduce the reader to version control software in more detail as well as the methodology used throughout the research project.

Chapter 4 introduces the reader to related literature and relevant research in adjacent fields. This is followed by a more in-depth description of the different user groups at Babbel, their requirements and the tasks that are performed using the tool (Chapter 5).

In the second half of the thesis the design process as well as the usability tests are covered. Chapter 7 is about the first prototype and the first round of user tests. This is followed by a description of the focus group (Chapter 8) and the second design iteration (Chapter 10).

Readers who are mostly interested in the outcome of the design phase and the research results can refer to Chapters 11 and 12.

Chapter 2

Version Control

The following chapter will introduce the reader in more detail to version control systems. A definition is provided as well as a brief introduction into the taxonomy and different concepts of version control. In the end, Git, the VCS most of this thesis is centered around, is explained more precisely.

2.1 Definition

A version control system, also referred to as revision control or source control, tracks changes that are made to a set of documents. The documents and all changes that have been made in the past are stored inside a store called *repository*. New versions of a document can be added by *committing* changes to the repository. This action creates a unique version number and associates author, time and a short description with the state of the content. This allows users to reverse changes and simplifies collaboration. For these reasons, and because it works well with text-based documents and source code in particular, version control is used heavily within the software development realm.

2.2 Taxonomy and Concepts

The general concepts and terms described below are mostly shared between different version control software. Nevertheless, no system is like the other and the naming can vary a lot from software to software. For this reason, Git, the most widely adopted version control software [**?**] was chosen as a basis. The features are explained conceptually, so that the reader will be equipped to follow the reasoning of this thesis. For a more technical and detailed description of the inner workings of version control other sources can be consulted [**?**] [**?**] [**?**].

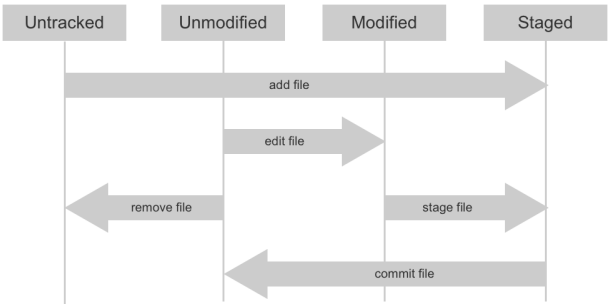

Figure 2.1: How the state of a file changes

- Repository: a database that holds file contents and all versions since creation. A new repository can be created by issuing the *init* command inside a directory. This means that from that moment all files in the directory and its sub-directories can be tracked by version control, unless told otherwise (see next bullet). Repositories can be either local (situated on the client machine) or remote (on the server).

- Tracked, untracked and ignored files: when a new repository is created it is empty. It has to be specifically told which files from the working directory should be tracked. This can be done by using the *add* and *commit* commands. If certain files should not be tracked, they can also be ignored permanently.

- Working directory: the working directory represents the state of the latest commit of the currently selected branch plus the changes that were made since then. It can also include untracked or ignored files.

- Staging area: this is a layer in between working directory and repository. It allows developers to control in a fine-grained way what exactly will be part of the next commit.

- Snapshotting: the process of adding a new version to the history of the repository is called snapshotting. It involves adding changes to the staging area (using the add command) and committing these together with a description of what has changed. Before staging one can also check what has changed compared to the previous version using the *diff* command. The different states a file can be in during its lifecycle are visualized in Figure 2.1.

- Branching: a branch is a named copy of the entire content stored inside the repository. It allows developers to build features without influencing the stable code. Branches can also be utilized to experiment in case one does not know whether a certain direction will lead to the desired solution.

- Merging: the opposite of branching, merging brings different versions of the same file back together. Usually this is automated, but sometimes conflicts occur, when both versions have changes in the same line of code.
- Pull Request: a pull request is a formalized way of letting collaborators review changes that are part of a particular branch before these changes are merged into another branch (usually the main branch). The feature enforces the best practice of *peer reviews* and simplifies the communication between developers. It is not part of Git itself but was introduced by Github (Figure 2.3).

Figure 2.2 provides an overview of a typical Git workflow. The cycle always starts with branching off the main development line and ends in merging back into it. In between the developer writes code and commits this code in coherent pieces, so that reviewing is made easier for collaborators. The pull request is just a way of formally asking these collaborators to review a particular chunk of code. It is a Github feature and not part of Git itself.

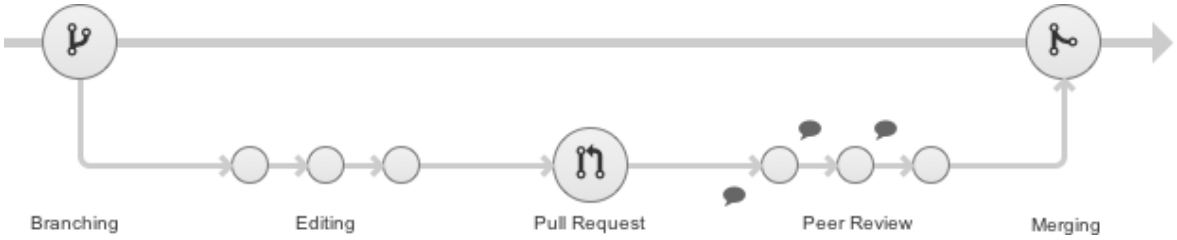

Figure 2.2: Version control workflow

2.3 Version Control Software

The concept of version control is older than some people might think. Early implementations, such as SCCS (Source Code Control System) or RCS (Revision Control System) date back to the 1970s and 80s [**?**] [**?**]. According to Sink [**?**] version control software evolved in three stages: first generation systems only allowed editing one file at a time that had to be locked while changing it. Second generation systems introduced networking and a central repository for simplifying collaboration. Additionally, the concurrent editing of files was now possible. Apache Subversion, still the second-most popular VCS as of 2015 [**?**], belongs to this generation. The third generation of version control systems were distributed (instead of centralized), no longer file-based and made branching more convenient. Distributed systems are faster, because the network latency for talking to a central repository is eliminated. Furthermore, third generation systems handle large code-bases more efficiently because these systems do not store copies of each file, but only the changesets. This

also makes branching a lot quicker and easier. The most widely used VCS among these third generation systems is Git, which is also described in more detail below.

2.4 Git and Github

Git is an open-source version control software that has its origins in the Linux kernel developer community. After running into licensing issues with a proprietary version control software called BitKeeper, the community developed Git in 2005 in order to replace it [**?**] [**?**]. The design of Git was based on BitKeeper and the requirements of the Linux developer community. Because it is distributed and changes do not need to be committed to a central repository it is faster than most other VCSs. Furthermore, it supports hundreds of parallel branches and handles large code bases like the Linux kernel efficiently. Git is also workflow agnostic. There is no access management or locking of files. Everyone can edit anything at any time. Conflicts, in case they emerge, are dealt with after the fact, when files are merged together again. As Linus Torvalds said in a presentation in 2007, Git is based on ”a network of trust” [**?**]. Companies can decide for themselves what kind of rules or workflows they want to enforce. These characteristics have made Git the most popular VCS in software development by far, beating Subversion as a distant second (69% vs. 37% adoption) [**?**].

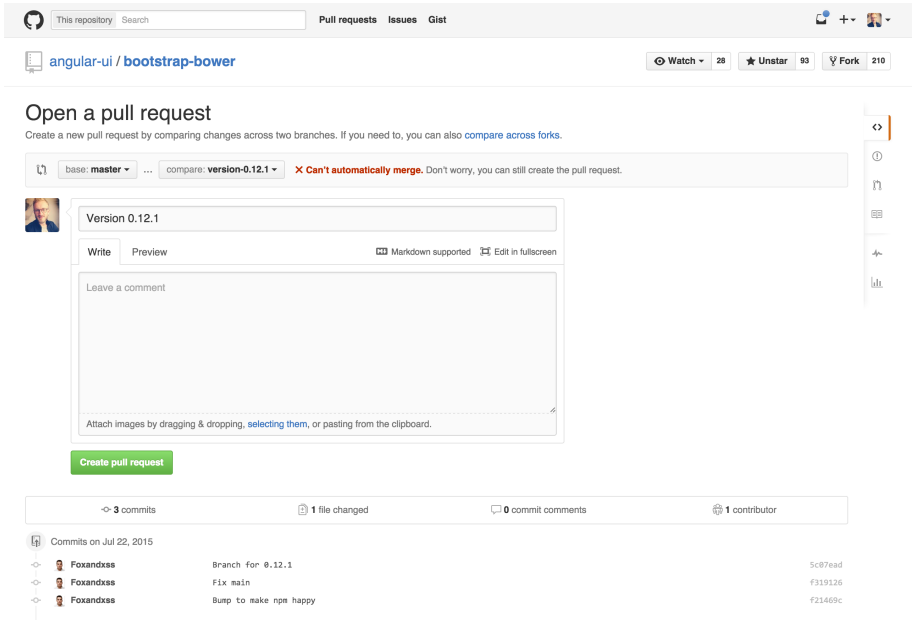

Figure 2.3: Screen for creating a pull request on Github

Github, like the name gives away, is based on Git. It is a repository hosting service

that adds additional features, such as access control and social networking capabilities [**?**]. The service eliminates the need of setting up a personal server in order to host a repository and makes collaboration simpler by offering wikis, task management and a mechanism for code reviews. All of this is offered through an easy to use graphical user interface. Furthermore, it has become the de-facto standard for sharing open-source projects.

2.5 Summary

Version control software is used to keep track of changing code-bases and for simplifying collaboration among developers. Even though some VCSs have been around for a few decades, the widespread adoption of version control has started only with the second generation of version control systems. Distributed VCSs as well as cloud services, such as Github, have further accelerated this adoption.

Now that the concept of version control is hopefully a little clearer, the next chapter will go on in describing the methodology used for this research project, which will help to ultimately answer the question whether version control can be incorporated into content authoring tools and whether these tools can benefit from it.

Chapter 3

Methods

The research project consists of two major phases. Phase 1 is all about analysis: Determining user requirements and finding out what goals these users are trying to achieve by using the software. Phase 2 is about design and user testing. Based on the analyses and the results of the literature study a first prototype is designed and tested. The insights inform the design of the second iteration, which is then tested again in a more thorough way and with more users. This last study not only assures that the implemented changes had a positive impact, but also verifies that all user requirements have been met and that the software allows users to perform all necessary tasks.

3.1 Phase 1: Analysis

The first phase of the project consisted of user research, which established requirements and shed more light on the users' needs. The research served as the basis for designing the first prototype, as described later on, and was also used at the end of the project to determine whether the newly designed software met its purpose.

3.1.1 User Requirements Analysis

In addition to segmenting the different users into groups requirements were gathered. Since most users are full-time employees at Babbel they were easily accessible. Because of this, requirements were gathered through observational field visits and interviews as proposed by Goodman et al. [**?**]. The observations were particularly fruitful, as users were using an existing content authoring tool already. This helped most of them verbalize what they did not like about it. During the observations the author stayed in the background and only inquired about details of a user's behaviour every now and then. In general, users were left alone and just watched. The field visits were arranged in advance to make sure that users had enough time

and were actually using the existing tool at that moment. Each session lasted about 20-30 minutes and was followed by a short interview or open discussion. The author made sure that what he had observed was not based on a wrong understanding and the users could raise concerns or point out especially annoying design flaws. At large, the atmosphere was kept informal so that users could behave natural and did not feel like being part of an artificial situation. The field visits resulted in a long list of requirements phrased as user stories. The stories were composed in a simple structure (see below) as suggested by Cohn [**?**]. Terms in angle brackets need to be replaced and square brackets signify an optional section. The stories were ranked by type (functional or quality requirement) and importance. They can be found in Chapter 5.

As a <role>, I want <goal/desire> [so that <benefit>]

Figure 3.1: User Story Template by Mike Cohn

3.1.2 Task Analysis

There are many different ways of analyzing tasks and structuring them in a coherent way. All task analyses are in some way focused on decomposing a task into its constituting parts. The result is often some kind of diagram that visualizes the hierarchy or flow that is inherent to the task.

For this project, a method called Hierarchical Task Analysis (HTA) was used [**?**]. It is a simple but effective method for studying work processes and was developed in the 1960s by two psychologists, Annett and Duncan [**?**]. Their aim was to understand the purpose of work by looking at human activity within organisations and systems. HTA, in the tradition of systems thinking, regards work as a complex system consisting of interacting sub-systems, which can be machines and humans [**?**]. These sub-systems interact by means of input and output. The interplay is constantly adjusted through feedback mechanisms, which allow a human operator, to adapt his or her behaviour.

Based on the analysis, diagrams were created, each of which represents a high-level task that is decomposed into sub-tasks. Each diagram provides an overview of the order in which tasks are performed as well as the mental processes behind them. The diagrams together with a short description of each task can be found in Chapter 5.

3.2 Phase 2: Iterative Design & Usability Testing

The second phase consisted of designing and iteratively testing the new interface. The first prototype was based on the insights gained through task and requirements analysis as well as the literature study. It served as a first starting point for discovering usability issues of the version control features of the system.

3.2.1 First Study

As a means of detecting usability problems early on the first study was conducted with a semi-interactive prototype that was based on simple graphical layouts. The prototype was mostly black and white and only those parts that were actually needed for the user tests were interactive. This meant that designing the prototype did not take a lot of time and subsequent modifications (i.e. after a design feedback round) could be realized quickly as well.

The sessions were scenario-based, which means that participants had to perform a number of tasks which reflected their everyday work. This put more emphasis on the actual behaviour of users instead of opinions and attitudes and allowed the test moderator to stay in the background. This method is often referred to as *assessment* or *summative study* [**?**] [**?**]. These kind of studies can yield more honest results because the method involves less interference than for example exploratory tests. The number of participants was kept rather small, informed by Nielsen's insight that 5 users are sufficient to find 85% of the usability problems if the user group is somewhat uniform [**?**].

Additionally, a few quantitative metrics were taken in order to support the results of the observations. Among them task completion rate, time for task completion and error rate. Because the sample size is somewhat small these metrics can rarely deliver statistically significant results, but they can serve as an indicator of potential problem areas that should be investigated further. How these metrics were measured is described in more detail in Chapter 7.

3.2.2 Focus Group

All participants of the first usability study were also part of a subsequent focus group. The group meeting served as a platform for discussing concerns and the impressions regarding the novel version control features that users were exposed to during the testing sessions. Furthermore, the goal was to identify potential stumble stones in regards to the version control terminology. The terminology used by Git and other version control systems is often rather technical and not necessarily

self-explanatory. But the aim is to make the system easy to learn for a non-technical user group as well.

Focus groups are particularly suitable for the early stages of a project. They can be an efficient means of gathering feedback in a short time-frame, because they involve more than one user at a time [**?**] [**?**]. When conducting focus groups researchers need to be aware of the fundamental difference between what people say and what they do. Only because a participant claims that she likes a product, does not mean that she will actually use it [**?**]. Therefore the outcome of the group should be put into perspective and not applied literally. Furthermore, the results of focus groups should be taken with a grain of salt, due to their small sample sizes. Nevertheless, the outcome can serve as a basis for further research [**?**].

3.2.3 Second Study

The second and final usability study was based on a functional web-based prototype. This prototype contained a number of changes that were made as a result of the findings of the first user study and the focus group. Most notably, the main navigation had changed as well as the way content was represented. Furthermore, a series of severe usability issues had been fixed, mostly related to the version control features of the system. On top of that, some of the main features were renamed, which had been identified as problematic during the focus group meeting. *Branches* were now called *working copies* and *pull request* was changed into a more descriptive *merge request*.

The main goal of this study was to verify whether the introduced changes had a positive impact on the user experience. Hence the name verification or validation study [**?**] [**?**]. For this reason, the setup of the study was similar to the first study, which made comparing the results easier. The web-based prototype, which covered a lot of functionality, allowed the moderator to stay in the background so that users could freely interact with the system. This proved to be quite successful in that a lot of new issues were found that had not been discovered during the first study.

Besides validating the changes the study was also aimed at confirming that the user requirements specified during the start of the project were met. This was done by simply observing whether users were able to perform the desired tasks. Furthermore, a post-study usability questionnaire was used that measured the perceived usability of the system. This helped to identify weak spots and potential areas that would need more attention in the future.

3.3 Summary

This chapter described the theoretical basis of the methodology used throughout this research project. The design of the user studies and the procedure during the sessions is described in more detail in Chapters 7 and 10. The following chapter will introduce the reader to other research that has been done in the field of version control systems, with a special attention to systems applied outside of the programming realm. The remaining project will build upon this insights and the first prototype was heavily influenced by these findings described in the next chapter.

Chapter 4

Related Work

The following section describes three different areas within the scientific debate on version control systems: 1. the usability of version control systems, 2. improvements to version control and 3. implementations of version control outside of the programming realm. The review serves as a foundation for the research described later on and provides the necessary context in which the results of this thesis should be seen. Furthermore, it enables the reader to assess the contribution of this work to the current state of research.

It seems like most research in the area of version control is concerned with the shortcomings of different systems, both in terms of usability and the conceptual design. In some research projects alternative solutions have been investigated, but those are usually situated within the programming realm. Unfortunately, there are not many examples in which version control has been applied to other domains. In general, most research is focused on the widely used open source VCSs Git and Subversion.

The review is structured into the three areas described above. All literature is shortly summarized, compared and set in relation to other work of the field. The search for literature was based on three main criteria: 1. work that produced specific results that could also be utilized for this project, 2. work that was concerned with the interfaces of VCSs or the usability and 3. work that investigated the use of version control in unusual areas or in innovative ways.

4.1 Usability of Version Control Software

During the 2011 Git User Survey participants were asked which parts of Git needed improvement. A total of 73% stated that the user interface needs improvement [**?**]. With this in mind, it is surprising how little research there is about the usability of version control systems in general and Git in particular. The two most notable studies

have investigated the user experience as well as the conceptual design of different version control systems, among them Git [**?**] [**?**].

Church, Sderberg and Elango have conducted an empirical study on the usage of command line-based version control inside two large IT companies (Google & Autodesk) [**?**]. The study was based on interviews and observations within these companies. Engineers who participated covered a diverse range of job positions and had varying experience. The study results show that common usage patterns emerge when version control systems such as Git or Subversion are incorporated into the software development workflow. In both companies, most engineers stuck to a small set of commands and showed a general risk aversion when using version control. Church et al. note that the observed ”ritualistic behaviour” of engineers is usually found among novice computer users and attribute this behaviour to a low user confidence in the version control systems used. They conclude that it is particularly problematic for a tool whose main premise it is to offer ”development without fear” to be perceived as risky to use.

Explaining these results, Church et al. present a cognitive dimensions analysis of Git. They point out that there are many *hidden dependencies*, such as between files and branches, local and remote repositories as well as between untracked files and the repository. Moreover, Git has an ”abstraction barrier”, meaning that certain concepts, i.e. branching, need to be learned, before the system can be used in a meaningful way. The authors reason that a combination of hidden dependencies and abstraction mostly contribute to the aforementioned usability problems. These conceptual issues are also the reason why the Graphical User Interface (GUI) client by Github [**?**] only offers a slight improvement in usability. It makes features more discoverable and offers a more visually pleasing interface, but it suffers from the same deficiencies as the command line interface. The authors close by highlighting that the discovered issues need to be solved before version control can take the leap from computer science to a wider audience.

The second study mentioned above, by Jackson and Perez De Rosso, was based on an analysis of Git’s conceptual model [**?**]. The researchers present the weaknesses in Git’s conceptual design and propose an improved alternative called Gitless.

They argue that concepts have a strong influence of how users think about an application and therefore conceptual integrity should be an important consideration in software design. By concepts the researchers mean ”the constructs and notions [...] that are invented for the purpose of structuring the functions of the system” (Page 38). Git’s concepts were scrutinized based on three main criteria that were defined by Frederick Brooks in the 1970s [**?**] and which he believed to be central to conceptual integrity. 1. orthogonality individual concepts should be independent

from each other, 2. propriety a software should only have the essential functions that are needed for its operation, and 3. generality a function should be applicable in different ways. After formally describing Git's main concepts the researchers go on in explaining which concepts violate these criteria and why. Regarding the first criterion, orthogonality, the researchers point out that Git's different file states, such as modified, staged and committed, are particularly troublesome. Commands that are primarily intended to modify one state, sometimes also affect the other. For example, if a user modifies a file, stages it and then goes on modifying the same file, the command git commit *file* will commit both states of the file (the staged and the unstaged). This happens, even though the commit command typically only effects the staging area, unless told otherwise by a particular flag. According to the researchers, the staging area also violates the second criterion propriety. They argue that an intermediate step between editing and committing is rarely useful and that most users just want to commit their changes right away. Although this is possible through an additional flag (commit -a) this approach has some drawbacks as well, e.g. that untracked files will not be included in the commit and that a high verbosity is required when committing only a few files. The last criterion, generality, is violated by the branch concept. Even though branches allow users to save different parallel versions of a file, this concept does not apply to the working directory or the staging area. There is only one working directory and one staging area. This means that changes made to a file need to be committed before branches can be switched. This is unnecessarily troublesome and requires the use of another function (git stash) in case the user does not want to commit a file, because the code is in a broken state for example.

Concluding, Jackson and Perez de Rosso suggest that software is designed from inside out and that interfaces can only be as good as the underlying concepts. Therefore, they propose a simplified Command Line Interface (CLI) for Git, called Gitless [**?**], that eliminates the aforementioned issues. Among other things the staging area was removed and a more general branch concept got introduced. Gitless is an open-source project that is built on top of Git. The researchers hope that it will be further improved by the community and that it sparks a discussion on how version control can be made simpler and more user-friendly.

The results of this study can partially explain why the engineers interviewed for the study by Church et al. [**?**] acted so risk-averse when using Git. The inconsistencies throughout the system might have prevented them from forming an accurate mental model.

4.2 Proposed Improvements to Version Control

Gitless is not the only attempt to make version control systems easier to use. The official Git Wiki lists as many as 36 different GUI clients for Git [**?**], whose main goal it is to make version control simpler and more accessible. Jackson and Perez De Rosso claim that these interfaces merely add an aesthetic layer to Git, but fail to make it simpler, because they are still operating within the boundaries of Git's inherent concepts. Below, projects are described that tried to dismiss these boundaries by looking into new ways of approaching version control.

Bicer, Koc and Tansel note that even though version control was supposed to make collaboration easier, it only does so as long as there are no conflicting changes [**?**]. But so called merge conflicts happen regularly during the development process. They can only be resolved by direct communication between the engineers who are responsible for the changes. This works well as long as both engineers share the same location, but nowadays large companies span the whole globe and open source projects are being developed by a diverse group of people from all around the world. Bicer et al. state that most version control systems do not offer a platform for resolving these merge conflicts. They suggest the introduction of a new command, called *peek*, which allows developers to take a look at local changes of other contributors in order to prevent conflicts from happening in the first place. The feature is accompanied by a social networking site that is believed to encourage communication between developers.

Even though Bicer et al. identified what seems to be a common problem with version control systems [**?**] [**?**] [**?**], their solutions did not succeed in reducing the number of conflicts. During a small experiment with 5 developers the users of the system were reluctant to utilize the new peek feature. The researchers blame the laborious process of having to request a peek first and the necessity of monitoring changes of colleagues regularly.

Because of the usability shortcomings of version control a solution to automate it entirely has been proposed [**?**]. The thesis describes a web-based IDE (Integrated Development Environment) that avoids conflicts by implementing an elaborate change awareness system. This system will warn developers in case they try to change code that team members are working on as well. The IDE utilizes different colors to highlight sections that are conflicting. When a conflicting line is highlighted, the user has the option to adopt the changes of a collaborator first and base her subsequent work on this updated state. This means that conflicts can be avoided right from the start and reduces the need for laborious conflict resolution during the merge process.

This mechanism is enabled by a feature called *AutoShare*, which basically means that changes are saved, committed and pushed automatically without any user interaction. Through this, users get a real-time view of what their collaborators are working on. This enables remote pair-programming for example, among other things. For programmers who prefer a more granular control all version control features can also be operated manually.

Even though case studies have been described, no user tests were performed as part of the thesis. This makes it hard to gauge whether the proposed solution could be potentially adopted by programmers. Eventual downsides of the proposed solution are a harder to navigate project history, because automated commit messages might not be as descriptive, and a possibly too disruptive IDE. For example if a developer consciously decides to ignore conflicting changes and the system keeps warning her.

4.3 Version Control Outside of Software Development

Given its usefulness it is surprising that version control has not become ubiquitous yet. As Grischenko notes, versioning was even part of the original hypertext concept [**?**], but today it is mostly found in highly specialized areas where collaboration on large sets of text-based documents takes place. Besides software development this is particularly digital publishing, where the emergence of Wikis has exposed the concept to a larger audience [**?**]. Within this area it is usually referred to as revision control (Wikipedia, to stay inside the subject matter, treats the terms revision control, version control and source control as synonymous [**?**]). The utilization of version control inside wikis is not surprising, since the process of creating a wiki holds similar requirements as the software development workflow. 1. collaboration is very common, therefore 2. concurrent editing is an essential feature and 3. transparent edits should allow reviewing and reverting changes if necessary [**?**].

Version control is especially hard to find in consumer products. The most prominent examples might be the *track changes* feature of Microsoft Word [**?**] and Google Docs revision history [**?**]. Both are automated version control systems that allow users to go back in time and look at different revisions of a document or changes that were introduced by collaborators.

When looking beyond publishing and text processing version control systems are less common. This might be due to the increased complexity of implementing version control for other domains. Computing and visualizing differences between text-files is much easier than for example between binary files on which most graphical and video file formats depend. There have been several attempts of bringing version control to graphics [**?**] [**?**] [**?**]. Most of them have used a rather simplistic ap-

proach, storing revisions as individual files. Thus, the user gets a temporal history of changes, but is not able to look at a delta of two revisions, because the system is blind to the semantics of changes.

A more advanced version control system for graphics has been proposed by Chen, Wei and Chang [**?**]. The researchers implemented a non-linear revision control system for GIMP, an open source graphical editing tool [**?**]. In order to offer a meaningful revision history the system records high-level user interactions. Based on these a graph-based representation is created that allows users to review spatial, temporal and semantic relations between different revisions. Nodes represent image editing operations whereas edges represent the relationship between these. On top of that, the graph nodes provide small thumbnails so that users can spot differences between revisions at a glance (Figure 4.1).

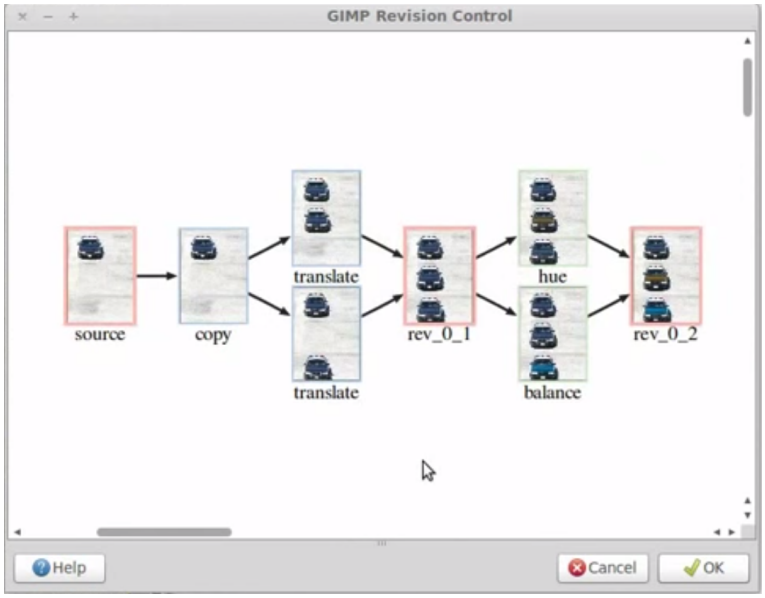

Figure 4.1: Revision control for GIMP Source

The system enables users to navigate different versions of a graphics file, compare them (diff) as well as create new branches and merge changes. Furthermore, the user can manually check in new revisions, even though this is generally unnecessary since the system tracks all user actions automatically. Common use cases cited by the researchers are image editing and digital sketching. Especially the latter one often involves trial-and-error experiments that make revision control particularly useful. During usability tests Chen et al. found that participants were happy to integrate revision control features into their workflow. Especially the visual representation of the history was deemed useful. Merging on the other hand, seemed to be more difficult for some users. The researchers conclude that even though the system is easy to use and helpful it might take some time and practice to master it.

4.4 Conclusion

It has become apparent that Git has some conceptual deficiencies that cannot be solved by just creating a new interface. Instead the underlying structure needs to be questioned and features should be redesigned or dismissed. The fact that experienced engineers at renowned software companies were anxious to freely interact with the system is proof that Git has some design flaws. These problems need to be eliminated before Git can be used by a non-technical audience.

On the other hand, fully automated version control does not seem to be the holy grail either. If users are not exposed to version control features at all it decreases the discoverability and might prevent them from forming an accurate mental model of the system. This might handicap their ability to solve problems in case the system does not work as intended. Furthermore, automated version control lacks a fine-grained control of how the work is structured and presented to collaborators (commit messages and history).

The findings of Bicer et al. showed that not every problem a software has can be fixed by adding new features. Sometimes, designers and researchers are better off accepting the boundaries of what software can achieve and focusing on building a lean system that can do everything that is necessary but not more than that. This again highlights the importance of *propriety*, offering only the smallest possible set of features, mentioned by Jackson and Perez De Rosso. Nevertheless, facilitating communication through a version control system is a promising and important feature that should not be ruled out, only because it did not work in one particular setting.

Concluding one could say that Git's conceptual design flaws need to be eliminated and that hidden dependencies should be reduced. This will be a challenging task given that the new system will still be based on Git and Github. Nevertheless, Jackson and Perez de Rosso have shown that enhancements can be made even within the constraints of an existing system.

Chapter 5

User & Task Analysis

Now that the concept of version control and the state of research should be a little clearer, the focus will shift from the abstract and theoretical to more practical matters. This chapter will introduce the reader to the different groups of people involved in creating new language lessons for the Babbel learning platform. In addition to outlining the responsibilities of different user groups, their needs and requirements the most common tasks are analyzed.

Even though the results of this research project should be applicable to a wide range of authoring tools, it is worth noting that, from now on, this thesis will be concerned with the usage of such a system in one particular case: the creation of language lessons by linguistic experts at Babbel. The so called *didactics department* consists of 25 full-time employees and almost 100 freelancers. The department is responsible for creating new language learning content for Babbel's 14 different languages and is divided into teams, based on their language expertise. Right now there are 7 different teams that consist of 2-4 people. Each team is led by a project manager who guides the process of creating new courses and lessons and also manages the collaboration with freelancers.

5.1 User Groups

Below, the different user groups are listed accompanied by a short description of their responsibilities. A more detailed explanation of what these responsibilities entail and which tasks these users perform on a daily basis can be found in the last section.

5.1.1 Content Authors

Content authors conceptualize and create new content. Together with their project managers they decide how a lesson looks like in detail. What is the lesson trying to

teach the end-user? What kind of vocabulary is introduced? Which type of exercise is suited best to convey this knowledge? If these decisions are made the editor goes on to ”script” a lesson, meaning that a spreadsheet is filled in that defines the exercises, vocabulary items and translations. Please note, that initially there is only one translation (English or German). The job of translating (localizing) a lesson into a new language is usually performed by a separate person since there is rarely a content author who has knowledge of all languages a particular lesson is translated to.

5.1.2 Project Managers

As mentioned above, project managers are the ones who guide the content creation process. They have to set priorities and decide what needs to be worked on. Furthermore, they need to manage freelancers and make sure that they stick to an agreed upon schedule. Project managers are also responsible for the content quality and have the last word before a new course is released.

5.1.3 Translators and Proofreaders

Translators and proofreaders are responsible for localizing lessons into new languages. They get a clearly defined work package and work in unison. First, the translator translates all vocabulary items, exercise titles and descriptions and then hands over her work to the proofreader. The proofreader corrects spelling errors and notes feedback on issues of style, grammar or semantics.

The role of being a translator or proofreader is not permanent, but only true for the duration of a project. Someone who is proofreading during one project could be translating during the next. This is why they are regarded as one user group. A more detailed description of tasks performed by translators and proofreaders can be found in the next chapter in section 5.2.1.

5.1.4 Speakers

Speakers may have the most straightforward job. They have to record the individual vocabulary items of a lesson. Of all users they need to collaborate the fewest with other people. Because of that and because it is not clear whether sound recording will be a feature of the new content authoring tool, speakers will be ignored for the rest of the thesis. They are just listed here for the sake of completeness.

5.1.5 Quality Assurance Professionals

Quality assurance is a systematic approach of finding bugs or problems within newly created content. It happens before new content is released. The check is carried out in the frontend as the end-user would use it. The CAT is only used when an error is found and needs to be corrected. A detailed account of what happens during quality assurance is given in section 5.2.1.

5.2 Task Analysis

As described in Chapter 3, a hierarchical task analysis was used to analyze the most common tasks performed within the process of authoring new language learning content. Insights were gained through interviews and observations over the course of several weeks. The task analysis in conjunction with the requirements described later on has served as a foundation for the design of the first prototype.

5.2.1 Tasks

It is the responsibility of the didactics department to constantly create new learning content. This work can be divided into four major tasks, which can be seen in Table 5.1. Three of these four major tasks were analyzed and then visualized using a diagram that shows the hierarchy and dependencies of different sub-tasks. Task 3 was not analyzed, because it is performed with a different tool and is quite separate as a task from the other ones. Note that task 1 and 4 are carried out by employees at Babbel, whereas task 2 and 3 are mostly performed by freelancers.

Level	Task Description
0.	Conceptualizing and creating new learning content
1.	Create new lesson
2.	Localize new lesson
3.	Sound recording for new lesson
4.	Final quality assurance

Table 5.1: Tasks that were analyzed

Task 1: Creating a New Lesson

Figure 5.1 shows the workflow for creating a new lesson. It should be noted that before a new lesson is entered into the system, as it is described here by the diagram, a long process of conceptualizing (what should we teach the user and in

which way?) and scripting already lies behind. What is referred to as scripting is the process of deciding on the right format of the content. For example what exercise types are best suited to convey a particular kind of knowledge about a language. All these decisions are put into a spreadsheet that is called the manuscript, therefore the term scripting. Creating a new course or lesson, as described by the diagram, is thus only a matter of transferring the manuscript into the content authoring tool. Whether this whole workflow can be centralized or combined inside the new tool remains to be seen.

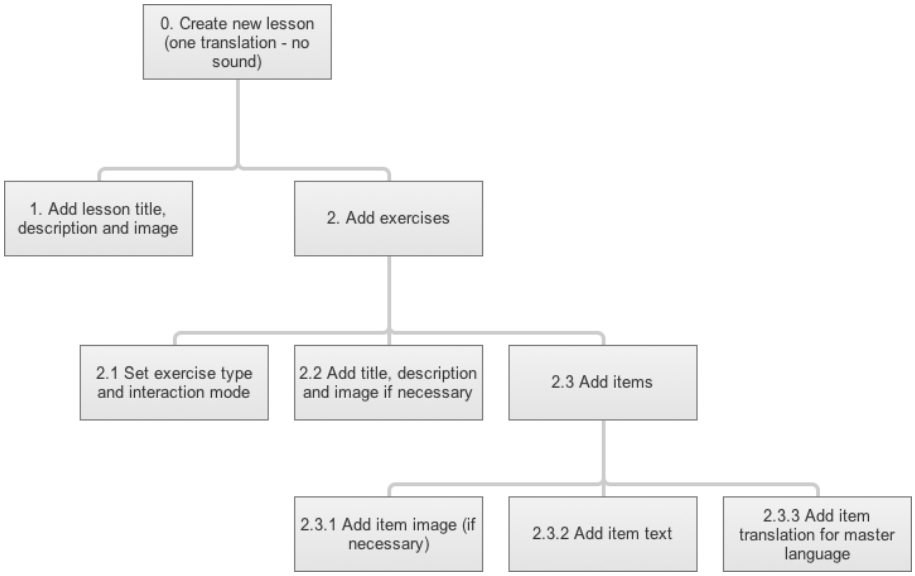

Figure 5.1: The tasks involved in creating a new lesson

Task 2: Localization of a Lesson

The following diagrams visualize the process of translating existing content into new (display) languages. A display language is the language a user chooses which defines the language of the user interface as well as the translations and explanations he gets while learning a new language. Usually the display language corresponds to the users mother tongue, but sometimes learners whose mother tongue is not offered on babbel.com choose English as alternative. Babbel offers 14 different learning languages right now which can be learned by using one of the 8 different display languages. This means that there are more than 100 unique language learning combinations (14*8) that need to be maintained as well as extended when new content is added to a language.

The task of localizing existing content is a joint effort of employees at Babbel and at least two different freelancers. Usually a Project Manager (PM) at Babbel prepares a content package (several lessons or a whole course) for translation and

manages the whole process. Afterwards a freelance translator takes over and translates all the necessary content based on a time plan. When he or she has finished a package the work is handed over to a proofreader who might do some corrections or suggest improvements. Figure 5.2 presents a high-level view of the localization process.

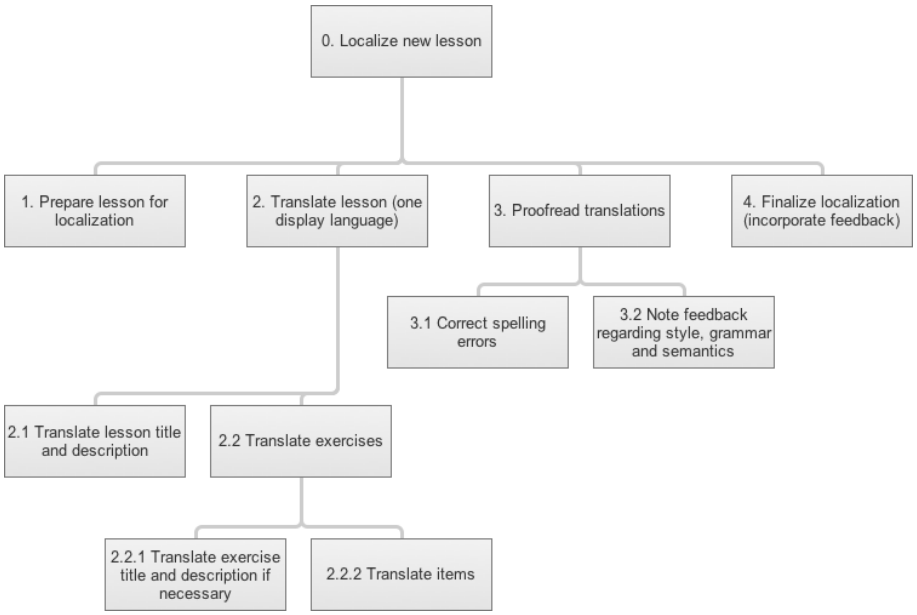

Figure 5.2: Overview of Localization Process

Task 3: Image Assignment

Task 4: Sound Recording

Sound recording is the process of turning the different vocabulary items into speech. Only the learning language is recorded, not the translations. The recordings are done by native speakers who work as freelancers. Whenever new content is ready for recording (usually several lessons or a whole course) the speakers get informed of what exactly needs to be recorded. They can either do this at home, provided they use a high-quality microphone, or come to the Babbel offices and record in a professional studio. Either way, the tool they use, is the same. The recorded sounds end up on the media server and the sound IDs are linked to the respective items. After the speaker has finished his or her work package a full-time employee checks whether the recordings meet the quality standards and if not some sounds need to be re-recorded.

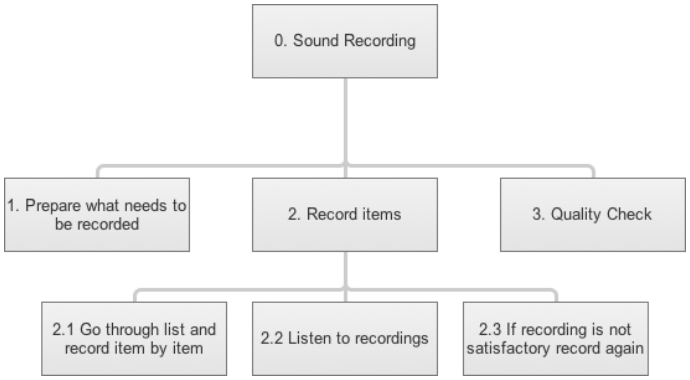

Figure 5.3: Sound Recording Process

Task 5: Quality Assurance

The final quality assurance happens right before new learning content is published. A Babbel employee clicks through the content in the frontend as a normal Babbel user would. This happens to check whether the content works as expected and the application does not break down at some point because a bug has been introduced. It is also the last chance of spotting spelling or grammar mistakes or checking whether the content complies with different conventions and has a consistent style.

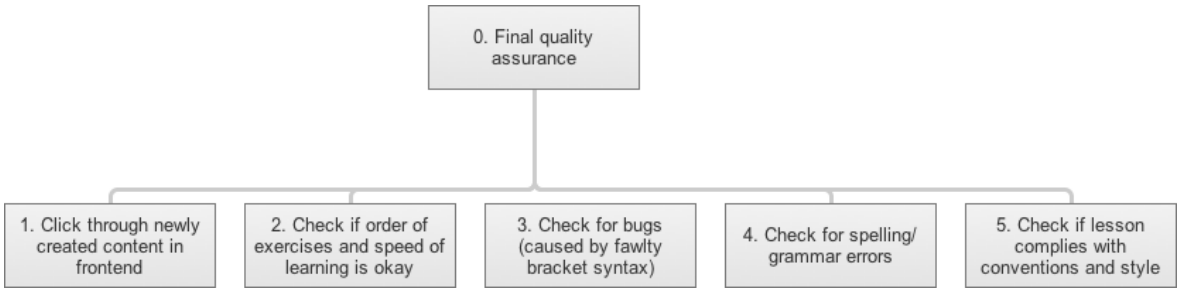

Figure 5.4: Quality Assurance Process

5.2.2 Conclusion

The analysis made a few things obvious. First of all, collaborating and communicating with colleagues is a reoccurring theme throughout most tasks. Not only the content creation process is divided between different user groups, but also some tasks themselves. For example, when a lesson is localized, there are at least three different individuals working together, who also need to communicate in some way. Right now, email and Google Sheets are used extensively for that, but it is conceivable that at least some of this communication could be facilitated by the new version control system. This could possibly speed up the whole process and streamline collaboration.

In general, version control features could potentially ease collaboration and allow full-time employees to give freelancers more freedom in their work, because the risk of breaking live content will be lower. Furthermore, a feature similar to Github's *pull request* could help to formalize the quality checks that happen throughout the process and provide a platform for communication between freelancers and full-time employees.

The next section introduces a list of formalized requirements that have been partially derived from the analysis of user groups and their daily tasks.

5.3 Requirements

As described in Chapter 3 this analysis was based on observations and interviews with members of the Didactics Department at Babbel. First, functional requirements are listed, which describe requirements based on a user's goal. In Section 5.3.2 the quality requirements of the whole system are explained. Please be aware that the following requirements are only concerned with the version control aspects of the content authoring tool. The list of requirements for the whole tool would contain much more.

5.3.1 Functional Requirements

The functional requirements are grouped based on the different user groups. Please note that these user groups are only approximations of the user's responsibilities and in reality are not as clear cut as it might seem here. For example, project managers share many requirements with content authors and the role of translators and proofreaders could be regarded as interchangeable as well.

The requirements are phrased as user stories, which emphasise the user's perspective and the ultimate goal they are trying to achieve by using the software.

5.3.2 Quality Requirements

The functional requirements above mostly answered the question *what* specifically users want to do with the tool. The quality requirements instead look at the *how*. The product quality model defined by the ISO/IEC standard 25010 [**?**] names usability as one of eight product characteristics that contribute to the overall quality. Usability is further divided into a set of 6 sub-characteristics: appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics and accessibility. Since the constraints of this project did not allow focusing on all of these

#	Requirement
PM-1	As a project manager I want to make sure that translators only work on the content they are supposed to work on so that no unnecessary work is done by them.
PM-2	As a project manager I want to have an overview that tells me whether content is broken or something needs attention.
PM-3	As a project manager I want to know which recent changes have been made to the language package I am responsible for.
PM-4	As a project manager I want to control when new content is published.
PM-5	As a project manager I want to know which content is published and which content is currently being worked on.
PM-6	As a project manager I want to know when translators or content authors have finished their work so that I can review it.

Table 5.2: Project manager requirements

#	Requirement
CA-1	As a content author I want to be able to edit content at any given moment so that I can be most efficient during my working hours.
CA-2	As a content author I want to be sure that I cannot break content while editing so that I can focus on what to edit and not how to edit.
CA-3	As a content author I want to collaborate with my colleagues on new content that I create.
CA-4	As a content author I want an overview of what I have changed during a session so that I can make sure that all the changes were intentional.
CA-5	As a content author I want to signal my project manager that I am done with my work so that the PM can review it.
CA-6	As a content author I want to edit content without fear of accidentally changing the live content so that I can experiment with some changes.

Table 5.3: Requirements of Content Authors

aspects, learnability and user error protection were chosen as the most important. Table 5.7 shows a list of quality requirements.

#	Requirement
TL-1	As a translator I want certainty that I am only translating content that is ready for translation so that I do not translate content that is not ready yet.
TL-2	As a translator I want to exactly know what I am supposed to translate.
TL-3	As a translator I want to inform the proofreader when my translations are ready for proofreading.
TL-4	As a translator I want to get an overview of what changes a proofreader has made to my content.
TL-5	As a translator I only want to see the information that is important for my job so that I am not distracted and can do my job efficiently.
TL-6	As a translator I want to be guided through the complex parts of the interface or given a tutorial so that I can focus on my job instead of learning a complex interface.

Table 5.4: Requirements of translators

#	Requirement
PR-1	As a proofreader I want to know exactly what I am supposed to proofread.
PR-2	As a proofreader I want to be able to provide feedback to the translator or comment on particular translations.
PR-3	As a proofreader I want to inform the translator when I have finished my work so that he or she can review what I did as soon as possible.
PR-4	As a proofreader I want an interface with a minimal set of features so that I can focus on the process of proofreading.
PR-5	As a proofreader I want to check how the content will look like to the end-user so that I can also check whether the text length is appropriate.
PR-6	As a proofreader I want to be guided through the complex parts of the interface or given a tutorial so that I can focus on my job.

Table 5.5: Requirements of proofreaders

#	Requirement
QA-1	As a QA professional I want to compare 2 different content releases so that I can see what has changed and identify possible problems.
QA-2	As a QA professional I want to see what has changed within a certain content package so that I know which parts of the content I need to review.

Table 5.6: Requirements of QA Professionals

#	Requirement
1	Users shall be able to start using the system without prior training.
2	The system shall be pro-active in offering help and advice.
3	Users shall be prevented from making errors wherever possible.
4	Errors shall not lead to permanent consequences, reverting should be easy.

Table 5.7: Quality requirements

Chapter 6

First Design Iteration

Based on the insights gained through task and requirements analysis as well as the literature study, a first semi-interactive prototype was designed. The design consisted of the most important views needed for covering the version control workflow. Additionally, a simple editing view was integrated, so that the tasks for the user study could be embedded inside a realistic scenario. The overall interface was strongly influenced by existing code hosting platforms, such as Github [**?**], Gitlab [**?**] and Bitbucket [**?**] as well as several Git GUIs [**?**]. The system can therefore be regarded as a breed between version control and content authoring tool. How the system compares to a traditional version control system is described in more detail below.

6.1 Comparison to Traditional VCSs

As compared to most other version control systems the interface only offers a minimal set of features. Some of them have been vastly simplified or adapted to the domain of language lesson authoring. A few features, such as the pull request and the history, are very close to their originals.

- There is no differentiation between *local and remote repositories* anymore. Church et al.'s [**?**] work has shown that hidden dependencies between these two often complicate matters for the user. Therefore, it was decided to have only one repository that is constantly up to date. This should, in theory, also simplify collaboration and avoid conflicts.
- Specifically *tracking* files is not necessary. Every course or lesson that is created using the tool will be under version control.
- There is no *staging* area anymore. As mentioned in chapter 4 this feature is sometimes problematic and often inconvenient, because every change has to be staged before it can be saved (committed).

- The *diff(erence)* view is presented by default before the user saves his or her changes. This ensures that the user knows what he or she is saving and provides an additional review mechanism. When using the Git CLI, diff is a separate command that needs to be executed when the user wants to look at what has changed.
- The diff view is enriched by a domain-specific design. It does not only display bare data as represented in the data format, but shows images, allows listening to sounds and visualizes boolean values, so that reviewing changes becomes simpler and is easier for users with a non-technical background.
- A *pull request* feature (as on Github) was added. Because Git offers no formalized way of reviewing code before it is merged this feature enforces best practices. Furthermore, the requirements analysis has shown that reviewing new content that was produced by freelancers is very important.
- Visualizations were added in order to help users understand some of the more abstract concepts, such as branches. This was inspired by Bitbucket, which is using different visualizations to explain certain features.

As the reader might have noticed, the prototype still uses the original version control terminology (except for committing which is now called saving). This was done in order to learn which terms would cause most problems and to discuss this topic separately within a focus group following the first user study.

6.2 Main Views

Below, the most important screens of the prototype are shown. The version control features are always accessible at the top in a horizontal navigation bar. The content in the editing view (Figure 6.2) can be navigated by expanding a tree structure that also reflects the inherent composition of the content.

A typical user flow would consist of creating a new branch (Figure 6.1), editing some content (Figure 6.2), saving it (Figure 6.3), optionally checking the edits in the history (Figure 6.4) and finally creating a pull request (Figure 6.5 and Figure 6.6).

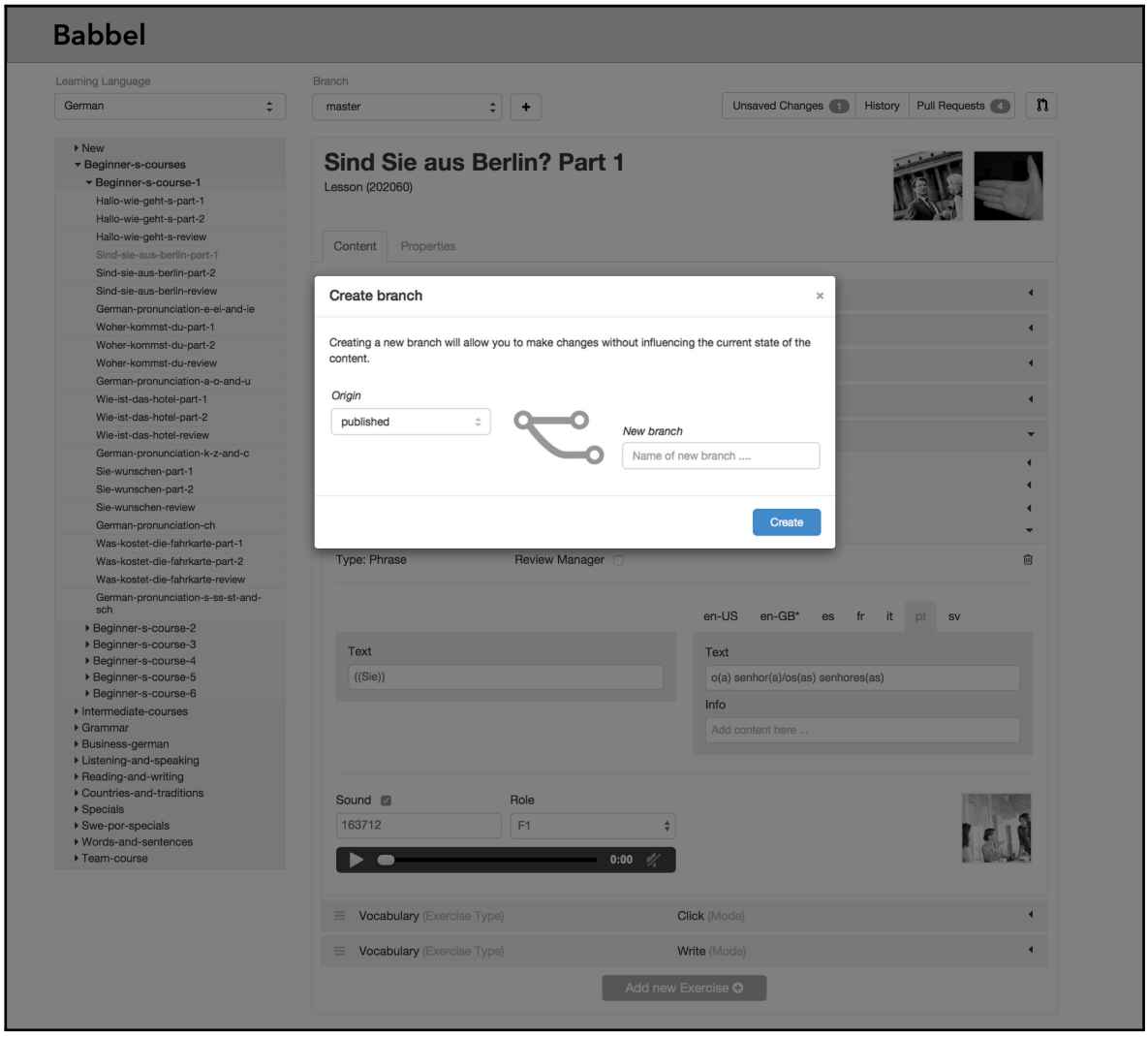

Figure 6.1: Create Branch

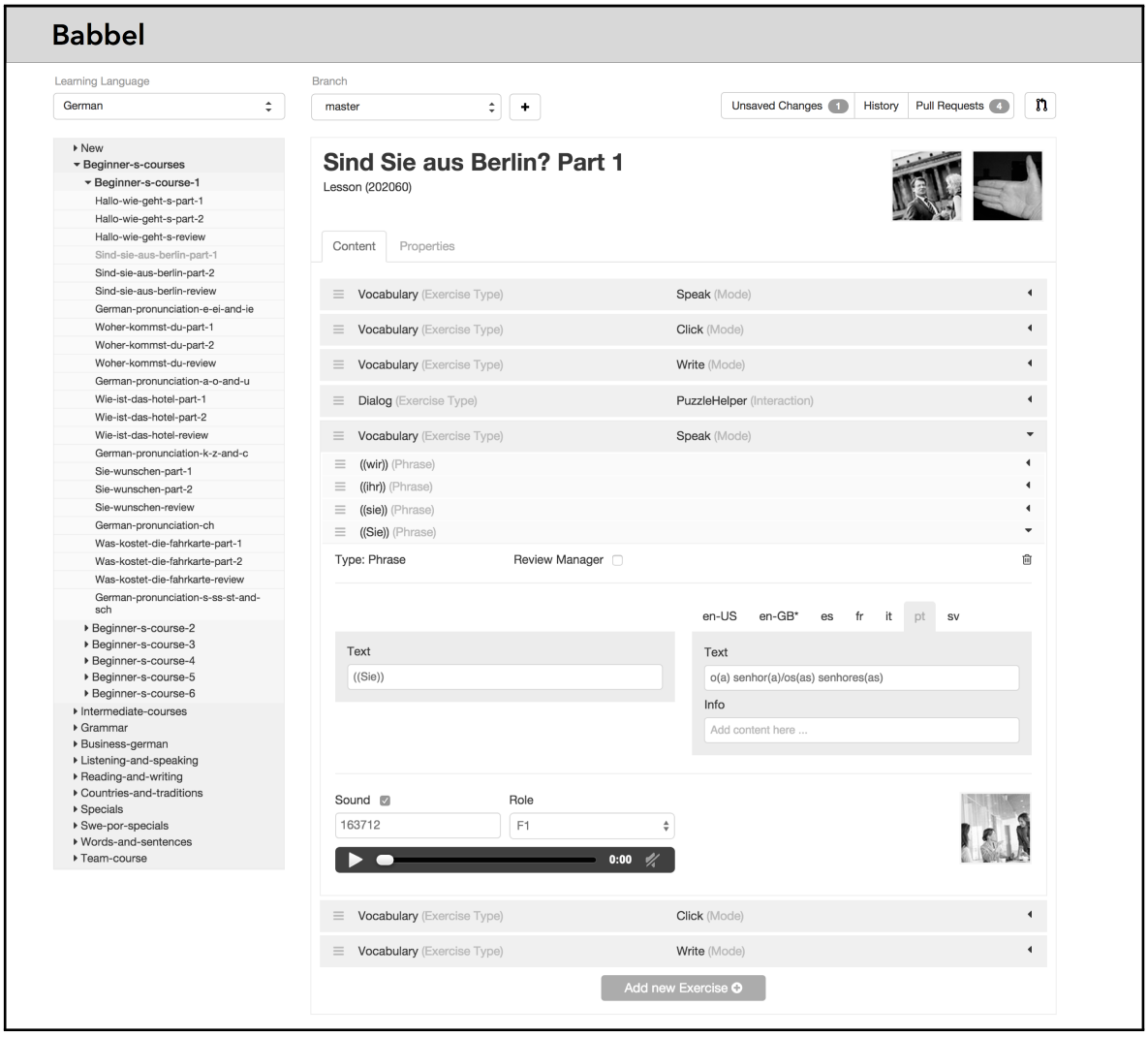

Figure 6.2: Editing View with Selected Lesson

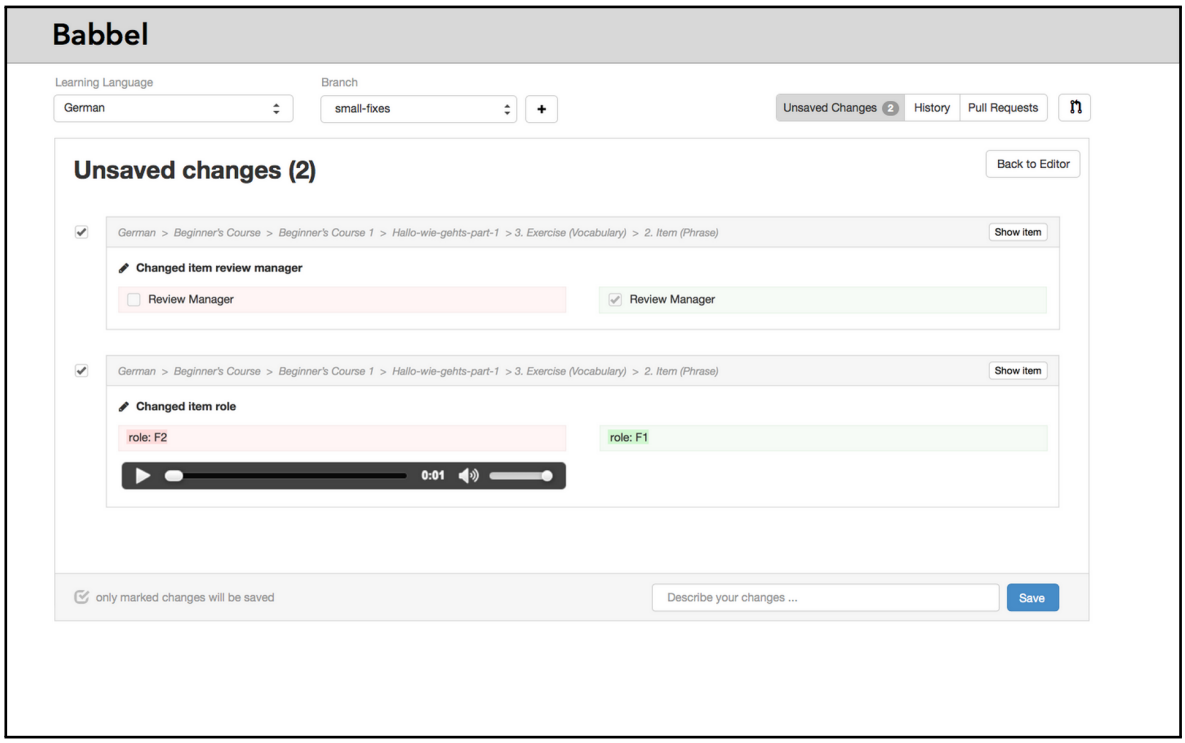

Figure 6.3: Unsaved Changes / Diff view

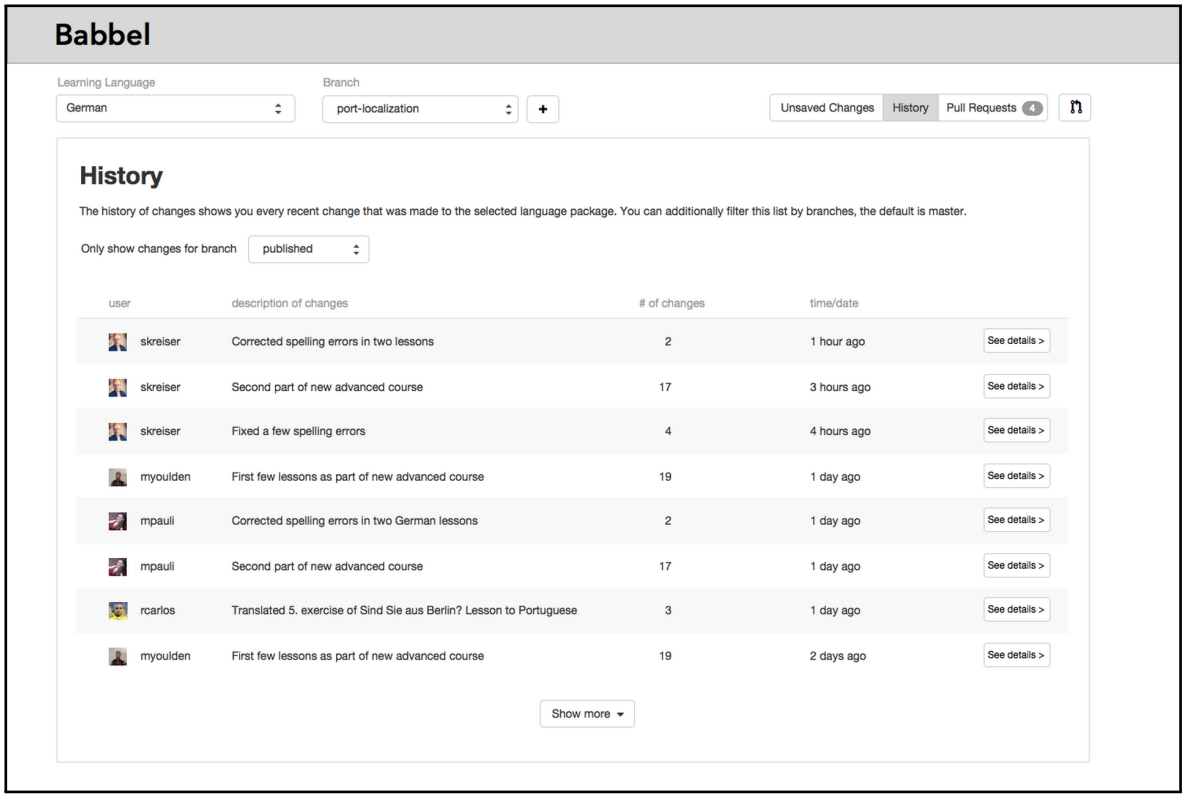

Figure 6.4: History

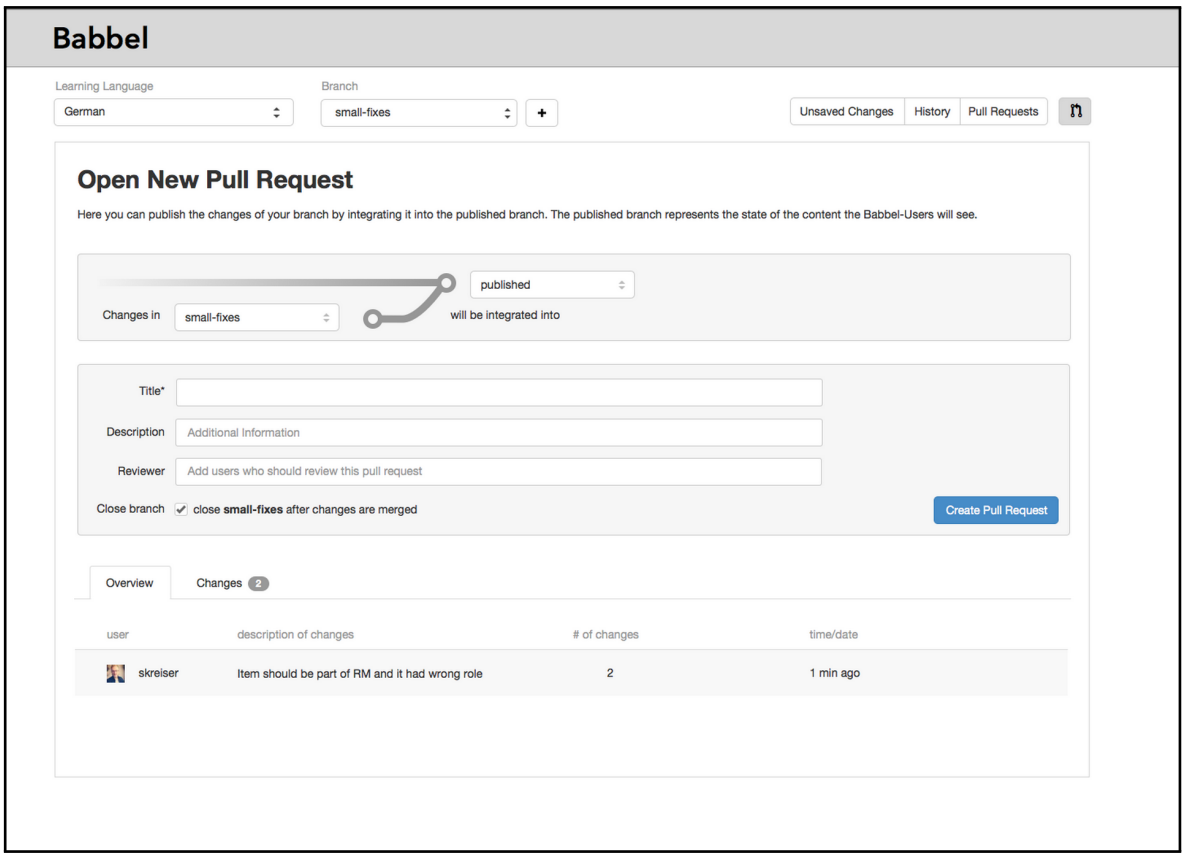

Figure 6.5: Create new pull request

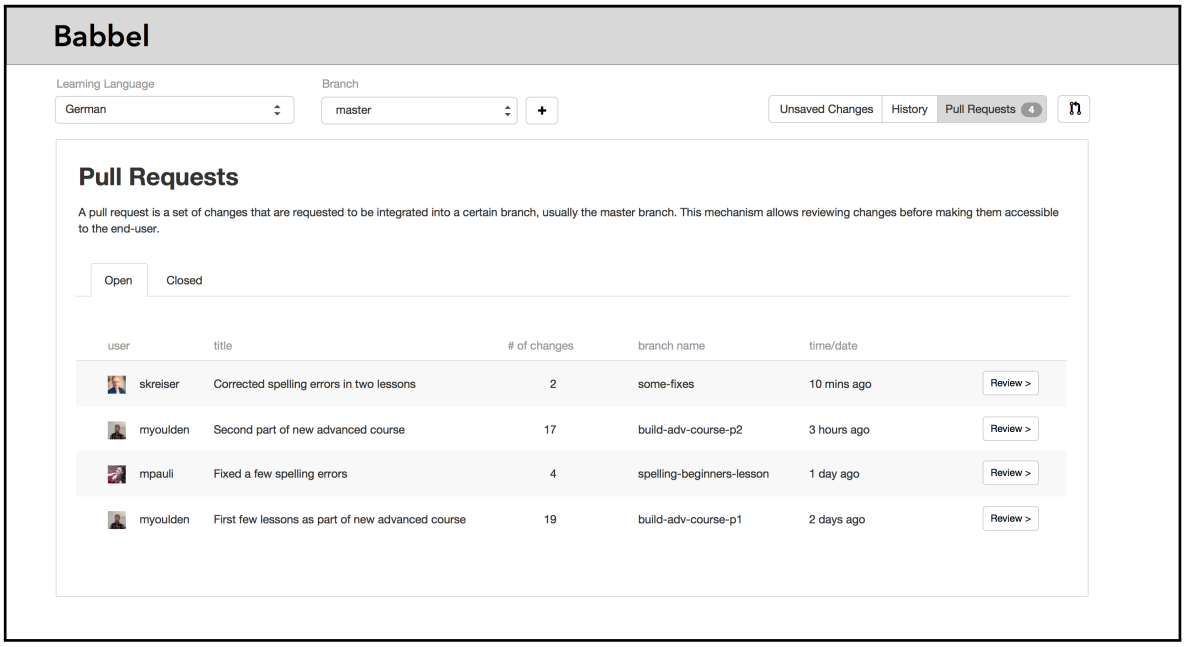

Figure 6.6: List of pull requests

Chapter 7

First Usability Study

Using the prototype described above a task-based usability study was performed. The goal was to evaluate whether the system could be used without prior training and how well the version control workflow could be integrated into the content authoring process. The 5 participants, all full-time employees at Babbel, were first-time users of version control.

Each session was divided into three phases. In the beginning a short pre-session questionnaire had to be answered, afterwards the participants had to perform three different tasks using the prototype and in the end an open discussion took place, during which participants could provide feedback on the interface. During each session the author was present as test moderator as well as one or two observers who took notes.

Even though the prototype was not fully functional, the study provided some valuable early feedback and revealed several usability issues (Section 7.2.2).

7.1 Study Design

Below a more detailed description of the study design is given. The purpose of this study was to make sure that severe usability issues were discovered as early as possible in the development process.

7.1.1 Participants

The participants were recruited randomly on a first come first serve basis. All participants were full-time employees at Babbel, but some of them have also worked as freelancers for Babbel before (Table 7.1). Most of the participants were content editors and some of them had special responsibilities, such as QA and localization. There was also one project manager who participated. This roughly reflects the distribution of content editors vs. PMs in the whole company.

Participant	Language Team	Job Position
1	French	Content Editor, QA
2	Portuguese	Content Editor
3	Portuguese	Content Editor, Localisation
4	Portuguese, Spanish	Project Manager
5	Turkish	Content Editor

Table 7.1: List of Participants

7.1.2 Metrics

Even though these quantitative measures are not statistically significant with a group of just 5 participants, they can still hint at potential problems, which should be investigated further.

- *Successful Task Completion*: percentage of tasks that were completed successfully
- *Time On Task*: time that was needed to perform a task
- *Error Rate*: participant deviated from ideal path of navigation, such as opening the wrong menu.

7.1.3 Scenarios

As suggested by Nielsen Norman Group the tasks given to the participants were situated within real-world scenarios [**?**]. This usually makes it easier for participants to engage with the interface in a realistic way. Furthermore, when some context is provided, the task itself does not need to be described as detailed, but rather the end-goal can be defined and the means of achieving this end-goal can be left open for the user. Three different task scenarios with an estimated duration of 4-8 minutes were created. The following task scenarios were given to the participants.

Scenario 1: Fix Error and Allow Review

A colleague (Firstname Lastname) has sent you a link to a lesson. She informed you that the second item in the vocabulary (write) exercise has the wrong speaker role (should be F1) and also should be added to the review manager. She asked you to correct the error and afterwards allow her to review your changes. Make sure youre not editing the live content.

Scenario 2: Publish Changed Content

A colleague (Firstname Lastname) has changed a lesson within the German language package. She asked you to review the changes. Look at the changes she made and if you dont find any errors make them available to the Babbel end-user.

Scenario 3: Eliminate Error and Save

A translator (user name) is currently working on a localization of German lessons (to Portuguese). Recently a translated exercise was published, but according to customer service users have complained that there is a translation missing within this exercise. Find the exercise which has been translated and locate the item with the missing translation. Add the translation and save your changes.

7.1.4 Sessions

Each session was scheduled to take no more than 45 minutes. In the beginning every participant received a short introduction to the reasons behind the study and the procedure of the session in order to make them feel more comfortable. It was highlighted that it is not them who were being tested but the interface. Furthermore, they were encouraged to speak aloud during the tasks, especially when running into problems. At the end of each session there was a short discussion where participants could voice their concerns or name the things they liked.

7.2 Findings

The results below are structured in the same way the testing sessions were. First, the results of the questionnaire are presented. Then, the quantitative metrics, recorded during the sessions, are shown in four different tables. In the end, problems that became apparent when observing the participants, are listed and described in detail.

7.2.1 Metrics

Several quantitative metrics were gathered during the user sessions. These include completion rates, error rates and time needed for the scenarios. The metrics serve as an add-on for the mostly qualitative user studies and help to interpret the observations and findings of the test sessions.

Task Completion Rate

The table below shows a summary of how many scenarios have been successfully completed. The completion rates for Scenario 1 and Scenario 3 could hint at potential problems, which are further discussed below.

User	**Scenario 1**	**Scenario 2**	**Scenario 3**
1	no	yes	yes
2	yes	yes	yes
3	yes	yes	no
4	no	yes	no
5	no	yes	yes
Success	2	5	3
Completion Rate	**40%**	**100%**	**60%**

Table 7.2: Task completion rate

Time on Task

The time on task was measured based on the video recordings of the sessions. The large differences between Scenario 2 and the other scenarios might be due to the different levels of difficulties inherent to the tasks presented in the scenarios. The table below highlights the shortest and longest times that were needed to finish a certain scenario. The times for Scenario 1 have the highest variance, which could point towards potential usability problems which were not encountered by all users. Scenario 1 asked users to not edit the live data, which meant that they had to create

a new branch. Since none of the participants had prior experience with version control, this was a particularly difficult scenario for some of them.

Scenario	User 1	User 2	User 3	User 4	User 5	Mean	Var
1	505	270	313	275	519	376	15624
2	81	226	73	170	90	128	4511
3	301	375	418	507	399	400	5550

Table 7.3: Time needed to perform scenarios (in seconds)

Error Rates

An interaction was counted as an error if the user departed from the ideal path and thus the interaction did not contribute to solving the overall goal of the scenario. The table below helps to paint a picture of how much trial and error was necessary. The error rate is not necessarily a sign for bad usability, it can also reflect the difficulty of a scenario. For example, Scenario 1 and 3 involved using similar parts of the interface, but Scenario 3 challenged a participant's mental model of the system whereas Scenario 1 was a little more straightforward.

Task	User 1	User 2	User 3	User 4	User 5	Mean	Var
1	3	1	3	3	1	2.2	0.96
2	0	5	0	3	0	1.6	4.24
3	3	6	7	3	2	4.2	3.76

Table 7.4: Number of errors done by each participant

7.2.2 Discovered Usability Issues

The following section describes the usability issues that have been discovered by observing participants performing the different scenarios. In total, 26 different problems were identified. 7 of them were so severe they prevented participants from finishing their tasks. 8 issues caused a significant delay or frustration for users. The remaining issues only caused minor problems or were suggestions for improvement.

In order to rank the issues by their impact on the user experience a severity score was used, as suggested by Sauro and Lewis [**?**]. The most severe issues, which prevented users from completing the tasks, were given a score of 10. 5 points were assigned if the issue caused a significant delay or frustration. 3 points were allocated for minor issues and just 1 point for improvements suggested by participants. To create an impact score the severity was multiplied by the frequency of

users who experienced the problem (severity * frequency / 10). A frequency of 40% means that 4 of 10 users encountered the problem, independent of whether they ran into the issue more than once. The scale ranges from 100 (most severe problem, experienced by all users) to 1 (suggestion by a single user).

The issues below are grouped by feature areas and ranked by their impact on the user experience. The most impactful issues are shortly explained.

Branches

As can be seen in Table 7.5 one of the most impactful usability issues was the difficult discovery of the branch creation functionality (Issue #1). During Scenario 1, 3 out of 5 participants spent more than half of the time by searching for this feature. Figure 7.1 visualizes this fact. The dark bars mark the point in time when users discovered the branch creation feature. The problem might have been a missing label for the button, which only consisted of a plus sign. Apparently, the proximity to the branch dropdown as well as the tooltip appearing upon hovering the button, did not suffice to make this feature discoverable.

There were two more issues with a severity score of 10. Issue #2: Some users dismissed the master branch warning (without reading it), which lead to even more confusion and prevented some of them from finishing the task.

The remaining issues highlight that the concept of a branch was only poorly understood (Issue #3) and that the naming of the default branch (master) is rather unfortunate (Issue #4), because it interferes with a different concept (master language) that describes the main translation language.

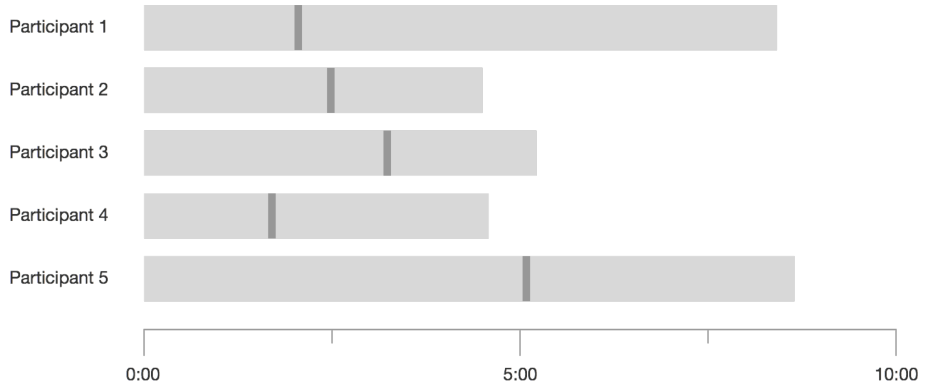

Figure 7.1: Total task completion time compared to discovery of branch creation button

#	Usability issue	Severity	Frequency	Impact
1	Create new branch button was not found	10	40%	40
2	Warning about editing content inside the master branch was ignored	10	40%	40
3	It is not clear that the master branch represents the public content and the content in all other branches is not visible to the end-user.	5	60%	30
4	The naming for the master branch was confused with the concept of a master language.	5	40%	20
5	After creating branch it is not clear that system switched to new branch already	5	20%	10

Table 7.5: Usability problems related to branches

Pull Requests

Pull requests are similar to branches in that they are one of the most complex concepts in version control. As suspected, they caused a couple of usability problems. The most important issue was related to the naming: deriving the functionality of a pull request just from its name is almost impossible (Table 7.6, Issue #6). When using a Git CLI the *pull* command fetches changes from the remote repository and merges them into the local repository. Therefore, a pull request describes a request to ”pull a new set of changes”. But since this command is not exposed to users inside the GUI of the content authoring tool, it is fairly difficult for them to guess what kind of functionality is concealed behind this name.

Diff

During Scenario 3, a high proportion of users had problems comprehending the list of changes presented to them (Table 10.7, Issue #11). In general, the diff view worked well, but when asked to find a missing translation, most users had a hard time understanding that something that had not been changed, would not appear in the diff view. Furthermore, the list of changes they did see, was confusing, because it showed empty fields, which some users mistook as the missing translation they were supposed to find (Figure 7.2).

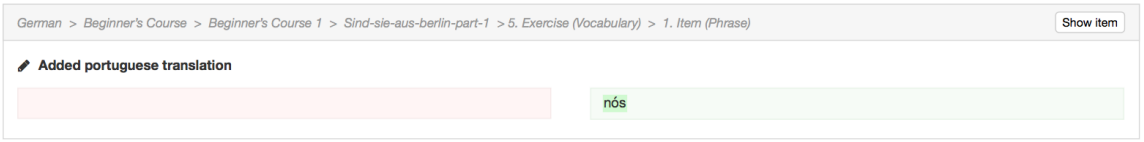

Figure 7.2: Part of diff view that displays an added translation

Saving Process

The most problematic issue related to saving was the non-existing save button at the bottom of the view (Table 7.8, Issue #14). Most users overlooked the saving feature in the navigation bar. It seems like most users would rather expect it to be attached to the lesson editor itself. The reason it is at the top is that saving is independent of content itself. In theory, several lessons could be edited and then only saved in the end altogether. This approach is very similar to how version control usually works, but it did not seem to resonate with most users. Many were anxious to navigate away from the current view after doing changes, because they feared of losing the changes. In most interfaces, saving is directly associated with a change that was done before. A solution needs to be found that is less counter-intuitive, but which still offers users a way of reviewing what was changed.

Main Editing View

The main editing view only caused a couple of minor problems. First and foremost, the way the content was structured and presented to the user was not the most intuitive (Table 7.9, Issue #19). Most users struggled with envisioning how the representation would translate into the actual exercises provided to the Babbel end-user. Additionally, the design did not provide a very good overview, because every exercise or item had to be expanded and at any given moment one could only see one at most.

The remaining issues were suggestions (Issue #21 & Issue #22) or related to the visual design of the interface (Issue #20).

Miscellaneous

In addition to the issues related to the major feature areas a couple of miscellaneous problems appeared throughout the system. None of the issues was of the highest severity, but they still caused some frustration. Especially Issues #23 and #25 reduced the smoothness with which users could move through the system. It seems that the naming for some of the navigation items was not descriptive enough and that icons without a label were more or less ignored. Some users proceeded by trial

and error, which does not speak for the clarity of the information architecture.

7.2.3 Conclusion

Concluding one could say that a long list of diverse usability issues has been identified. A number of severe issues prevented users from finishing their tasks, especially in Scenarios 1 and 3. This is also reflected in the somewhat low task completion rates for these scenarios. Furthermore, using central features, such as branching and saving posed some serious problems, which reflected on the entire system. Last but not least, the use of technical terminology (e.g. branch or pull request) prevented some users from forming an accurate mental model of the system. But, despite these flaws, a majority of users still succeeded in reaching their goals, which is more important than having an accurate understanding of the underlying concepts, which will probably improve over time.

Nevertheless, there are many things that need to be improved for the next iteration of the interface. The following chapter looks at one aspect of that: the terminology used within the interface. After that, Chapter 9 presents how the discovered issues were resolved and translated into a new interface design.

#	Usability issue	Severity	Frequency	Impact
6	Concept of a pull request is not clear (term might be confusing)	5	80%	40
7	Some users missed the reviewer field	10	20%	20
8	Confused pull request title with lesson title	5	20%	10
9	Branch visualization is unclear (release branch)	3	20%	6
10	Changes vs. Overview unclear (pull request detail view)	3	20%	6

Table 7.6: Usability issues related to Pull Requests

#	Usability issue	Severity	Frequency	Impact
11	New translations listed in the diff view are confusing, because the left column is empty (some users expected to see the learning language text)	10	80%	80
12	Not clear how to get from the diff view to the editing view	10	40%	40
13	red and green are confusing as colors (red is associated with a mistake)	3	20%	6

Table 7.7: Usability issues related to the diff view

#	Usability issue	Severity	Frequency	Impact
14	Save button was not found (expected at bottom of screen)	10	80%	80
15	Expects saved changes to be public right away	10	20%	20
16	Confused back-to-editor-button with review functionality	5	40%	20
17	Wording on saved changes confirmation page is somewhat confusing	3	20%	6
18	Concept of separated saving not entirely clear (across lessons)	3	20%	6

Table 7.8: Usability issues related to the saving process

#	Usability issue	Severity	Frequency	Impact
19	Visual representation/hierarchy of items/exercises not entirely clear	3	60%	18
20	Learning language text is overlooked (looks different than translations)	3	20%	6
21	Suggested a search functionality to find content faster	1	40%	4
22	Suggested pro-active system for translations	1	20%	2

Table 7.9: Usability issues related to the main editing view

#	Usability Issue	Severity	Frequency	Impact
23	Main navigation not completely obvious, clicking through (trial & error)	5	40%	20
24	History is not recognized for what it is. Most users take long to find it. (naming might not be obvious)	5	40%	20
25	Navigation: Icons without text are not understood ”What is this?” (pull request icon)	3	20%	6
26	No help offered when needed	1	20%	2

Table 7.10: Miscellaneous usability issues

Chapter 8

Focus Group

As the user tests have shown, terminology is a major usability issue. Participants have been exposed to entirely new concepts they were not familiar with before. Errors that have been made during the sessions were often related to a misunderstanding of what a feature does. How things are phrased influences a users comprehension of a system, her mental model, heavily. As described in the related work section earlier Gits conceptual model is flawed in certain ways, which is also reflected in the terminology. In order to avoid adopting these flaws for the content authoring tool, some Git concepts will be revised, reconsidered or combined in new ways. Therefore, new names need to be found for these concepts that reflect their purpose and help the user form an accurate mental model of the system.

8.1 Objectives

First and foremost the focus group is intended to provide a better understanding of the users thought process. Which terms are hard to understand or even misleading? Can they derive a functions purpose based on its name? Since the participants were already exposed to the terms and corresponding features they will be able to provide feedback based on their own experience. Besides gaining insights the focus group also aims at producing suggestions for improvements. How can terms be rephrased to allow an easier understanding?

8.2 Procedure

In the beginning the participants will get an overview of the agenda and a short introduction. The first phase consists of an open discussion among the participants. This should take no longer than 15 minutes. During this time the participants are going to talk about their difficulties when using the prototype in special regards to

the naming of the functionality. In case the discussion steers off into a different direction the group moderator will try to focus the discussion on the naming again by asking specific questions.

In the second phase the participants are expected to come up with alternatives to the current names. During the user tests it became clear that most participants did not really understand what version control is and why they have to perform certain steps. For the focus group a more informed understanding of version control will be helpful so that the participants can reflect on why they struggled. Therefore, a short introduction to version control is given by the moderator. Afterwards the participants will be asked to imagine alternatives for the most problematic terms, which have been determined through a survey and the previous usability study. First, participants will generate ideas by themselves and write them down on paper. Afterwards the whole group will discuss those ideas and the pros and cons of the proposed solutions.

8.3 Results

8.3.1 Pre-session survey

In a short survey the participants were asked about how well they had understood different version control concepts based on their names. The table below lists the ones that were perceived to be most difficult to understand. The scale reached from 1 to 5, 1 reflecting a hard to understand term/concept and 5 an easy one.

Score (mean)	Term
3.0	Pull Request
3.4	Create (Pull Request)
3.4	Branch
3.6	Merge (Pull Request)
3.6	Reviewer

Table 8.1: Least understood version control concepts

8.3.2 First Phase

The first phase, the open discussion, showed that still a lot of confusion prevailed among the participants regarding the most common version control concepts. It is hard to say whether this is because of the names used for these concepts or because the concepts are inherently hard to understand. Branch and pull request,

for example, seemed to be most problematic, but they are also conceptually the most complex. One of the participants even thought they are more or less the exact same thing. Another one mentioned, she could not tell the difference between the three concepts of unsaved changes, history and pull requests and had no idea in which order to use them. To conclude, one could say, that none of the participants (except for one) had formed an accurate mental model of the system, which is not surprising, given that they have only used it once and were not introduced to it before.

The participants agreed that it is easier to understand the concepts once they are explained first, but they also stated that freelancers likely will not have this luxury. One positive aspect that was highlighted is that the illustrations in the prototype helped form a more accurate comprehension of certain version control concepts (Figure 1). History, as one of the few concepts, seemed to be clear to most people. This also reflects the 4.0 average score in the pre-session survey. Something that is peculiar, is that the term unsaved changes fared well during the survey (4.2), but seemed to pose conceptual problems to some participants.

Figure 8.1: Illustration that visualizes what happens when two branches are merged

8.3.3 Second Phase

At the beginning of the second phase the moderator gave a short introduction to version control and the most important concepts. This was done to provide the participants with a solid understanding of version control and its concepts, thus making it easier for them to come up with alternative names. The brainstorming proved to be more difficult than expected, but yielded a handful of alternative terms for each concept that had been deemed hard to comprehend earlier. What made it difficult to find different names for some concepts was the large domain they have to work for. For example, a translator creates a pull request in order to request proofreading, but one can not name it request proofread, because for some users it fulfills a different function. Some of the terms listed in Table 8.2 have a slight deficit in that they do not properly reflect the underlying concept. For example, Publish Changes will not work, because one can also merge changes into a branch that is not public (not the master branch). Furthermore, Request of Changes sounds more like someone is

requesting to change something and not like he or she has already changed something. Besides these negative examples, there are a couple of viable options that, after further consideration, could become part of the next prototype.

Pull Request	**Merge (Pull Request)**	**Branch**
Check Changes	Finalize	Working version
Request of Changes	Publish Changes	Experimental version
Request Approval	Accept Changes	Copy
Request Review	Approve Changes	Draft
	Save	Public/Private Version
	Authorize	

Table 8.2: Suggested alternative terms

8.4 Conclusion

The focus group has shown why naming things is one of the hardest parts of programming. One needs a deep understanding of the concepts that shall be named. Furthermore, it needs to be clear what the name is supposed to signify and to which target group. What became apparent during the group session, is that it is almost impossible to detach a users understanding of a concept from its name. A weak mental model of a system can be the result of inconsistent concepts or poorly given names, or both. Nevertheless, some viable alternatives were perceived, which will serve as an inspiration for creating the next iteration of prototypes.

Chapter 9

Second Design Iteration

The previous usability study revealed several usability issues that prevented users from achieving their goals. This chapter outlines how the most severe issues have been addressed and presents the revised interface design. The sections below are structured based on the different feature areas of the interface. Figure 9.1 might help to visualize the relations between them: The main editing view as well as the diff comparison are very central, whereas the remaining features are more or less equally important. Please note that the diff is not an independent view of itself but is embedded in several other views, such as the history, the saving page and the merge request details.

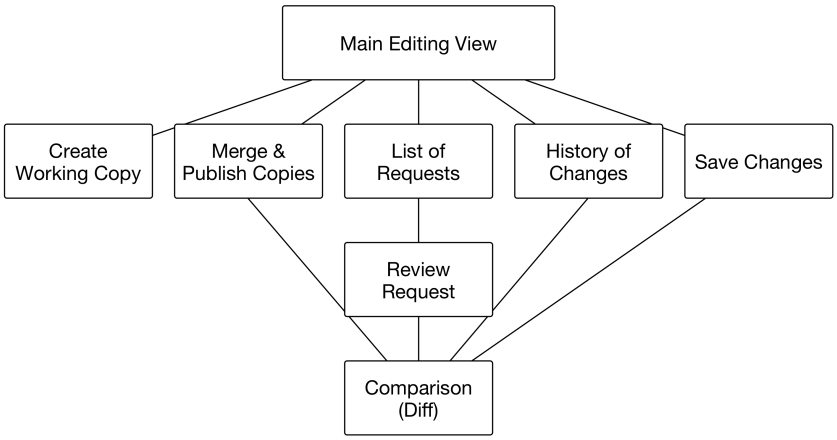

Figure 9.1: Main parts of the application

9.1 Navigation and Terminology (Issues #1, #6, #23, #24, #25)

During the first usability study as well as the subsequent focus group, the version control terminology has been identified as a major usability issue (Issues #4, #6

and #23). Users especially struggled with grasping the concepts behind branches and pull requests. Based on the results and the input from the focus group, new names were conceived for some of the main features. Most notably, branches are now called working copies, or just copies, and pull requests are referred to as merge or publish requests. The term copy was chosen, because it is part of everyday language and much less abstract than the term branch. ”Pull” was swapped for ”merge” or ”publish”, because it describes the action of combining two different states of the content much better.

Additionally, the names of most features now signify an action (unsaved changes vs. save changes, pull requests vs. review requests, create pull request vs. merge & publish). The intention behind this was to signal the result of an action to the user upfront and thus allow him or her to anticipate the result of an interaction.

Besides improving the used terminology, the design of the navigation bar was also slightly tweaked. Most importantly, all items now have labels and icons that visualize the feature, which could help users to orient themselves in a better way (Issues #1 and #25). Because this takes more space now, the repository (language package) selection has been moved to the upper right corner (Figure 9.2).

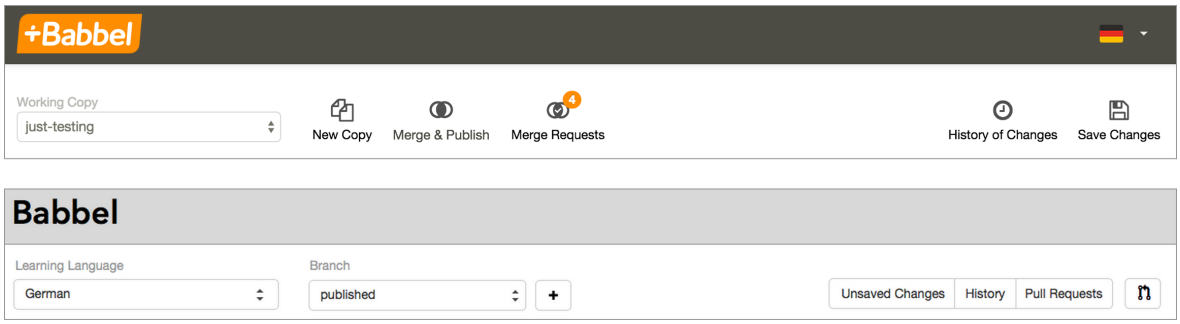

Figure 9.2: New and old navigation bar

9.2 Branches/Working Copies (Issues #1, #2, #3, #4)

During the previous usability study the branch concept presented some serious problems to users. First of all, most participants were confused when being informed that editing the main branch is not possible (Issue #2) and just ignored the message. Afterwards, many took some time till finding the functionality of creating a new branch (Issue #1).

The warning message in the previous prototype (Figure 9.3) did not offer any solutions to the users. This lead to a situation where many users were lost after the message appeared. In order to mitigate this problem the new warning message

now offers solutions right away. Users can choose to create a new working copy or dismiss the warning and edit the live content anyway.

Additionally the main branch was renamed from *master* to *published* to make it even clearer which kind of content is inside, thus fixing Issue #3 and #4.

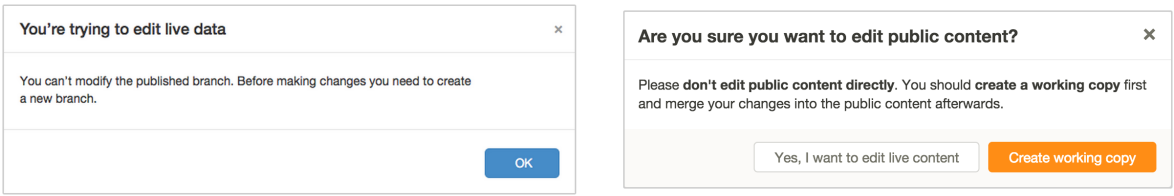

Figure 9.3: Live data warning old vs. new

9.3 Improved Diff View (Issues #11, #12, #13)

The diff view posed some problems to users when looking at added translations (Issue #11). Because the previous state was no translation at all the previous state was empty and therefore the view showed a red empty box. Some users mistook this for a missing translation or an indication of an error, because of the red color (Issue #13). The new design used a gray box instead to show that there was nothing before (Figure 9.4).

Issue #12 was addressed by underlining the breadcrumb, so that it becomes clearer to users that they can interact with it. Furthermore, instead of the button saying ”show item” it was now labeled ”edit lesson”, which is hopefully clearer as well.

Figure 9.4: Improved diff view

9.4 Saving Process (Issues #14, #16, #18)

As Figure 9.5 shows the editor view now offers a shortcut for saving changes. In the design of the first iteration users were forced to use the save changes view (Figure 9.10), which also shows the differences between the old and the new state of the content. This is of course useful if a lot of changes have been done, especially

across several lessons. But, the last study has shown, that users do not necessarily work that way (Issue #18) and that it is often more convenient to save small changes right away. Therefore a simple save button at the bottom of the content table was introduced.

Furthermore, Issue #16 was fixed by replacing the ”back to editor” button with a simple *x*, which is often used for closing overlays.

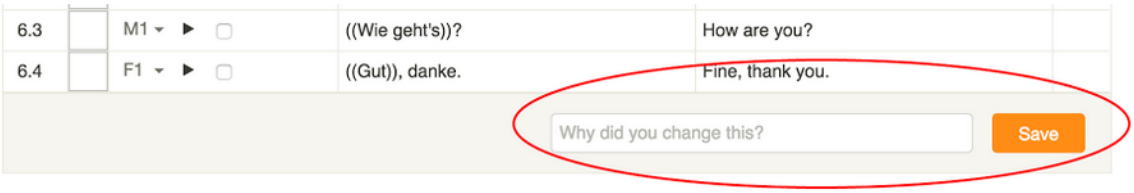

Figure 9.5: Saving shortcut

9.5 New Content Representation (Issues #19 and #20)

For the new interface a different content representation was designed (Figure 9.6). The old one, which was using so called accordions, hid a lot of information in the beginning. It was one of the reasons why Scenario 3 presented such a hurdle for most users during the last study (Issues #19 and #20). The new design uses a table layout which presents most of the important information at a glance. The intention is to provide a better overview and thus enable users to find errors or problems in the overall lesson structure more quickly.

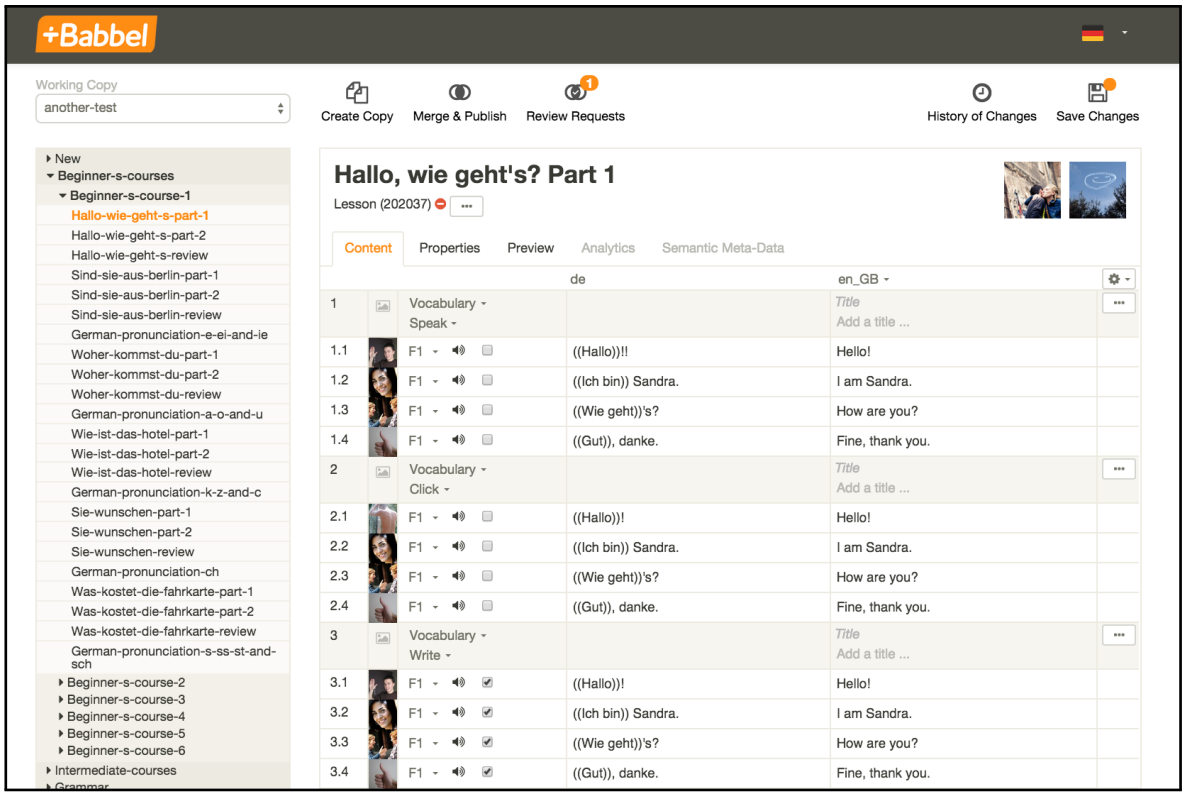

Figure 9.6: Main editing view

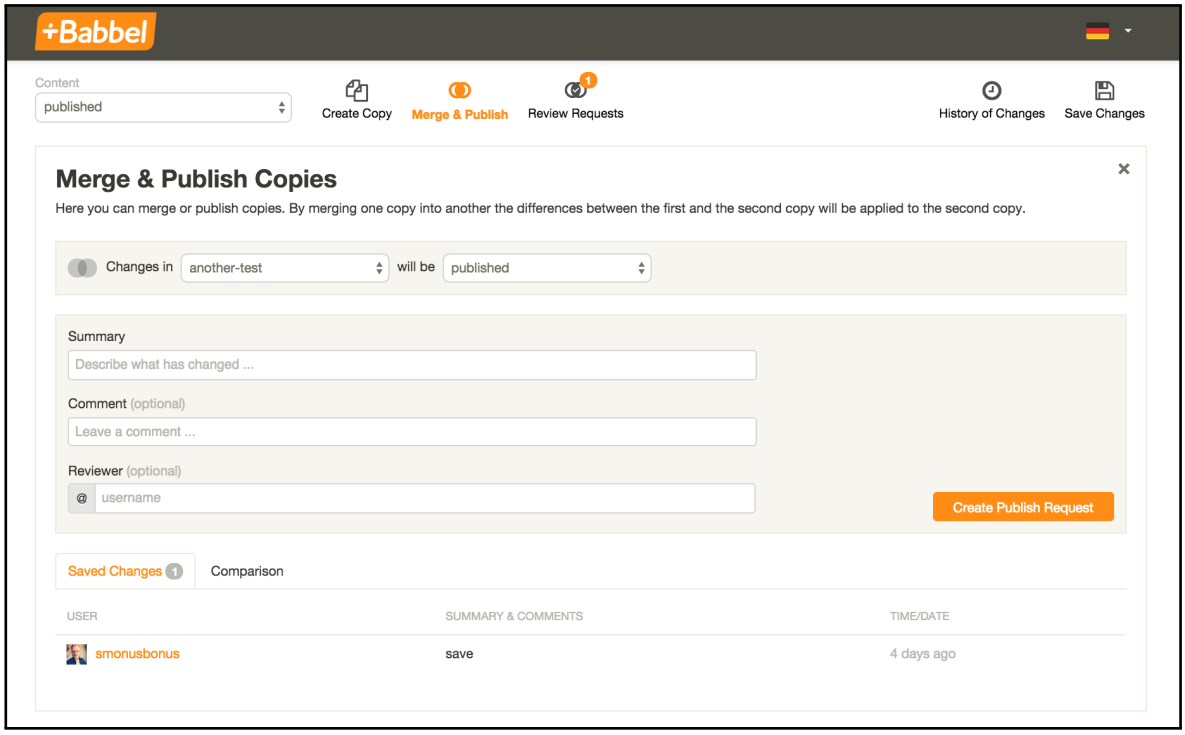

Figure 9.7: Merge and publish copies

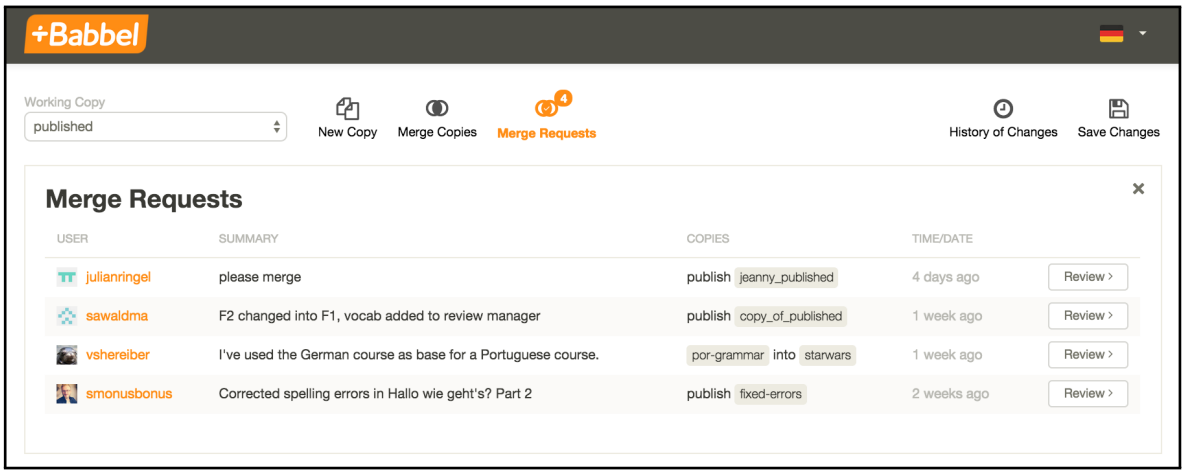

Figure 9.8: List of requests

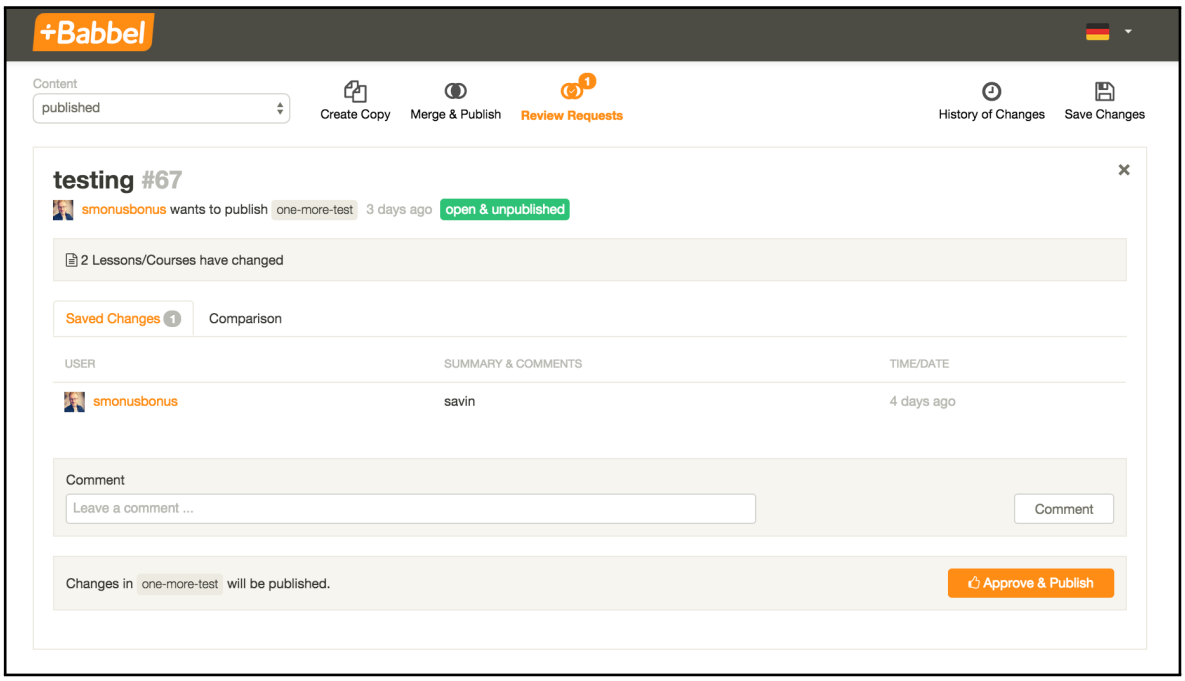

Figure 9.9: Review request

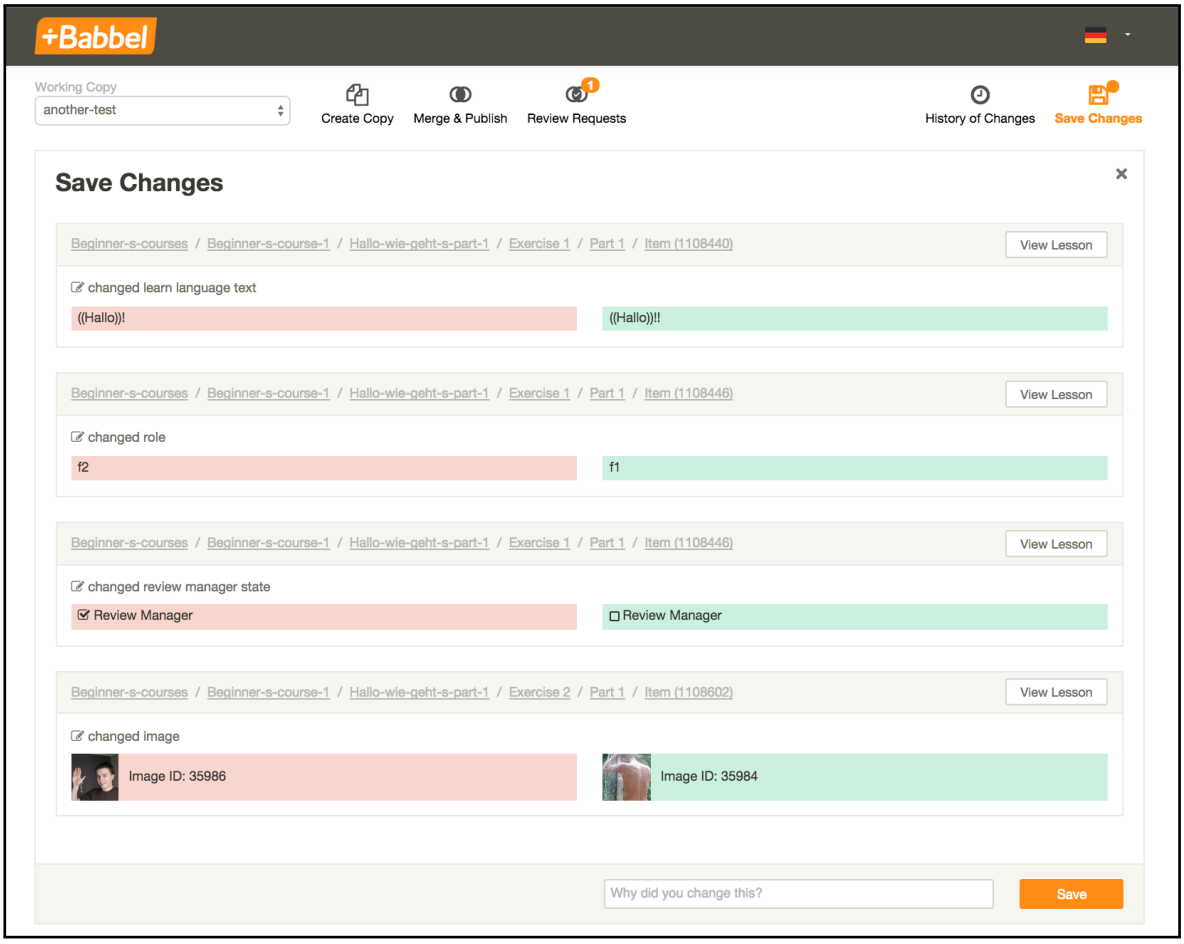

Figure 9.10: Save changes

Chapter 10

Second Usability Study

The previous chapter described how the insights gained during the first study have been translated into redesigned interactions and new interface components. This chapter now looks at how, during the second usability study, the effects of these changes were measured.

10.1 Study Design

The first study was mostly about evaluating the high-level aspects of the interface and finding out which areas of the application caused the most serious problems. In contrast, this study is more about validating that the introduced changes had a positive impact and that the application is functional and usable from end-to-end. This type of study is usually called verification or validation study [**?**] [**?**]. Whereas the prototype in the previous study still had many deficiencies, the system that was tested in this study was already functional to a large degree. The purpose of the study was to assess whether the application is fit for usage in the didactics department and to establish a set of standards for the first time. These standards are determined by the metrics and the post-study questionnaire described below. The metrics for example, can answer whether a certain goal can be reached with a low number of errors or within a certain time-frame. The questionnaire answers the question how well the system is perceived in general and which areas of the system still have room for improvement. These standards can be used as a reference point for verifying that all subsequent changes to the interface have a positive effect.

10.1.1 Participants

The participants of the second usability study were different from the ones during the first study. This was done to ensure that no learning effect would influence the results. Furthermore, it was important that users encountered the concept of

version control for the first time when performing the tasks. All participants were selected randomly on a first come first serve basis. As can be seen in Table 10.1 they represented a diverse group of people. Most of them were part of different language teams and also had different responsibilities within the content creation process. Most participants were content editors and one was a project manager.

As compared to the first study the number of participants was doubled. This was done in order to get more reliable results for the metrics and the newly introduced post-study questionnaire.

Participant	Language Team	Job Position
1	French	Content Editor, QA
2	Swedish	Content Editor
3	German, Polish	Project Manager
4	French	Content & Image Editor
5	Portuguese	Content Editor
6	French	Project Manager
7	English	Content Editor
8	Russian	Content Editor
9	Russian	Content Editor
10	German	Content Editor

Table 10.1: List of Participants

10.1.2 Metrics

The same quantitative metrics as during the first study were collected. Comparing those might provide helpful insights into what has improved or not.

- *Successful Task Completion*: percentage of tasks that were completed successfully
- *Time On Task*: time that was needed to perform a task
- *Error Rate*: participant deviated from ideal path of navigation, such as opening the wrong menu.
- *Scenario Specific Metrics*: such as how long it takes to find the missing translation in Scenario 3 or how quick users discover the create copy function

10.1.3 Scenarios

Scenarios 1 to 3 were identical to the ones used in the first study (Described in Section 7.1.3). This was done in order to make comparing the results of the two studies simpler. The interface could be regarded as the independent variable. This means that new results are not influenced by a changed session procedure but instead can be directly attributed to the changes in the interface. The fourth task has been added to the study in order to lay more focus on the new content representation (main editing view).

Scenario 4: Create Exercise and Fill With Content

Add a new memory exercise to this lesson (Link provided). Please dont edit live content. The title of the exercise should be awesome memory. The translation visibility partial. Add 3 items to the exercise. Assign random images to the items. Assign all items the speaker role F1. Add all items to the review manager. Finally save your changes.

10.1.4 Post-Study Questionnaire

As an additional means of gathering feedback users were asked to answer a post-study questionnaire. A standardized questionnaire called PSSUQ (Post-study System Usability Questionnaire) was used. It consists of 16 questions that produce 4 different scores, which signify how well the system was perceived (Overall, System Quality, Information Quality, Interface Quality). The goal of utilizing this questionnaire was to reveal potential weak spots of the interface that were not discovered by observing participants. Note that the third version of the questionnaire was used, as recommended by Sauro and Lewis [**?**].

10.1.5 Sessions

Each session was scheduled for one hour. In the beginning participants were introduced to the procedure and the purpose of the study. Afterwards they had to fill in a short questionnaire which asked about their experience with version control systems and the current content authoring tool.

The largest part, going through the scenarios and performing the described tasks, was scheduled to take about 30 minutes. Some users took longer than that, but most finished within the expected time-frame.

Lastly, an open discussion was initiated were questions that arose during the scenarios could be answered and users could provide some general feedback. Usually

they were asked about their general impression of the interface and what they liked and disliked most about it. The session concluded with the aforementioned PSSUQ.

10.2 Findings

The following section describes the most important findings of the second usability study. Where meaningful the findings are compared to those of the first study. These include quantitative metrics as well as the discovered usability problems.

10.2.1 Metrics

As during the first study quantitative metrics were recorded. Task completion rate, time on tasks and error rate. Furthermore, a few task-specific metrics were measured. For example for Task 1 it was measured how much time users needed to find the create copy/branch feature. This allowed for a more meaningful comparison as is described in the section below. For task 3 the time for finding the missing translation was recorded.

The tables below show a comparison between the metrics recorded during the first user study and the second. It should be noted that the studies carried out are focused on qualitative findings, which are mostly based on observations and discussions. The metrics only extend and support these findings, but they do not stand for themselves. Because of the relative small sample sizes and partially large variances this data should not be overrated.

Task Completion Rate

The task completion rates did not change that much except for Task 1 for which it increased 11 percentage points. One possible explanation is the newly designed warning message, that now offers a direct solution instead of just informing users (explained in more detail in the scenarios section below). Unfortunately, 61% is still an unsatisfactory completion rate. Exactly why it is still so low is hard to tell. One reason for the low completion rate, especially in comparison with the other tasks, might be that users had to get used to the interface at first. The task order was not randomized but always remained the same. A randomized task order might have resulted in a more balanced rate, in particular between Task 1 and 3, which according to the participants were the most difficult tasks (4.8 and 2.7 on a scale from 1-7 where 1 equals very difficult).

Time on Tasks

As Table 10.3 shows the biggest difference is again observed for Task 1. Participants during the second study needed about 75 seconds less for finishing this task as participants in study 1. The scenario-specific metric provides an explanation for

Scenarios	First Study	Second Study	Difference in pp
Scenario 1	50%	61%	+11 pp
Scenario 2	100%	94%	-6 pp
Scenario 3	70%	72%	+2 pp
Scenario 4		100%	

Table 10.2: Comparison of task completion rates during first and second user study

this. As can be seen in Figure 1, participants of the second study needed much less time to locate the create branch/copy feature then participants in study 1 who almost spent half of the time searching for this feature.

Scenarios	First Study (mean)	Second Study (mean)	Difference
Scenario 1	367	292.25	-74.75
Scenario 2	128	146.57	18.57
Scenario 3	395.5	369.42	-26.08
Scenario 4		280	

Table 10.3: Time on tasks (in seconds)

Errors

Even though the other metrics show most improvements for Task 1 this is not the case for the error rate. Here, there is only a slight decrease of errors for Task 1, whereas errors during the other tasks (2 and 3) have halved. A possible explanation for this might be the changed terminology, which allowed users to better orient themselves (action verbs and more everyday language than technical terms). It seems there was less trial and error when finding the right feature. The observations support this assumption; there were less users asking things like What is a pull request?.

Scenarios	Study 1	Study 2	Difference in Errors
Scenario 1	2.2	1.87	-0.33
Scenario 2	1.6	0.5	-1.1
Scenario 3	4.2	2.12	-2.08
Scenario 4		0.5	

Table 10.4: Comparison of error rates during first and second user study

10.2.2 Impact of Design Changes

The major changes implemented for this study had different levels of impact. Some of the changes were quite noticeable during the sessions whereas others are harder to validate.

Navigation Bar & Changed Terminology

The new navigation bar did not attract a lot of attention, which is probably a positive sign. The reduced error rate could at least be partially attributed to the improved naming and the icons (Table 10.4). In general, there seemed to be less trial and error when users navigated the application. During the previous study remarks like ”What is a pull request?” and ”I’m just going to click through, I have no idea” suggested that there is a usability problem. This time, these remarks were much less frequent. Especially the new terms ”working copy” and ”merge request” seemed to be much better understood than their predecessors. Furthermore, using action verbs to highlight what a feature can do instead of passively describing it cleared things up as well.

Saving Process

A positive outcome of the newly added saving shortcut was that it was used a lot. The downside on the other hand was that some users got confused by the existence of two different saving features. The shortcut save sped up editing and users did not seem to miss the list of changes they would got with the ”advanced” saving feature. What has become apparent is that the old saving feature, which concept-wise, is very much inspired by the staging area of version control systems, might not be the most intuitive solution for content editors. In programming there is often a need of touching many files at once to implement a new feature. In content editing this is seldom the case. Changes are usually logically tied to a single lesson.

Live Content Warning

Task 1 asked users to apply two small changes inside a lesson. The difficulty was that they were not supposed to edit the live content. That meant that they had to create a copy (branch) of the content first, make the changes, save them and finally merge the copy with the public content again. During the first study participants wasted a lot of time finding the create branch/copy feature after they had been alerted to not edit the live content. As Figure 10.1 shows this time could be greatly reduced through the new design in the second study. The dark grey bars show the moment in time when users discovered the feature. During the first study users

spent almost half of the time with finding this feature whereas during the second study they discovered it a lot earlier. This also had an effect on the average time taken for Task 1 in total and the completion rate. These improvements are probably due to the redesign of the warning message as described in the previous chapter (Figure 9.3).

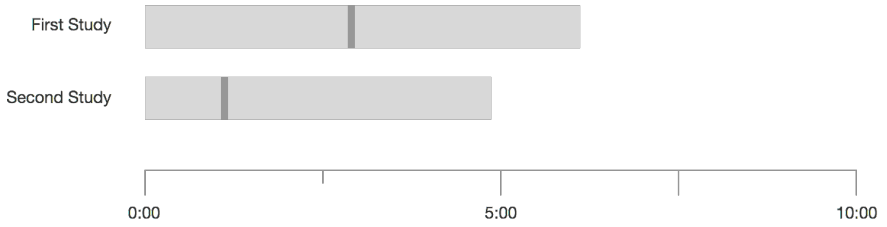

Figure 10.1: Average task completion time and time of branch/copy creation

Content Representation

In general the new content representation performed quite well. This became apparent in Task 3 when users had to find a missing translation. Whereas, during the previous study, some users had problems locating this missing field within the content representation (because it was initially hidden), this was no longer a problem with the new design. Once users had arrived at the content of a lesson and selected the right language they could locate the missing translation quite quickly. The only noteworthy issues of the new design were malfunctioning popovers (did not close automatically) and labels that looked placeholders and were therefore selected by some users (instead of the actual input field).

10.2.3 Discovered Usability Issues

Even though several usability issues were fixed as compared to the first study, there were also a lot of new ones coming up. Some of this might be due to the fact that users could now freely interact with the tool and were not constrained by the prototype design as in the first study. In total, the number of rather severe issues (Severity score of 10 or 5) was reduced from 15 in the previous study to 11 during the second study. Also the impact of most issues was lower this time, which can be attributed to less users experiencing the problem.

The following section lists the usability issues discovered during the test sessions. As for the first study, the impact score by Sauro and Lewis [**?**] was calculated for each issue and the tables are sorted by the highest impact scores. Please note,

in order to avoid confusion later on, the issue numbers are continuous and thus start with 27.

Working Copies

Working copies, which were previously called branches, were in general quite well understood. Nevertheless, they also caused some usability issues. A common misconception was that a working copy only consists of one lesson, where in reality it contains the whole language package (repository). Furthermore, the interface did not provide appropriate feedback when errors occurred, the feedback was either too general or completely non-existent (Table 10.5). Additionally, some users did not realize that after creating a new copy it was already selected. The name in the dropdown changed, but it is barely noticeable if you do not know where to look.

#	Usability issue	Severity	Frequency	Impact
27	User does not know whether newly created copy is already selected	10	40%	40
28	No error message when copy with existing name is created	10	20%	20
29	Error message too general when spaces are used in copy name	5	30%	15
30	Users have wrong understanding of working copy (think that lesson is copied)	3	40%	12
31	Copies accumulate quickly - results in long dropdown list which is hard to scan quickly	1	20%	2

Table 10.5: Usability issues related to working copies

Merge Requests

As Table 10.6 shows merge requests were the root of various usability issues. First of all, there was a missing error message when users tried to create a merge request based on a copy that did not contain changes (as compared to the *published* content). This happened when users forgot to save and went straight to the merge request view. Furthermore, the reviewer functionality was not always discovered, because it is not prominent enough.

#	Usability issue	Severity	Frequency	Impact
32	User tried creating merge request before saving (no error message)	10	20%	20
33	User does not know how to let colleague review changes	10	10%	10
34	When approving a merge request and commenting first comment confirmation is forgotten	3	30%	9
35	Commenting in request detail view (diff) does not result in feedback	3	10%	3
36	The list of requests does not show the reviewers at a glance	3	10%	3

Table 10.6: Usability issues related to merge requests

Diff

The diff view, which was only subject to a small set of changes as compared to the last iteration, caused the most problems. This is surprising, since these issues have not been discovered during the first study. Maybe this time users were better able to articulate themselves or the root of their frustration was more obvious in a fully functioning system. The main problem with the diff view, which design is heavily inspired by Github, was that it does not provide enough context. The listed changes were only of limited value, if not meaningless, without the context they originated from (lesson or exercise). In programming a single method or class can usually be assessed in isolation, but for language learning it is much more important to know the context of a certain element.

#	Usability issue	Severity	Frequency	Impact
37	It is hard to deduce the structure of the content from the diff view	10	90%	90
38	Diff view is missing context (i.e. text that was translated)	5	80%	40
39	It is not clear how to get from the diff view to the content	10	20%	20
40	Change description is above red box in diff view which is confusing (what has changed?)	5	30%	15

Table 10.7: Usability issues related to the diff view

Saving process

The saving process, as described before, now offered two options, a direct save below the editing view and a more advanced option where users would see their changes again. The positive aspect is that many users used the new saving feature and most of them did not even notice that there were two options. The flipside of this is, when users did notice there were two options, they were severely confused. Two more things that caused problems were again related to feedback and error messages (Problem 1 & 2, Table 10.8. When users try to save without entering a description (commit message) the input field turns red, but there is no explanation on why this is needed. Furthermore, after saving changes using the advanced option, the *blank slate* [**?**] appears telling users ”no unsaved changes”. This is rather confusing, because the blank slate is a design pattern that is aimed at helping users out in case they encounter a dead-end or empty view. Instead there should be a confirmation informing users about a successful save.

#	Usability issue	Severity	Frequency	Impact
41	Difference between two saving features unclear	10	30%	30
42	User tried saving without entering description	3	90%	27
43	No unsaved changes? Blankslate after save is confusing - no confirmation (same for shortcut)	5	50%	25
44	There should be a suggestion after saving changes what to do next	1	30%	3

Table 10.8: Usability issues related to the saving process

Main Editing View

As mentioned before there were relatively few problems arising in the main editing view. The most severe one only influenced a single user who got confused because the translation column was showing a different language than the one she had selected before. The view does not ”remember” the display language setting. It always switches back to a default when the user comes back from a different view. The other issues were only minor ones that users could easily recover from. Nevertheless they should be fixed to improve the editing workflow.

#	Usability issue	Severity	Frequency	Impact
45	Clicked on label that looked like placeholder in order to enter data	3	40%	12
46	There is only one (not obvious) way of closing a popover	3	30%	9
47	A comment functionality was suggested	1	10%	1

Table 10.9: Usability issues related to main editing view

10.2.4 Post-study Questionnaire

Table 10.10 shows the resulting scores of the Post-Study System Usability Questionnaire (PSSUQ). The lower a value the better (Scale from 1 to 7: 1 being strongly agree and 7 strongly disagree). These scores are based on the mean of 16 items that comprise the PSSUQ. The list below shows which items contribute to which score. The complete catalog of questions together with the average ratings can be found in the appendix (Table 10.10).

- *Overall:* Average of items 1 through 16
- *System Quality:* Average of items 1 through 6
- *Information Quality:* Average of items 7 through 12
- *Interface Quality:* Average of items 13 through 15

As can be seen the overall feedback was quite positive although there is still room for improvement (2.83 vs. the neutral value 4). Participants seemed to be especially pleased with the interface quality, which basically describes the aesthetic and functional quality of the system. The high score could partially be due to the sharp contrast between the existing content authoring tool, which is very cluttered, and the new one. System quality, which assesses ease of learning and the general usability received a good average score as well (2.51). The fourth score, information quality, did not fare as well as the other ones (3.38). Information quality describes how much documentation is offered, how well error messages are designed and how easy it is for users to recover from their mistakes. The reasons for the low score are probably a non-existent documentation and still a lot of missing error messages, as described earlier. Some users ran into issues where the system did not react to their input but also did not provide any feedback which would have allowed users to recover.

4 scores	Rating (mean)
Overall	2.7
System Quality	2.51
Information Quality	3.31
Interface Quality	2.07

Table 10.10: Average scores for the 4 dimensions

10.2.5 Conclusion

What has become apparent by looking at the findings above is that almost all changes that were introduced had a positive impact. For some of these improvements the metrics speak for themselves: the reduced time for Task 1 due to the redesigned live content warning and the lower error rate (all tasks) due to an improved navigation bar and a better terminology. For changes like the new saving feature and the improved content representation a shift in behaviour could be observed. But, these changes also introduced new problems. Even though the saving shortcut was used a lot and users did not have to search for a save button anymore as in the previous study, those that noticed the presence of two disparate saving mechanisms were usually confused by it.

Most surprising was the fact that severe usability issues were discovered related to the diff view, which did not surface during the last study, even though the design had only changed in a few details. The fact that other problems had been eliminated, for example with the content representation, might have contributed to making these problems more obvious.

In general it is to say that testing a more ore less functioning application is very different from testing a prototype. Users are much more demanding and expect the interface to work properly. On the other hand it needs less interference by the moderator and therefore more problems are discovered. Users can freely move inside the application and might do things that the designers of the system did not conceive before. From this, interesting new ideas can evolve.

The next chapter presents the final design of the system, which is based on the findings of this usability study.

Chapter 11

Final Design

The last usability study has shown that the interface still has two major problem areas: the diff view and the saving process. In this chapter solutions for these problems are proposed as well as fixes for the smaller usability issues discovered during the previous study.

11.1 Difference View (Issues #37, #38, #39, #40)

The most severe and impactful usability issues occurred in connection with the diff view during the last study. The main problem was that the representation of content in the diff view was very fragmented and fundamentally different from the way content was represented to the user when editing it (Issues #37 and #38). Furthermore, some users did not understand how to use the breadcrumb navigation in order to jump back and forth between the list of changes and the actual editing view (Issue #39). For some users, it was not even clear which part of the diff showed the old and which part the new state (Issue #40).

In order to eliminate these issues a new design for the diff view was developed (Figure 11.1). The new layout shows two tables next to each other, that are more or less identical to the ones in the editing view, except that they are read-only and cannot be edited. The old state of the content is shown in the table on the left and the changes are marked in red. The new state is shown on the right-hand side and changes are marked in green. The view only displays those rows that have been edited, the remaining ones are hidden by default, but can be shown by clicking on the button at the bottom. Content that has not changed is somewhat transparent to visually lay more focus on the edited content.

Using the same representation as in the editing view makes recognizing the structure of the content a lot easier. Jumping between diff view and editing view only to get a feeling of the structure of the content is no longer necessary. By this,

the most problematic parts of the old design are removed and the mental effort to review changes should be smaller now.

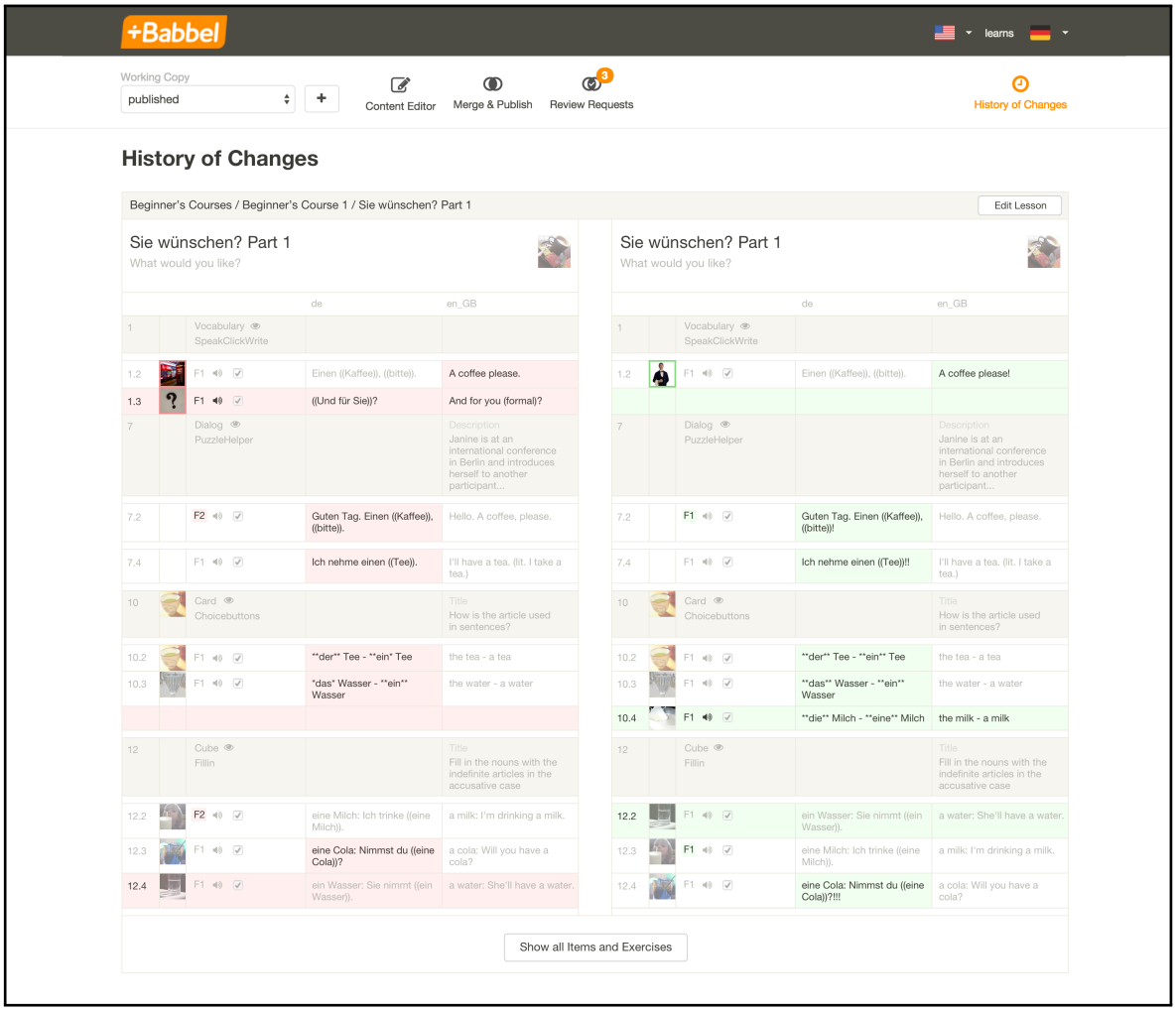

Figure 11.1: New difference view

11.2 Saving Process (Issues #41, #43)

Saving changes was one of the major pitfalls for users during the last usability study. The redesigned saving process addresses the issues #41 and #43 listed in Table 10.8. Most of all, users were confused by the existence of two separate saving features (Issue #41). Even though the saving ”shortcut” at the bottom of the editing view was used frequently, it caused confusion when users discovered there is also a second saving button at the top. For this reason, the two separate saving features were combined into one by trying to maintain the advantages of both. The new saving process (Figure 11.2) allows users to save their changes on the same page as the edits were done while at the same time giving them the opportunity to review

what has changed. The changes are highlighted in green and when hovered reveal the state prior to editing it. Users can decide whether they want to enable this highlighting by toggling a checkbox at the bottom of the view.

Additionally to reducing the steps required to save changes, the new feature also provides better feedback (Issue #43), because the highlighting disappears after the user saved, signalling that there are no current changes that were not saved yet.

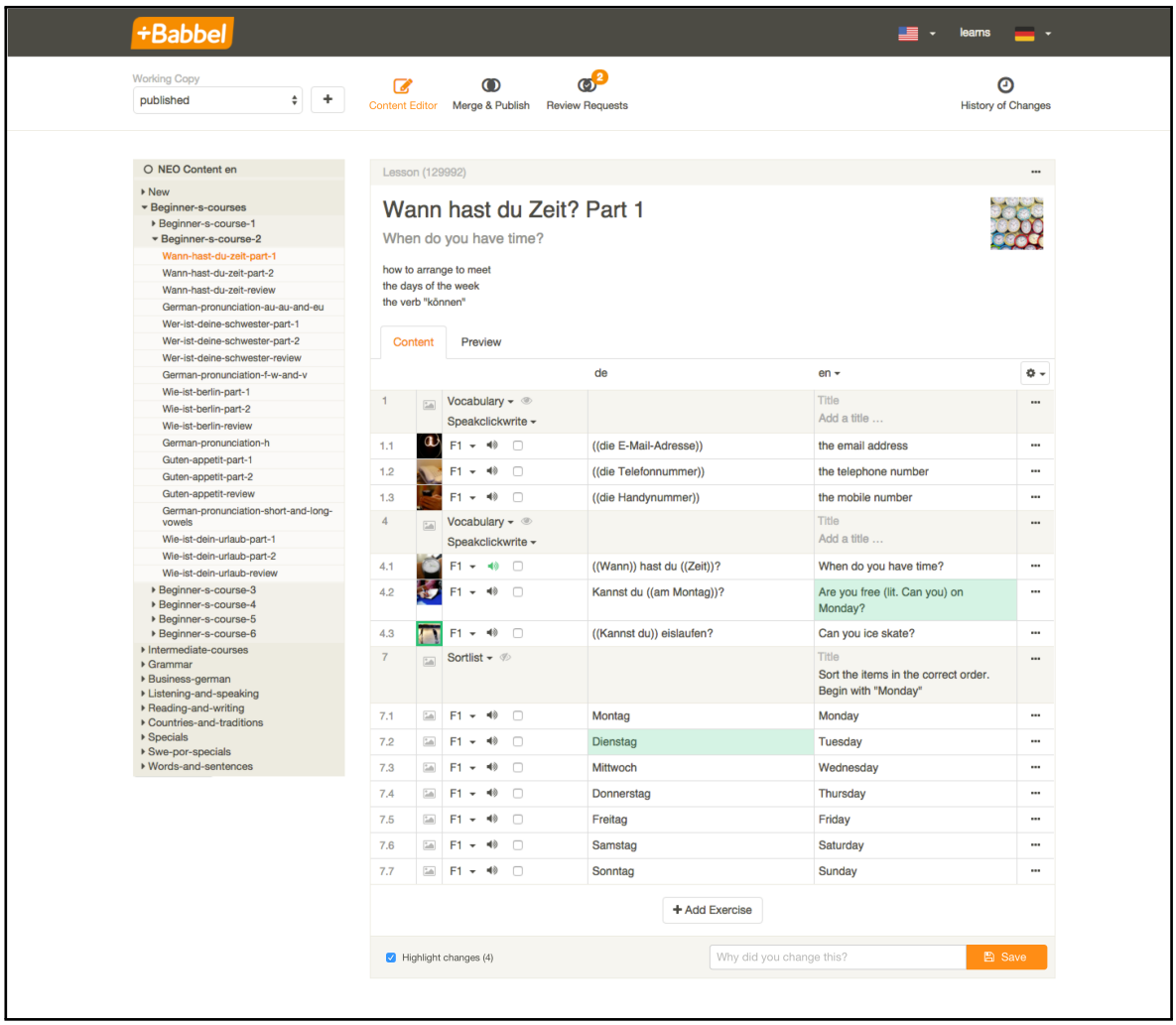

Figure 11.2: Highlighted changes after editing

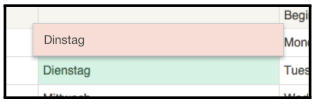

Figure 11.3: Hovering a changed text

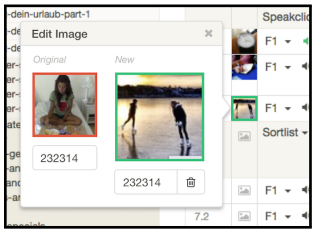

Figure 11.4: Hovering a changed image

11.3 Error Messages (Issues #28, #29, #32)

Another problem that occurred quite frequently during the previous testing sessions were missing or insufficient error messages. Especially during the creation of new working copies, users got stuck, because the interface did not provide appropriate information for recovering from a mistake. The two most common problems where using invalid characters or trying to use a name that was already taken. Now, there are error messages informing users about this (Figure 11.5).

Furthermore, the system is more pro-active now. Even though it is technically possible to create an empty merge request, in reality there is no value in doing that. Therefore, the interface now informs users when they are about to do it (Figure 11.6).

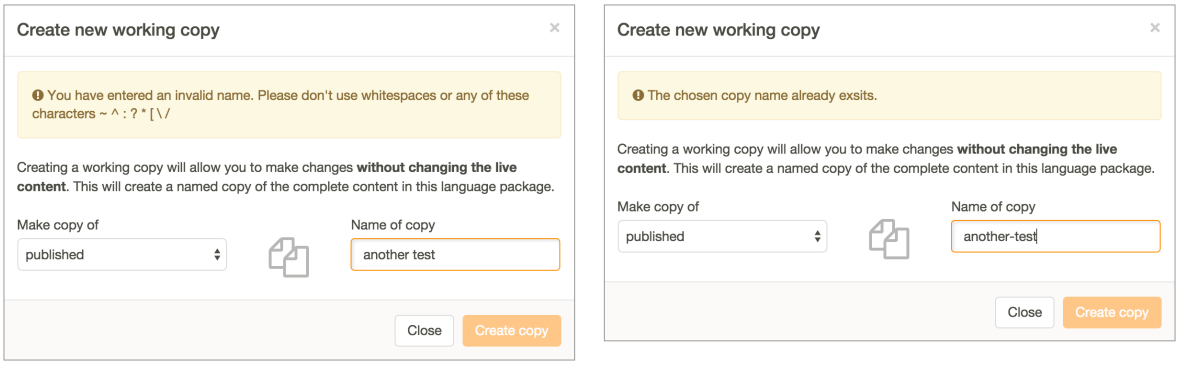

Figure 11.5: Improved error message inside working copy modal

11.4 Navigation Bar

The top-level navigation was changed conceptually, which also had to be reflected in the layout of the navigation bar (Figure 11.7). Instead of treating the different views as ”layovers”, which were closed with a little *x* in the upper right corner, the navigation now works more like a classical tab-style navigation. Therefore, a new icon for the content editor had to be introduced, which is now placed next to the working copy selection. The ”new copy” button was visually grouped with the selection, be-

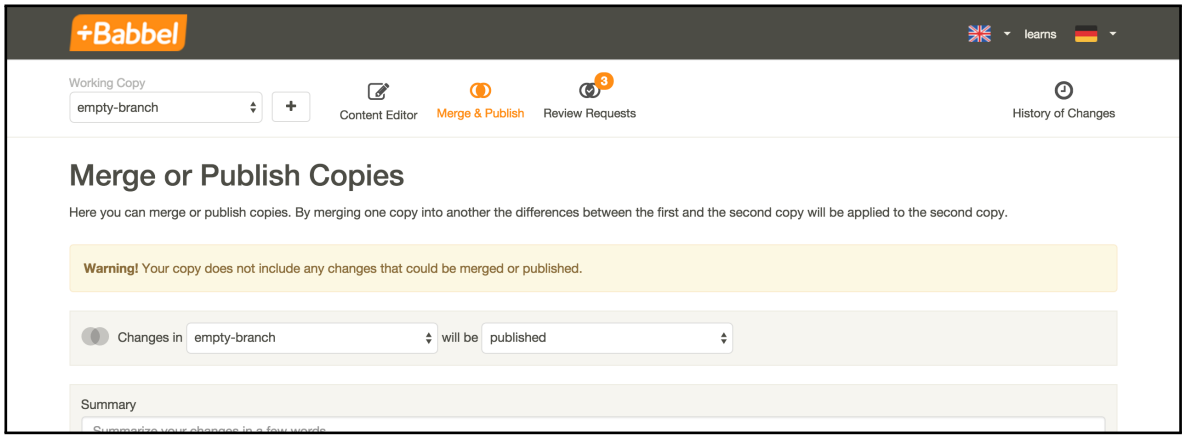

Figure 11.6: A warning informing users that the selected working copy is equal to the target of the merge

cause these two features conceptually belong together and because it triggered a different behaviour (opening a popup) then the remaining buttons, which result in a complete a view change.

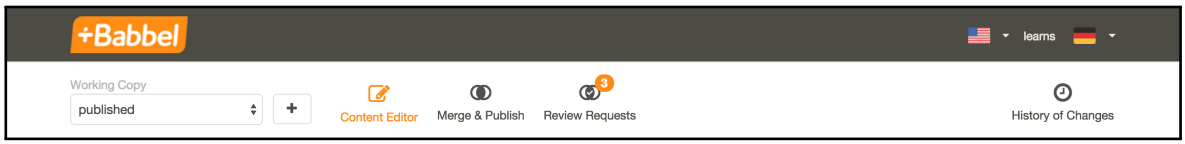

Figure 11.7: Redesigned navigation bar

11.5 Miscellaneous Changes

In order to make the new diff view work, the lesson editing view had to be revised as well. So far, the lesson properties, which include the title, subtitle and a description for every display language, were hidden behind the properties tab. This design was impractical for the new diff view, because everything related to a lesson should be reviewable with just one glance. Therefore, the lesson properties were moved into the header of the editing view and editing was made possible by an inline-editing solution (Figure 11.8) [**?**]. This means that the text turns into an input field when clicked and can be edited.

The new lesson header design described above required another redesign. Now that changing the display language not only affected content that is inside the translation column, but also the text in the lesson header, it seemed odd to select this language in the table header as before. For this reason, the selection was moved to the main navigation bar at the top (Figure 11.9) next to the selection of the learning language.

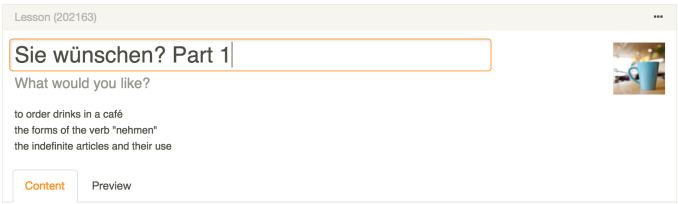

Figure 11.8: New lesson header

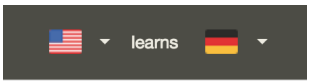

Figure 11.9: Display language selection

Besides the conceptual changes and those that addressed usability issues directly, a few visual improvements were made in order to reduce the visual clutter. The form for merging or publishing copies was simplified and the input field for the second reviewer is now hidden by default (Figure 11.10). Furthermore, the list of requests (Figure 11.11) and the list of changes (Figure 11.12) were simplified by removing the ”show details” button and instead making the whole row clickable.

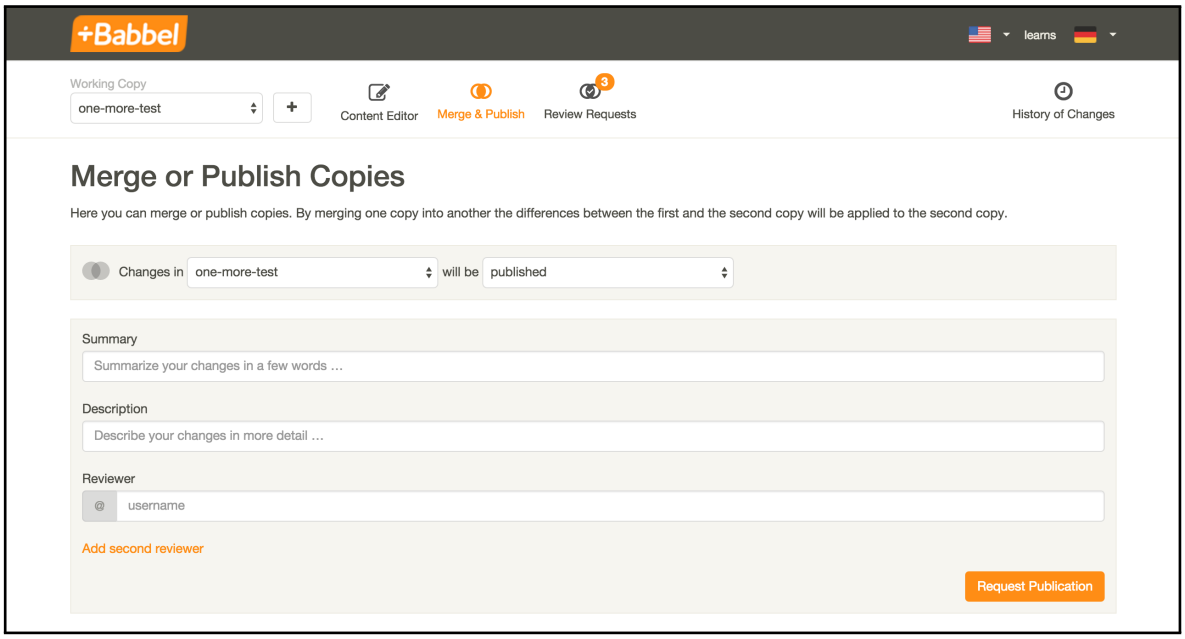

Figure 11.10: Merge or publish copies

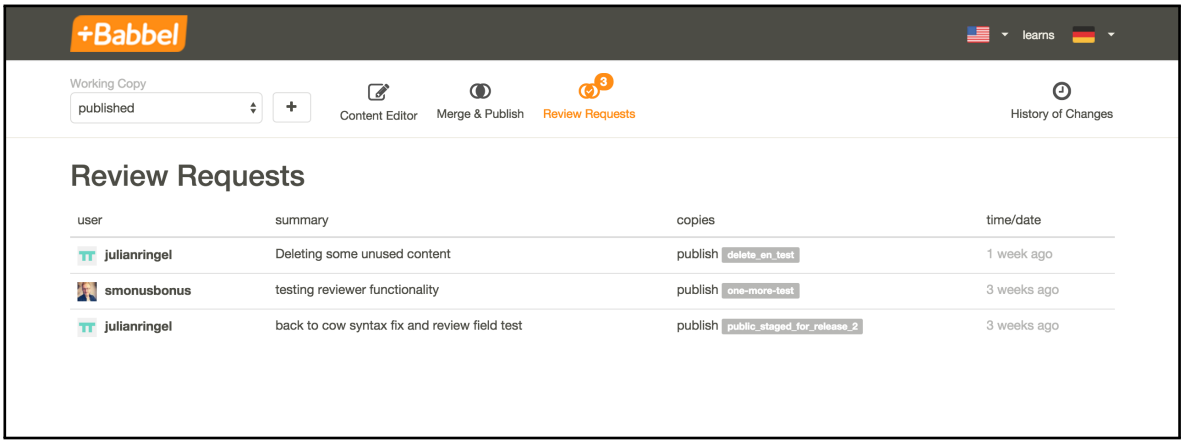

Figure 11.11: List of requests

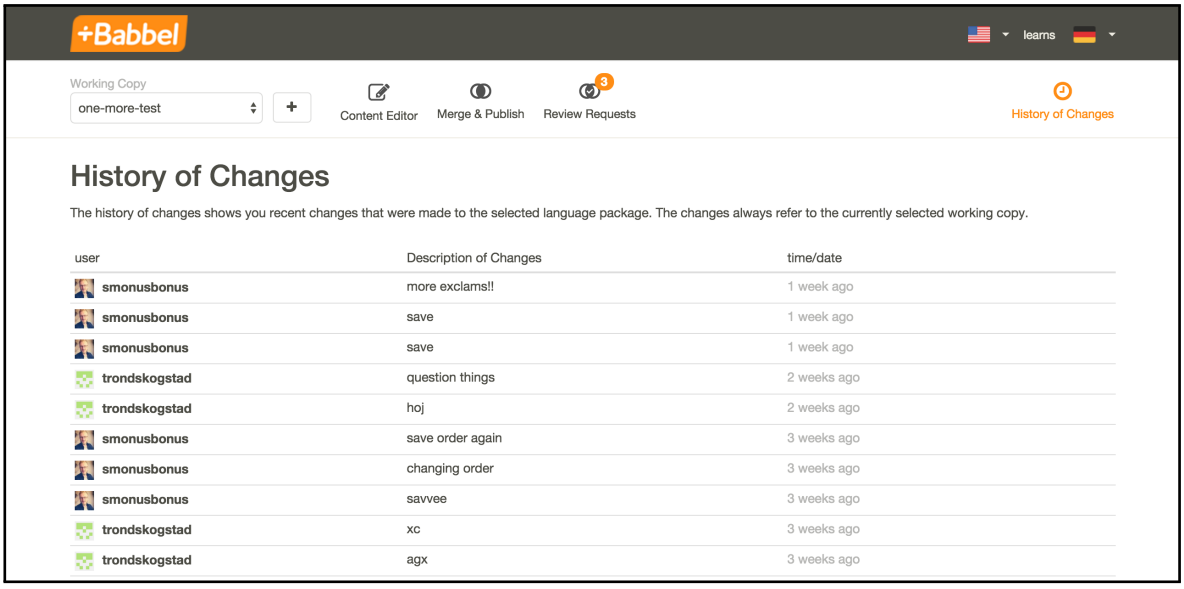

Figure 11.12: History of changes

Chapter 12

Results and Conclusion

The research question asked in a rather general manner, whether authoring tools can benefit from version control. The thesis at hand tried to answer this question by documenting the user-centered design approach used for designing such a system. The design presented above shows that this it is indeed possible to combine version control and content authoring, but is this liaison beneficial to the tool's users? By looking at the sub-questions first it might become easier to answer the main research question later on.

12.1 Sub-question 1: Feature Reduction and Simplification

Most version control systems are quite powerful and cover a wide range of use cases. A result of that is a large number of features that come in many different varieties depending on which flag is used and in which context they are used. For example when using Git on the command line the *checkout* command is used for both switching branches as well as creating new ones (using the -b flag). All of this needs to be learned and remembered, which increases the entry barrier for new users. Therefore, one of the questions raised in the beginning was whether certain features could be simplified or eliminated, while still preserving the core functionality of version control. Since the new authoring tool is based on Git, the question was actually, which features to expose to the users and in which way.

The decision which features to expose was based on the requirements and task analysis as well as the literature review. For example, the review has shown that version control users often regard the staging process as laborious and superfluous [**?**]. Based on this finding a simpler edit and saving workflow was designed that made staging unnecessary. Additionally, the manual tracking of files was eliminated entirely. As soon as a new content package is created inside the tool it is also tracked

by the version control system.

Furthermore, the separation between local and remote repository was eliminated. Users always interact with a remote repository that is stored on Github. Although, this comes with a few disadvantages, such as reduced speed and a single point of failure, overall it simplifies the system and removes hidden dependencies, which Church et al. [**?**] identified as a usability problem. In general, managing repositories is no longer a responsibility of the user, since there is a fixed number of them representing the 14 different learning languages. The user can switch between them, but cannot create new ones or alter them in any way.

The Git reference consists of 53 commands, excluding the low-level features [**?**]. The git command line help still lists 21 ”commonly used” commands. Compared to that the interface presented in this section consists of 7 main features related to the version control capabilities. Of course, comparing command line tools with a GUI is not the best way of answering this research question, but it still hints at the reduced complexity.

12.2 Sub-question 2: Improve Learnability

The second sub-question asked how the learnability of the system could be optimized. According to ISO/IEC 9126 [**?**] learnability is a major contributor to a system's usability. As mentioned before, most version control systems, particularly those controlled via command line, require a lot of upfront knowledge about the different commands and the conceptual model of the system. But, in order to make the authoring tool attractive to a non-technical audience, this complexity should be mostly hidden or exposed only gradually to users. Therefore, a couple of measures were taken to design a system that has a high learnability.

First of all, the initial design was based on insights gained through the literature study and two major studies that investigated the usability of VCSs by Church et al. [**?**] and Jackson and Perez De Rosso [**?**]. How these studies influenced the interface design is described in Chapter 6.

Secondly, the iterative design process and the user tests ensured that hard to comprehend features were discovered and simplified. Only users with no prior experience in version control software were chosen for the tests (Appendix A.2) so that the sessions were a realistic picture of novice users encountering the system for the first time. The outcome of the tests resulted in a couple of design decisions, such as the use of illustrative icons in the navigation bar and warning messages (i.e. for editing live content) that helped users to adopt best practices.

A third and very crucial means of improving the learnability of the system was the focus group that identified terms that were difficult to understand and suggested

alternatives. This helped to design an interface which uses terms that are rooted in everyday language rather than one that is comprised of technical jargon.

Of course it is hard to tell whether the implemented version control mechanism is easier to use than one of the many GUI-clients for Git or other version control systems, since the context of use is quite different (code vs. language lessons). But there are a couple of indicators which show that the design of the system has indeed resulted in a good learnability: First of all, the task completion rates for almost all scenarios are quite high (above 60% during 2nd study). Especially when considering that all participants encountered the system for the first time and basically had to learn it 'on the fly' while performing the tasks, without any introduction or prior tutorial. Additionally, the scores of the post-study questionnaire (Appendix B.2) for items 5 (”It was easy to learn to use this system” - 2.11) and 6 (”I believe I could become productive quickly using this system.” 1.77) show that users are generally happy about the learnability of the system and feel confident to learn it quickly in the future.

12.3 Sub-question 3: Reducing Initial Overhead

The third and last research question asked whether the overhead, initially introduced through a version control system, could be reduced. Most version control systems, especially those used in software development, require an additional effort from the user. The reason for this is that instead of just editing and saving files, the user has to administrate repositories, manage branches, define which files should be tracked and describe his or her edits. All this additional effort results in a delayed gratification when the project has increased in size and more and more users are collaborating. But, as is usually the case with delayed gratification, humans are not very good at ignoring small short-term rewards (simpler and faster editing) in favor of greater long-term rewards (file tracking, reversibility, easier collaboration). The challenge therefore was to design a system that reduced the entry barrier as compared to traditional version control systems and helped users to pick up best practices as fast as possible. Was this achieved and if yes how?

The final design offered a couple of solutions to address these issues. First of all, version control comes ”pre-installed” with the authoring tool. There is no need to setup repositories or define which files should be tracked. This work ”out of the box”. Second, users are free to ignore most version control features. If they choose to do so they can just keep on using the authoring tool as they would use other tools without a version control system. Using branches, entering saving descriptions (commit messages) or creating merge requests is entirely optional. Nevertheless, the system tries to teach best practices to the user. When altering public content

(master branch) the user is reminded that it is advisable to create a new branch first. If edits are saved without a description, a tooltip explains that a description helps colleagues to better understand why something was changed. Through this mechanisms the user is slowly introduced into version control without the need for him or her to go through an extensive tutorial first. The completion rates (Chapter 10, Figure 10.2) as well as the relatively low error rates (Chapter 10, Figure 10.4), especially during the second study, have shown that it was indeed possible for most users to start using the system without any prior training.

On the other hand there is also room for improvement. The post-study questionnaire had the weakest scores for information quality (3.31), which measures how well the system is documented and how helpful error messages are to the user. This was partially due to the incomplete prototype that was missing functionality, but nonetheless shows that some users felt there was too little help for using such a system. For those occasions a context-ware help feature could be useful.

Summing up one can say that the overhead of using version control was certainly reduced, but of course not completely eliminated. A system that is more powerful and offers more features will always be harder to learn and use than a simpler system.

12.4 Conclusion

Now that we have looked at the sub-questions, let us get back to the main research question. Can authoring tools for e-learning benefit from version control? Ultimately answering this question is difficult, since users have not started using the system in a real-world environment yet. Time will tell whether content authoring really benefits from an integrated version control system. But, what has become clear, is that it is in fact possible to design a version control system that is easier to use and learn than most traditional VCSs and that integrates well with the content authoring process. The system enables users to manage and control the content creation process in ways that were simply not possible before. The authoring tool's capabilities were greatly improved while at the same time increasing the complexity of the tool only moderately. It can be concluded, that version control is indeed a valuable addition to the authoring process of e-learning content and its possible application should be investigated further. The next chapter looks at how this could be done.

Chapter 13

Future Work

As mentioned in the previous chapter, the research mostly focused on determining how an easy-to-use interface for version control could be integrated into a content authoring tool. The long-term implications of utilizing such a system were not studied. A possible follow-up study could investigate whether the long-term benefits of the system really outweigh the short-term increase in labour. Furthermore, it should be confirmed which features of the system are actually used and in which way. For this purpose a Web Analytics tool could be used, which would provide more quantitative data and enable a more detailed insight into the usage patterns of the system.

A second area of future research could explore whether the findings of this project also apply to a wider range of tools than just content authoring for e-learning. Given the increasing amount of content created and published online, especially by amateurs, it is conceivable that general purpose systems, such as Wordpress [**?**] or Drupal [**?**] could benefit from this as well. In order to find out, more research would be needed in regards to the ability of occasional users to learn and remember such a system.

Summing up, one could say, that the expansion of version control outside of software development is a recent development, which has slowly shifted the focus to usability and simpler interfaces. Probably, there are more use cases for these systems than the current research landscape suggests. If the HCI community continues to prove, that version control is also viable outside the technical domain, we might increasingly see it appear inside consumer products as well.

Appendix A

Background of First User Study

A.1 Scenarios

The following task scenarios were given to participants of the first iteration user studies. The link leads to the original prototype that was used.

A.1.1 Scenario 1

- End goal: Let colleague review changes
- Estimated time: 8 mins
- Link: https://marvelapp.com/c89590

A colleague (Firstname Lastname) has sent you a link to a lesson. She informed you that the second item in the vocabulary (write) exercise has the wrong speaker role (should be F1) and also should be added to the review manager. She asked you to correct the error and afterwards allow her to review your changes. Make sure youre not editing the live content.

A.1.2 Scenario 2

- End goal: Publishing changed content
- Estimated time: 4 mins
- Link: https://marvelapp.com/1011fjj

A colleague (Firstname Lastname) has changed a lesson within the German language package. She asked you to review the changes. Look at the changes she made and if you dont find any errors make them available to the Babbel end-user.

A.1.3 Scenario 3

- End goal: Eliminating error and save
- Estimated time: 6 mins
- Link: https://marvelapp.com/102f4b4

A translator (user name: rcarlos) is currently working on a localization of German lessons (to Portuguese). Recently a translated exercise was published, but according to customer service users have complained that there is a translation missing within this exercise. Find the exercise which has been translated and locate the item with the missing translation. Add the translation and save your changes.

A.2 Pre-session Questionnaire

User	**Familiar with version control?**	**Used it?**	**Experience with current tool**	**Used other content management or authoring systems?**
1	No	No	1 - 3 years	
2	No	No	1 - 3 years	
3	No	No	3 months - 1 year	WordPress
4	No	No	more than 3 years	
5	No	No	1 - 3 years	

Table A.1: Pre-session questionnaire

Appendix B

Background of Second User Study

B.1 Participant Quotes

B.1.1 Working Copies

- ”Am I in the copy right now?”
- ”I have created a copy but I’m not sure whether I have selected it”

B.1.2 Saving Process

- ”Why do I have to answer why?”
- ”No unsaved changes?” (related to blank slate)

B.1.3 Merge Requests

- ”I think I didn’t read the text properly because I dont really get what merge and publish means”
- ”The merge and publish is confusing me”
- ”It would be easier if you had it straight after the save changes”
- ”I don’t see what is addressed to me” (list of requests)

B.1.4 Navigation

- ”If I wanna go back can I close this?”

B.2 Average Scores for 16 PSSUQ Items

#	Item	Rating
1	Overall, I am satisfied with how easy it is to use this system.	2.55
2	It was simple to use this system.	3.00
3	I was able to complete the tasks and scenarios quickly using this system.	3.44
4	I felt comfortable using this system.	2.22
5	It was easy to learn to use this system.	2.11
6	I believe I could become productive quickly using this system.	1.77
7	The system gave error messages that clearly told me how to fix problems.	3.77
8	Whenever I made a mistake using the system, I could recover easily and quickly.	3.11
9	The information (such as online help, on-screen messages and other documentation) provided with this system was clear.	3.33
10	It was easy to find the information I needed.	3.55
11	The information was effective in helping me complete the tasks and scenarios.	3.11
12	The organization of information on the system screens was clear.	3.00
13	The interface of this system was pleasant.	1.66
14	I liked using the interface of this system.	1.55
15	This system has all the functions and capabilities I expect it to have.	3.00
16	Overall, I am satisfied with this system.	2.11

Table B.1: All 16 items of the PSSUQ