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

Simon Kreiser
M.Sc. Thesis
March 2016

Supervisors:
Dr. Mariët Theune
Prof. Dr. Dirk Heylen
Julian Ringel

Human Media Interaction 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 is an account of a research project concerned with the application of version control, a proven paradigm used in software development, to the authoring process of 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 a non-technical audience.

The interface design was based on a user analysis as well as insights gained from related research on the usability of version control. During the course of the project the interface went through 3 different design stages. Each iteration of the interface was evaluated by actual users during moderated usability studies. The results informed the evolution of the system and ultimately led to the final design, presented at the end of this thesis.

The research has shown that version control can be vastly simplified and thus benefit a non-technical user-base. The final usability study has shown that the system is intuitive enough to be sufficiently grasped during first-time-usage without prior training (82% task completion rate on average). Furthermore, users expressed great confidence to "become productive quickly using the system" (1.77 on a scale from 1 to 7, where 1 is strong agreement). Additional improvements could be made regarding error messages and documentation (Information Quality: 3.31). In general, the user acceptance was high (Overall satisfaction: 2.11), which indicates a positive attitude towards the novel version control features and a likely adoption of the system in the future.

The project’s intention is to kick off a discussion on the viability of version control outside the realm of programming and make advanced features, such as merging and diffing, available to a wider audience.
4.1.4 Translators and Proofreaders ............................................. 20
4.1.5 Quality Assurance Professionals ...................................... 20
4.2 Conclusion ........................................................................... 21

5 First Design Iteration .............................................................. 23
  5.1 Comparison to Traditional VCSs ....................................... 24
  5.2 Prototype ........................................................................... 25
    5.2.1 Navigation ..................................................................... 25
    5.2.2 Branches ....................................................................... 25
    5.2.3 Saving Process and Diff View ....................................... 25
    5.2.4 Content Editing ............................................................... 26
    5.2.5 Pull Request ................................................................... 27

6 First Usability Study ................................................................. 29
  6.1 Study Design ................................................................. 29
    6.1.1 Participants ................................................................. 30
    6.1.2 Metrics ......................................................................... 30
    6.1.3 Scenarios ...................................................................... 31
    6.1.4 Sessions ...................................................................... 32
  6.2 Findings ............................................................................ 32
    6.2.1 Metrics ......................................................................... 32
    6.2.2 Discovered Usability Issues ........................................ 33
    6.2.3 Conclusion ................................................................... 38

7 Focus Group ........................................................................... 41
  7.1 Procedure ......................................................................... 41
  7.2 Most Problematic Terms .................................................. 42
  7.3 First Phase .......................................................................... 42
  7.4 Second Phase .................................................................... 42
  7.5 Conclusion ........................................................................ 43

8 Second Design Iteration .......................................................... 45
  8.1 Navigation and Terminology ............................................. 45
  8.2 Branches/Working Copies ................................................. 46
  8.3 Improved Diff View ............................................................ 47
  8.4 Saving Process ................................................................... 47
  8.5 New Content Representation ............................................ 48

9 Second Usability Study ............................................................ 51
  9.1 Study Design ..................................................................... 51
    9.1.1 Participants .................................................................. 52
9.1.2 Metrics ................................................. 53
9.1.3 Scenarios ............................................. 53
9.1.4 Post-Study Questionnaire ....................... 53
9.1.5 Sessions .............................................. 54

9.2 Findings ................................................. 54
9.2.1 Metrics ................................................ 54
9.2.2 Impact of Design Changes ....................... 56
9.2.3 Discovered Usability Issues ..................... 58
9.2.4 Post-study Questionnaire ....................... 61
9.2.5 Conclusion .......................................... 62

10 Final Design .............................................. 65
10.1 Difference View ...................................... 65
10.2 Saving Process ....................................... 66
10.3 Error Messages ....................................... 68
10.4 Navigation Bar ........................................ 69
10.5 Miscellaneous Changes ......................... 69

11 Results and Conclusion .............................. 71
11.1 Sub-question 1: Feature Reduction and Simplification ........................................... 71
11.2 Sub-question 2: Improve Learnability ........... 72
11.3 Sub-question 3: Reducing Initial Overhead ... 73
11.4 Conclusion ............................................. 74

12 Future Work .............................................. 75

Bibliography ................................................. 77

Appendices

A Background of First User Study .................... 81
A.1 Pre-session Questionnaire ......................... 81

B Background of Second User Study .................. 83
B.1 Pre-session Questionnaire ......................... 83
B.2 Average Scores for 16 PSSUQ Items .............. 84
List of acronyms

CAT content authoring tool
CLI command line interface
DL display language
GUI graphical user interface
LL learn language
MOOCs massive open online courses
SD standard deviation
QA quality assurance
VCS version control system
Preface

The following thesis is an abridged public version of the original. The reason for this is that Babbel, the company at which the research was carried out, has requested to make some facts described in the thesis confidential and therefore not available to the public. A more extensive and complete version has been made available to the official thesis committee at the University of Twente.
Chapter 1

Introduction

The emergence of massive open online courses (MOOCs) has spawned a sharp growth in the e-learning industry in recent years [1]. In 2012, as much as one third of all enrollments in the US were registrations for online courses [2]. In addition, 77% of US companies offer advanced professional training based on e-learning to their employees [3]. 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 [4], [5].

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 continuously. But, with the increasing complexity of learning content and the rising numbers of contributors, these systems also quickly 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 services 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 a company offering interactive language courses online. It 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 almost one year at the Babbel headquarters in Berlin. The author was part of a four-member team and responsible for user research and interaction design. The team was created as part of an effort to completely rethink the way language lessons were authored and maintained at Babbel. One particular aspect of this process, the collaboration between authors, translators and quality assurance professionals, was the main research theme of this thesis, which is described in more detail below.

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 10-15 individuals who have participated in the production of the course, often even more. The authoring tool currently used at Babbel does not facilitate this workflow very well and has a couple of inherent problems:

1http://www.babbel.com/prices
**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 [6].

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 [7], [8]. In the context of this thesis it was investigated 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, as defined in the previous research proposal [9], reads as follows:

*How could version control support the authoring process of 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 a rather practical approach was adopted, which involved designing and testing an interface for a new content authoring tool. In general, the research project consisted of two major phases: the analysis phase and the design and prototyping phase. During the analysis phase a literature review was conducted as well as two additional analyses performed. The literature review examined 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 were mainly centered around the future users of the system. Different analyses were performed, which ensured that user needs and goals were properly addressed during the subsequent design phase.

Based on the insights gained during the first phase a prototype was designed that incorporated the most important version control features. The prototype was semi-interactive and tested with 5 users during a task-based usability study. This initial study was followed by a focus group meeting during which users discussed the problems they encountered. The focus group put special emphasis on the version control terminology since finding alternatives to the technical and abstract terms used in most version control systems was a key factor for improving the usability of the interface.

The results of the focus group and the first usability study were then used to design an improved version of the interface, which was tested again during a second usability study. The findings informed the final design of the authoring tool and additionally helped to answer the research question described above.

1.5 Thesis Structure

The next chapter (Chapter 2) will introduce the reader to version control software in more detail. Afterwards, Chapter 3 introduces the reader to relevant research in adjacent fields. This is followed by a more in-depth description of the user groups and the tasks that constitute the content authoring process in Chapter 4.

In the second half of the thesis the design process as well as the usability tests are covered. Chapter 6 is about the first prototype and the first round of user tests. This is followed by a description of the focus group (Chapter 7) and the second design iteration (Chapter 9).

Readers who are mostly interested in the outcome of the design phase and the research results can refer to Chapters 10 and 11. In the last chapter (Chapter 12) recommendations and suggestions for future research are made.
Chapter 2

Version Control

The previous chapter posed the question whether version control can simplify the content authoring process of e-learning content. Before continuing, this chapter introduces the reader to the concept of version control in more detail. A solid understanding of version control, its functionality and the context it is used in, will be necessary in order to follow the argument of this thesis during the subsequent chapters.

First, version control is defined and a summary of its most common features is provided. Afterwards, the history of version control is outlined and the most popular version control software is shortly introduced. The last section focuses on Git and Github, as it is the most widely adopted VCS as of today, and also the one that was used throughout this project.

2.1 Definition

Perhaps, version control is summarized best with the words of Hunt and Thomas: "A source code control system [is] a giant UNDO key–a project-wide time machine" [10]. Version control, as it is referred to throughout this thesis, goes by several other names as well, such as source code control, revision control or source code management. Nevertheless, these different terms all refer to more or less the same thing: a system that keeps track of changes made to a file or set of files in a particular location [11]. A new version can be created by committing (explained below) a changed file 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 retrieve a particular version of a file from the past. Furthermore, version control simplifies collaboration by offering automated merging of distinct changes made to the same file. For these reasons, and because it works well with text-based documents and source code in particular, version con-
Figure 2.1: The lifecycle of a file, illustration adapted from Chacon [14, p. 45].

version control is used heavily within the software development community [12].

2.2 Taxonomy and Features

The features described below exist in most version control systems. Nevertheless, no system is like the other and the naming can vary somewhat from software to software. For this reason, Git, the most widely adopted version control software [12] 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 [13]–[15].

- **Repository**: a database that holds meta data about all the files tracked by the version control system. This includes a historical record of the changes made to a file over time. A new repository can be created by issuing the `init` command inside a directory. This means that from that moment on all files in the directory and its sub-directories can be tracked by the version control system. Repositories can either be local (situated on the client machine) or remote (on the server). In a distributed version control system, such as Git, local repositories hold the same (complete) information as the remote repository.

- **Tracking files**: when a new repository is created it is empty. It has to be specifically told which files should be tracked. This can be done by using the `add` and `commit` commands.

- **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. Changed files can be added to the staging area using the `add` command.
2.3. WORKFLOW

- **Commit**: When a file was changed and staged, it can be committed to the repository in order to create a new version of the file. Usually a commit message is added which describes why the file was changed. This makes retrieving past versions at a later point of time simpler.

- **File states**: Once a file is added to the version control system it is in one of three different states: unmodified, modified and staged. If certain files should not be tracked, they can also be ignored permanently. The different states a file can be in during its life cycle are visualized in Figure 2.1.

- **Diff**: Before staging a changed file a user can check how the new version differs from the previous one by using the `diff` command. This tool can also be used to compare historical file versions to one another.

- **Branching**: a branch is a named copy of the latest file versions as represented in the repository. A branch allows developers to build features without influencing the stable code of the main branch. 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 two versions were changed in the same place.

- **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).

2.3 Workflow

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.
2.4 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 [17]. According to Sink [18] 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 [12], 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.

2.5 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 [11], [14]. 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"\(^1\). 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) \(^1\).\(^2\)

![Figure 2.3: Pull request view on Github](image)

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\(^2\). 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.

### 2.6 Conclusion

Version control software is used to keep track of changing code-bases and to simplify the collaboration between 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. Open source systems, such as Git and Subversion, have further accelerated this development. Furthermore, social

---

\(^1\)https://www.youtube.com/watch?v=4XpnKHJaok8  
\(^2\)http://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/
coding platforms, such as Github, have become more and more popular during recent years. Nowadays, almost any serious programmer uses version control when developing software. Despite its popularity among developers, version control has not made the leap into many other areas yet. The next chapter will illuminate some of the reasons for this.
Chapter 3

Literature Review

The previous chapter outlined the most common features of version control and introduced Github and Git, the most widely adopted version control software. The following chapter presents research that is concerned with version control and its different applications. The review aimed at identifying work, which is directly related to version control systems and their usability. The search for literature was based on three main criteria: (1) work which produced specific results that could be utilized for this project, (2) work that was concerned with the usability or interfaces of VCSs and (3) work that investigated the use of version control outside of the domain of programming.

The reviewed literature was key to create a foundation for the research described in this thesis. The first design iteration has drawn heavily on the insights gained through the review, which is described in more detail in Section 5.1. Furthermore, this review provides the necessary context in which the results of this thesis should be seen and enables the reader to assess the contribution of this work to the current state of research.

3.1 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 [19]. 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 [7], [8].

Church, Söderberg and Elango have conducted an empirical study on the usage of command line-based version control inside two large IT companies (Google & Autodesk) [7]. 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[^1] 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 [8], 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" [8, p. 38]. Git's concepts were scrutinized based on three main criteria that were defined by Frederick Brooks in the 1970s [20] 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

[^1]: https://mac.github.com/
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\(^2\) 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. \(^7\) acted so risk-averse when using Git. The inconsistencies throughout the system might have prevented them from forming an accurate mental model.

### 3.2 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\(^3\), whose main goal it is

\(^2\)http://gitless.com
\(^3\)https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools
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 [21]. 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 [22]–[24], 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 [25]. 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.
3.3. Version Control Outside of Software Development

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.

3.3 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 [26], 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 [27]. 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 [27].

Version control is especially hard to find in consumer products. The most prominent examples might be the track changes feature of Microsoft Word [28] 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

---

4https://en.wikipedia.org/wiki/Revision_control
5https://support.google.com/docs/answer/190843
version control to graphics\textsuperscript{5,7,8}. Most of them have used a rather simplistic approach, 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 \cite{29}. The researchers implemented a non-linear revision control system for GIMP, an open source graphical editing tool\textsuperscript{9}. 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 3.1).

![GIMP Revision Control](https://example.com)

**Figure 3.1:** A screenshot from the GIMP revision control system

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

\textsuperscript{5}\url{https://blogs.adobe.com/jnack/2009/03/new_version_control_system_for_photoshop.html}

\textsuperscript{7}\url{http://adobe.wikia.com/wiki/Adobe_Version_Cue}

\textsuperscript{8}\url{http://www.alienbrain.com/}

\textsuperscript{9}\url{http://www.gimp.org/}
3.4 Conclusion

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.

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. Section 5.1 describes in more detail how this review influenced the design of the first prototype. Chapter 7 outlines how some of the inconsistencies were addressed.
Chapter 4

User Analysis

The last chapter has shown that version control comes with several problems. Most systems are hard to use and therefore discourage even experienced users. This might be one of the reasons, why there are only few examples of version control applied outside of software development.

The following chapter offers a description of the different user groups and their respective responsibilities. This might help the reader to visualize the interplay between the different individuals involved in the content creation process and the tasks that have to be performed in order to produce a new language course. This analysis plays an important part in ensuring the new system’s usefulness to its user base and the ultimate goal of creating an easy to use version control system.

4.1 User Groups

The so called didactics department consists of about 30 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 different teams, based on each person’s language expertise. Right now there are 7 different teams that consist of 2-5 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.

4.1.1 Project Managers

Project managers guide the content creation process and manage the various professionals involved. 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.
4.1.2 Content Editors

Content editors have different responsibilities, ranging from authorship over translation to reviewing content. Editors ensure that new content meets a high quality benchmark and fulfills its didactic purpose. Their goal is to create content that allows Babbel’s end-users to successfully acquire new language skills. When involved in the localization of existing content, Editors make sure that specific language characteristics, such as idioms, are considered when translating from one language into another.

4.1.3 Authors

Authors conceptualize and create new language lessons and courses. Based on guidelines and specifications provided by project managers they decide how lessons look 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 author 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 for each lesson, which is usually English or German. The job of translating (localizing) a lesson into a new language is always performed by separate people, who are native speakers of the respective language.

4.1.4 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 the work to the proofreader. The proofreader corrects spelling errors and provides 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.

4.1.5 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 in order to emulate how the end-user would experience the content. Once an error is found, it is either corrected right away (simple typos or spelling
mistakes) or noted in a spreadsheet and send to the responsible editor or project manager. The responsible person then uses the content authoring tool to locate and fix the error.

4.2 Conclusion

The analysis made a few things obvious. First of all, collaborating and communicating with colleagues is a reoccurring theme throughout the content authoring process. For example, when a lesson is localized, there are at least three different individuals working together (project manager, translator, proofreader), who also need to communicate in some way. Right now, email and spreadsheets 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 assurance that happen throughout the process and provide a platform for communication between freelancers and full-time employees.
Chapter 5

First Design Iteration

The analysis presented in the previous chapter has shown that the content authoring process at Babbel is a highly collaborative process involving a lot of tasks. In order to simplify the collaboration and potentially speed up the content creation, the integration of version control features has been suggested earlier.

This chapter describes an early prototype, which exposed several version control features, which were deemed potentially useful for the content authoring process. Since the research presented throughout this thesis was mainly concerned with the applicability of version control to the content creation process, the prototype is an important first step towards answering this research question.

The chapter starts by highlighting the main differences between the integrated version control features and those of a more traditional VCS. Afterwards, the most important views comprising the prototype are shown and shortly explained. Figure 5.1 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 5.1: Overview of main sections of the application](image)
5.1 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 content 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 [7] work has shown that hidden dependencies between these two repositories 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 3 this feature is sometimes problematic and often inconvenient, because every change has to be staged before it can be saved (committed).

- The difference view is presented by default before the user saves his or her changes. This ensures that the user knows what will be saved 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 the things that have 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 user analysis has shown that reviewing new content, which 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.
5.2 Prototype

The overall interface was strongly influenced by existing code hosting platforms, such as Github\(^1\), Gitlab\(^2\) and Bitbucket\(^3\) as well as several Git GUIs\(^4\). The system can be regarded as a crossbreed between version control and content authoring tool. Below, the main views of the prototype are shown.

5.2.1 Navigation

The upper-most navigation bar (Figure 5.2) offers a quick access to the most important version control features as well as the language package and the current branch. Next to the branches dropdown a little plus-icon signifies how a new branch can be created. Furthermore, some buttons provide additional information, such as indicating whether changes can be saved or open pull requests exist.

![Figure 5.2: The top-bar navigation exposing version control features](image)

5.2.2 Branches

Branching is a fundamental concept of most version control systems and allows working in an isolated state. A lot of time went into the consideration of whether to include this feature or not. Finally, it was decided, to make this feature visible to users, which has the advantage of them being able to share branches with colleagues or point to a particular state of content that is still being created.

5.2.3 Saving Process and Diff View

As mentioned before, when a user has edited some content, which shall be saved, a difference view is presented. This allows the user to review her changes before committing to a change. As is common with most version control systems, a commit message needs to be entered before saving. This allows collaborators to comprehend why a particular piece of content was edited.

\(^1\)https://github.com/about
\(^2\)https://about.gitlab.com/
\(^3\)https://bitbucket.org/
\(^4\)http://git-scm.com/downloads/guis
5.2.4 Content Editing

The content in the editing view can be navigated by expanding a tree structure (Figure 5.5, left-hand side), which also reflects the inherent composition of the content. Furthermore, when a lesson is selected, its exercises are listed and the content can be exposed through so-called accordions. Each exercise consists of several items which themselves consist of text, images and sounds.
5.2.5 Pull Request

Pull requests are a formalized way of requesting a "merge" of two branches. A typical version control flow consists of creating a new branch, editing some content, saving it and finally creating a pull request (Figure 5.6). A pull request allows collaborators to review changes before merging them with the main branch and thus ensures the quality of the content.
Figure 5.6: Form allowing the creation of new pull requests
Chapter 6

First Usability Study

The prototype described in the previous chapter presents a first attempt at offering version control features inside a content authoring tool. Even though most of these features have been simplified, the system is still very feature-rich and might present a hurdle to first-time users.

The usability study presented in this chapter aimed at identifying possible weak spots and usability issues of the interface. The goal was to determine whether the system could be used without prior training and if the content authoring workflow could ultimately benefit from version control.

Below, the design of the study is described in more detail. In total, five users participated, each of which went through three different scenarios. The scenarios were designed in a way that users had to utilize the most important features of the interface in order to reach their goals. During the sessions participants were encouraged to speak out loud and interference by the moderator was kept to a minimum. In addition to the observations, several metrics were recorded, which helped interpret the results.

6.1 Study Design

The purpose of this study was to evaluate whether users would be able to utilize the different version control features as part of the content authoring workflow. Additionally, the aim was to expose severe usability issues that could prevent users from getting their jobs done. Therefore, 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 behavior 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 [30], [31]. These kind of studies can yield more honest results because the method involves less in-
terference 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 more or less uniform [32].

In addition to observing users during the scenarios, a few quantitative metrics were measured, such as task completion or error rate. Because of the small sample size, these metrics are unlikely to deliver statistically significant results, which is why they should not be relied upon alone. Nevertheless, they can serve as an indicator of potential problem areas, which could then be investigated further (e.g. in the following focus group).

The study was conducted using 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. It should be noted, that the prototype already went through several iterations before being tested. Because of its rather low fidelity, tweaking or modifying the prototype, e.g. after design feedback rounds, was relatively easy.

### 6.1.1 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 had also worked as freelancers before (Table 6.1). So, even though most of them worked as content editors at the time, they also had experience in translation, proofreading and quality assurance. This was important in order to cover all aspects of the content authoring process and test the tool regarding the different requirements of these groups.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Language Team</th>
<th>Job Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>French</td>
<td>Content Editor, QA</td>
</tr>
<tr>
<td>2</td>
<td>Portuguese</td>
<td>Content Editor</td>
</tr>
<tr>
<td>3</td>
<td>Portuguese</td>
<td>Content Editor, Translator</td>
</tr>
<tr>
<td>4</td>
<td>Spanish, Portuguese</td>
<td>Project Manager</td>
</tr>
<tr>
<td>5</td>
<td>Turkish, German</td>
<td>Project Manager</td>
</tr>
</tbody>
</table>

**Table 6.1:** List of Participants

### 6.1.2 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.
6.1. Study Design

- **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.

6.1.3 Scenarios

As suggested by Nielsen Norman Group the tasks given to the participants were situated within real-world scenarios [33]. 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 you are 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 do not 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.
6.1.4 Sessions

In the beginning of each session the participant received a short introduction to the reasons behind the study and the procedure of the session. In order to make him or her feel more comfortable, it was highlighted that the interface was being tested and not the participant. Each session was structured into three different parts: First, a short questionnaire had to be answered, then the participant had to go through the three different scenarios (as described above) and lastly, he or she was asked some follow-up questions and encouraged to provide feedback on the interface. While performing these tasks, participants were asked to think aloud, especially when running into problems. During each session the author was present as test moderator as well as one or two observers who took notes. The sessions usually took no longer than 30 to 45 minutes.

6.2 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.

6.2.1 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.

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 different durations (in seconds) that were needed
6.2. Findings

<table>
<thead>
<tr>
<th>User</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>Success</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Completion Rate</td>
<td>40%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6.2: Task completion rate

to finish a certain scenario. The mean for Scenario 1 has the highest standard deviation (SD), 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.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>505</td>
<td>270</td>
<td>313</td>
<td>275</td>
<td>519</td>
<td>376</td>
<td>133.1</td>
</tr>
<tr>
<td>2</td>
<td>81</td>
<td>226</td>
<td>73</td>
<td>170</td>
<td>90</td>
<td>128</td>
<td>67.2</td>
</tr>
<tr>
<td>3</td>
<td>301</td>
<td>375</td>
<td>418</td>
<td>507</td>
<td>399</td>
<td>400</td>
<td>85.2</td>
</tr>
</tbody>
</table>

Table 6.3: 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.

6.2.2 Discovered Usability Issues

The following section describes the usability issues that were discovered by observing participants during the sessions. In total, 26 different problems were identified. 7 of them were of the highest severity, preventing 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 combination of severity and frequency was used to derive an impact score. The severity of each problem was determined based on four different definitions, based on Sauro and Lewis [34, p.14]:

- 10 points: Severe usability issues preventing users from completing the task scenarios
- 5 points: Usability issues causing a significant delay or frustration
- 3 points: Minor usability issues which are merely a nuisance
- 1 point: Improvements suggested by participants

To calculate the 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. Issues with a high impact are shortly explained.

### Branches

As can be seen in Table [6.5] 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 [6.1] 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.

<table>
<thead>
<tr>
<th>#</th>
<th>Usability issue</th>
<th>Severity</th>
<th>Frequency</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create new branch button was not found</td>
<td>10</td>
<td>40%</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Warning about editing content inside the master branch was ignored</td>
<td>10</td>
<td>40%</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
<td>5</td>
<td>60%</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>The naming for the master branch was confused with the concept of a master language.</td>
<td>5</td>
<td>40%</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>After creating branch it is not clear that system switched to new branch already</td>
<td>5</td>
<td>20%</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.5: Usability problems related to branches

![Graph showing task completion time vs branch creation button discovery](image)

Figure 6.1: Total task completion time compared to discovery of branch creation button
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 6.6, 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.

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

Table 6.6: Usability issues related to Pull Requests

Diff

During Scenario 3, a high proportion of users had problems comprehending the list of changes presented to them (Table 6.7, 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 6.2).

Figure 6.2: Section of difference view that displays an added translation
## Usability issues related to the diff view

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

Table 6.7: Usability issues related to the diff view

### Saving Process

The most problematic issue related to saving was the non-existing save button at the bottom of the view (Table 6.8, 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 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.

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

Table 6.8: Usability issues related to the saving process
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 6.9, 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).

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

Table 6.9: Usability issues related to the main editing view

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.

6.2.3 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
### Table 6.10: Miscellaneous usability issues

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

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 8 presents how the discovered issues were resolved and translated into a new interface design.
Chapter 7

Focus Group

As the previous usability study has shown, the naming of features was a major usability issue. Participants have been exposed to entirely new concepts. 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 user’s comprehension of a system, his or her mental model, strongly. As described in Chapter 3, Git’s conceptual model is flawed in certain ways, which is also reflected in its terminology. In order to avoid adopting these flaws for the content authoring tool, some Git concepts had to be revised, reconsidered or combined. This is why new names needed to be found for these concepts, which would properly reflect their purpose and help users to form an accurate mental model. The goal of the focus group was to identify these concepts that are particularly problematic as well as ideating on better names. The expected outcome were new ideas as well as valuable input for the second design iteration.

7.1 Procedure

The focus group was scheduled for 90 minutes with a short break in between. The first part consisted of an open discussion among the participants of the first usability study, during which they talked about their difficulties during the test sessions. They were briefed to put particular emphasis on the language used within the interface. In case the discussion steered off into another direction, the author, who acted as the facilitator, tried to direct it towards the terminology aspect again.

During the second phase, participants were expected to come up with alternatives for the current naming. In a short survey, prior to the meeting, participants had voted which features and terms they found most puzzling (See below). The results of the survey as well as the outcome of the previous usability study, formed the basis for the discussions during the second phase. In order to provide participants with a
more solid understanding of version control, a short presentation was given before the start of the second phase. Afterwards participants were asked to conceive alternatives for the most problematic terms. At first for themselves and later on together with their peers.

7.2 Most Problematic Terms

In a short survey prior to the meeting participants had been asked how well they had understood different version control concepts. The least understood terms or features were *Pull Requests*, *Branches*, *Merging* and the Reviewer functionality. Furthermore, the previous usability study has shown that naming of the default branch ("master branch", Issue #4) and the history (Issue #24) can cause problems as well.

7.3 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 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, but they also stated that freelancers likely will not have this luxury. Most people agreed that illustrations, explanations and tooltips can be a great help while using the system.

7.4 Second Phase

At the beginning of the second phase the author 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 cannot name it “request proofread”, because for some users it serves a different purpose, such as publishing changes.

Some of the terms listed in Table 7.1 have deficits 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 (i.e. every branch except 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.

<table>
<thead>
<tr>
<th>Pull Request</th>
<th>Merge</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Changes</td>
<td>Finalize</td>
<td>Working version</td>
</tr>
<tr>
<td>Request of Changes</td>
<td>Publish Changes</td>
<td>Experimental version</td>
</tr>
<tr>
<td>Request Approval</td>
<td>Accept Changes</td>
<td>Copy</td>
</tr>
<tr>
<td>Request Review</td>
<td>Approve Changes</td>
<td>Draft</td>
</tr>
<tr>
<td></td>
<td>Save</td>
<td>Public/Private Version</td>
</tr>
<tr>
<td></td>
<td>Authorize</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1: Suggested alternative terms

7.5 Conclusion

The focus group has shown that it is almost impossible to detach a user’s understanding of a concept from its name. A weak mental model of a system can be the result of inconsistent concepts or bad names, or both. In this regard, the focus group did not help to illuminate this problem area. On the other hand, some viable alternatives to existing terms were found. The next design iteration, which is described in the next chapter, utilizes some ideas and the insights gained through this focus group. The subsequent usability study then evaluates whether the new naming scheme contributes to an improved usability.
Chapter 8

Second Design Iteration

The previous usability study revealed several usability issues that prevented users from reaching their goals. Most users experienced problems when using the diff view and during the saving process. Furthermore, some version control terms were confusing, a fact that was also discussed during the focus group meeting described in the previous chapter.

This chapter describes how these problems have been addressed and how the results of the user study as well as the focus group informed the next iteration of the interface design. The design presented below presents a significant step towards the ultimate goal of providing a usable version control system to a non-technical user group.

8.1 Navigation and Terminology

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 is now save changes, pull requests is now review requests, create pull request is now merge or 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
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 8.1).

### 8.2 Branches/Working Copies

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).

![Figure 8.2: Live data warning old (left) vs. new (right)](image)

The warning message in the previous prototype (Figure 8.2) 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 in order to emphasize its purpose of representing the state visible to the Babbel end-user.
8.3 Improved Diff View

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 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 8.3).

Issue #12, which prevented users from navigating from the diff view to the editing view, was addressed by underlining the breadcrumb, so that it became clearer that one 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 8.3: Improved difference view showing an added translation](image1)

![Figure 8.4: The saving shortcut allowing users to save changes with one click](image2)

8.4 Saving Process

As Figure 8.4 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 8.5), 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, which was related to the naming of the "back
to editor" button and the fact that users confused its meaning with a "real" content editor, was solved by using a simple x instead, which is often used for closing views or popups.

![Image of the advanced saving view]

**Figure 8.5:** The advanced saving view, which lists all changes since the last save

8.5 **New Content Representation**

For the new interface a different content representation was designed (Figure 8.6). The old one (Figure 5.5), which was using so called accordions, hid a lot of information in the beginning. The main reason for using accordions, according to the Yahoo Developer Network, is "to compress a large amount of options into a limited space.\(^1\) This design pattern has the advantage to provide users with a quick overview, but the downside is that some information can only be exposed through a user interaction. Bret Victor explained this phenomenon as "harmful interaction" and provides

\(^1\)https://developer.yahoo.com/ypatterns/navigation/accordion.html
the insight that good software design is often just good graphic design. The accordions were one of the reasons why Scenario 3 presented such a hurdle for most users during the last study (Issues #19 and #20). Therefore, 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 without necessary interaction and thus enable users to find errors or problems in the overall lesson structure more quickly. Furthermore, this spreadsheet-like design should be very familiar to most users, since spreadsheets were used a lot as an auxiliary tool before, as described in Chapter 4.

Figure 8.6: The main editing view of the content authoring tool
Chapter 9

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.

The setup of the second study was very similar to the first one: The same set of scenarios was used, except that one additional scenario was added. This made comparing the results of study 1 and 2 easier. Furthermore, a post-session questionnaire was used in order to evaluate the perceived usability of the system. In total, ten users with diverse backgrounds participated.

9.1 Study Design

Whereas the first usability 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, the second study was more about validating that the introduced changes had a positive impact on the user experience. Furthermore, the general suitability in regards to the envisioned purpose of the system was evaluated. In order to make the results of the first and second study comparable, the setup, especially in regards to the user scenarios, was kept similar.

The screens shown in the previous chapter were part of a web-based prototype, that utilized a framework for rapid application development, called AngularJS\(^1\). Whereas the prototype in the previous study still lacked a couple of important features, such as error messages, the system that was tested in this study was already functional to a large degree. This meant that users could freely interact with the system and therefore the sessions provided a fairly realistic picture of how the system would be used in a real-world environment later on.

\(^1\)https://angularjs.org
Additionally to verifying that the introduced changes had indeed improved the usability, a benchmark was established for the very first time. The benchmark was based on the metrics and the post-study questionnaire. Whereas the metrics answered the question whether a goal is reachable within a certain time-frame and by only committing a small number of errors, the questionnaire provided insights into how well the system was perceived in general and which feature areas still need improvement. The benchmark serves as a baseline for other researchers who want to draw on the ideas presented in this thesis and can also be used to evaluate future developments of the system at Babbel.

9.1.1 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 9.1 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.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Language Team</th>
<th>Job Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>French</td>
<td>Content Editor, QA</td>
</tr>
<tr>
<td>2</td>
<td>Swedish</td>
<td>Content Editor</td>
</tr>
<tr>
<td>3</td>
<td>German, Polish</td>
<td>Project Manager</td>
</tr>
<tr>
<td>4</td>
<td>French</td>
<td>Content &amp; Image Editor</td>
</tr>
<tr>
<td>5</td>
<td>Portuguese</td>
<td>Content Editor</td>
</tr>
<tr>
<td>6</td>
<td>French</td>
<td>Project Manager</td>
</tr>
<tr>
<td>7</td>
<td>English</td>
<td>Content Editor</td>
</tr>
<tr>
<td>8</td>
<td>Russian</td>
<td>Content Editor</td>
</tr>
<tr>
<td>9</td>
<td>Russian</td>
<td>Content Editor</td>
</tr>
<tr>
<td>10</td>
<td>German</td>
<td>Content Editor</td>
</tr>
</tbody>
</table>

Table 9.1: List of Participants
9.1.2 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.

9.1.3 Scenarios

Scenarios 1 to 3 were identical to the ones used in the first study (as described in Section 6.1.3). The fourth task has been added to the study in order to lay more focus on the new content representation (main editing view).

The procedure, especially in regards to the scenarios, was kept more or less consistent 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 mostly attributed to the changes in the interface. Of course, given the small number of participants, the different abilities and professional backgrounds of the participants can affect the results as well.

**Scenario 4: Create Exercise and Fill With Content**

Add a new memory exercise to this lesson (Link provided). Please don’t 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.

9.1.4 Post-Study Questionnaire

As an additional means of gathering feedback users were asked to answer a post-study questionnaire. For this, a standardized questionnaire called PSSUQ (Post-study System Usability Questionnaire) was used. PSSUQ was specifically devel-
oped for scenario-based usability evaluations at IBM and produces four different scores (one overall score and three sub-scales), which signify how well the computer system was perceived by users [36]. The goal of utilizing this questionnaire was to reveal potential weak spots of the system, not discovered through the observations, and to get an overall picture of how well the system was perceived. Note that the third version of the questionnaire was used, which consists of 16 items rather than 19 (3 items that contributed rather little to the reliability were removed for this version) [34]. Each item has to be rated on a scale from 1 to 7, where 1 signifies strong agreement and 7 strong disagreement. Users could also choose to skip an answer. Ultimately, the questionnaire resulted in four scores describing the overall satisfaction with the system as well as the system quality, information quality and interface quality [34].

9.1.5 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.

9.2 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.

9.2.1 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, which is explained in more detail in Section 9.2.2. Unfortunately, 61% is still an unsatisfactory completion rate. Exactly why it is still so low is hard to tell. Users still seemed to have problems with the dual purpose of merge requests, which serve as a means of reviewing changes. Some users simply forgot to add a reviewer to their requests, others said they would probably send a link to their colleagues and ask them to have a look at the changes.

Another 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.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>First Study</th>
<th>Second Study</th>
<th>Difference in pp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>40%</td>
<td>61%</td>
<td>+21 pp</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>100%</td>
<td>94%</td>
<td>-6 pp</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>70%</td>
<td>72%</td>
<td>+2 pp</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>100%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 9.2: Comparison of task completion rates during first and second user study*

**Time on Tasks**

As Table 9.3 shows the biggest difference is again observed for Scenario 1. Participants during the second study needed about 75 seconds less for finishing this scenario than participants in study 1. For Scenario 3, participants needed about
half a minute less than during the previous study. Because of the small sample sizes these numbers are not statistically significant and should therefore be taken with a grain of salt.

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>First Study (mean)</th>
<th>Second Study (mean)</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>367</td>
<td>292.25</td>
<td>-74.75</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>128</td>
<td>146.57</td>
<td>18.57</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>395.5</td>
<td>369.42</td>
<td>-26.08</td>
</tr>
<tr>
<td>Scenario 4</td>
<td></td>
<td>280</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.3: 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, e.g. there were fewer users asking things like “What is a pull request?”. Nevertheless, users still encountered serious usability issues that led to confusion and errors. Those are described in more detail below (Section 9.2.3).

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Study 1</th>
<th>Study 2</th>
<th>Difference in Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>2.2</td>
<td>1.87</td>
<td>-0.33</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>1.6</td>
<td>0.5</td>
<td>-1.1</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>4.2</td>
<td>2.12</td>
<td>-2.08</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.4: Comparison of error rates during first and second user study

9.2.2 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.
9.2. FINDINGS

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 9.4). 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 have gotten 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 9.1 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 8.2).
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 like placeholders and were therefore selected by some users (instead of the actual input field).

9.2.3 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 system and were no longer constrained by the semi-functional prototype as in the first study. In total, the number of severe issues (Severity score of 10 or 5) was reduced from 15 in the previous study to 11 during the second study. Furthermore, the impact scores of most issues were lower this time, which can be attributed to fewer users actually experiencing problems.

The following section lists the usability issues discovered during the test sessions. As for the first study, the impact score by Sauro and Lewis [34] 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 9.5). 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.

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

Table 9.5: Usability issues related to working copies

**Merge Requests**

As Table 9.6 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.

**Diff**

The diff view, which was only subject to a small set of changes as compared to the last iteration, caused the most severe problems (Figure 9.7). 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 the missing context as well as the
<table>
<thead>
<tr>
<th>#</th>
<th>Usability issue</th>
<th>Severity</th>
<th>Frequency</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>User tried creating merge request before saving (no error message)</td>
<td>10</td>
<td>20%</td>
<td>20</td>
</tr>
<tr>
<td>33</td>
<td>User does not know how to let colleague review changes</td>
<td>10</td>
<td>10%</td>
<td>10</td>
</tr>
<tr>
<td>34</td>
<td>When approving a merge request and commenting first comment confirmation is forgotten</td>
<td>3</td>
<td>30%</td>
<td>9</td>
</tr>
<tr>
<td>35</td>
<td>Commenting in request detail view (diff) does not result in feedback</td>
<td>3</td>
<td>10%</td>
<td>3</td>
</tr>
<tr>
<td>36</td>
<td>The list of requests does not show the reviewers at a glance</td>
<td>3</td>
<td>10%</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 9.6: Usability issues related to merge requests

entirely different structure of the content (as compared to the main editing view). 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. The listed changes were only of limited value, if not meaningless, when presented in this way. One participant described it as being “totally lost inside this view”.

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

Table 9.7: 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 9.8). 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.

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

**Table 9.8: 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.

### 9.2.4 Post-study Questionnaire

Table 9.10 shows the resulting scores of the Post-Study System Usability Questionnaire (PSSUQ). The lower the value, the more satisfied users were with the particular aspect of the system. These scores are based on the mean of 16 items that

---

2http://ui-patterns.com/patterns/BlankSlate
### Table 9.9: Usability issues related to main editing view

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

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 9.10).

- **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.

#### 9.2.5 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 clearer terminology. For changes like the new saving feature and the improved content representation a shift in behavior 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. Furthermore, testing a functioning application is very different from testing a prototype. Users are much more demanding and expect the interface to work properly.

In general, the results from this study should be taken with a grain of salt. The differences in the metrics are not actually statistically significant (because of the small sample size) and the scenarios given to users only emulate a real-world environment. Once the tool is used on a daily basis, new issues might arise, that were simply not discovered by first-time users during the testing sessions. More experienced users might demand convenience features, such as shortcuts, in the future.

The next chapter presents the final design of the system, which is based on the findings of this usability study.
Chapter 10

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.

10.1 Difference View

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 10.1). 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 10.1: Newly designed difference view, which provides more context

10.2 Saving Process

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 9.8. 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 10.2) 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, signaling that there are no current changes that were not saved yet.

Figure 10.2: The new editing view combines reviewing and saving changes and reduces the amount of clicks needed to save

Figure 10.3: Hovering a highlighted changed exposes the previous value
10.3 Error Messages

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 10.4).

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 10.5).

Figure 10.4: Improved error message inside working copy modal

Figure 10.5: A warning informing users that the selected working copy is equal to the target of the merge
10.4 Navigation Bar

The top-level navigation was changed conceptually, which also had to be reflected in
the layout of the navigation bar (Figure 10.6). Instead of treating the different views
as "layovers", which were closed with a little x in the upper right corner, the navi-
gation 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-
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 10.6: The redesigned navigation bar featuring fewer icons and a consistent
way of navigating to the content editor

10.5 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\(^1\)
solution (Figure 10.7). This means that the text turns into an input field when clicked
and can be edited.

Figure 10.7: New lesson header enabling in-place editing

\(^1\)http://ui-patterns.com/patterns/InplaceEditor
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 10.8) next to the selection of the learning language.

**Figure 10.8:** 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. For example, the form for merging or publishing copies was simplified and the input field for the second reviewer is now hidden by default (Figure 10.9).

**Figure 10.9:** Merge or publish copies
Chapter 11

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 it is indeed possible to unify a content authoring tool with a version control system, but the question remains, whether this liaison is beneficial to the tool's users? By looking at the research questions posed in the beginning, we might be able to come closer to an answer.

11.1 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 user 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 \[8\]. 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. [7] 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.

11.2 Sub-question 2: Improve Learnability

The second sub-question asked how the learnability of the system could be optimized. According to ISO/IEC 9126 [37] 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. [7] and Jackson and Perez De Rosso [8]. How these studies influenced the interface design is described in Chapter 5.

Secondly, the iterative design process and the usability studies ensured that hard to comprehend features were discovered and simplified. Only users with no prior experience in version control software participated in the studies (Appendix A.1) and no additional instructions were given before the scenarios. This ensured 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). This helped users to navigate the system and recover from errors more easily.

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. Even though, none of the alternatives were used directly, the results inspired a design that utilizes terms 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 were quite high (82% on average during the 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 training. 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 become proficient quickly in the future.

11.3 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 these 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 9, Figure 9.2) as well as the relatively low error rates (Chapter 9, Figure 9.4), 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-aware 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.

11.4 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.
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\(^1\) or Drupal\(^2\), 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 a purely technical domain, it might increasingly appear inside more mainstream products in the future.

\(^1\)https://wordpress.org
\(^2\)https://www.drupal.org


Bibliography


Appendix A

Background of First User Study

A.1 Pre-session Questionnaire

<table>
<thead>
<tr>
<th>User</th>
<th>Familiar with VCSs</th>
<th>VCS Used</th>
<th>Experience with current tool</th>
<th>Other CMS or authoring tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>No</td>
<td>3 months - 1 year</td>
<td>WordPress</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td>more than 3 years</td>
<td>no</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td>no</td>
</tr>
</tbody>
</table>

*Table A.1: Pre-session questionnaire of first user study*
Appendix B

Background of Second User Study

B.1 Pre-session Questionnaire

<table>
<thead>
<tr>
<th>User</th>
<th>Familiar with VCSs</th>
<th>Used VCS</th>
<th>Experience with current tool</th>
<th>Other CMS or authoring tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>No</td>
<td>Less than 3 months</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td>Wordpress</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>No</td>
<td>1 - 3 years</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>No</td>
<td>more than 3 years</td>
<td>Drupal, Adobe E-Learning Tool and iBooks</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>No</td>
<td>more than 3 years</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>No</td>
<td>Less than 3 months</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>No</td>
<td>1 - 3 years</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>No</td>
<td>Less than 3 months</td>
<td>WordPress, Imperia 8</td>
</tr>
</tbody>
</table>

Table B.1: Pre-session questionnaire of second user study
## B.2 Average Scores for 16 PSSUQ Items

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

**Table B.2:** All 16 items of the PSSUQ