Characteristics, Work Modes, and the Design of Digital Collaboration Tools

A survey and analysis of collaboration apps and research on them
Master thesis in Information Systems Technology by Fabian Meerkötter
Date of submission: February 11, 2025

1. Review: Prof. Dr.-Ing. Mira Mezini
2. Review: Dr.-Ing. Ragnar Mogk
3. Review: Julian Haas M.Sc.
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

INFORMATIONS SYSTEM TECHNIK

Software Technology Group
TU Darmstadt | FB Informatik

Characteristics, Work Modes, and the Design of
Digital Collaboration Tools
A survey and analysis of collaboration apps and research on them

Master thesis in Information Systems Technology by Fabian Meerkötter

Date of submission: February 11, 2025

Darmstadt

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-311386
DOI: https://doi.org/10.26083/tuprints-00031138
Jahr der Veröffentlichung auf TUprints: 2025

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
https://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Abstract

This thesis investigates digital collaboration tools (DCTs), including the work modes supported by them and their general characteristics. This is done to inform current research on local-first software [1], which are new types of collaboration apps aiming to shift computing from the cloud back to local devices without compromising the ability to collaborate with other people. Existing literature on the topic of collaboration and computer-supported collaborative work (CSCW), spanning the past 40 years, is reviewed and discussed. Older research is contrasted with more recent research, and existing definitions and taxonomies are collected. An analysis of 45 currently available collaboration apps is performed. The apps are categorized and investigated regarding supported work modes and general properties. A user survey with 37 participants is conducted in the form of a digital questionnaire, asking questions about people's usage of DCTs, positive as well as negative experiences with them, and their opinions on different work modes. Based on these results, definitions of collaboration and digital collaboration tools are given, as well as a taxonomy and characterization of the latter. The results are further discussed and recommendations for the development and design of modern DCTs are made. The main findings include that DCTs are diverse in nature, and their support for different work modes heavily depends on the type of application and work being done with it. The asynchronous work mode can be considered the default, except for the case of pure meeting apps. The synchronous work mode is beneficial for most apps, and should be properly supported with appropriate awareness mechanisms. Local-first software is promising, but still has several limitations that hinder its development and adoption. Automatic conflict resolution is found to have several drawbacks, therefore transparency and preservation of user intent is emphasized instead. The thesis concludes by outlining suitable areas for future research, including evaluation of a synchronous-independent work mode, and improvements in conflict-free replicated data types (CRDTs).

Contents

List of Acronyms

1. Introduction

Collaboration is fundamental to human behavior. Collaborative behavior emerges at a young age and is an innate skill [2]. Collaboration offers benefits that no single individual could achieve on their own. Modern digital collaboration is mainly concerned with improving efficacy, efficiency, and coordination. It has been around inside and outside of the workplace for decades, and has become even more important during the 2020 pandemic when physical collaboration was not an option anymore [3, 4]. Collaboration between people in remote and ubiquitous computing environments where a single piece of equipment (like a computer or whiteboard) cannot be shared is a challenge even beyond the pandemic. This can apply to field workers that need to stay in touch and exchange data with their base of operations, or remote office workers in trans- or international teams that work across countries and time zones.

Collaboration nowadays even plays an important role in industries like food services, manufacturing, or aviation[1]. Collaborative or asynchronous learning is another area that makes heavy use of digital collaboration tools (DCTs). Supporting students' social behavior and connectivity constraints is essential to successfully employ modern teaching techniques [5], which can in turn improve students' ability to collaborate effectively [6]. Collaboration is a broad term, and there are many ways to collaborate with other people. Not everyone works the same way; people have different preferences, schedules, and social behavior. Some people thrive when working with others in real time, some people prefer to work in isolation, and others prefer something in-between. Sometimes work modes are switched spontaneously, when certain physical or digital social cues nudge people towards a different type of collaboration [7, p. 15]. Working without a reliable internet connection is also still a common occurrence. Researchers or journalists might have to take notes at remote locations [1]. Even in corporate settings, where one might assume that internet access is all but guaranteed, there are scenarios in which connectivity is not available. Employees of a power company need to collect meter readings. Service personnel might have to update hardware status or contract information. *Requiring* internet connectivity for these tasks would entail that those people need to wait until they're connected to the internet again before they can record their changes. That means either remembering everything or using supplemental tools to keep track of notes and data. Tools that can be used even when offline could improve this workflow.

Motivation Given the previously mentioned use cases for DCTs in various settings, it seems like there would be an abundance of popular tools that serve all kinds of needs for different people. Remarkably, the majority of collaboration tools appears to be rather homogeneous, with tools usually employing a centralized architecture that requires almost constant connectivity. On-premise, offline, or distributed usage is underrepresented, especially among recent tools.

There certainly are tools that use different approaches, one of them being *Git*[2]. It is a version control system that serves as an industry standard in software development, and is

[1] `https://ditto.live/customers`, visited on 2024-11-21
[2] `https://git-scm.com/`

claimed to be used by at least 90% of Fortune 100 companies, with well over 100 million users worldwide[3]. *Git* is decentralized and works offline for the most part, using network connectivity only to periodically synchronize changes with collaborators (which can even happen through alternative side channels, meaning *Git* itself doesn't require internet at all [8]). Why is a tool that is radically different from most other DCTs so popular, and why aren't there more tools outside of software development that work similar to it? *Git*'s focus on working offline is also peculiar, since there is a distinct dependence on internet connectivity for modern software development. Other work is arguably less reliant on internet access than software development, but there seems to be a lack of collaboration tools that break the centralized and always-online norm.

This begs a question: Why are collaboration tools so often using this centralized and always-online approach? There clearly are good reasons for building and using collaboration tools that can tolerate a lack of connectivity or support independently working on a project. It is also definitely possible to build good tools with other properties, which is evident by *Git*'s popularity. So what is the reason for this dichotomy of requirements and available tools? What is stopping collaboration tools from supporting offline and independent work? And what makes collaboration tools successful and usable to begin with? There isn't an obvious answer to these questions, and only little research on the topic. Research on collaboration apps has so far been narrow in scope, with no clear and generally applicable suggestions for the design and behavior of DCTs. There has also been little research on the work modes of these tools, which describe the distinct ways in which collaboration happens through them, like asynchronously or in real time.

This thesis was originally sparked by research into local-first software [1]. Local-first software actively tries to switch to more decentralized approaches for software design, to support independence, improve ownership and privacy, and to improve performance as well as the User Experience (UX). To this end it tries to apply new algorithms and data structures to both new and existing problems. Research on local-first software started in the late 2010s, and the research on the fundamental algorithms and data structures often used for it started some years earlier. Almost a decade later, the first consumer and business-grade local-first applications are starting to emerge. Despite some clear theoretical advantages, adoption of local-first software seems slow. Since local-first software mainly focuses on collaborative tools, investigating possible reasons for slow adoption is of interest for this thesis.

Goal of this Thesis This thesis aims to establish the fundamental properties of DCTs, and make suggestions on what functionality they should provide. It also wants to find criteria for apps that help to explain which types of collaboration apps are suitable for which kinds of work modes. A distinction is made between four different work modes: synchronous, asynchronous, independent, and offline. These modes should cover most of the observed DCTs, clarifying their basic properties and capabilities as well as helping to categorize similar tools. The goal is to find universal characteristics of DCTs. The thesis is supposed to act as a framework that helps to explain how DCTs work and why, as well as identify common patterns and challenges among these tools. As part of this an effort is made to find out why there are so few apps supporting the independent and offline work modes. Additionally, this thesis attempts to educate future research on local-first software, conflict-free replicated data types (CRDTs), programming languages, and framework

[3] `https://github.com/about`, visited on 2024-11-21

design to make it easier to build good, usable, and satisfying applications for digital collaboration. It therefore sits somewhere at the junction between human-computer interaction (HCI) research and programming language research.

Methodology & Structure Existing literature on this topic is presented and discussed as part of a systematic literature survey. Currently available DCTs are researched, analyzed, and also discussed. These range from more traditional text-based collaboration tools to those that offer unique data types and challenges. Meeting tools are only touched on peripherally because their unique scope, focus on communication, and lack of editing capabilities makes them much less diverse than other collaboration tools. Based on this, this thesis develops a taxonomy of DCTs, describing their basic principles and features. A user survey is presented that highlights the research topic from another perspective. In the end, a range of properties of DCTs are presented, and recommendations for their design are made.

This thesis is divided into six chapters. Chapter 2 presents and discusses existing literature, definitions, and taxonomies on the topic. It also distills them into a new set of definitions. Chapter 3 provides an analysis of currently available collaboration apps, along with required functionality of DCTs. Chapter 4 presents the results of the conducted user survey. Chapter 5 discusses the results and insights of the previous chapters and contrasts them. Chapter 6 concludes the thesis by summarizing the insights into and recommendations for the design of DCTs, as well as proposing future research opportunities.

2. A History of Literature on Collaboration

In this chapter, a detailed overview of research into the areas of computer-supported collaborative work (CSCW), groupware, collaborative editing (CE), conflicts, and modern DCTs is given. CSCW is a frequently-used umbrella term for the process of working together digitally. Groupware is a branch of software that emerged in the late 1980s and deals with software meant for group activities. CE is a subset of CSCW that focuses mainly on creative work like collaborative writing. This will serve as a foundation for the analysis and taxonomy presented in later chapters. Since this thesis is mainly concerned with work modes, features, behavior, and requirements of DCTs, the literature survey is also focused to these areas.

2.1. Evolution of Collaboration Tools

This section presents and discusses existing research, ordered roughly from oldest to newest. Exceptions are made, for example in cases where it makes sense to contrast older with more recent research. This should give a good understanding of how the perception and general understanding of this research topic have evolved over the years. It also allows contrasting research results with currently existing applications later on, as well as understanding why collaborative applications work in certain ways.

2.1.1. Early Research on Groupware

Overview In 1991, Ellis et al. published an article about groupware [9]. They describe "a change in emphasis from using the computer to solve problems to using the computer to facilitate human interaction" [9, p. 2]. Specifically, they write about how advances in computation and digital communication enable people to work in a group through "user-to-user interaction" [9, p. 3], and describe important properties for software that enables this. They focus on synchronous (real-time) groupware and system design considerations. To this end they present a definition of groupware as well as two taxonomies, their definition being: "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [9, p. 3].

The first taxonomy is a "Time Space Taxonomy" (based on a book by Johansen [10]) that categorizes applications through the dimensions of time and space. These categories are not exclusive, so that a groupware system could for example support both *same time - same place* and *different time - same place* scenarios. The taxonomy is also expandable to additional dimensions like the group size. The second taxonomy deals with features and functionality offered by various groupware systems. It tries to cluster related applications together based on similar functionality, rather than by different scenarios of collaborator presence. They include categories like *message systems*, *multiuser editors*, and *intelligent agents*, to name a few. In general, groupware is described as a spectrum, where the extent to which software provides information about "the participants", "the current state of the project" and "the social atmosphere" [9, p. 4] determines at which point of the spectrum they exist. Even e-mail is considered groupware, but "low on the groupware spectrum" due to its static nature and lack of "environmental cues" [9, p. 4].

In addition to the taxonomies, Ellis et al. also present what they call "key disciplines [...] for successful groupware" [9, p. 7], those being distributed systems, communications, human-computer interaction, artificial intelligence, and social theory. They maintain that while any of these disciplines can be used for groupware on its own, approaching groupware from multiple disciplines will improve the "understanding of the theory and practice" [9, p. 7] of it. They also comment on important design considerations for groupware systems, highlighting the importance of user-friendliness, flexibility, technological control, and the overall effects on users, as well as their interactions. Finally, they present *GROVE*, which is a groupware prototype, along with experiences and design issues discovered during its use by several groups of people.

Observations The first interesting thing to note is that Ellis et al.'s definition specifically mentions a "shared environment". Groupware works by letting different users interact with this environment, which leads to them having access to the same information and being able to see each other's work. The shared environment is where the data is being exchanged, and is needed for collaboration. Ellis et al. also give an example of designers working together on a computer-aided design (CAD) project. They mention explicitly checking files in and out to work and to share their work with collaborators, and call for shared environments that facilitate seamless collaboration without these manual updates and locking mechanisms. Still, even over 30 years later, modern CAD tools often don't support simultaneous editing of the same object[4].

Another aspect worth mentioning is how they highlight social processes. Even at the very beginning of groupware and CE research it was clear that the social behavior of users plays an important role for how well such software can be used. Different people have different preferences, and when their work isn't isolated anymore it becomes paramount that these differences don't lead to conflicts or inefficiency.

Also notable is their design issue of *robustness* and concurrency control. They offer several fundamental strategies for being robust against concurrent changes due to network failure or simultaneous edits[5]. These range from *simple locking* to *Operation Transformation* (which nowadays is known as Operational Transformation (OT) and used in many popular collaborative editors).

2.1.2. Considerations for Building Groupware

Overview There have also been other early attempts at building collaborative applications. Edwards et al. developed *Bayou*, a distributed and replicated storage engine [11], and used it to implement several collaborative applications. In addition, their paper characterizes collaborative systems and explores design issues. The advantages of asynchronous work modes are highlighted. In their opinion it is important that the support for asynchronous work is considered during the design of an application.

[4]This is the case for both Siemens NX (`https://plm.sw.siemens.com/en-US/nx/`) in combination with Teamcenter (`https://plm.sw.siemens.com/en-US/teamcenter/`), and Autodesk Fusion (`https://www.autodesk.com/products/fusion-360/overview`).

[5]The maximum time span between changes for them to be considered *concurrent* heavily depends on external factors, like connectivity or network latency. If a collaborator makes a change while offline (no connectivity) while another make a change while connected to a server or peer, these two changes are still concurrent, even if there are hours or even days between them.

It is also mentioned that a range of factors can influence which work mode is best suited for collaboration, including the type of task, the collaborators' own preferences, and the functionality provided by the available tools. They maintain that tasks that are generally suitable to be done with asynchronous applications usually only "require little interactive coordination" or exchange of work artifacts [11, p. 120]. This results in collaborators being able to work independently for short or even extended time spans. This independence is later described as "perhaps the key trait of asynchronous work" [11, p. 120]. They refer to various degrees of independence, which can be anything from getting less distracted by collaborators (or their changes), to working on a completely isolated replica, without being connected to any peer or server. The premise of this kind of independence is then used as a constraint which requires applications supporting asynchronous collaboration to employ weakly consistent replication. This is said to be the only way to achieve fully disconnected functionality when working with a shared data set, since strong consistence would require immediate data exchange with peers. In this context "eventual consistency" is also mentioned and deemed "desirable".

One other issue Edwards et al. consider is explicit control over when information is shared with collaborators. This is meant as a way to withhold certain changes that are not yet fully complete, actively keeping the shared data in an inconsistent state (as in not fully consistent with other replicas) for longer. They touch on the aspect of automatic conflict resolution, stating that this can be used as a means to reduce the amount of coordination required for collaboration. To this end, applications should be able to specify concrete semantics on how conflicts can and should be resolved without user intervention. Lastly, Edwards et al. highlight the challenge of "fluid transition between synchronous and asynchronous modes of operation" [11, p. 127], and expect further exploration in that area.

Observations Edwards et al. touch on a range of aspects that are important for the design of DCTs. The importance of both asynchronous and synchronous work modes, the factors influencing which work modes are most suitable for an application, and their considerations of independence and control over synchronization are also partially seen in other literature. The way in which they try to combine centralized and decentralized architectures as a means of switching between synchronous and asynchronous work modes is a unique approach. It raises several important questions regarding how and when data is synchronized with collaborators, and how that is influenced by users' preferences and the functionality of the applications. A lot of the behavior described by Edwards et al. can be seen in modern collaboration apps, especially local-first ones.

The issue of independence is discussed in detail by Edwards et al. Several benefits are mentioned, including a reduced need to share changes with collaborators, which they argue is common for asynchronous work in general. Avoiding distractions by collaborators is also noted. But independence also comes with certain drawbacks, for example an increased risk of conflicts, which have to be handled somehow. The loss of independence which Edwards et al. warn about doesn't appear to be problematic either, as evident by the rarity of support for the independent work mode observed in Chapter 3. Edwards et al. also justify the importance of independence by highlighting frequent issues of low network bandwidth or disconnection from the internet, which generally isn't as much of an issue nowadays.

2.1.3. Focusing on Group-Work

Overview Kate Ehrlich describes groupware as "the applied side of CSCW (Computer Supported Cooperative Work)" [7, p. 21] in their chapter on the design of groupware applications. They also state that "groupware is about group-work" (as opposed to task-focused work), specifically about "provid[ing] computer support for group work" [7, p. 21], where group work is any "written and spoken communication, meetings, shared information, and coordinated work" [7, p. 21]. This work can happen both synchronously or asynchronously. To Ehrlich, four categories (*communication*, *meetings*, *information sharing*, and *coordinating work processes*) characterize groupware applications. As an example they mention email as the "perhaps [...] most widely used groupware application" [7, p. 22], which falls into the (asynchronous) communication category.

Ehrlich continues with describing the requirements and design of groupware applications. This deals with going from observations to concrete design requirements, without forgetting less observable or implicit aspects of the work. They also focus on how the unique requirements of certain tasks can be analyzed to build apps that fulfill these requirements. Aside from some general remarks regarding the design research for groupware applications[6], Ehrlich specifically mentions four concrete considerations for the design of applications. These are *informal communication*, *synchronous and asynchronous awareness*, *anonymity*, and *customization*. Aside from *customization*, these are all traits that affect the behavior of people engaged in group work, through social cues or lack thereof. Groupware applications should support these traits to successfully mimic non-computer supported group work, and to avoid impacting efficiency through over-formalization, according to Ehrlich. Reduced efficiency can also be avoided by allowing different levels of customization, which enables apps to be adapted to the current tasks and contexts.

Finally, Ehrlich discusses the adoption of groupware applications and factors that influence it. These factors include the need for purchasing multiple copies (everyone participating needs access to the application), ease of setup and integration (deploying the application), as well as *organizational implications* (how compatible the groupware application is with the customer's style of work). To improve chances of successful adoption, two strategies are discussed: *incentives and motivation*, and *critical mass*. *Incentives and motivation* deals with convincing people that an application is worth using by describing its benefits, or by mandating its use. While especially the latter option will usually lead to widespread adoption, it doesn't necessarily lead to successful adoption, if for example the organizational implications don't match. *Critical mass*, or simply peer pressure, is about applications reaching a critical point, at which people try to motivate their collaborators to adopt the app for their own convenience. This is a point where adoption has become significant enough that non-adopters start to hinder collaboration.

Ehrlich concludes by reiterating that "single-user applications are about tasks", while "groupware applications are about work", the difference being that tasks are "explicit, observable, concrete", and work is "tacit, invisible, amorphous" and "about people, habits and culture" [7, p. 42].

[6] This mostly reflects the design research for single-user applications, but with additional considerations regarding how work is being done *outside* the app or task, as well as the context or environment where the work is being done.

Observations Kate Ehrlich's [7] definition of groupware takes a different approach than Ellis et al.'s [9]. Where Ellis et al. use very specific categories and are concerned with the time and place in which collaboration is happening, Ehrlich delegates to very general categories and focuses on the social dynamics and work patterns of users. In the case of email, Ellis et al. barely consider it groupware due to its mostly static and rather blunt nature which lacks social and environmental cues. In contrast, Ehrlich sees it as one of the most popular and successful instances of groupware. This can be attributed to its ease of adoption, symmetric benefit, and high flexibility.

Another interesting aspect is that while Ehrlich's distinction based on simple categories at first seems mostly superficial (in that it doesn't really deal with how these apps behave, just what general kind of work they're used for), it actually influences the properties of apps in certain categories. As an example, the *meetings* category only features synchronous apps. Supporting work in a meeting happens with all participants present at the same time, and it is not possible to interact with the same shared data (the meeting notes, votes, etc.) in the same way later on. While it would be conceivable that such an application would save the meeting notes for later asynchronous interaction, this then falls into the category of *information sharing* instead. Ehrlich also mentions project management systems as application types that are not generally considered groupware, but is of the opinion that they should be. This was clearly a valid assumption since nowadays, many of the most popular collaboration tools are project management tools[7]. This is further supported through a survey by Xu et al. [12], which is discussed in the next section.

The focus on requirement analysis, and more importantly user-focused design, also hints at an important progression in the design process towards a more user-centered approach like in modern UX design, following early pioneers in this space [13, 14, p. 70]. While the initial design guidelines given by Ehrlich do not mention any form of iterative design, they later on mention an "especially good example of a successful collaboration" that employed iterative, user-focused design [7, p. 33]. In general, this focus on the user and group, as well as their behavior, becomes very apparent in the chapter and is well aligned with the principles of UX design.

Another takeaway from the chapter is that collaboration tools and groupware tend to be *unanimous* in nature. They work best if everyone (within a group of people working together) is using them. If only a part of the collaborators are using them, collaboration becomes less effective. This seems to be because they change what Ehrlich calls *protocols*. If some members of a group use a certain tool and others don't, they are interacting through incompatible protocols. Unifying the use of a certain tool will align these protocols again. Adoption of a tool here becomes a path of least resistance, where if there are more adopters than non-adopters, it is easier and more compelling to have everyone adopt the tool than to use an alternative.

2.1.4. Asynchronous Collaboration Tools

Overview Xu et al. offer a survey specifically of asynchronous collaboration tools [12]. They analyze ten different tools that range from professional business tools to free open-source tools. To start off they give their definition of collaboration as "all processes where people work together to achieve results" [12, p. 2]. They give a brief overview

[7] Examples of this would be Jira (`https://www.atlassian.com/software/jira`) or Trello (`https://trello.com/`).

of the different approaches of the tools, including different architectures (centralized, decentralized), various synchronization solutions, and types of supported clients.

They continue by describing common features they found within the apps, and dividing them into four functional categories: *communication*, *information sharing*, *group calendar*, and *project management*. This is accompanied by a list of example features for each category. The *communication* category also includes examples of synchronous features, although these are described as "basic but handy" [12, p. 4], along with a note that true synchronous collaboration tools offer more advanced features.

They describe two tools that make use of peer-to-peer communication, *Collanos Workplace* and *Microsoft Office Groove 2007*. Both employ a hybrid approach, where besides peer-to-peer communication, a central server is additionally available. In both cases the server helps with bootstrapping of the peer-to-peer network, and in the case of *Groove* the server also caches updates to distribute them to collaborators in the absence of the authoring peer. Another common property is that both tools exclusively create updates client-side and support doing this even when disconnected from the network. There is no main replica that would act as a ground truth; each client has an authoritative copy. Xu et al. then list a table of features that compares the presence of features across the ten tools. The most common features are file sharing (present in all tools), tasks for project management (present in 8 out of 10 tools), and a shared calendar (also present in 8 out of 10 tools). *PHProjekt* and *Microsoft SharePoint Server* offer most of these features, but no tool offers them all. The authors note a general lack of integration between features.

Observations Both Xu et al. [12] and Ehrlich [7] make use of similar categories, although Xu et al. replace *meetings* with *group calendar* and specify *coordinating work processes* as *project management*. The former can be explained by Xu et al.'s focus on asynchronous tools, which aren't suited for meetings (since those are always synchronous [7, p. 21]), while the latter can be seen as a progression of Ehrlich's notion that project management tools should be considered groupware. Overall Xu et al. make a clear distinction between synchronous and asynchronous groupware, and don't consider tools to belong to both types, even if they offer synchronous and asynchronous features. It even appears as if their notion of groupware only includes asynchronous collaboration tools, but not synchronous ones. This is further emphasized by them trying to find categories that are specifically suited to asynchronous tools. In a general sense their way of referring to these tools as *asynchronous collaboration tools* as opposed to *groupware* suggests a shift in the definition and scope of research, where the general category of groupware is deprioritized and the focus instead is on specific work modes of these tools. The general absence of reactivity in 2008 could explain the clear distinction between synchronous and asynchronous tools. Building an app with asynchronous features that can be made synchronous through real-time updates wasn't something to be considered back then. Instead, the general functionality and its use case were the deciding factors determining the work mode.

One interesting observation is that "peer-to-peer collaboration tools allow users to update the data offline" [12, p. 3]. This statement entails that peer-to-peer tools could never be purely synchronous. This is obviously not the case, as evident by peer-to-peer features which are synchronous in nature, like modern WebRTC[8] video communication. However,

[8] `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API`

this notion could indicate that synchronous applications are less suited to peer-to-peer architectures, for instance due to poor scaling with network size.

Finally, Xu et al. mention that "[t]he best collaboration tool is the one that meets users' needs" [12, p. 7], hinting that the quality of a tool can never be fully determined through the quality of its features alone. Instead, a holistic view of the users, their tasks, and their work and social environments is needed to assess how useful a tool is *to a certain user*. This is in accordance with Ehrlich's guidelines for the design of groupware applications [7], and with modern UX design.

2.1.5. Collaborative Writing

Overview Posner and Baecker [15] investigated groupware in 1992, but their focus was on collaborative writing. They present a taxonomy of collaborative writing, characterizing it through four properties. These properties are *roles*, *activities*, *document control methods*, and *writing strategies*. They base their findings on a number of interviews they performed with people of various occupations who had previously worked together on documents. They also propose requirements for the design of collaborative writing apps. Their main goal was to determine if the features offered by existing collaborative writing applications solved users' actual needs.

One main finding was that collaborators start the collaboration process with certain expectations about the writing itself and the quality of the resulting work. They also observed various conventions employed in collaborative writing (such as authorship attribution), hierarchical effects, as well as issues due to incompatible working styles. Trust was a major factor among all participants, and was in fact "most often mentioned as important to the success of a group" [15, p. 2]. Posner and Baecker also note that "[t]echnology and distance can contribute to this lack of trust and, thus, need to be carefully managed to facilitate group projects" [15, p. 3]. Almost all participants used face-to-face meetings for collaboration, with varying success. Use of non-collaborative text editors proved difficult, and communication tools caused more issues than they solved. Low bandwidth was also mentioned as a serious obstacle for long-distance collaboration, citing close proximity as the best solution.

Their taxonomy consists of four different categories, which allow fine-grained definition of collaborative writing processes. The *roles* include *writer*, *consultant*, *editor*, and *reviewer*. They determine how a member contributed to the collaboration, and sometimes also to what extent. *Activities* are divided into *brainstorm*, *research*, *initial plan*, *write*, *edit doc*, *final edit*, and *review*. These activities describe different stages and tasks of the collaboration. *Document control methods* consist of *centralized*, *relay*, *independent*, and *shared*, where notably *independent* is used most frequently (although not always as the preferred method) and *shared* (where everyone can work on the same version of a document at the same time) is used least. Finally, *writing strategies* feature *single writer*, *scribe*, *separate writer*, and *joint writing*. These somewhat overlap with the *roles*, but the focus is more on the text creation and less on the individuals themselves. Except for the *joint writing* strategy, all strategies were at times used due to lacking technological support for alternative approaches. The *joint writing* strategy was generally most useful early on in the project for creating and quickly revising a rough outline for the document. Later on it became more difficult to employ, and was least successful when combined with the *independent* document control method because it frequently lead to conflicts between participants. One thing to note here is that *joint writing* is not the same as synchronous

writing, but specifically refers to a scenario where all writers edit all parts of a document together. This can even happen asynchronously.

Posner and Baecker conclude by discussing "Implications for System Design" [15, p. 8], and listing a collection of requirements. In essence, they highlight the need for "preserv[ing] collaborator identities" [15, p. 9] along with their changes and allowing communication between collaborators, "mak[ing] collaborator roles explicit" [15, p. 9], supporting different writing activities (including planning) and transitioning between them, providing version control and a change history, controlling access to documents as well as sections within the documents, and finally supporting writing by single writers and multiple writers, both synchronously and asynchronously. They continue by applying these requirements to six existing collaborative writing tools and discovered that the tools generally support many of the requirements, but none support all of them. They determine that improvements are still possible. Posner and Baecker summarize their findings by emphasizing the importance of flexible tools that support various writing strategies with minimal distractions.

Observations The paper by Posner and Baecker [15] performs an analysis that is very close to that of Kate Ehrlich [7]. This includes observing real-world usage for a certain type of work and deriving a set of specific requirements for it that inform application design. It does however not observe the actual usage, since Posner and Baecker rely solely on interviews[9]. An important issue touched on by Posner and Baecker is the trust between participants. This is something that is very hard to affect through the tools used. An application isn't easily able to make a collaborator more trustworthy to the user, at least not directly. They proposed increasing trust by enabling more familiarity between remote collaborators through the use of "video and audio connections" [15, p. 3].

Another issue mentioned was that communication bandwidth can cause delays when transferring documents. Nowadays, this is no longer an issue in most cases; if internet connectivity is available, it is generally fast enough to transfer text-based documents quickly[10]. If the collaborators are in close proximity to begin with, a face-to-face meeting can indeed be an option, although that depends on the *activity*, *document control method* and *writing strategy* used.

The *writing strategies* used often also determine the work mode. The *scribe* strategy is inherently synchronous (since it's used during meetings), whereas the *single writer* strategy is asynchronous (where notes are transformed into a document later on). Although Posner and Baecker never explicitly discussed work modes, their taxonomy was later refined and extended by Lowry et al. to include work modes, which describe the proximity and synchronicity of collaboration [17]. Posner and Baecker's recommendation of building flexible tools doesn't appear to fit well with making contributor roles explicit, and modern collaboration tools consequently usually don't enforce roles.

2.1.6. Modern Collaborative Writing

Overview Dakuo Wang has published a dissertation [18] along with an overview of it [19] in 2016. Around 25 years after Posner and Baecker released their paper, they once again performed intensive research into collaborative writing to get an up-to-date

[9] Which are opinionated, not objective, and possibly flawed, as shown by Suchman [16].

[10] Even if not, or if no connectivity is available at all, this is usually solved by finding a better-connected workspace nearby instead of decreasing the physical distance between collaborators.

picture of how the technology, the people using it, and collaborative writing in general have changed. Wang used a mix of qualitative and big data quantitative research methods, analyzing both what people think about collaborative writing and how they actually write. They interviewed 30 participants with different occupations for the qualitative part, and analyzed collaboration traces and logs recorded by *Google Docs*[11] to build two visualization tools that help interpret collaboration semantics based on this data.

Olson et al. [20] published part of the research outlined by Wang [19] in 2017. They based their research on previous work, mainly by Posner and Baecker [15] and Lowry et al. [17]. They focused on investigating collaborative writing based on actual usage data instead of reports by the users themselves. To do this they analyzed collaboration traces from a total of 96 collaborative documents created by groups of undergraduate students for a course assignment. They analyze which quantitative measures (like duration of work, distance of last edit from the final deadline) have an impact on different measures of quality. They also performed a qualitative analysis of the documents contents and their evolution throughout the writing process.

Findings include that students tend to make use of modern synchronous collaboration tooling to work on the same document in sessions of simultaneous work. The vast majority of documents were created using a mix of synchronous and asynchronous work modes. The documents created with balanced contribution by all collaborators generally achieved better quality, which Olson et al. attribute to a diverse group constellation that involves reading, commenting, and editing others' work. Tending to work synchronously was associated with finishing the assignment earlier, but also with spending more time working on the document in total. They discovered that the roles filled by collaborators during the writing were not static but changed fluently over time. The document itself was used beyond just the writing of text, with additional content being added and later removed during the life cycle of the document, ranging from example documents over comments and highlights to humorous images and notes. This additional content was not part of the final submission of any group.

Olson et al. conclude their report by highlighting the need for additional research to help validate their findings and outlining suggestions for future application design. These include building flexible and unopinionated tools that are easy to understand and use, considering how changes from collaborators can be signaled to users without interrupting them, and encapsulating the document editor with additional functionality that helps support the collaboration through metadata and references (like version control). Finally, they also touch on their expectations for the future, where they believe that "collaborative creation" will start moving beyond document editing and include "a variety of digital objects" [20, p. 32].

Observations The work by Wang [18, 19] and Olson et al. [20] is a thorough analysis of collaborative writing that draws parallels to research done around 25 years ago. This gives numerous insights into how existing research has been applied in modern applications, how well these applications are received, and what kind of behavior results from their usage.

Google Docs is a popular collaborative writing application even today, but has had some changes since Olson et al.'s publication. Notably the integration of voice and video calls

[11] `https://docs.google.com/`

within the app, a quick way to see the latest edits to a document, and recently introduced "document tabs" for notes related to the document but not part of the final version (like meeting notes and task trackers). This fulfills suggestions made by Olson et al. [20] as well as Posner and Baecker [15], at least to some extent. One thing that stands out in Olson et al.'s work is how in all cases of collaboration they observed, the collaborators had *shared* control, as defined by Posner and Baecker [15]. This is in fact the default behavior of *Google Docs*, where newly added collaborators will be an *editor* by default. This stands in stark contrast to what Posner and Baecker originally observed in 1992, where *shared* control was the least employed document control method [15]. It seems like the earlier lack of *shared* control methods was mainly due to technological constraints instead of user preferences. Olson et al. label the ability to work together in real-time as "a distinct advantage" [20, p. 28].

Olson et al. also confirm Posner and Baecker's impression that trust has a large impact on collaborative writing. It affects the tendency of collaborators to directly perform edits versus adding comments and suggestions. This can also have an effect on the work modes used, since leaving comments for another collaborator to address later on is inherently asynchronous, while directly editing the text can be done synchronously [21, p. 61]. Another observation related to trust and direct editing is people's belief that having work reviewed by other people increased its quality, but having others directly edit that work "felt intrusive, and lowered their sense of ownership" [20, p. 4]. This could further support the benefit of an independent approach to writing, where each collaborator mostly works on a separate section. Completely rejecting direct editing instead could result in a switch to a more asynchronous work mode, which can cause notable slow-downs for short collaborations [22]. It's worth noting that Olson et al. observed no commenting at all in 57% of the 96 documents. Of those who added comments, many did not use the built-in commenting features of *Google Docs*, but Olson et al. could not observe any detrimental effect of this. This further speaks to the importance of flexibility of collaborative tools, allowing users to easily work around minor issues without explicit support from the application.

Lastly, Olson et al. stated that students mentioned how working face-to-face was easier than working remotely. It seems like this face-to-face work was accompanied by a synchronous session in *Google Docs*, although it could also have involved the *scribe* writing strategy, for example. They mentioned that a lack of metadata and supporting tools in *Google Docs* could have had an influence on this, preventing students from easily discussing their work.

2.1.7. Local-First Software

Overview In their article on "Local-First Software", Kleppmann et al. describe their vision for the future of software [1]. They introduce a new type of application that tries to combine the best parts of cloud-based applications (like ease of collaboration and effortless usage across multiple devices) with those of fully-local (or "old-fashioned") applications (fast, long-lived, privacy & security, control of one's own data, and no dependence on network connections). They recognize the strengths and weaknesses of both kinds of apps, and propose a hybrid approach. These local-first apps are built from the ground up with a focus on the local device but the capability to make use of the network (or other means of data exchange) beyond that, enabling easy collaboration and synchronization where possible. The core principle here is that all application logic and state are kept

on-device, and updates to the state are exchanged with collaborators and integrated locally, eventually leading to a consistent state across all involved devices.

Kleppmann et al. were motivated by the problematic nature of cloud apps, which require a central server outside of the user's control to function, meaning that if the server stops working, the cloud app also stops working, and any user-created data could be lost or inaccessible. This dependence on a "cloud", combined with their significant latency and lack of privacy and ownership, is what local-first apps are meant to avoid, ideally without sacrificing any of the benefits of cloud apps. To define what a local-first app should be, they present "seven ideals for local-first software" [1, p. 2]. These include "no spinners", "your work is not trapped on your device", "the network is optional", "seamless collaboration with your colleagues", "the long now", "security and privacy by default", and "you retain ultimate ownership and control" [1, pp. 2–8]. These can be summarized as fast, decentralized apps with locally-stored data that can be transferred to other devices, which can keep working indefinitely as long as it's possible to execute them. The authors also analyzed existing app architectures for their adherence to these principles and found that while there are some apps that cover many of these ideals, none of them cover all.

Kleppmann et al. see CRDTs as an enabling technology for building applications that can cover all ideals. CRDTs are "multi-user from the ground up" [1, p. 14] and replace traditional data types in collaborative applications. They originated from distributed systems research and work by offering atomic, transformable, and consistent updates that can be exchanged between two CRDTs of the same type, leading to a consistent state that includes the combined result of all updates. This makes them a fitting choice both for synchronizing state between collaborators and, in many cases, merging simultaneous edits to a shared data structure without leading to a conflict. As a proof of concept, Kleppmann et al. built three different prototype apps using CRDTs. These apps are meant to resemble existing cloud apps, but use a local-first architecture. They generally found CRDTs to be a viable technology for these apps, although some unsolved problems remained. The merging of changes between users worked well and explicit conflicts due to concurrent changes were rare. Additionally, they considered the UX of these apps to be "splendid" [1, p. 18], especially the ability to continue working offline without major restrictions. Finally, they also recognized that cloud-based servers are a useful addition to local-first apps since they can facilitate data exchange, store backups, or provide computing services beyond the capabilities of the local device. Kleppmann et al. also note that there are certain types of apps that don't work offline and are therefore not suitable to a local-first approach.

They conclude their article by outlining future work, where they mention offering more control over data exchange and versioning, as well as investigating new user interface (UI) design approaches that can support the semantics of local-first apps. They also touch on building tooling and solutions for networking peers, facilitating data sync and backup, and trying to bring existing apps as close to the local-first ideals as possible.

Observations While Kleppmann et al. contrast cloud apps with "old-fashioned" software [1, p. 2], stating that the latter doesn't support seamless collaboration, this chapter made it clear that collaborative applications have been around for over three decades. In fact, the rise of groupware application started even before the internet was ubiquitously used, let alone the world-wide web. As such, decentralized collaborative apps synchronizing over the local network are not a new idea. When the internet became more popular and easier to make use of, these groupware applications started employing new ways

of connecting peers, given that local network discovery wasn't a suitable approach for discovering peers across the internet[12]. Having central servers for bootstrapping became a necessity, and at some point, these servers became the main way of synchronizing. So in a way, today's cloud apps started out as decentralized collaborative groupware apps[13].

But when Kleppmann et al. published their article, there were very few popular decentralized apps around. *Git* was popular, but didn't offer seamless synchronous collaboration. Cloud apps were dominant, in which the "user's local software [...] is a thin client" [1, p. 9], and almost nothing is processed locally[14]. In local-first apps, this is different. Here, the cloud servers act as buffers ("cloud peers" [1, p. 20]) that help peers to synchronize their state.

Kleppmann et al. have an interesting idea related to branching and supporting different versions in local-first apps. They included a *Git*-inspired tool for branching and merging in one of their prototype apps. They also note that "users must have the freedom to reject edits made by another collaborator, or to make private changes to a version of the document that is not shared with others" [1, p. 20], and comment on arriving changes from collaborators potentially distracting or annoying users. This was also noted by previous works [11, 15]. Avoiding distractions by either temporarily excluding changes from collaborators or by switching to an independent version seem like valid approaches. Using separate versions as a means to deal with conflicts has also been proposed recently [24] and will be discussed in the next paragraph. Another thing Kleppmann et al. discovered that has previously been confirmed through additional research [25] is that users tend to avoid creating conflicts with their collaborators. D'Angelo et al. called the type of editing close together which can lead to these conflicts "spacetime collaboration" [25, p. 2]. In the cases where such conflicts appear, Kleppmann et al. suggest choosing an arbitrary version as a solution.

Finally, local-first apps could theoretically have much better performance than cloud apps because they store all their data on-device and can use short local network links with low traffic and high bandwidth for synchronization[15]. But Kleppmann et al. discovered issues with performance and resource usage in their prototypes, mostly due to the change history kept by their CRDTs[16]. This could potentially be worked around using state-based synchronization [29] or using some form of garbage collection [30].

Besides local-first software, some authors of the original paper by Kleppmann et al. [1] have recently proposed what they call "Local-first Cooperation"[17]. This seems to have

[12] This refers to methods like port scanning or User Datagram Protocol (UDP) broadcasts, which generally only work within the local area network (LAN).

[13] Like in the case of Lotus Notes, which evolved from using replicated storage across "rarely-connected networks" [23] to a client-server application.

[14] This is also described by Johansen as "the personal computer [...] becoming the interpersonal computer" [10, p. 1]. This reflects how cloud apps, specifically centralized collaboration tools, work nowadays. The users' personal computer has become a terminal to accessing the interpersonal computer - the server in the cloud. Hardly any work is really done on the local computer, it's just used as an input device. Even when offline, any changes made are only buffered, to be sent to the server later on. The interpersonal computer connects these terminals and serves as the single source of truth.

[15] Round-trip times (RTTs) on the internet range from tens to hundreds of milliseconds (depending on the server location) since "No matter how hard you push and no matter what the priority, you can't increase the speed of light" [26], and the perceived thresholds for instantaneous interaction are somewhere around 100 ms to 400 ms [27, 28].

[16] Another real-world example for this is AFFiNE (`https://affine.pro/`, v0.18.1), which frequently shows loading spinners or is slow to respond even for non-collaborative work using a local workspace.

[17] `https://www.local-first-cooperation.org`, visited on 2024-12-18

a more general scope, aiming to apply local-first principles to all kinds of distributed computing. They describe five principles, as opposed to the original seven proposed in 2019, that are about fault-tolerance, autonomy, and representing state, connectivity, and changes. For DCTs the original ideals still seem more relevant, although communicating the connectivity status and helping users reason about the age of the data they're working with as well as the effects of their changes is certainly an important part of the UX.

2.1.8. Supporting Inconsistency, Non-Linearity, and Branching

Overview Schiefer et al. propose an alternative approach to handling updates from collaborators in local-first applications [24]. They propose a solution that postpones conflict resolution to a later time by splitting into parallel histories for each conflicting version. They coin this approach "forking histories" [24, p. 2] and argue that it is specifically useful in what they call "digital gardens" [24, p. 1]. To them, a digital garden is an application where users manage a significant amount of data that grows over extended time spans, where preserving user intent is paramount. They describe four properties of digital gardens: "accretion over time", "unbounded scope", "medium size", and "immense value" [24, p. 3]. Their issue with using CRDT-based approaches for these digital gardens is that CRDTs try to resolve conflicts as soon as possible, either by applying automatic merge semantics or by prompting the user. They argue that digital gardens need a wider scope for conflicts (i.e., transactions) in order to best preserve user intent, which would lead to an excessive amount of conflicts that need to be handled.

With their forking histories approach they avoid frequent conflicts by allowing multiple histories, which can continue to be edited independently after a conflict occurred. Each history is free of conflicts in itself, but does conflict with parallel histories. The user can keep working on any history, until they are ready to reconcile the histories by resolving any conflicts between them. They describe this behavior as "deferred conflict resolution" [24, p. 4]. As a concrete instance of this they propose *TreeDB*, which they describe as a "meta-CRDT" [24, p. 7] that manages histories instead of application data. *TreeDB* uses events and change sets to detect conflicts, and if there is no order in which events can be applied to avoid a conflict, the history forks and the conflicting events are applied to different conflict-free histories. This means that the current state of an application is the reduction of all events that are part of the current history, even though other histories with additional events might exist. Histories can therefore be considered as branches or versions that can be switched between. The authors highlight this by stating that "events can be available without having been applied" [24, p. 5]. By modifying the change set, certain behavior like transactional changes can be implemented. Schiefer et al. are even able to emulate simple CRDTs or force a fork in the history using *TreeDB*. In summary, Schiefer et al.'s approach to local-first collaborative digital gardens is to "merge what you can, fork what you can't" [24, p. 4].

Observations Schiefer et al.'s paper [24] builds on top of Kleppmann et al.'s work [1], while showing the limitations of custom merge semantics in CRDTs for certain types of applications. They mention that CRDT-based application design might "pretend that data can always be automatically merged without issues" [24, p. 6], but argue that it's important to recognize that sometimes non-trivial conflicts happen that should be resolved manually by a user. This is different from Kleppmann et al.'s resolution approach of arbitrarily picking one version over the other [1, p. 19]. It is however important to note that Kleppmann et al.'s three prototypes should not be classified as *digital gardens*,

although *PushPin* might come close. Additionally, McKelvey et al. later discovered that "automatic merging is necessary but not sufficient" [31].

An interesting insight by Schiefer et al. is their take on eventual consistency, which emphasizes the property of eventuality. They argue that in digital gardens it is of little concern if different users' views of the data stays inconsistent for extended periods. It is even encouraged, giving users time to finish up their work before resolving conflicts and merging into a consistent state. It becomes obvious that the time span in which data can be allowed to be inconsistent heavily depends on the type of application. This is evident from workflow patterns in *Git*, where fetching changes from collaborators but then manually integrating them into one's own branch of work is common. Depending on the size of the change, users might spend weeks working on a separate version before integrating external changes. Notably Schiefer et al.'s take on eventual consistency still qualifies as *strong eventual consistency* [32], just not for the state of the application itself, but for the state of the history tree.

The versioning found in *Git* is also an idea that transfers well to local-first applications and *digital gardens*. Schiefer et al. dealt with this concept and explored methods of forcing forks in the history in absence of a conflict [24, p. 8], and McKelvey et al. investigated this with *Upwelling* [31]. A need for additional research into UIs supporting multiple versions and comparing changes was highlighted by both Schiefer et al. and McKelvey et al. [24, 31]. Schiefer et al. also note how *Git* uses static, plaintext-focused conflict semantics, while *TreeDB* supports application-specific conflict semantics. This is notable because it allows *TreeDB* to deal with other types of data. *Git*'s lack of such flexibility leads to most non-text files being treated as binary blobs [1, p. 10].

One potential problem that remains with Schiefer et al.'s approach to digital gardens is that broad dependencies (conflict sets) combined with transactional changes will make conflicting edits more likely. The longer users keep working on a separate version, the higher the chances of collaborators introducing additional conflicts. At some point, resolving all of those conflicts could become tedious and unattractive to users, resulting in the different versions diverging more and more, or in a loss of productivity [33, p. 47]. This is definitely not the goal of collaboration, and it should be investigated how this can be avoided.

2.1.9. Dealing with Synchronous and Asynchronous Conflicts

Overview Hoai Le Nguyen investigated conflicts in collaborative editing in their dissertation from 2021 [33]. They considered both asynchronous and synchronous collaboration, but from different perspectives. For asynchronous collaboration, they analyzed several large *Git* repositories for merge conflicts and how those were handled. For synchronous collaboration, they analyzed logs of a ShareLaTeX[18] server to determine under which conditions conflicts appeared.

Their results show that while conflicts in *Git* are generally common, there are certain types of conflicts (adjacent-line conflicts) that are often false-positives, leading to users resolving them by simply applying both changes. They suggest that *Git* should by default not consider adjacent-line conflicts as true conflicts, but instead just warn the user if those occur. Nguyen also touched on the topic of semantic conflicts that can appear when during

[18] `https://www.sharelatex.com/`

a merge no conflict is detected by the collaboration tool, but the resulting state of the document is semantically incorrect.

The collaboration logs from ShareLaTeX reveal that there are two types of conflicting edits (*border conflicts* and *insertion conflicts*), but both are generally rare (around 8.5% and 2% of edits for the lowest considered threshold). They comment that edits can generally be grouped into clusters, and within a cluster, each author usually edits in a continuous fashion, with clear borders between the edits of two authors. However, they do note that there are exceptions where two authors edit close together both in space and time, and that those are the locations of potential conflicts, which in most cases lead to actual conflicts. Their suggestion for avoiding these conflicts is to add awareness mechanisms into the editor that warn users if they're editing close together and risking conflicts. Additionally, Nguyen highlights that synchronous collaboration in C̣Ẹ leads to more editing being done more quickly and over longer durations. They summarize this by stating that "people are more 'productive' in coauthored sessions than in single-authored sessions" [33, p. 98].

Observations Hoai Le Nguyen gave an overview of the differences in asynchronous and synchronous conflicts [33]. They acknowledged that these two types of collaboration can have very different semantics, user behavior, and design considerations, and aren't easily compared. However, there clearly is a connection between synchronous and asynchronous collaboration, where an asynchronous application can often be made more synchronous by increasing the frequency of change synchronization between collaborators, or a synchronous app being used by one user at a time.

A compelling observation they made is that semantic conflicts are often hard to detect [33, 34]. Sometimes things like automatic tests can help discover them, but tests can be flawed or incomplete, making both false positives and false negatives likely. Semantic conflicts are also essentially impossible to resolve in an automated fashion [34, p. 452]. Probable solutions can sometimes be determined based on the context, but choosing the right one is up to the user. An automated fix that chooses a wrong solution could have severe implications. Semantic conflicts are nonetheless common when merging changes by multiple collaborators [34].

Regarding synchronous collaboration, Nguyen's results confirm previous observations by Olson et al., D'Angelo et al., Larsen-Ledet & Korsgaard, and Kleppmann et al. [1, 20, 25, 35] with regard to the occurrence of conflicts due to user behavior. Users tend to work on separate sections of the document when possible, therefore rarely causing any conflicts at all. Nguyen's results do however highlight an intriguing fact: when starting to write a new document from scratch, the document is short, limiting the maximum distance between edits from different users. This leads to an increased number of conflicts early on. This observation could additionally be explained by young documents still being rough and subject to changes, where different writing activities and processes like brainstorming or outlining might be happening [20].

Nguyen additionally touched on the topic of awareness, which has been a long-standing research topic [7, 9, 11, 17, 18, 20, 35–38]. Making users aware of their collaborators' location and activities helps to avoid conflicts. The issue with Nguyen's suggestion of an awareness mechanism is that conflicts are more common for asynchronous work modes, but many awareness mechanisms are not easily applicable to them[19].

[19]There are proposals for conflict-avoiding awareness mechanisms in asynchronous-independent applications [39], but those are rarely seen in modern apps. Building live and always up-to-date awareness mechanisms for the offline work mode is not possible at all.

2.1.10. Conclusion

There is a long history of research on digital collaboration and tools for it. Many different names describe roughly the same topic, although with slightly different semantics. Collaboration is generally considered useful, offering better efficiency, quality, and transparency than working alone. Tools have become more synchronous and centralized over time, which comes both with benefits (seamless synchronization, consistent state, access from everywhere, good awareness) and drawbacks (cloud dependency, lack of ownership and privacy, slower performance, less manual control). Conflicts between collaborative edits can arise in any work mode (synchronous and asynchronous), but are not always automatically detectable or fixable. Users' behavior, preferences, and social environment always play a significant role for collaboration and the design of collaboration tools. These tools should be flexible, allowing users to work in a way that's most comfortable for them instead of being forced to work in any specific way. Recently, researchers have started to explore alternative ways of building DCTs that offer more diverse work modes and fewer drawbacks in general. But older research is still valid and applicable for these new applications, even if there are some new problems to solve. Additionally, researchers have theorized and shown interest in collaboration tools that support more complex digital data beyond just text, and tooling for this is starting to emerge.

2.2. Existing Taxonomy

In this section existing definitions and taxonomies related to collaboration are discussed. *Collaboration*, *digital collaboration tools*, and *work modes* are dealt with in particular. This is done to have a good foundation for discussing the definitions used in this thesis later on.

Collaboration Ellis et al. provide several definitions of collaboration in their article from 1991 [9]. They define it as "user-to-user interaction" [9, p. 3] which requires the sharing of information [9, p. 3], but also as working on a shared environment [9, p. 3]. Herskovic et al. share that latter definition [40, pp. 191–192]. Olson et al. additionally highlight that collaboration involves conversations [20, p. 17]. Edwards et al. note that "collaboration involves sharing: the sharing of data, artifacts, context, and ultimately ideas" [11, p. 119]. Xu et al. are more concerned with the work itself, stating that "collaboration refers to all processes where people work together to achieve results" [12, p. 2]. Similarly, Dakuo Wang adopts the definition by Dillon in 1993 in that "any document production activities that involve more than one person is collaborative writing" [18, p. 48], which can be understood in such a way that collaboration is the process of working together with other people, which is also supported by Ignat et al., Nguyen, and Kainberger [33, 39, 41]. However, Kainberger restricts this definition to "edit[ing] a shared document [...] simultaneously and in real-time" as well as "ensur[ing] that the document opened and edited by different users remains synchronized"[41, p. 1]. Kleppmann et al. seem to think of collaboration as a combination of working together, communicating, and coordinating [1, p. 1]. They also mention that it "typically requires that several people contribute material to a document or file" [1, p. 4].

Digital Collaboration Tools Many of the existing literature deals with *groupware* as the software used for collaboration. Since that term isn't popular anymore and related

terms like CSCW aren't suited to describing the applications themselves, this thesis refers to them as *digital collaboration tools* instead. Ellis et al. again offer several definitions here: from "an organization wide system that integrates information processing and communication activities" [9, p. 2], over "the class of applications, for small groups and for organizations, arising from the merging of computers and large information bases and communications technology" [9, p. 2], to "computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment" [9, p. 3]. In short, they describe it as technology that helps groups of people work. This is very similar to the definition by Kate Ehrlich in 1999 who describes that DCTs "provide computer support for group work" [7, p. 21]. More recently, DCTs are often referred to as being *multiplayer*, for example by engineers at Figma and GitHub [42, 43]. Nextcloud interestingly differentiates between groupware and other collaboration apps, where *Nextcloud Groupware*[20] includes a calendar, contacts, and mail application, as well as a Kanban-style project management tool, but leaves out their collaborative *Office* suite. Kleppmann et al. don't give an immediate definition of collaboration apps, but highlight the importance of good connectivity to their usefulness [1, pp. 1–2, 9].

Work Modes Posner and Baecker include a number of categories in their taxonomy from 1992 [15]. These include *roles*, *activities*, *document control*, and *writing strategies*. They also touch on the fact that collaboration happen in face-to-face settings or remotely, as well as synchronously or asynchronously. However, only Lowry et al. actually consider this in their extended taxonomy in the form of *work modes* [17, pp. 87, 89]. The terms *synchronous*, *asynchronous*, and *distributed* are mainly used to describe them. These are arranged in a 2×2 matrix by Lowry et al., Ellis et al., and Johanson to categorize both the spatial and temporal aspect of collaboration [9, 10, 17]. Ellis et al. mainly differentiate between *real-time* and *non-real-time* groupware, where the former includes both synchronous and asynchronous usage, while the latter only includes asynchronous usage. In contrast, Edwards et al. note that asynchronous collaboration "needn't necessarily happen at the same time" [11, p. 120], thereby including synchronous functionality in their definition. They also mention that "asynchronous collaboration is characterized by the degree of independence collaborators have from one another" [11, p. 1]. In contrast, synchronous applications are those "in which users share some 'thing' at the same time" [11, p. 1]. Kate Ehrlich describes synchronous working as "people interact[ing] with each other at the same time" and asynchronous working as "work[ing] together at different times" or "over a period of time" [7, p. 21]. Xu et al. also describe asynchronous collaboration tools as "allow[ing] users to collaborate at different times" [12, p. 1].

Herskovic et al. divide the synchronous work mode into two levels: a weaker notion of being "reachable", which constitutes the ability to exchange information and receive a response in a certain period of time, and a stronger notion of being "simultaneously present", or being "available to work synchronously" [40]. Their definition of synchronous also includes parallel working. The Wikipedia page for "AJAX"[21] defines asynchronous as "in the background", which can be seen as happening without the user being actively aware of it. Stack Overflow uses a distinct definition for their 2024 developer survey. Asynchronous tools are described as "collaborative work management and/or code documentation tools", while synchronous tools are simply "communication tools" [44]. More recently, McKelvey et al. describe asynchronous collaboration as sharing batches of

[20] `https://nextcloud.com/groupware/`, visited on 2024-11-21
[21] `https://en.wikipedia.org/wiki/Ajax_(programming)`, visited on 2024-11-21

changes instead of fine-grained real-time updates [31], giving a definition of synchronous collaboration at the same time. Lee and Paine suggest a revised version of the 2x2 collaboration matrix, called the "Model of Coordinated Action (MoCA)", which features 7 distinct dimensions: *synchronicity*, *physical distribution*, *scale*, *number of communities of practice*, *nascence*, *planned permanence*, and *turnover* [45]. Finally, Dourish uses the term "multi-synchronous" to refer to a collaboration where multiple people work independently on their own version and later merge their changes [37, p. 4].

2.3. Refined Taxonomy

As a counterpart to Section 2.2, the definitions for *collaboration*, *digital collaboration tools*, and *work modes* used in this thesis are given in this section.

Collaboration In contrast to some of the definitions from prior work, this thesis doesn't restrict collaboration to specific scenarios or activities. Instead, collaboration is seen as sharing a workload with others for a (perceived) benefit. The first part of this definition deals with the need to involve multiple people in the collaboration process, while the second part is about giving a reasoning for the collaboration to happen. A person initiating a collaboration wants to make the resulting artifact better or more efficient to obtain than when working alone. This definition also ensures that use cases like writing an article for others to read is not considered collaboration, in line with the position of Kate Ehrlich [7, p. 22].

Digital Collaboration Tools This thesis defines DCTs as interconnected applications that enable working together on digital artifacts in shared environments. The main focus of this thesis are *collaborative editing (CE) tools*, a subcategory of DCTs, where users are able to edit (modify) previously-created artifacts (by other users or by themselves). Purely-synchronous meeting apps like Zoom[22] are also considered DCTs, even though those are not specifically dealt with as part of the thesis, due to them being very narrow in scope.

Work Modes In the context of this thesis an alternative set of work modes is proposed. The *synchronous* and *asynchronous* work modes are kept, but their meaning is slightly changed. Instead of *synchronous* meaning that the collaboration happens at the same time, this thesis defines it as collaborators being immediately aware of their collaborators activities and any changes they made. The temporal aspect is still kept, because the definition considers the delay between a collaborator making a change and the other collaborators being able to notice it. Similarly, *asynchronous* means that there is a significant delay before collaborators notice changes, either because they are not working at the same time, or because they are not able to exchange updates with each other. This can happen either sequentially (collaborators taking turns) or concurrently (working in parallel, but on different versions or without seeing each other's changes). Differentiating based on physical proximity seems like an ill fit for the purposes of this thesis, since there seem to be few collaboration apps that specifically support face-to-face meetings.

[22] `https://www.zoom.com/`

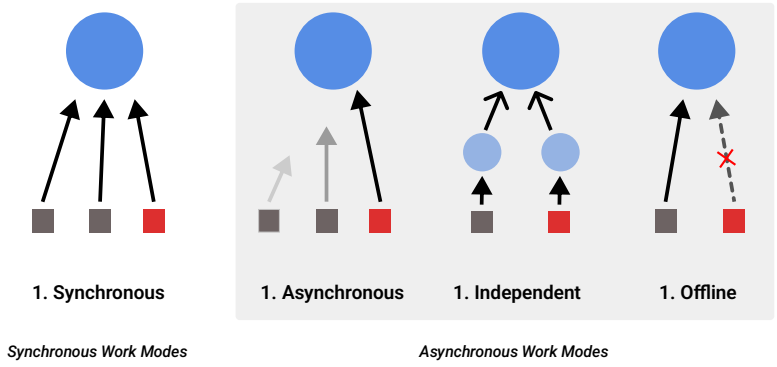


Figure 1.: The four main work modes considered in this thesis, visualized.
The red square marks the user's client. The blue circle marks the shared environment. Arrows from client to shared environment indicate an editing session. Arrows that aren't fully extended represent sessions in the future. Thin arrow heads represent synchronization instead of editing.

Instead, two new work modes are introduced: *independent* and *offline*. The *offline* work mode describes a state of temporary disconnectedness, where collaborators are working without being able to exchange updates for extended periods of time. This work mode is therefore inherently *asynchronous*. The *independent* work mode similarly is about a lack of integration of updates from collaborators, but not because of missing connectivity. Instead, working independently means working on a separate, isolated version of a digital artifact that is modified either by no or only some additional collaborators. All changes to the isolated version of the digital artifact are then added to its default version at a later point. Notably, the user has to manually decide when to merge the changes.

It is theoretically possible to support a *synchronous-independent* work mode, which allows working on independent versions while still being connected to other collaborators and immediately being aware of any changes made by them. However, this has not been observed in any DCT so far. The term "multi-synchronous", as proposed by Dourish in 1995 [37, p. 4], is not considered a work mode here. It mainly describes the case of concurrent editing during *asynchronous* usage, and doesn't match the definition of *synchronous* used in this thesis. In conclusion, we deal with four different work modes as part of this thesis: *synchronous*, *asynchronous*, (asynchronous-)*independent* and (asynchronous-)*offline*. These are visualized in Figure 1.

Work Modes - An Example To help understand which work modes can be considered supported by various applications, a few examples are given in the following. *Google Docs* is an app for collaborative writing. It supports sharing a document with multiple users for reading, editing, and reviewing. Users always work on the same document, accessing it either at the same time and seeing any changes made by their collaborators immediately, or using it at different times. This means that both the *synchronous* and *asynchronous* work modes are supported. There are awareness mechanisms for both modes, for synchronous in the form of telepointers that show other users' cursors within the document, and for asynchronous in the form of a change history. The *independent* work mode is not supported, since users always interact with the same document and cannot work on a separate version independently before merging their changes into the

default version[23]. The *offline* work mode is mostly supported, since it is possible to work on a document while being disconnected for extended periods. However, this is not possible by default. Special programs or add-ons are needed, and the user needs to explicitly prepare the document for offline work.

Git is a tool for both version controlling files and collaborating on source code with others. In *Git*, changes are made locally to a repository on the user's device in the form of commits, and later transferred to other repositories. Exchanging these commits happens deliberately and not in real time. This makes *Git* an *asynchronous* tool. It is also possible to create independent versions of a repository in the form of branches, which can diverge from the default version but can be merged back into the default version if desired. Through this, *Git* supports the *independent* work mode. In fact, since *Git* is decentralized and each repository is a separate replica, any changes made are *always* made to an independent version. But when exchanging commits between the default versions of these replicas, they are usually automatically applied and not kept independently. *Git* also supports the *offline* work mode by default; connectivity is only needed for exchanging commits with collaborators, but not during the actual editing.

[23]This is of course possible by creating a copy of the document and then manually transferring changes to the original document, but the application doesn't actively support this in any way.

3. Characterizing Digital Collaboration Tools

This chapter presents an analysis of currently available collaboration tools. These tools have been analyzed for their supported work modes, the general type of app, as well as related features and behavior. An overview is presented, and discovered patterns are discussed.

3.1. Analysis of Modern Collaboration Tools

In order to get a good understanding of the current landscape of DCTs, 45 different apps were analyzed for their support of the four main work modes discussed in this thesis. The analysis was focused on applications that supported *editing* work done by other people, so this analysis excluded meeting apps[24]. A number of *invariants* of collaboration tools were distilled based on the discovered properties of these tools as well as the existing literature discussed in Chapter 2. The results, including the type of app, supported work modes, and additional notes, are shown in Table 1. The table is sorted by type of app, with five main categories: *Version Control*, *Collaborative Editing*, *Project Management*, *Notes & Knowledge Bases*, and *File Sharing*.

When looking at Table 1 it immediately becomes apparent that apps from the same category have very similar support for work modes. Version control apps don't usually support synchronous work. Project management apps are largely usable both asynchronously and synchronously, but require constant connectivity. A few project management apps offer mobile companions that additionally support offline usage. Collaborative editing apps always support asynchronous and synchronous work modes, and can usually support offline work if really needed. In contrast to project management apps, in the case of CE only the full apps offer offline support, while the companion apps do not. Apps for notes & knowledge bases have even better support for asynchronous and offline work, but are less focused on synchronous collaboration. Finally, file sharing apps don't support real-time exchange of files and are mostly used passively. Offline work is well-supported, but with some constraints. This *clustering* of work mode support based on application type can be more clearly observed in Figure 2.

[24] After initially including meeting apps, it was discovered that those generally don't have a shared environment containing digital artifacts, since they're all about communication and speech. Therefore, work modes like independent or offline principally can't be supported. While there are some asynchronous communication tools (like `https://www.loom.com/`), these shouldn't be considered collaboration tools according to Kate Ehrlich [7, p. 22].

Table 1.: Supported work modes in various collaboration apps

App	Sync	Async	(Async-) Indep.	Offline	Notes
Version Control					
Git	✗	✓	✓	✓	
Mercurial	✗	✓	✓	✓	
Darcs	✗	✓	✓	✓	
Pijul	✗	✓	✓	✓	
Subversion	✗	✓	✓	(✗)	Offline use restricts collaboration
Collaborative Editing					
Visual Studio Live Share	✓	(✗)	(✗)	(✗)	Asynchronous functionality through Git
JetBrains Code With Me	✓	(✗)	(✗)	(✗)	Asynchronous functionality through Git
Etherpad	✓	✓	✗	✗	
Grammarly	✓	✓	✗	✗	
Multicollab	✓	✓	✗	✗	
Mural	✓	✓	✗	✗	Awareness mechanisms can be disabled
Canva	✓	✓	✗	✗	
Nextcloud Office / Collabora	✓	✓	✗	✗	Nextcloud Office uses the Collabora editor suite internally
Kapwing	✓	✓	✗	✗	
Figma	✓	✓	(✓)	(✗)	Limited functionality while offline, only currently loaded files are accessible. Branching available on business plans.
FigJam	✓	✓	✗	(✗)	Only pre-loaded files accessible when offline
Figma Slides	✓	✓	✗	(✗)	Only pre-loaded files accessible when offline
Dropbox Paper	✓	✓	✗	(✗)	Offline use available for mobile (with pre-requisites)
ShareLaTeX / Overleaf	✓	✓	(✗)	(✗)	Supports independent and offline via a Git integration
GWDG Pad / HedgeDoc	✓	✓	✗	✗	
typst	✓	✓	(✗)	(✓)	Can't be launched offline. Git integration available
Google Docs/Sheets/Slides	✓	✓	✗	(✓)	Offline use has additional pre-requisites
Microsoft 365 Word/Excel/PowerPoint	✓	✓	✗	(✓)	Offline use not available for the web clients
Apple Pages/Numbers/Keynote	✓	✓	✗	✓	Awareness mechanisms can be disabled

App	Sync	Async	(Async-) Indep.	Offline	Notes
Zoom Whiteboard (new)	✓	✓	✗	✗	
DaVinci Resolve	(✓)	✓	(✓)	✗	Synchronous editing uses locking, changes need to be manually accepted
Siemens NX + Teamcenter	✗	(✓)	(✗)	✓	Asynchronous editing uses locking
Upwelling	✗	✓	✓	✓	
Project Management					
ClickUp	✓	✓	✗	✗	
Asana	✓	✓	✗	✗	
Salesforce	✓	✓	✗	✗	
Miro	✓	✓	✗	✗	
Attio	✓	✓	✗	✗	
Trello	(✓)	✓	✗	(✗)	No awareness mechanisms, offline use only for mobile apps
Jira	(✓)	✓	✗	✗	No synchronous awareness mechanisms
Airtable	(✓)	✓	✗	✗	No awareness mechanisms
Memento	✗	✓	✗	(✗)	Offline use available for mobile
Notes & Knowledge Bases					
AFFiNE	✓	✓	✗	✓	
Skiff	✓	✓	✗	✓	
AnyType	✗	✓	✗	✓	
Notion	✓	✓	✗	(✗)	Limited functionality while offline, only currently loaded files are accessible
Whimsical	✓	✓	✗	✗	Allows forking of files, but not merging
File Sharing					
Dropbox	✗	✓	✗	(✓)	Offline use only for desktop apps
Google Drive	✗	✓	✗	(✓)	Mobile offline use has additional pre-requisites
OneDrive	✗	✓	✗	(✓)	Mobile offline use has additional pre-requisites and is read-only

Notably, not a single analyzed app supported all four work modes. Some came close: *ShareLaTeX* & *Overleaf* enable additional asynchronous flexibility by offering a *Git* integration [46] that lets users work on their project as a separate copy even when offline, but the LaTeX editor frontend is not available in that scenario. Given that the additional asynchronous functionality is provided by a different app, the work mode support can't be fully attributed to *ShareLaTeX* in this case. But the idea of combining multiple separate approaches to create a whole is interesting. Another remarkable observation is that every single analyzed application supports an asynchronous work mode. This shows that generally only meeting apps are made exclusively for synchronous work modes, and essentially all other collaboration apps, where one user can edit another user's work, can be used in an asynchronous fashion. This was also touched on by Ellis et al. [9, pp. 5–6, 9]. It could also be that this is a trait of centralized apps, where the shared environment lives on a central server and is therefore always available, but further research would be needed to answer this. Strikingly the synchronous and independent work modes are almost never encountered together in a single app, especially not with full support for both modes. CRDT-based apps in general tend to offer the asynchronous-offline work mode, but require additional work to properly support synchronous usage. Apps using some form of OT algorithm are inverse of that, where they are well-optimized for keeping a consistent state in a synchronous setting, but require explicit engineering for offline support. Overall, hardly any app from the *collaborative editing* and *project management* categories has *good* support for the offline work mode. Some categories also have one or more *outliers*, i.e., apps that behave differently than the *normal* apps.

There are a few more interesting observations regarding the analyzed apps and their behavior. For one, there are some apps that support disabling the telepointers from other collaborators to avoid distractions. Apps that support this include *Mural* and *Apple Pages/Numbers/Keynote*. The benefit of avoiding distractions was previously noted by Posner and Baecker, Kate Ehrlich, Mark et al., and Kleppmann et al. [1, 7, 15, 38]. Disabling telepointers could be seen as an alternative to offering a separate working copy through the independent work mode, and is a form of user customization [7, p. 36]. Another set of apps employs locking mechanisms to control the collaboration. This includes *DaVinci Resolve*, Apache *Subversion*, *Multicollab*, and *Microsoft 365 Word*. They are generally used for the synchronous work mode to prevent collaborators from concurrently editing the same region of the data. These apps either don't support the offline work mode at all, or provide tooling for manually resolving conflicts. In the case of *Subversion* locks can be optionally enabled on a file-by-file basis and require permission from the server to be opened. With all these apps, the locks were added retroactively along with general support for collaboration.

Some apps also support explicitly handling conflicting changes. *Figma* [47], *Microsoft Word*[25], and essentially all *version control* apps do this. In all cases, this is used to handle larger divergences in state, like when working independently or offline. Apps that don't offer tooling for handling conflicts, like *Trello*'s mobile app, tend to rely on more drastic measures, like overwriting changes on a last-write-wins (LWW) basis. *Notion*'s lack of offline support[26] compared to most other apps of the *notes & knowledge bases* type is another interesting topic. The missing support for the offline work mode is a frequent point of discussion among *Notion* users, with many article explaining what's possible and what can be worked around [49–51]. Notion is now actively looking to resolve this

[25] Mentioned in the "Can I co-author offline?" section of a support page [48].

[26] Notion introduced improved offline functionality in August 2025, after this thesis was originally submitted

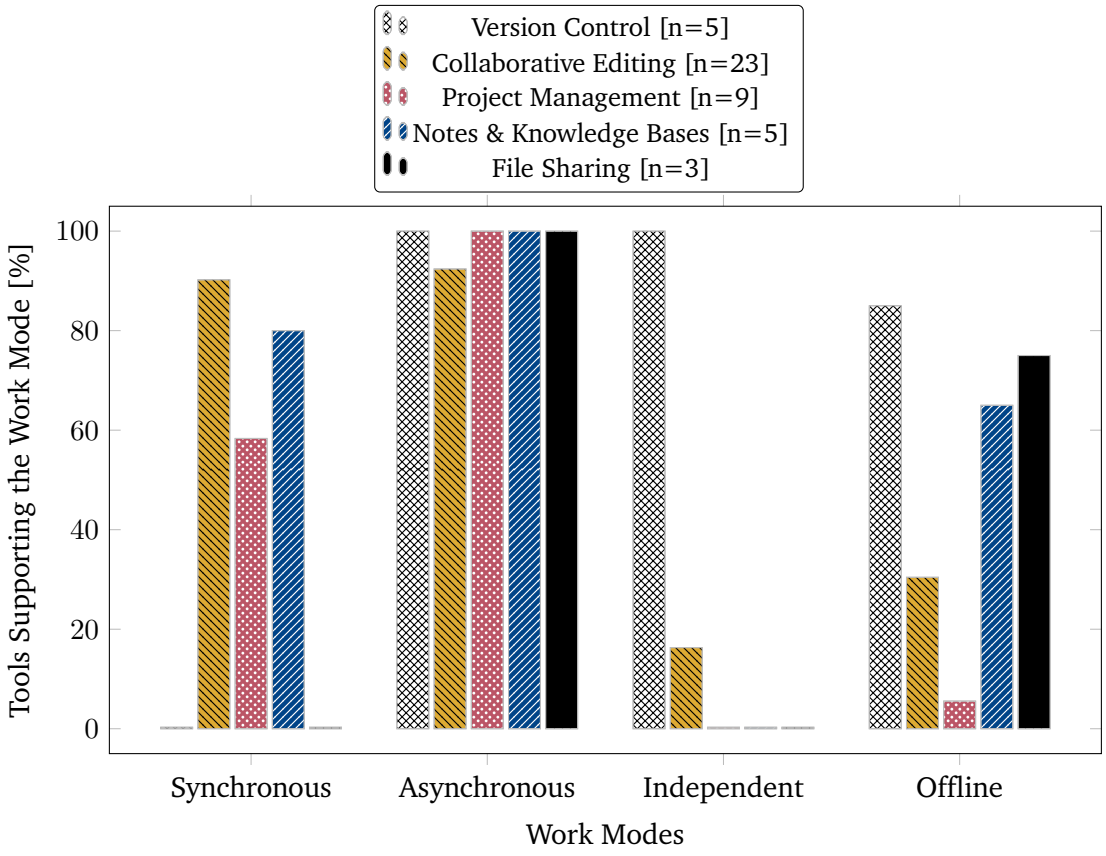


Figure 2.: The 45 analyzed apps laid out based on their supported work modes. Different types of apps support distinct sets of work modes.

issue, as mentioned in their *Made with Notion* conference in October 2024 [52], which is also evident by their recent acquisition of *Skiff*[27]. *Skiff*, a former competitor of *Notion* in the *notes & knowledge bases* category, has proper offline capabilities since it is based on CRDTs [53].

Similarly, there are apps that don't support a synchronous work mode by default, but where attempts are made to externally add support recently. One example for this is *Git*, which normally requires manually *fetching* changes from collaborators and remote repositories. Here some editors and integrated development environments (IDEs) step in by periodically and automatically fetching changes in the background[28,29]. They complement this through new awareness mechanisms like showing the user's latest commit in the context of the remote repository [54]. So while *Git* still is inherently asynchronous from a workflow perspective, tooling tries to add more synchronicity to it for improving awareness and collaboration, as well as avoiding conflicts.

[27] `https://web.archive.org/web/20241209010419/https://skiff.com/`, visited on 2024-12-09

[28] While this is disabled in Visual Studio Code by default, new installations prompt the user to enable it. The default interval is 3 minutes. On the web version, `https://vscode.dev`, the setting is enabled by default with an interval of 1 minute.

[29] The JetBrains family of IDEs support this via a popular plugin (`https://plugins.jetbrains.com/plugin/7499-gittoolbox`)

3.2. Invariants & Common Traits

There are some common traits or *invariants* that apply to DCTs. These become apparent when taking a step back from the details and specifics of the analyzed apps, and instead looking at the bigger picture, the similarities, and the differences in general.

i No *project management* app that was analyzed supported the independent work mode. This observation can be extended, establishing that *project management* tools are never suited to an independent work mode. This is because they are meant to provide a consistent view of the most up-to-date state of a project.

ii So-called *offline-first* apps are rare in the realm of collaboration apps, because collaboration is all about exchanging data with collaborators, which fundamentally requires some form of connectivity.

iii Collaborative editing apps are inherently asynchronous. Editing is the process of changing an existing artifact and is therefore sequential in nature.

iv Some types of apps can be considered asynchronous-only. Most notably these include collaborative calendars, which generally don't feature what Ellis et al. call "sessions" [9]. This partially extends to *project management* apps, although synchronous work modes become slightly more important for these.

v Following the observation that meeting apps are frequently synchronous-only by design, it can inversely be concluded that any collaboration tool that is meant to support meetings needs to offer a synchronous work mode [7, p. 21].

vi Awareness mechanisms are most useful and best applicable to synchronous work modes. While awareness mechanisms are undoubtedly useful in asynchronous settings as well, being aware of what your collaborators are doing becomes exponentially more useful the more up-to-date that information is.

vii The acceptable maximum time span for which the state within an app may be inconsistent depends on the type of app and the nature of the work being done.

3.3. The Anatomy of Digital Collaboration Tools

After the investigation of both literature and existing DCTs, the relevant parts of such tools can now be dissected and discussed. This should improve the understanding of how these tools work and how the different parts interact with and depend on each other.

A DCT is intended to involve more than one person. This is in line with the definition of collaboration, but is also an important boundary to apps for distributed personal working, where a single user synchronizes data between multiple personal devices. At the same time, it is generally important that a collaboration app is also usable by just a single user [55, p. 104, 15]. If an app can be used for work by a single user just as well as when actively collaborating, context switches during different type of collaboration can be reduced [55, p. 104]. This requirement applies less to meeting apps which only support *synchronous* work modes, but applies in all other cases. This is further supported by the findings on social dynamics by Posner and Baecker, Olson et al. and Wang et al. [15, 20, 56]. The requirement to support single users naturally extends to the fact that DCTs almost always support an *asynchronous* work mode, which is therefore considered the default. Working asynchronously is in essence nothing else than an artifact being edited multiple times

(usually sequentially), just that the editing user can potentially be a different person each time[30]. It is therefore important that tools are designed to explicitly support asynchronous collaboration as best as possible, and include awareness mechanisms where needed. But DCTs should still offer a synchronous work mode in addition to the default asynchronous one [56, p. 15, 21].

Collaboration is about creating and working on something together, and without a way to interact with others' work, there's little chance to actually contribute to that work. This means that the digital artifacts exist in some form of "shared environment" [9]. Most apps should further support interacting with other users' work. While feedback mechanisms like commenting or reacting can be helpful, a suggestion feature is much more suited to sharing the workload, since suggestions made by others can easily be accepted without additional overhead, and include attribution. Publishing or sharing documents with others in a read-only way also isn't sufficient, as explained by Ehrlich in 1999 [7, p. 22]. A DCT needs to have a built-in way of exchanging data and updates with collaborators. If this has to be done manually or through side channels, the app is more akin to a single user application. This also implies that programs for annotating portable document format (PDF) files are often not true DCTs, since the user needs to manually share the annotated PDF with collaborators. The tool needs to offer a way to exchange data by itself, and exchange should be as quick as possible. Depending on the supported work modes, exchange should be done automatically and as frequently as reasonable, unless manually disabled by the user (e.g. an offline mode).

Tools also need to offer digital presence and awareness mechanisms where applicable. Building awareness of collaborators' actions is immensely important for successful and efficient collaboration, reducing duplication of work, conflicts, and the need for explicit communication [9, pp. 11–12, 39]. There are many different awareness mechanisms for both synchronous and asynchronous work modes. General examples include locking and *blaming*[31]. Synchronous examples are telepointers & shared animations, activity/status indicators, spotlight/*follow me*-type interactions, or voice & video peeking [7, p. 34], and examples for asynchronous work modes include change highlighting, change (commit) messages, and update notifications. These mechanisms can be distributed around the app. Lastly, as previously noted by Kate Ehrlich [7], DCTs focus on work. Here this refers mainly to producing and editing content, as opposed to coordination and communication. Discussing the work and coordinating how work should be done can of course facilitate that. But it usually isn't the main aspect of collaboration.

[30] It does get a bit more complex when changes haven't been fully synchronized between users before the next editing sessions starts.

[31] This is a way of identifying where a change originated, i.e., which collaborator performed the change.

4. User Survey

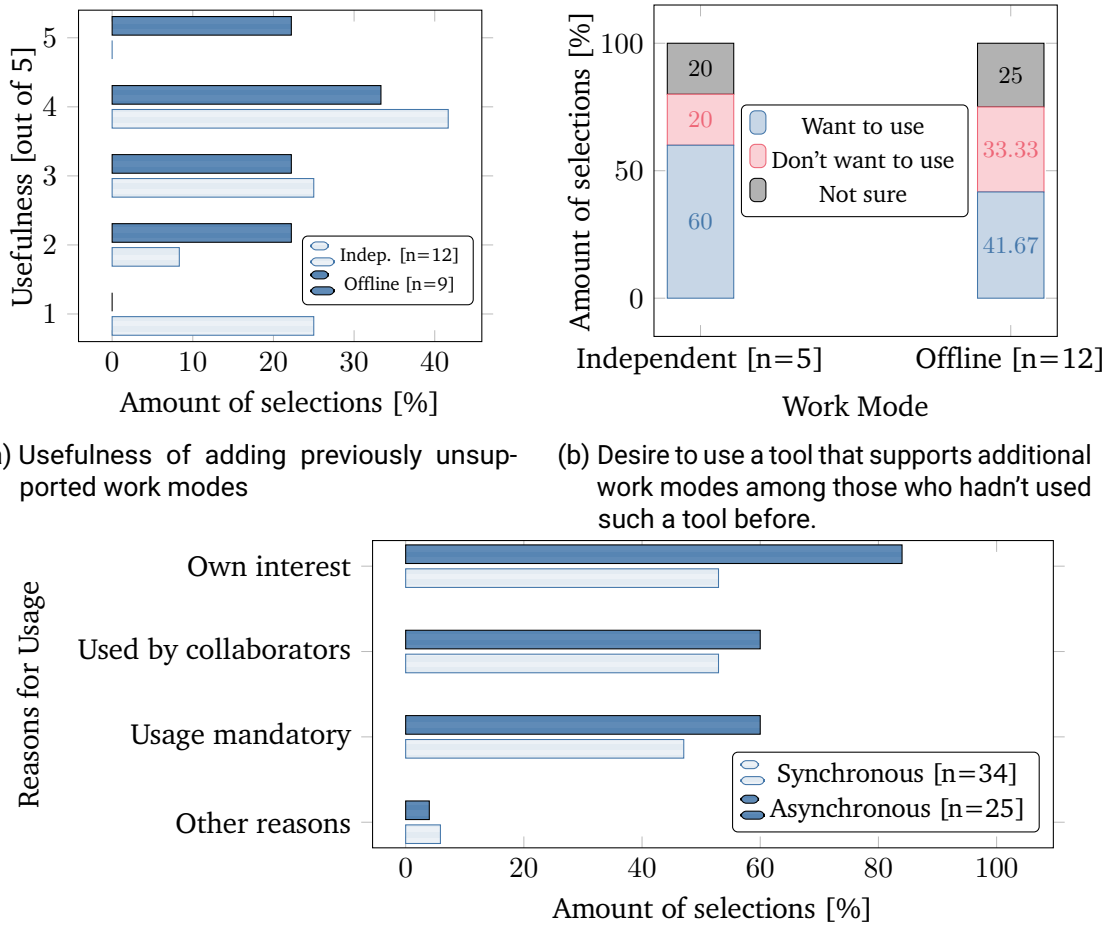
A user survey was conducted in addition to the survey of existing research (Chapter 2) and the analysis of existing applications (Chapter 3). This was done to complement the other data sources in a way that provides multiple points of view to answer the research question of this thesis. The user survey allowed getting feedback from actual users of DCTs. It also allowed asking very specific questions.

Methodology The user survey was conducted through an online questionnaire using Formbricks[32]. A link to the survey was distributed mainly to members of Technical University of Darmstadt (TU Darmstadt), although participants were free to share the link further. Participants received no compensation for their participation. A total of 50 questions were asked in English as part of the questionnaire, ranging from single choice questions to free text ones. Users were free to answer in either English or German (or a mix thereof) for each question. All questions, along with the responses, can be found in Appendix A. Responses were translated analogously into English if needed. The questionnaire began with a short description of the topic followed by a privacy agreement. Before the questions started, a more detailed explanation of the topic and the work modes was given, along with a prompt to carefully read the explanation. Participants were also shown an earlier version of Figure 1. The questions were divided into two sections. The first section dealt with the synchronous work mode, while the second section dealt with the three asynchronous work modes (asynchronous, independent, and offline). The questionnaire used conditional logic to skip certain question based on previous responses by the participant. Therefore, not all questions were shown to every participant. Sometimes this was done because certain questions were not applicable, other times to avoid asking the same question multiple times. Responses were collected from 2024-11-19 to 2024-12-31.

Results A total of 37 participants submitted valid responses, and 34 fully completed the questionnaire. The full results can be seen in Appendix A. The following statements reflect the results of the user survey, and not necessarily facts. Responses from 37 people clearly only reflect a tiny portion of all users, and are therefore not representative.

Some particularly interesting results are shown in Figure 3. Tools supporting synchronous work modes are common and popular. More than 90% of respondents had used such tools before, and almost 90% of those use them at least once a month. Their overall experience with synchronous work modes were positive. Reasons for usage were balanced between own interest, collaborators already using the tools, and usage being mandatory, with each reason being chosen by around half the respondents (Figure 3c). The participants that hadn't used synchronous work modes before weren't interested in starting to use them. Asynchronous work modes are also common and popular. Almost 90% of respondents had used such tools before, and over 80% of those use them at least once a month. More than 80% of respondents used these tools out of own interest (Figure 3c). Like with synchronous tools, participants who hadn't used asynchronous work modes before weren't interested in starting to do so. The offline work mode was considered more useful when

[32] `https://formbricks.com`



(a) Usefulness of adding previously unsupported work modes

(b) Desire to use a tool that supports additional work modes among those who hadn't used such a tool before.

(c) Reasons for usage of tools depending on supported work modes

Figure 3.: A few notable results from the user survey, visualized. There are clear variations, but the number of respondents is unbalanced.

added to an existing tool that supported neither the independent nor offline work mode (Figure 3a). 60% of respondents who hadn't used any tools supporting the independent work mode yet showed interest in doing so in the future, and 40% of those who hadn't used offline-capable tools wanted to change that (Figure 3b).

Overall, collaborative writing tools (*Microsoft Word*, *Google Docs*, *ShareLaTeX*) dominated both the synchronous and asynchronous tools (61 out of 102 selections among synchronous tools, 48 out of 97 selections among asynchronous tools). *Microsoft PowerPoint* was also often mentioned. *Git* was the most used asynchronous tool, and was often used for all three reasons (own interest, collaborators using it, mandatory usage). For the independent work mode, people were most familiar with *Git* (20 out of 27) and *Microsoft Word/Excel/PowerPoint* (6 out of 27). The work mode was used by over 90% of respondents, of which more than 75% used it at least once a week. The independent work mode was rated at 4.56 out of 5, and 60% of those who didn't use tools that offered such a work mode would like to start using it. Results were similar for the offline work mode, with the most popular tools again being *Git* (17 out of 25) and *Microsoft Word/Excel/PowerPoint* (8 out of 25). In total, more than 80% or respondents used offline-capable tools, and almost 90% actually used that functionality. 50% used it at least once a week. The work

mode was rated at 4.36 out of 5, and over 40% of those who didn't use offline-capable tools would like to start using such a work mode. Around 33% of respondents used tools that didn't offer the independent or offline work modes. The main reasons for using these tools were that collaborators were already using them. For those tools, the usefulness of adding the independent or offline work modes were rated at 2.83 and 3.56 out of 5, respectively.

Participants reported various issues with both synchronous and asynchronous usage of tools. For synchronous usage, this included difficulty to keep track of changes and comments, issues with synchronization (speed, consistency, missing network, and loss of work), mistakes made by collaborators affecting everyone (like syntax errors), and use of locking preventing efficient collaboration. For asynchronous usage, it also included difficulty to keep track of changes and synchronization issues (slow, missing network, non-transparent), but also difficulties with merging multiple divergent versions. There were some participants who hadn't used some of the discussed work modes and weren't interested in trying them. Their motivation for that included real-time collaboration being unnecessary (synchronous work mode), not being interested in collaboration at all (asynchronous work mode), seeing no benefit compared to synchronous tools (independent work mode), and having no need for it, seeing no benefit for their type of work, or even not being allowed to work offline by company policy (offline work mode). Other responses highlighted the benefits of various work modes. For the synchronous work mode, people liked that it worked well, was useful, and eased their work, or appreciated the fast working pace. For the asynchronous work mode, version control, versatility (especially for synchronization of data), and efficient collaboration were highlighted. The independent work mode was mentioned to work well for finalizing changes in private, keeping multiple versions (possibly focused on different problems, or for trying out ideas), using a version history, letting changes be reviewed before integrating them into the default version, working independently of each other without disturbing or distracting others, distributing tasks among collaborators, offering a good overview, and preventing loss of work. The offline work mode naturally enables working without a (fast) internet connection, privacy, increased security (e.g. no reliance on public networks), flexibility, fewer distractions, and control over the data.

When asked if they would like to change anything about the current tools they were using, responses included better offline support (including local replicas that synchronize with the server) and support for more synchronous collaboration, also in combination with the asynchronous-independent work mode. There was also demand for better version control, freezing final versions, and better annotation and commenting features. Participants additionally wished for better awareness mechanisms, reactive apps that don't require manual refreshes, and collaboration not relying on locking.

5. Discussion

This chapter discusses insights from all three surveys (literature, available applications, and the questionnaire) and makes design recommendations based on this. These include notes on local-first software and the use of CRDTs, conflict handling, and which work modes to implement in a DCT.

5.1. Desired Functionality (Best Practices)

When it comes to implementing DCTs, there are more aspects to take into account than just functionality. These other considerations should also guide the design and development of applications to make them usable, pleasant, and productive. One fundamental characteristic of a collaboration app should be that it's designed with collaboration and the type of work in mind [7, 15, 42]. While collaboration apps are about supporting group work [7], the goal is not to build do-it-all applications that support any kind of work. And while it is possible to *hack on* collaboration features at a later point, this often comes at the cost of proper integration; the functionality often ends up shallow or limited[33]. Different types of work have different types of requirements and semantics, and therefore require different types of apps. The narrower in scope an app is, the better it can implement those specific semantics without trade-offs[34]. Designing the app, its UI, and the functionality according to the constraints and properties of the collaborative work will result in a much more coherent app.

Another aspect that was mentioned by existing research mainly with regard to collaborative writing but is generally applicable is attribution. Attribution, which is about identifying the person (or system) responsible for a change, serves two purposes: preserving authorship during collaboration, and enhancing traceability of changes. Preserving authorship is a social issue, where its absence can be detrimental to the collaboration quality, by reducing motivation, and leading to confusion or even social conflicts among collaborators [15, 18]. Traceability is about making it easy for users to comprehend changes, seeing who changed what where, and possibly when. If there is no attribution for changes, awareness mechanisms like this cannot function properly, making the collaboration unnecessarily mentally demanding [9]. Participants of the user survey also mentioned confusion and annoyances during collaboration due to a lack of traceability.

One issue with modern collaboration apps highlighted by Kleppmann et al. is the lack of privacy and independence [1]. Since many apps try to support the synchronous work mode by using a central server, all data generated with and by these apps will end up somewhere in the cloud. This can be especially problematic for businesses who need to keep certain information confidential. But privacy from other collaborators is also important. Sometimes people don't want others to see their drafts or experiments [15, 56]. DCTs should try to preserve privacy wherever possible. Another issue related to

[33] Examples for this include the use of locking in Microsoft Word and DaVinci Resolve. Where Google Docs was built to be collaborative from the start, Microsoft Word wasn't.

[34] This may seem like an obvious insight, but has profound implications. A collaborative plain-text editor could be used for many different things, but isn't a perfect fit for any type of work. The usefulness comes from the specific additional functionality that is added beyond text editing.

privacy is independence. If the authoritative copy of the data is stored on a third-party server, there is not only a single point of failure to the collaboration, but also a risk of complete data loss. And even if the data is stored on collaborators' devices, collaboration is no longer possible without the central server [1]. Adding alternative means of data persistence and data exchange to collaboration apps will therefore make the use of such apps less risky in general. Participants of the user survey reported fear of data loss, the desire to have full control over their data, and being paranoid about privacy issues related to the cloud.

A related topic is the need for keeping track of conflicts, should they appear. Some tools (like Evernote[35] [1, p. 5], or most file sharing apps) will create multiple copies of a conflicting file with two slightly different names. The user might get informed about this, but no support is given for resolving the conflict and going back to having a single, consistent version. Good collaboration apps should always support the user in resolving conflicts [33, 57, 58]. Apps should also prompt the user to resolve the conflict in due time to facilitate keeping a consistent state. Proper handling and tracking of conflicts becomes more important the less immediate their resolution needs to be. For apps that can tolerate great divergence, the chances of collaborators forgetting about conflicts and inconsistent versions increases, and therefore good tooling is needed to help them eventually return to a consistent version [24].

One more thing that is immensely beneficial for creating good DCTs is reactivity [1]. Modern collaboration apps shouldn't need to refresh lists or reload pages to show the latest data. This only increases the likelihood of conflicts appearing in the meantime. If new changes arrive at the client and a compatible work mode is used, the user should see them immediately. Reactivity enables this by making sure the state of the app's data structure and the UI are always consistent, which is essential for supporting synchronous work modes. Issues with shifting text in a document or similar issues when content is added or removed should be handled through awareness mechanisms that clearly identify what was changed and by whom. Directly integrating changes by default also makes it easier to scale from an asynchronous behavior to a quasi-synchronous one, simply by controlling when and how frequently changes are retrieved from collaborators. Employing reactivity instead of manually integrating incoming changes is also easier to implement, making development faster and enabling potential support for a broad set of work modes that improve the collaboration experience for users. During the user study, some participants made comments about not wanting to manually refresh websites to see the latest status, as well as wishing for better real-time collaboration and awareness mechanisms. All of this could be partially improved through reactivity.

5.2. Insights from the User Survey

Aside from the raw results of the user survey presented in Chapter 4, there are some additional insights that can be gained from the responses. Many issues, ideas, and suggestions mentioned in existing research were reflected in the responses to the questionnaire. Some respondents mentioned the ability to finish tasks without interruptions, and only sharing their work once the task was completed. This aligns both with the premise of the independent work mode, and with Edwards et al.'s notion of asynchronous and independent work [11, p. 120]. Posner and Baecker highlighted the need for "preserv[ing] collaborator identities" [15, p. 3] along with their changes, providing version control and a change

[35] `https://evernote.com/`

history, and supporting different working styles (including single- and multi-user, and working synchronously as well as asynchronously). User demand for all of these proposed needs can be seen in the responses of the user survey. A lack of these features resulted in confusion among users, making it hard to keep track of fast-paced work and when many collaborators worked together. The user survey also confirmed that the suitability of a certain work mode for an application heavily depends on the nature of the work done with that application. For instance, the asynchronous-independent work mode was almost exclusively used (and supported) by *Git*. Respondents also saw it as the default work mode for software development, through comments like "it is the default" or "well, software development works like that" when asked about the independent work mode. One respondent also explicitly suggested combining the asynchronous-independent work mode of *Git* with the synchronous work mode of collaborative writing apps. This real-time collaboration combined with independent changes that can be worked on and integrated once completed exactly describes the theorized synchronous-independent work mode, which so far isn't properly supported by any of the analyzed DCTs. As for usage reasons, Kate Ehrlich theorized that both the need for purchasing multiple copies and peer pressure affected how well groupware tools were adopted by people. The survey results were inconclusive here, but generally it doesn't seem like the most popular apps were mainly used because collaborators were already using them. Since all considered tools had a free version available, no comment can be made on how license costs affect adoption.

It's important to note that a survey only provides indirect information, where people describe their opinion, the issues they see, and the solutions they would like to see, instead of their actual behavior being observed. These might be vastly different for every user, and not necessarily actually indicate the best solution for them. The survey results can therefore only be seen as a rough representation of requirements and user behavior. The trust in the user survey's results may be increased if they align with suggestions and results from other sources. Many participants had a background in software development[36] and were therefore familiar with *Git*. Some participants selected Microsoft Word/Excel/PowerPoint as the tool supporting the independent work mode which they were most familiar with, although these apps were deemed to not support the independent work mode in Table 1. It could be that participants misunderstood the functionality of the independent work mode, even though an example using *Google Docs* was given at the beginning. A reason for this could be that English likely wasn't the first language for most of the participants.

5.3. Towards Local-first Collaboration Apps

Since Kleppmann et al. coined the term "local-first" in 2019 [1], many people have investigated these kinds of apps. The premise of apps that are at least as usable as today's collaboration apps, while being faster, independent, private, and decentralized is enticing. Currently available apps look promising in most of these regards, but are still struggling mainly with supporting real-time synchronous work modes and living up to the performance expectations set for them. Almost all existing apps that call themselves local-first use CRDTs to store and synchronize data, but from the analysis done for this thesis it is evident that this alone is not sufficient to completely fulfill all goals of local-first collaboration software. Apps need to actively implement synchronization strategies that support additional work modes, add awareness mechanisms to facilitate collaboration, and stay efficient and fast to avoid loading times.

[36]This was determined by tracking how and where each link was shared.

The popularity of web apps makes developing true local-first apps more difficult, since web apps are everything but local-first by nature. They are centralized, require an internet connection, can only store limited amounts of data locally, and are always subject to loading times and latency. While there are ways to alleviate most of these issues, that requires additional development effort, making them not the ideal candidate for being the foundation of local-first apps. This results in local-first apps requiring users to download native apps to unlock the full functionality and UX, which is an added inconvenience compared to non-local-first apps.

CRDTs & Consistency Local-first software uses eventual consistency to allow for independent and decentralized data storage. Consistency must only be reached once the peer network is quiescent[37]. How often and quickly consistency is reached heavily depends on the work mode that is employed. A synchronous work mode requires that the state becomes consistent across peers as quickly as possible, while an independent work mode encourages prolonged inconsistency by design [24]. If the offline work mode is supported, consistency can't be reached at all by disconnected peers.

CRDTs were proposed as a promising technology for local-first development, given that they are able to decentralize apps at the data storage layer. They were designed for distributed systems, especially any systems or applications that could benefit from eventual consistency [1, 29, 32]. This included mostly backend systems[38], but client-side adoption is slowly picking up speed. There are even entire companies built around local-first collaboration[39]. These focus on providing tooling and platforms for synchronizing data between edge devices. Data synchronization mainly targets collaboration between on-site devices that form an independent and mostly isolated network[40]. Synchronization is often provided through CRDTs, but complex merge semantics are not necessary. Instead, changes are mostly applied in a LWW fashion for individual properties. Conflicts are not handled explicitly, but user intent is partially preserved.

But, in many cases, it isn't necessary to implement complex CRDT-based synchronization at all. If concurrent usage only happens rarely or for short periods, there will be little difference in the merge result between CRDTs and simpler algorithms. In synchronous work modes conflicts rarely appear, and only in scenarios that D'Angelo et al. call "spacetime collaboration" [25, p. 2]. So CRDTs' main benefit is enabling decentralized and consistent merging during *extended concurrent usage*, which happens in the independent and offline work modes, where many opportunities for concurrent changes exist [29, p. 13]. Apps that don't intend to support these work modes should consider using simpler and less taxing algorithms [1, 25, 33]. Implementations of the OT algorithm often optimized to rely on a central server, and have poor support for end-to-end encryption [64][41], which makes them less suitable compared to CRDTs.

[37] Meaning that all current changes have arrived at all peers and no new changes are made.
[38] As was the case with the likes of SoundCloud [59], Facebook [60, 61], and Redis [62, 63].
[39] Like Ditto (`https://ditto.live/`).
[40] Examples of this are in aviation, autonomous tactical units, or point-of-sales systems for restaurant floors (`https://ditto.live/customers/`, visited on 2024-11-21).
[41] While *Google Docs* offers an encrypted variant for businesses, this does not support synchronous editing. Documents get locked when someone starts editing them [65], making it much less collaborative than the regular version.

Conflicts, Merging & Preserving the User's Intent While CRDTs try to keep a sensible semantic for concurrent updates, this semantic is app- and data type-dependent, and can never be truly semantically correct. One of CRDTs main purposes is providing a semantic for concurrent updates that is as close to correct as possible, where *correct* would be exactly the change the user would have intended to make if they had manually merged the changes. However, for certain types of conflicts, called *semantic conflicts*, this is not realistically achievable. McKelvey et al. describe these semantic conflicts as "conflicts [...] not in the structure of the text but rather in its meaning" [31]. Algorithms that merge concurrent conflicts, like CRDTs or OT, cannot properly preserve the intent in these cases, and preserving intent across merges is difficult in general [66]. *Semantic conflicts* are difficult or even impossible to detect and fix. This means that users need to meticulously review merges based on the changes between the old and new version. In the case of a *semantic conflict*, a user would need to manually resolve it.

Automatic conflict resolution isn't needed for DCTs at all. Through good tooling, conflicts could be highlighted and manually resolved when convenient. This could be done similarly to how suggestions are shown in *Multicollab* or *Google Docs*, allowing the user to quickly compare and toggle between versions in-place. An alternative approach is the one used by Schiefer et al. [24], where the intent behind both conflicting versions is preserved by simply not merging them. Keeping both conflicting versions and allowing the user to manually resolve the conflict at a later time greatly increases flexibility and transparency, at the cost of encouraging prolonged divergence. Both approaches allow users to "continue working, regardless of the actions taken by coworkers" [11, p. 120]. A popular third approach is to choose the most recent change and discard older ones. This LWW approach works well for apps that are more *synchronous* and objective in nature. For *asynchronous* applications, especially *asynchronous-independent*, this approach could lead to a severe loss of work, making it not universally applicable.

Asynchronous conflict management needs to be optimized for large change sets, offering powerful tools for comparing changes between versions [57, p. 116]. There are well-established solutions for text-based applications [67], but other data types are more challenging to compare[42]. Synchronous conflict resolution instead needs to be as quick as possible, or easily ignored and postponed. Given that conflicting edits in synchronous work modes are very rare [20, 33], manual handling is possible even in fast-paced editing environments.

5.4. Choosing Work Modes

Which work modes are best suited for a given application depends on many factors, most notably the type of application, the type of work, and the users' preferences and social environment [11, 15, 20, 21, 24, 70]. There are certain DCTs that can make good use of all work modes. These are mostly apps supporting creative work, where users can work on certain parts together in real time, but also take their time and work on some task

[42] One approach could be that of *DaVinci Resolve*, with its "Visual Timeline Comparison" feature [68]. Here, instead of comparing the resulting video directly, the comparison is done between the timelines of both versions. This allows the user to get a quick overview of how the timeline was modified between versions. Comparing a more abstract representation of the digital artifact (the timeline in this case) helps to simplify complex data for easier comparison. Such an approach could also be used for other complex types like 3D modeling, where many programs (like *Autodesk Fusion* or *Shapr3D*) already offer a timeline of changes that represents the 3D object. *GitHub* has also tried to add better visual diffing tools for 3D objects without relying on a timeline [69].

in an isolated fashion. None of the analyzed apps supported all work modes. Most apps lean toward a smaller region of the synchronicity spectrum, supporting most often two, sometimes three of the work modes. Except for pure meeting apps, every tool supports the asynchronous work mode. It can therefore be considered the default work mode [56, p. 15, 18, 21]. The synchronous work mode is a nice addition for almost every app since it has very few drawbacks while offering more efficient collaboration [20, p. 28], but it does require a reactive foundation [1]. Synchronous work is usually only maintained within sessions [9], and becomes asynchronous in-between those sessions. The independent work mode can be useful for highly complex but subjective tools that need to go beyond simple change tracking, by allowing users to avoid distractions and finalize their work before integrating it. The offline work mode is generally least useful[43] and has several implications for the app design, but might be easily implemented depending on the framework used.

The lack of certain work modes can often be compensated for to a certain degree: side channels can be used for additional coordination instead of a synchronous work mode, and the offline work mode can somewhat replace an independent work mode. However, explicitly offering a work mode is much more powerful. In the end, an applications or task's tolerance for inconsistencies decides which work modes can and should be supported. Apps that rely on up-to-date and consistent state *need* a synchronous work mode. Apps used in places without network connectivity *need* an offline work mode. These requirements should be analyzed before the tool is designed and before decisions about the framework and software stack are made. But no matter which work modes are supported, conflicts can always appear in collaboration apps. How frequently they appear and how they are handled depends on the work mode. Conflicts never have to be handled immediately, and manual conflict resolution is the most transparent solution that best preserves the users' intents, and should therefore be used much more often.

[43] While there are undoubtedly situations in which being able to work offline is incredibly useful, these are rare and the offline functionality is seldom required during regular usage. Dealing with intermittent network interruptions is not considered offline support here.

6. Conclusion

This thesis took an in-depth look at both the history and the current state of digital collaboration tools. It investigated from three different angels: existing literature, currently available collaboration apps, and a user survey in the form of a digital questionnaire. Several properties and features of DCTs were collected, presented, and discussed, including recommendations for how these tools should behave. A fundamental taxonomy for DCTs was compiled, based on several existing taxonomies and properties typically encountered in modern collaboration apps.

DCTs are interconnected applications that enable working together on digital artifacts in shared environments. They can support several different work modes, including synchronous, asynchronous, independent, and offline. While the asynchronous-independent work mode is common for certain types of apps, the synchronous-independent work mode is currently not supported by any of the analyzed DCTs, although there seems to be user demand for it. The analysis of currently available collaboration apps revealed that certain types of apps tend to support a certain selection of work modes, while often being incompatible with certain other work modes. DCTs evolved from decentralized groupware applications focused on bringing any form of collaboration to traditional software over the local network, to centralized cloud-based applications supporting advanced synchronous and asynchronous editing, at the cost of privacy, independence, and ownership. Given that collaboration apps are nonetheless immensely popular and ubiquitous in the modern workplace, researchers are now trying to combine some of the benefits of older (collaboration) software with those of modern cloud tools, to create a new family of *local-first software* that offers the best of both worlds.

One main finding of this thesis is that DCTs should support the asynchronous work mode as the default, in addition to the synchronous work mode. They should be collaborative from the ground up, and support both single and multiple users. Appropriate multi-user awareness mechanisms should be employed for all work modes. The supported work modes are ultimately determined by the type of application and the concrete work being done with it, since it dictates the application's tolerance for inconsistencies among collaborators. The users' social environment, preferences, and style of work also play an important role, and need to be adequately considered by employing UX design techniques. Data exchange should happen automatically and as frequently as possible. Interacting with collaborators' work should be supported, ideally by suggesting specific changes. Finally, digital collaboration apps should strive to offer privacy, independence, and data ownership to their users. Further research is needed to determine if the asynchronous work mode is the default for all DCTs, or only for centralized ones.

For actually building DCTs, CRDTs look promising, but they inherited some assumptions from traditional distributed systems that need to be adapted for user-facing collaboration tools. This mainly includes their tendency to automatically resolve conflicts (*intent merging*). Keeping track of conflicts and supporting users in resolving them is necessary, but semantic conflicts and user intent are impossible to handle automatically. Letting users manually resolve conflicts seems to be possible in every work mode. Future research into CRDTs should focus less on automatic conflict resolution, and more on offering transparency to the users as well as preserving user intent. Performance should be

improved, and support for synchronous collaboration should be emphasized. Awareness mechanisms should be easy to integrate into CRDT-based apps. Suggestions like forking histories [24] seem valid, but encouraging divergence still needs proper evaluation. At some point, resolving all conflicts between diverged versions could become tedious and unattractive to users, resulting in the versions diverging more and more, or in a loss of productivity [33, p. 47]. This is definitely not the goal of collaboration, and it should be investigated how this can be avoided.

The principles of local-first software and CRDTs differ immensely from the nature of web apps, which makes it difficult for local-first apps to gain a foothold in the modern software landscape without suffering the drawbacks of centralized computing. Local-first collaboration apps seem to be thriving especially in areas with limited connectivity, as originally envisioned by past papers [11, 23, 37, 71, 72]. While low-bandwidth connectivity doesn't seem to be as much of an issue nowadays as reported by Posner and Baecker [15], the user survey revealed that there still are instances where slow connections can cause problems with collaboration. However, it is unlikely that new local-first applications would be adopted just for these rare occurrences.

The limitations of asynchronous work modes point to a general need for effortless real-time synchronization for all apps (if desired by the user) [20, p. 28]. This should be supported on a programming language- or framework-level. Most importantly, DCTs should be reactive by default, always and reliably offering up the latest data available as soon as it's available [1].

Olson et al. believed that "collaborative creation" will start moving beyond document editing and include "a variety of digital objects" [20, p. 32], a sentiment shared by Ellis et al. [9, p. 3]. So far only few, specialized tools seem to do this, and often lack extensive awareness mechanisms. More design and engineering explorations towards collaboration on complex data types are needed, especially towards improving attribution, change histories, and comparing changes. However, the enormous popularity of large language model (LLM)-based artificial intelligence (AI) might slow this development down, since text-based work seems to have regained importance for now.

The goal of this thesis was to inform the design and development of DCTs to help build better apps that offer a great UX. Past research has had major influences on the landscape of collaboration apps, and suggestions made by literature are eventually implemented[44].

Even though this thesis employed user research in the form of a questionnaire, more research in this area is still needed. A larger survey with a more diverse group of participants could help paint a clearer picture of user needs, and alternative research methods such as field or diary studies might yield more trustworthy data that better reflects actual usage and user behavior. There are also several opportunities for building and evaluating prototypes that offer broader support for different work modes. This could clarify if the synchronous-independent work mode offers any advantages and if it has any unforeseen drawbacks, or if available connectivity can be used to offer new and improved awareness mechanisms in the asynchronous-independent work mode to streamline collaboration. It could also explore if users who are not familiar with *Git* can easily make proper use of an independent work mode.

[44] For instance, suggestions from both Posner and Baecker [15] as well as Olson et al. [20] have been recently implemented in *Google Docs*, like version control, voice and video calls, or separate spaces for related notes that are not part of the actual document.

List of Figures

List of Tables

Bibliography

[1] Martin Kleppmann et al. “Local-first software: you own your data, in spite of the cloud”. In: *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Onward! 2019. New York, NY, USA: Association for Computing Machinery, 2019-10, pp. 154–178. DOI: 10.1145/3359591.3359737.

[2] Brian McLoone and Rory Smead. “The ontogeny and evolution of human collaboration”. en. In: *Biology & Philosophy* 29.4 (2014-07), pp. 559–576. ISSN: 1572-8404. DOI: 10.1007/s10539-014-9435-1.

[3] Peter Evans-Greenwood, Sue Solly, and Robbie Robertson. *Reconstructing the workplace*. en-us. 2021-07. URL: https://www2.deloitte.com/us/en/insights/focus/technology-and-the-future-of-work/working-digitally.html (visited on 2024-10-30).

[4] Angelika Bullinger-Hoffmann et al. “Computer-Supported Cooperative Work – Revisited”. en. In: *i-com* 20.3 (2021-12). Publisher: Oldenbourg Wissenschaftsverlag, pp. 215–228. ISSN: 2196-6826. DOI: 10.1515/icom-2021-0028.

[5] S. H. Halili, R. Abdul Razak, and Z. Zainuddin. “Enhancing collaborative learning in flipped classroom”. en. In: *International Conference on Science, Engineering and Built Environment*. Bali, Indonesia, 2014-11. URL: https://eprints.um.edu.my/11973/ (visited on 2024-10-31).

[6] Sang-Hong Kim, Nam-Hun Park, and Kil-Hong Joo. “Effects of Flipped Classroom based on Smart Learning on Self-directed and Collaborative Learning”. ko. In: *International Journal of Control and Automation* 7.12 (2014-12), pp. 69–80. ISSN: 2005-4297. URL: https://www.earticle.net/Article/A239156.

[7] Kate Ehrlich. “Designing Groupware Applications: A Work-Centered Design Approach”. en. In: *Computer Supported Co-operative Work*. Ed. by Michel Beaudouin-Lafon. Vol. 7. Trends in Software. John Wiley & Sons Limited, 1999, pp. 1–28. DOI: 10.1016/B978-0-7506-7111-8.50009-3.

[8] Drew DeVault. *The advantages of an email-driven git workflow*. 2018-07. URL: https://drewdevault.com/2018/07/02/Email-driven-git.html (visited on 2024-10-31).

[9] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. “Groupware: some issues and experiences”. In: *Commun. ACM* 34.1 (1991-01), pp. 39–58. ISSN: 0001-0782. DOI: 10.1145/99977.99987.

[10] Robert Johansen. *Groupware : computer support for business teams*. eng. New York : Free Press ; London : Collier Macmillan, 1988. ISBN: 978-0-02-916491-4. URL: http://archive.org/details/groupwarecompute0000joha.

[11] W. Keith Edwards et al. “Designing and implementing asynchronous collaborative applications with Bayou”. In: *Proceedings of the 10th annual ACM symposium on User interface software and technology*. UIST ’97. New York, NY, USA: Association for Computing Machinery, 1997-10, pp. 119–128. DOI: 10.1145/263407.263530.

[12] J. Xu et al. “A Survey of Asynchronous Collaboration Tools”. en-gb. In: *Information Technology Journal* 7.8 (2008), pp. 1182–1187. DOI: 10.3923/itj.2008.1182.1187.

[13] Gail L. Rein and Clarence A. Ellis. “The Nick Experiment Reinterpreted: Implications for Developers and Evaluators of Groupware”. en. In: *Office Technology and People* 5.1 (1989-01). Publisher: MCB UP Ltd, pp. 47–75. ISSN: 0167-5710. DOI: 10.1108/EUM0000000003528.

[14] Susanne Bødker et al. “Computer support for cooperative design”. In: *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*. CSCW ’88. New York, NY, USA: Association for Computing Machinery, 1988-01, pp. 377–394. DOI: 10.1145/62266.62296.

[15] I.R. Posner and R.M. Baecker. “How people write together (groupware)”. In: *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*. Vol. iv. 1992-01, 127–138 vol.4. DOI: 10.1109/HICSS.1992.183420.

[16] Lucy A. Suchman. “Office procedure as practical action: models of work and system design”. In: *ACM Trans. Inf. Syst.* 1.4 (1983-10), pp. 320–328. ISSN: 1046-8188. DOI: 10.1145/357442.357445.

[17] Paul Benjamin Lowry, Aaron Curtis, and Michelle René Lowry. “Building a Taxonomy and Nomenclature of Collaborative Writing to Improve Interdisciplinary Research and Practice”. en. In: *The Journal of Business Communication (1973)* 41.1 (2004-01). Publisher: SAGE Publications, pp. 66–99. ISSN: 0021-9436. DOI: 10.1177/0021943603259363.

[18] Dakuo Wang. “Exploring and Supporting Today’s Collaborative Writing”. en. PhD thesis. UC Irvine, 2016. URL: https://escholarship.org/uc/item/7441493c.

[19] Dakuo Wang. “How People Write Together Now: Exploring and Supporting Today’s Computer-Supported Collaborative Writing”. In: *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. CSCW ’16 Companion. New York, NY, USA: Association for Computing Machinery, 2016-02, pp. 175–179. DOI: 10.1145/2818052.2874352.

[20] Judith S. Olson et al. “How People Write Together Now: Beginning the Investigation with Advanced Undergraduates in a Project Course”. In: *ACM Trans. Comput.-Hum. Interact.* 24.1 (2017-03), 4:1–4:40. ISSN: 1073-0516. DOI: 10.1145/3038919.

[21] Tom Boellstorff et al. “Words with friends: writing collaboratively online”. In: *interactions* 20.5 (2013-09), pp. 58–61. ISSN: 1072-5520. DOI: 10.1145/2501987.

[22] Paul André, Robert E. Kraut, and Aniket Kittur. “Effects of simultaneous and sequential work structures on distributed collaborative interdependent tasks”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’14. New York, NY, USA: Association for Computing Machinery, 2014-04, pp. 139–148. DOI: 10.1145/2556288.2557158.

[23] Irene Greif et al. “Replicated document management in a group communication system”. In: *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*. CSCW ’88. New York, NY, USA: Association for Computing Machinery, 1988-01, p. 395. DOI: 10.1145/62266.1024798.

[24] Nicholas Schiefer, Geoffrey Litt, and Daniel Jackson. “Merge what you can, fork what you can’t: managing data integrity in local-first software”. In: *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC ’22. New York, NY, USA: Association for Computing Machinery, 2022-04, pp. 24–32. DOI: 10.1145/3517209.3524041.

[25] Gabriele D’Angelo, Angelo Di Iorio, and Stefano Zacchiroli. “Spacetime Characterization of Real-Time Collaborative Editing”. In: *Proc. ACM Hum.-Comput. Interact.* 2.CSCW (2018-11), 41:1–41:19. DOI: 10.1145/3274310.

[26] Ross Callon. *The Twelve Networking Truths*. Request for Comments RFC 1925. Num Pages: 3. Internet Engineering Task Force, 1996-04. DOI: 10.17487/RFC1925.

[27] Jakob Nielsen. “Response Time Limits”. en. In: *Nielsen Norman Group* (1993-01). URL: https://www.nngroup.com/articles/response-times-3-important-limits/ (visited on 2024-11-26).

[28] Walter J. Doherty and Ahrvind J. Thadani. *The Economic Value of Rapid Response Time*. Catalog Number: 102751398 Category: Technical Report Credit: Gift of Bill Worthington. 1982-11.

[29] Nuno Preguiça. “Conflict-free Replicated Data Types: An Overview”. en. In: *arXiv.org*. 2018-06. DOI: 10.48550/arXiv.1806.10254.

[30] Petru Nicolaescu et al. “Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types”. In: *Proceedings of the 2016 ACM International Conference on Supporting Group Work*. GROUP ’16. New York, NY, USA: Association for Computing Machinery, 2016-11, pp. 39–49. DOI: 10.1145/2957276.2957310.

[31] Karissa Rae McKelvey et al. *Upwelling: Combining real-time collaboration with version control for writers.* en-us. 2023-03. URL: https://www.inkandswitch.com/upwelling/ (visited on 2024-10-11).

[32] Marc Shapiro et al. “Conflict-Free Replicated Data Types”. en. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Xavier Défago, Franck Petit, and Vincent Villain. Berlin, Heidelberg: Springer, 2011, pp. 386–400. DOI: 10.1007/978-3-642-24550-3_29.

[33] Hoai Le Nguyen. “Study of Conflicts in Collaborative Editing”. en. PhD thesis. Université de Lorraine, 2021-01. URL: https://hal.univ-lorraine.fr/tel-03203102.

[34] T. Mens. “A state-of-the-art survey on software merging”. en. In: *IEEE Transactions on Software Engineering* 28.5 (2002-05), pp. 449–462. ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1000449.

[35] Ida Larsen-Ledet and Henrik Korsgaard. “Territorial Functioning in Collaborative Writing”. en. In: *Computer Supported Cooperative Work (CSCW)* 28.3 (2019-06), pp. 391–433. ISSN: 1573-7551. DOI: 10.1007/s10606-019-09359-8.

[36] Ronald M. Baecker et al. “The user-centered iterative design of collaborative writing software”. In: *Proceedings of the INTERACT ’93 and CHI ’93 Conference on Human Factors in Computing Systems*. CHI ’93. New York, NY, USA: Association for Computing Machinery, 1993-05, pp. 399–405. DOI: 10.1145/169059.169312.

[37] Paul Dourish. “The Parting of the Ways: Divergence, Data Management and Collaborative Work”. en. In: *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work ECSCW ’95*. Ed. by Hans Marmolin, Yngve Sundblad, and Kjeld Schmidt. Dordrecht: Springer Netherlands, 1995, pp. 215–230. DOI: 10.1007/978-94-011-0349-7_14.

[38] Gloria Mark, Victor M. Gonzalez, and Justin Harris. “No task left behind? examining the nature of fragmented work”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’05. New York, NY, USA: Association for Computing Machinery, 2005-04, pp. 321–330. DOI: 10.1145/1054972.1055017.

[39] Claudia-Lavinia Ignat et al. “Providing awareness in multi-synchronous collaboration without compromising privacy”. In: *Proceedings of the 2008 ACM conference on Computer supported cooperative work*. CSCW ’08. New York, NY, USA: Association for Computing Machinery, 2008-11, pp. 659–668. DOI: 10.1145/1460563.1460665.

[40] Valeria Herskovic et al. “The Iceberg Effect: Behind the User Interface of Mobile Collaborative Systems.” In: *J. UCS* 17 (2011-01), pp. 183–201.

[41] Barbara Kainberger. “Collaborative Editing in Web Applications: Exploration & Implementation of a collaborative real-time web application with React, Yjs and modular Rich-Text editor frameworks”. en. MA thesis. Wien: University of Applied Sciences FH Campus Wien, 2022-11. URL: http://pub.fh-campuswien.ac.at/obvfcwhsacc/8510828.

[42] Evan Wallace. *How Figma’s multiplayer technology works*. en. 2019-10. URL: https://www.figma.com/blog/how-figmas-multiplayer-technology-works/ (visited on 2024-11-04).

[43] Jake Donham and Devon Rifkin. *GitHub Next | Realtime GitHub*. en. 2023-11. URL: https://next.github.com/projects/rtgh (visited on 2024-12-09).

[44] Stack Overflow. *Technology*. en. 2024. URL: https://survey.stackoverflow.co/2024/technology/ (visited on 2025-01-24).

[45] Charlotte P. Lee and Drew Paine. “From The Matrix to a Model of Coordinated Action (MoCA): A Conceptual Framework of and for CSCW”. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. CSCW ’15. New York, NY, USA: Association for Computing Machinery, 2015-02, pp. 179–194. DOI: 10.1145/2675133.2675161.

[46] Overleaf. *Git integration*. en. URL: https://www.overleaf.com/learn/how-to/Git_integration (visited on 2025-01-24).

[47] Figma. *Get updates from main files*. en-US. URL: https://help.figma.com/hc/en-us/articles/5665728006423-Get-updates-from-main-files (visited on 2025-01-24).

[48] Microsoft. *Document collaboration and co-authoring*. URL: https://support.microsoft.com/en-us/office/document-collaboration-and-co-authoring-ee1509b4-1f6e-401e-b04a-782d26f564a4 (visited on 2025-01-24).

[49] Dominic Anderson. *How To Use Notion Offline*. en-US. 2022-06. URL: https://www.alphr.com/use-notion-offline/ (visited on 2025-01-24).

[50] Younès Benallal. *Ultimate Guide to Using Notion Offline*. 2023-11. URL: https://notionist.app/notion-offline-guide-can-you-use-it (visited on 2025-01-24).

[51] Giorgia Dalla Valle. *Can Notion Work Offline: Everything you Need to Know (2024)*. en. 2024-07. URL: https://www.notionavenue.co/post/notion-offline (visited on 2025-01-24).

[52] Simo Elalj. *How To Use Notion Offline in 5 Easy Steps*. en. 2024-12. URL: https://2sync.com/blog/how-to-use-notion-offline (visited on 2025-01-24).

[53] Skiff. *Security Whitepaper*. 2023-02. URL: https://web.archive.org/web/20241203195908/https://skiff.com/security-model (visited on 2025-01-24).

[54] Microsoft. *Visual Studio Code September 2024 Update*. en. 2024-10. URL: https://code.visualstudio.com/updates/v1_94#_source-control-graph-view-improvements (visited on 2025-01-24).

[55] Atul Prakash. "Group Editors". en. In: *Computer Supported Co-operative Work*. Ed. by Michel Beaudouin-Lafon. Vol. 7. Trends in Software. John Wiley & Sons, 1999, pp. 103–133. ISBN: 0-471-96736-X. URL: https://www.lri.fr/~mbl/Trends-CSCW/fulltext.html.

[56] Dakuo Wang, Haodan Tan, and Tun Lu. "Why Users Do Not Want to Write Together When They Are Writing Together: Users' Rationales for Today's Collaborative Writing Practices". In: *Proc. ACM Hum.-Comput. Interact.* 1.CSCW (2017-12), 107:1–107:18. DOI: 10.1145/3134742.

[57] Nicholas Nelson. "Supporting merge comprehension when resolving merge conflicts". en. Publisher: Oregon State University. Dissertation. Oregon State University, 2024-01.

[58] Antti Nieminen. "Real-time collaborative resolving of merge conflicts". In: *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. 2012-10, pp. 540–543. DOI: 10.4108/icst.collaboratecom.2012.250435.

[59] Peter Bourgon. *Roshi: a CRDT system for timestamped events*. en. 2014-05. URL: https://developers.soundcloud.com/blog/roshi-a-crdt-system-for-timestamped-events (visited on 2025-01-24).

[60] Xiao Shi et al. "FlightTracker: Consistency across Read-Optimized Online Stores at Facebook". en. In: 2020, pp. 407–423. ISBN: 978-1-939133-19-9. URL: https://www.usenix.org/conference/osdi20/presentation/shi.

[61] Sander Mak. *Facebook announces Apollo at QCon NY 2014*. 2014-06. URL: http://branchandbound.net/blog/conferences/2014/06/facebook-announces-appolo-qcon/ (visited on 2025-01-24).

[62] Redis. *Diving into Conflict-Free Replicated Data Types (CRDTs)*. en. 2022-03. URL: https://redis.io/blog/diving-into-crdts/ (visited on 2025-01-24).

[63] Cihan Biyikoglu. *Under the Hood: Redis CRDTs (Conflict-free Replicated Data Types*. 2020-10. URL: https://web.archive.org/web/20201030020404/https://lp.redislabs.com/rs/915-NFD-128/images/WP-RedisLabs-Redis-Conflict-free-Replicated-Data-Types.pdf (visited on 2025-01-24).

[64] Michael Clear et al. “Collaboration-Preserving Authenticated Encryption for Operational Transformation Systems”. In: vol. 7483. 2012-09, pp. 204–223. DOI: 10.1007/978-3-642-33383-5_13.

[65] Google. *Collaborate on encrypted files in Docs, Sheets & Slides*. URL: https://support.google.com/docs/answer/10519035 (visited on 2025-01-24).

[66] Chengzheng Sun et al. “Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems”. In: *ACM Trans. Comput.-Hum. Interact.* 5.1 (1998-03), pp. 63–108. ISSN: 1073-0516. DOI: 10.1145/274444.274447.

[67] Yusuf Sulistyo Nugroho, Hideaki Hata, and Kenichi Matsumoto. “How different are different diff algorithms in Git?” en. In: *Empirical Software Engineering* 25.1 (2020-01), pp. 790–823. ISSN: 1573-7616. DOI: 10.1007/s10664-019-09772-z.

[68] Blackmagic Design. *DaVinci Resolve 19 – Collaboration*. URL: https://www.blackmagicdesign.com/products/davinciresolve/collaboration (visited on 2025-01-24).

[69] Mike Skalnik. *3D File Diffs*. en-US. 2013-09. URL: https://github.blog/news-insights/product-news/3d-file-diffs/ (visited on 2024-12-27).

[70] Peter Evans-Greenwood, Rosemary Stockdale, and Tim Patston. *The digital-ready workplace*. en-us. 2021-05. URL: https://www2.deloitte.com/us/en/insights/focus/technology-and-the-future-of-work/supercharging-teams-in-the-digital-workplace.html (visited on 2024-10-30).

[71] Bernd Essmann and Thorsten Hampel. “A design pattern for mobile-distributed knowledge spaces”. In: *Proceedings of the 2005 symposia on Metainformatics*. MIS ’05. New York, NY, USA: Association for Computing Machinery, 2005-11, 3–es. DOI: 10.1145/1234324.1234327.

[72] Yasushi Saito and Marc Shapiro. “Optimistic replication”. In: *ACM Comput. Surv.* 37.1 (2005-03), pp. 42–81. ISSN: 0360-0300. DOI: 10.1145/1057977.1057980.

A. Responses from the Digital Questionnaire

A total of 37 responses were collected through the questionnaire, of which 34 were full completions. The questionnaire began with an explanation of the four main work modes discussed in this thesis, including an example categorization. These work modes are synchronous, asynchronous, (asynchronous-) independent, and (asynchronous-) offline. All questions were optional and good be skipped or left blank. Conditional logic was used to leave out non-applicable or duplicate questions.

Questionnaire Introduction

Introductory Text

Hi!
Thanks for helping us with our research about digital collaboration tools!
Our goal is to find out what makes these tools useful, which types are used most often, and also how people end up using them.

The survey should only take around 10-15 minutes!
You can answer in English or German for each question, whichever is easier.

Topic Explanation (Part 1)

This survey is about digital collaboration tools, more specifically any tools (apps, programs, or websites) that let you work on a project together with others and that let you **modify** work made by other people (and let other people modify your work).

We will ask you about different types of collaboration tools/apps/programs. The next page will have some examples, but here's an overview of the four types:

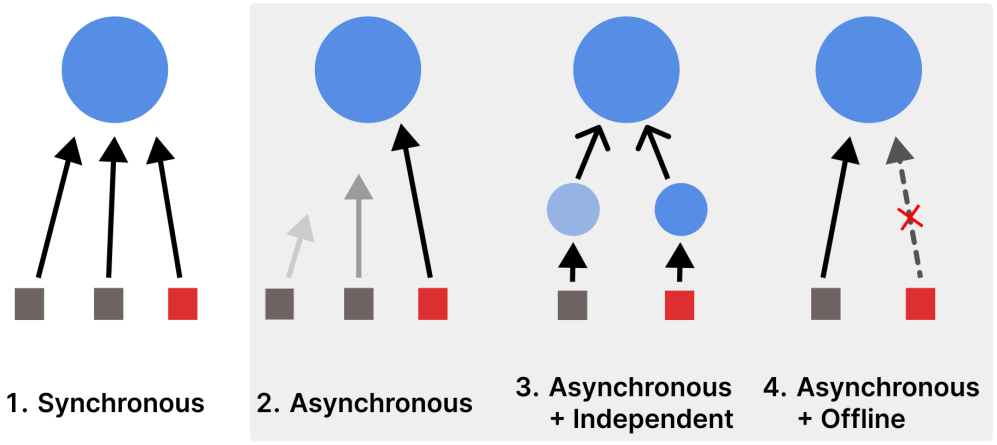
1. **Synchronous ("Real-Time")**
2. **Asynchronous**
3. **Asynchronous + Independent**
4. **Asynchronous + Offline**

In tools supporting **Synchronous** work, multiple people can work together at the same time.

In **Asynchronous** tools, changes are made or exchanged at different times, like you working alone after your collaborators added some changes a few hours ago. Types 2-4 are all some form of asynchronous tools.

Tools can belong to multiple types. For example, a tool might allow you to keep working on a project if no others are online (asynchronous), but once they come back online, you can work together in real time (synchronous).

Topic Explanation (Part 2) - The four types of collaboration tools



1. **Synchronous:** Drawing together on an online whiteboard or having a co-writing session in Google Docs. Synchronous is anything where multiple people are working together at the same time.
2. **Asynchronous:** Making edits to a shared document on your own (before and/or after others make changes), doing some work on your local copy of a project (e.g. programming on your laptop), or working on something while you are offline. Asynchronous is anything where you and your collaborators are not working at the same time or not on the same version of a project.
3. **Asynchronous-Independent:** Any tools that let you add changes to something like a private ”draft” before sharing your changes with your collaborators, or working on a local branch in Git. Asynchronous-Independent always offers a separate version of a project where you can work on without getting distracted (or distracting others) with new changes, before sharing the changes with others.
4. **Asynchronous-Offline:** Working on a shared document while you don’t have internet (like on a plane). Asynchronous-Offline apps let you keep working even when you’re offline for an extended duration.

Lets look at a **concrete example**: Google Docs

Google Docs allows you to work on a (shared) document on your own, whenever you have time, even if none of your collaborators are online. It is therefore an **asynchronous** tool. But if your collaborators are online at the same time as you, you can work together on the document and see each other’s changes in real-time. That means it is also **synchronous**. If you have the Google Drive app or an additional browser extension installed you can even download your documents and continue to work on them even when you are offline. However, as soon as you come back online, your changes are immediately applied to the online version of the documents that is shared with your collaborators. That means Google Docs supports working **asynchronous+offline**, but it does NOT support asynchronous+independent (because your changes always end up being applied to the shared version of the document).

Questions & Responses

1. Have you ever used any digital tools/apps/programs to collaborate with other people SYNCHRONOUSLY / IN REAL TIME? *(n=37)*

Yes	91.89%
No	8.11%

2. Which tools/apps/programs did you use to work synchronously with others? *(n=34)*

Shown only to those who stated they used synchronous tools before.

Google Docs/Sheets/Slides	73.53%
Microsoft 365 Word/Excel/PowerPoint	55.88%
ShareLaTeX / Overleaf	50.0%
Miro	26.47%
Trello	17.65%
Notion	14.71%
Visual Studio (Code) Live Share	14.71%
Jira	14.71%
Figma / FigJam	8.82%

Custom Responses:

- JetBrains (IntelliJ IDEA) Code With Me (x3)
- Strato HiDrive Office (x1)
- GWDG Pad (x1)
- Zoom Whiteboard (x1)
- typst (x1)

3. Of these tools/apps/programs, which one are you most familiar with? *(n=34)*

Shown only to those who stated they used synchronous tools before.
Some responses listed more than one tool.

- Google Docs (x11)
- Microsoft Word/Excel/PowerPoint (x9)
- ShareLaTeX / Overleaf (x7)
- Jira (x2)
- Miro (x2)
- typst (x1)
- JetBrains Code With Me (x1)
- GWDG Pad (HedgeDoc) (x1)
- All (x1)

4. Have you used *<most familiar synchronous tool>* in the past 6 months? *(n=34)*

Shown only those who stated they used synchronous tools before.

Yes	94.89%
No	5.11%

5. Do you use *<most familiar synchronous tool>* at least once a month? *(n=32)*

Shown only to those who stated they used synchronous tools before.

Yes	87.5%
No	9.38%
Not sure	3.13%

6. Why did you use *<most familiar synchronous tool>*? *(n=34)*

Shown only to those who stated they used synchronous tools before.

Because I wanted to use it myself	52.94%
Because my collaborators were already using it	52.94%
Because I had to use it (i.e. demanded by employer)	47.06%

Custom Responses:

- Because it‘s free and/or in our students program available
- leicht zu beschaffen und zu installieren (*Easy to obtain and install*)

7. What did you use *<most familiar synchronous tool>* for? *(n=34)*

Shown only to those who stated they used synchronous tools before.

- Work for my job Collaborating with friends to plan events
- Taking meeting notes and writing reports for work
- Übungen (*exercises*)
- Project management, document editing, presentations and slide creation
- Group work uni
- Group work, where many people had to work and access on the same file
- Everythibg
- University document creation
- Collaboratong on Papers
- sharing docs

- Erstellung von Präsentationen gemeinsam mit Kollegen (jeder hat seinen Teil aber alles in einer Präsentation) (*creation of presentations with colleagues (everyone has their part but all in one presentation)*)
- Note taking, preparing for University courses
- Uni Assignments, Documents that should look professional
- Planning Trips with Others and private stuff
- work on (group-) exercises/tasks with a written submission
- I used it for every Uni assignment where I needed to collaborate with others and hand in a PDF.
- Presentation of technical analyses
- Gemeinsame Erarbeitung von Studienleistungen (Hausarbeiten, Präsentation) (*collaborative compilation of course work (essays, presentations)*)
- record rough ideas with others make quick todo lists
- Creative work (songwriting)
- Writing documents
- online collaborative document editing
- Wir arbeiten mit GoogleDocs gemeinsam an Projektanträgen, Einverständniserklärungen, Ethikanträgen, etc. (*We use Google Docs to work together on project applications, consent forms, ethic applications, etc.*)
- Writing a specification document
- We use it for planning and to organize and document our work. We have boards of all kinds (scrum and kanban).
- Gathering ideas for - projects - improvement of work conditions
- Ideensammlung Teamevent (*idea collection team event*)
- Thesis, Projects, Seminares
- RFCs, Technical Designs, Documentation
- Group Project for university
- Coding
- voluntary work - protocol, documentation
- Exercises for my courses

8. Overall, what kind of impact did *<most familiar synchronous tool>* have on your life or work life? *(n=34)*

Shown only to those who stated they used synchronous tools before.

Rating type:	1	2	3	4	5
Smileys	(Negative)				(Positive)
Percentage	0%	0%	5.88%	41.18%	52.94%
Average rating:					4.47 / 5

9. Did you have any issues or interesting experiences while working SYNCHRONOUSLY / IN REAL TIME with any of these tools in general (not just *<most familiar synchronous tool>*)? *(n=21)*

Shown only to those who stated they used synchronous tools before.

- I had issues with people's comments being missing in Word sometimes. I also found it difficult to keep track of everyone's changes and comments when there were a lot of people collaborating on a document.
- The synchronisation of Microsoft Word 365 is embarrassingly slow and buggy
- Sync Fehler, zu langsame Synchronisierung (*sync errors, too slow synchronization*)

- Depending in the device (used laptop/macbook) there can be different errors impacting everyone because of failed update uploading
- Sonetimes people Fleete work
- Sometimes connection is lost but I don't get any feedback. Then suddenly changes applied were removed because it got synced again.
- I quite like it. The only bummer is, that the pads are per default public. Collaboration works like a charm
- In the case of JetBrains, the state of the changes can sometimes desync.
- No always work really well.
- See exactly what changes have been made by colleagues
- Microsoft 365 word: Man sieht schnell, was andere schreiben (*one quickly sees what others are writing*)
  Microsoft PowerPoint: nicht möglich, gleichzeitig auf einer Folie zu arbeiten (*not possible to work on a single slide at the same time*)
  Google Docs: reagiert langsam, wenn man was schreiben will (*reacts slowly when wanting to write something*)
  ShareLaTeX: sehr schwer Fehler zu entdecken, wenn einer etwas falsch gecodet hat, sehr schnell reagiert das Programm dann nicht mehr (*very hard to spot errors when someone coded something incorrectly, the program doesn't respond quickly / quickly doesn't respond in that case*)
- Docs - unexpected moving of text when someone inserts an image above
  sharelateLatex - Renederissues because of otherwise edited code
- Hard to retrace who wrote which part; laborious to compare comments or text fields
- No
- Sehr praktisch, erleichtert die gemeinsame Arbeit wirklich enorm. (*very convenient, really makes the joint work a lot easier*)
- Sometimes it's not available. But mostly everthings works fine.
- With many people (>10)involved in the same MS Teams meeting, keeping track of all the changes was getting hard, as I had to constantly scroll and zoom in on items on the screen. A larger monitor would have helped a lot...
- Yes, sometimes Microsoft Word 365 it doesn't sync great / in real realtime
- Very fast working pace
- Online dependence, bad versioning
- Sometimes when multiple people write at the same location the result is unexpexted

10. Would you like to start using such real-time collaboration tools/apps/programs? *(n=3)*

Shown only to those who stated they never used synchronous tools

Yes	0%
No	33.33%
Not sure	66.67%

11. Why would you not like to use real-time collaboration tools/apps/programs? *(n=1)*

*Shown only to those who stated they never used synchronous tools **and** selected "No" in the previous question*

- I have zero need to "collaborate in real-time" in any sort of work

12. Have you ever used any digital tools/apps/programs to collaborate with other people ASYNCHRONOUSLY / AT ANY TIME (even when you or they aren't online)? *(n=36)*

Yes	88.89%
No	11.11%

13. Which tools/apps/programs did you use to work asynchronously with other people? *(n=32)*

Shown only to those who stated they used asynchronous tools before.

Git	68.75%
Microsoft 365 Word/Excel/PowerPoint	59.38%
Google Docs/Sheets/Slides	50.0%
ShareLaTeX / Overleaf	40.63%
Miro	15.63%
Jira	15.63%
Trello	12.5%
Notion	9.38%
DaVinci Resolve	9.38%
Figma / FigJam	6.25%
Siemens NX / Teamcenter	3.13%

Custom Responses:

- Visio (x1)
- Obsidian (x1)
- OneNote (x1)
- Pads/Docs 3rd party (x1)

14. Of these tools/apps/programs, which one are you most familiar with? *(n=31)*

Shown only to those who stated they used asynchronous tools before.

- Git (x14), GitHub (x1)
- Microsoft Word/Excel/PowerPoint (x7)
- Google Docs (x5)
- OneNote (x1)
- Jira (x1)
- ShareLaTeX (x1)
- Pads (x1)

15. Have you used <*most familiar asynchronous tool*> in the past 6 months? *(n=25)*

Shown only to those who stated they used asynchronous tools before.
If the most familiar asynchronous tool was also the most familiar synchronous tool, this question was skipped, since it was answered previously.

Yes	96.0%
No	4.0%

16. Do you use <*most familiar asynchronous tool*> at least once a month? *(n=24)*

Shown only to those who stated they used asynchronous tools before.
If the most familiar asynchronous tool was also the most familiar synchronous tool, this question was skipped, since it was answered previously.

Yes	83.33%
No	12.5%
Not sure	4.17%

17. Why did you use <*most familiar asynchronous tool*>? *(n=25)*

Shown only to those who stated they used asynchronous tools before.
If the most familiar asynchronous tool was also the most familiar synchronous tool, this question was skipped, since it was answered previously.

Because I wanted to use it myself	84.0%
Because my collaborators were already using it	60.0%
Because I had to use it (i.e. demanded by employer)	60.0%

Custom Responses:

- Organization of a family event

18. Overall, what kind of impact did <*most familiar asynchronous tool*> have on your life or work life? *(n=25)*

Shown only to those who stated they used asynchronous tools before.
If the most familiar asynchronous tool was also the most familiar synchronous tool, this question was skipped, since it was answered previously.

Rating type:	1	2	3	4	5
Smileys	(Negative)				(Positive)
Percentage	0%	0%	12.0%	32.0%	56.0%
Average rating:					4.44 / 5

19. Do any of these tools/apps/programs let you work on your own version, INDEPENDENTLY (asynchronous + independent) of your collaborators? *(n=31)*

Shown only to those who stated they used asynchronous tools before.
The list of previously selected asynchronous tools was shown.

Yes	87.1%
No	12.9%

20. Which of the tools/apps/programs that let you do this (work independently / on a separate version) are you most familiar with? *(n=27)*

Shown only to those who stated they used independent-capable tools before.

- Git (x19), GitHub (x1)
- Microsoft Word/Excel/PowerPoint (x6)
- Google Docs (x1)

21. Did you ever use that functionality of *<most familiar asynchronous-independent tool>*? *(n=27)*

Shown only to those who stated they used independent-capable tools before.

Yes	92.59%
No	7.41%

22. What was your reason for using that functionality of *<most familiar asynchronous-independent tool>*? *(n=22)*

Shown only to those who stated they used that functionality.

- I wanted to work on a draft without everyone seeing my changes that were in progress.
- It is the default
- No Internet Connection
- To save many versions with changes and to be able to go back to an older version
- Working on different problems so that they can be merged into the main version. Mostly work related use
- Software developmemt
- Well, software development works like that
- My changes can be reviewed by the others, before merging them into the final product. Also, people can work simultaneously without breaking each other's changes
- To add functionality to already working Programms/scripts
- Tasks were distributed in the Group
- Different people can work on different aspects of the project without interfering with another. And it is easy to merge the work in the main project.
- Offline Working
- quick protoype - try something
- Different feature
- working on my own branch, writing horrible code to fix before pushing

- Thats just how git works
- programming
- To develop features seperately
- Starting to write a document, locally, without sharing it (at first)
- Works well
- I can test and play around with my version, without endangering other peoples work.
- Feature branches where different people work on different features

23. How often do you use that functionality of *<most familiar asynchronous-independent tool>*? *(n=24)*

Shown only to those who stated they used that functionality

At least once a day	20.83%
At least once a week	54.17%
At least once a month	25.0%

24. What do you think of this functionality of *<most familiar asynchronous-independent tool>*? *(n=25)*

Shown only to those who stated they used that functionality.

Rating type:	1	2	3	4	5
Stars	(Hate it)				(Love it)
Percentage	0%	0%	4.0%	36.0%	60.0%
Average rating:					4.56 / 5

25. Overall, what kind of impact did *<most familiar asynchronous-independent tool>* have on your life or work life? *(n=11)*

Shown only to those who stated they used that functionality ***and*** *who hadn't answered that question about this tool earlier.*

Rating type:	1	2	3	4	5
Smileys	(Negative)				(Positive)
Percentage	0%	0%	27.27%	54.55%	18.18%
Average rating:					3.91 / 5

26. Would you like to start using such tools/apps/programs that let you work independently on a separate version? *(n=5)*

Shown only to those who stated they never used independent-capable tools.

Yes	60.0%
No	20.0%
Not sure	20.0%

27. Why would you like to start using tools/apps/programs that let you work independently on a separate version? *(n=3)*

*Shown only to those who stated they never used independent-capable tools **and** selected "Yes" in question 26.*

- Unausgereifte Ideen werden nicht sofort geteilt. Man kann dann nur teilen, was auch wirklich am Ende relevant ist. (*immature ideas aren't shared immediately. one can then share only what is really relevant in the end*)
- Hopefully more overview, less distraction
- to prevent fear of loss of data caused by collaboration

28. Why would you not like to start using tools/apps/programs that let you work independently on a separate version? *(n=1)*

*Shown only to those who stated they never used independent-capable tools **and** selected "No" in question 26.*

- Scheint gegen synchronen Tools keine Vorteile zu bringen (zumindest nicht bezüglich meiner Arbeiten). (*seems to offer no benefits compared to synchronous tools (at least not regarding my work)*)

29. Do any of these tools/apps/programs let you work WHEN YOU ARE OFFLINE (asynchronous + offline)? *(n=31)*

The list of previously selected asynchronous tools was shown.

Yes	80.65%
No	19.35%

30. Which of the tools/apps/programs that let you do this (work offline) are you most familiar with? *(n=25)*

Shown only to those who stated they used offline-capable tools before.

- Git (x16), GitHub (x1)
- Microsoft Word/Excel/PowerPoint (x8)
- Obsidian (x1)

31. Did you ever use that functionality of *<most familiar asynchronous-offline tool>*? *(n=20)*

Shown only to those who stated they used offline-capable tools before.

Yes	88.0%
No	12.0%

32. What was your reason for using that functionality of <*most familiar asynchronous-offline tool*>? *(n=20)*

Shown only to those who stated they used that functionality.

- It is the default
- Normale Nutzung/ Kein Internet (*normal usage / no internet*)
- So it is just for me and also i am scared and paranoid about the cloud, so private/sensitive stuff is all offline
- Independency of internet access. Work office uses DSL and can't handle as much synchronous applications simultaneously.
- Code development
- Kein Internet (*no internet*)
- I was offline, the trains in Germany don't habe great internet connections
- Local Changes can bei stored and updated If i am online
- I had no internet connection
- No internet.
- If working online is not safe enough (airport, railway, etc.)
- Kein Internetzugang (*no internet access*)
- on the go with poor connection
- No stress
- no internet on train
- No internet access - I used a local copy of MS Word
- To further develop my code
- Offline, that almost doesn't exist 😅. But, just without thinking about the internet, at first. Yes. Creating some documentation.
- Portability, flexibility
- No Internet while traveling Full control of data

33. Do you use <*most familiar synchronous tool*> at least once a month? *(n=20)*

Shown only to those who stated they used that functionality.

At least once a day	15.0%
At least once a week	35.0%
At least once a month	20.0%
At least once a year	30.0%

34. What do you think of this functionality of <*most familiar asynchronous-offline tool*>? *(n=22)*

Shown only to those who stated they used that functionality.
Using smileys instead of stars for the rating was an oversight and is inconsistent with question 24.

Rating type:	1	2	3	4	5
Smileys	(Hate it)				(Love it)
Percentage	0%	0%	4.55%	54.55%	40.91%
Average rating:					4.36 / 5

35. Overall, what kind of impact did <*most familiar asynchronous-offline tool*> have on your life or work life? *(n=8)*

*Shown only to those who stated they used that functionality **and** who hadn't answered that question about this tool earlier.*

Rating type:	1	2	3	4	5
Smileys	(Negative)				(Positive)
Percentage	0%	0%	12.5%	37.5%	50.0%
Average rating:					4.38 / 5

36. Would you like to start using such tools/apps/programs that let you work offline? *(n=12)*

Shown only to those who stated they never used offline-capable tools.

Yes	41.67%
No	33.33%
Not sure	25.0%

37. Why would you like to start using tools/apps/programs that let you work offline? *(n=3)*

*Shown only to those who stated they never used offline-capable tools **and** selected "Yes" in question 36.*

- Mehr Flexibilität (*more flexibility*)
- less distraction - independence from internet access
- For Trainrides/Commutes

38. Why would you not like to start using tools/apps/programs that let you work offline? *(n=3)*

*Shown only to those who stated they never used offline-capable tools **and** selected "No" in question 36.*

- I don't need it
- Same, scheint keine Vorteile zu bringen bei unserer Projektarbeit. (*the same, it doesn't seem to bring any advantages for our project work*)
- By policy, I can't work offline.

39. Did you use any tools/apps/programs that did NOT allow you to work either on your own version (asynchronous+independent) or offline (asynchronous+offline)? *(n=30)*

Yes	33.33%
No	66.67%

40. Which tool/app/program that wouldn't let you do that are you most familiar with? *(n=12)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.

- ShareLaTeX / Overleaf (x4)
- Google Docs (x3)
- Miro (x2)
- OneNote (x1)
- Discord (x1)
- Work related stuff (x1)

41. Why did you use <*most familiar non-independent+non-offline tool*>? *(n=9)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.
If the most familiar non-independent+non-offline tool was also the most familiar synchronous or asynchronous tool, this question was skipped, since it was answered previously.

Because I wanted to use it myself	33.33%
Because my collaborators were already using it ..	66.67%
Because I had to use it (i.e. demanded by employer)	33.33%

42. What did you use <*most familiar non-independent+non-offline tool*> for? *(n=9)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.

- Price comparison and generally excel sheet work
- Collaborating in creating documents
- Uni assignments.
- writing stuff for university
- Collaborative work
- Dokumentation, planning tasks, notes
- University homework
- Team communication
- protocol

43. What do you think of <*most familiar non-independent+non-offline tool*>? *(n=9)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.

Rating type:	1	2	3	4	5
Stars	(Hate it)				(Love it)
Percentage	0%	11.11%	33.33%	44.44%	11.11%
Average rating:					3.56 / 5

44. Overall, what kind of impact did *<most familiar non-independent+non-offline tool>* have on your life or work life? *(n=9)*

*Shown only to those who stated they used tools that support neither the independent nor the offline work mode **and** who hadn't answered that question about this tool earlier.*

Rating type:	1	2	3	4	5
Smileys	(Negative)				(Positive)
Percentage	0%	0%	33.33%	44.44%	22.22%
Average rating:					4.89 / 5

45. How USEFUL do you think it would be if *<most familiar non-independent+non-offline tool>* would let you work on a SEPARATE VERSION before adding your changes to the main version (asynchronous+independent)? *(n=12)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.

Rating type:	1	2	3	4	5
Numbers	(Not useful / unchanged)				(Very useful)
Percentage	25%	8.33%	25.0%	41.67%	0%
Average rating:					2.83 / 5

46. How USEFUL do you think it would be if *<most familiar non-independent+non-offline tool>* would let you work even WITHOUT AN INTERNET CONNECTION (asynchronous+offline)? *(n=12)*

Shown only to those who stated they used tools that support neither the independent nor the offline work mode.

Rating type:	1	2	3	4	5
Numbers	(Not useful / unchanged)				(Very useful)
Percentage	0%	22.22%	22.22%	33.33%	22.22%
Average rating:					3.56 / 5

47. Did you have any issues or interesting experiences with any of these asynchronous tools in general? (Please mention the tools you're referring to) *(n=10)*

Shown only to those who stated they used asynchronous tools before.

- Docs, sharelatex: no possibility of working on a version without any internet access.
- Merging in Git can be frustrating. I like version control feature.
- my whole life runs on git and i love it. it is all i need for everything i do. there is literally no use for any other synchronization tools (apart from real-time collaboration). i love git and plain text.

- OneNote has had issues with synchronization and account logins in the past
- Keeping track of changes and comments others have done in MS Office documents can get a little confusing ;-)
- Bisher wenig Erfahrung. Diese waren positiv, da viele schnell gemeinsam bearbeiten konnten. (*so far little experiences. these were positive, since many could quickly and jointly edit.*)
- The worst of all is when using Microsoft Word 365, and that you sometimes don't have the latest version, and changes others make CAN take a lot time.
- Microsoft 365 Word / Excel / PowerPoint it is a pain to merge versions

48. Would you like to start using such asynchronous collaboration tools/apps/programs? *(n=5)*

Shown only to those who stated they never used asynchronous tools.

Yes	0%
No	40.0%
Not sure	60.0%

49. Why would you not like to use asynchronous collaboration tools/apps/programs? *(n=1)*

*Shown only to those who stated they never used asynchronous tools **and** selected "No" in question 49.*

- I generally only work independently and usually solo on any online-type task.

50. Is there anything you would like to change about the collaboration tools/apps/programs (of ANY type) you're currently using? *(n=20)*

- No (x7)
- If somehow a version was saved on my PC that synchronizes with the online version if internet access was available. That would be nice
- More IDEs to have the ability to Code together.
- Overleaf's vim mode isn't great. So I would like it if it improved. But to be fair, general vim motion doesn't work great in browsers in general.
- Freeze the final version of the document
- Schnelleres gleichzeitiges Arbeiten auf einer Folie/Seite (*faster simultaneous work on a single slide/page*)
- full offline support good version control
- Better visibility of different authors while working live and maybe ways to interact (sign them to stop or direct their focus on a certain part)
- Google Docs having version control and having full versions' features.
- Sometimes I need to refresh the website to see if someone is still working on a document - it could refresh itself
- Something to annotate others work better. Something to easily show my collaborators where I am and what I'm doing (f.ex. a pointer/mouse made visibly for every team member or a marker to highlight a certain region in the document)
- I would like to have realtime Microsoft Word like Google Docs mostly is, as far as I remember.
- Using tool for planning and project management
- I want a live tool which integrates with delayed pushing of independent local changes. A bit like HackMD with GitHub